

目录

转换语言参考-版权.....	6
Developer 转换语言参考-版权.....	9
前言.....	9
Informatica 资源.....	10
转换语言.....	11
转换语言概览.....	11
表达式语法.....	12
向表达式添加注释.....	15
保留字.....	15
常量.....	16
DD_DELETE.....	16
DD_INSERT.....	17
DD_REJECT.....	17
DD_UPDATE.....	18
FALSE.....	18
NULL.....	19
TRUE.....	20
运算符.....	21
运算符优先级.....	21
复杂运算符.....	22
算术运算符.....	30
字符串运算符.....	31
比较运算符.....	31
逻辑运算符.....	33
变量.....	33
内置变量.....	33
事务控制变量.....	38
局部变量.....	39
日期.....	39
日期概览.....	39
日期格式字符串.....	43

TO_CHAR 格式字符串.....	44
TO_DATE 和 IS_DATE 格式字符串.....	47
了解日期算术运算.....	50
函数.....	51
函数类别.....	51
ABORT.....	59
ABS.....	60
ADD_TO_DATE.....	61
AES_DECRYPT.....	64
AES_ENCRYPT.....	64
ANY.....	65
ARRAY.....	67
ASCII.....	67
AVG.....	68
CAST.....	70
CEIL.....	70
CHOOSE.....	71
CHR.....	72
CHRCODE.....	73
COLLECT_LIST.....	74
COLLECT_MAP.....	75
COMPRESS.....	76
CONCAT.....	76
CONCAT_ARRAY.....	78
CONVERT_BASE.....	79
COS.....	79
COSH.....	80
COUNT.....	81
CRC32.....	84
CREATE_TIMESTAMP_TZ.....	84
CUME.....	85
DATE_COMPARE.....	86
DATE_DIFF.....	87
DEC_BASE64.....	90
DECODE.....	91
DECOMPRESS.....	93

ENC_BASE64.	94
ERROR.	94
EXP.	95
FIRST.	96
FLOOR.	97
FV.	98
GET_DATE_PART.	99
GET_TIMEZONE.	101
GET_TIMESTAMP.	102
GREATEST.	103
IIF.	104
IN.	107
INDEXOF.	108
INITCAP.	109
INSTR.	109
ISNULL.	112
IS_DATE.	114
IS_NUMBER.	116
IS_SPACES.	118
LAG.	119
LAST.	120
LAST_DAY.	121
LEAD.	122
LEAST.	124
LENGTH.	125
LN.	126
LOG.	127
LOOKUP.	127
LOWER.	129
LPAD.	130
LTRIM.	131
MAKE_DATE_TIME.	132
MAP.	133
MAP_FROM_ARRAYS.	134
MAP_KEYS.	135
MAP_VALUES.	136

MAX (Dates).	136
MAX (Numbers).	137
MAX (String).	139
MD5.	140
MEDIAN.	141
METAPHONE.	142
MIN (Dates).	146
MIN (Numbers).	147
MIN (String).	148
MOD.	149
MOVINGAVG.	151
MOVINGSUM.	152
NPER.	153
PERCENTILE.	154
PMT.	156
POWER.	157
PV.	158
RAND.	158
RATE.	159
REG_EXTRACT.	160
REG_MATCH.	162
REG_REPLACE.	163
REPLACECHR.	164
REPLACESTR.	167
RESPEC.	170
REVERSE.	171
ROUND (Dates).	172
ROUND (Numbers).	176
RPAD.	178
RTRIM.	179
SETCOUNTVARIABLE.	180
SET_DATE_PART.	182
SETMAXVARIABLE.	184
SETMINVARIABLE.	186
SETVARIABLE.	188
SIGN.	189

SIN.	190
SINH.	191
SIZE.	192
SOUNDEX.	193
SQL_LIKE.	195
SQRT.	196
STDDEV.	197
STRUCT.	198
STRUCT_AS.	200
SUBSTR.	200
SUM.	203
SYSTIMESTAMP.	204
TAN.	205
TANH.	206
TIME_RANGE.	207
TO_BIGINT.	208
TO_CHAR (Dates).	209
TO_CHAR (Numbers).	214
TO_DATE.	216
TO_DECIMAL.	219
TO_DECIMAL38.	221
TO_FLOAT.	222
TO_INTEGER.	223
TO_TIMESTAMP_TZ.	224
TRUNC (Dates).	225
TRUNC (Numbers).	228
UPPER.	230
UUID4.	231
UUID_UNPARSE.	231
VARIANCE.	232
创建自定义函数.	233
创建自定义函数概览.	233
步骤 1. 获取存储库 ID 属性.	234
步骤 2. 创建表头文件.	234
步骤 3. 创建实现文件.	236
步骤 4. 构建模块.	248

步骤 5.创建存储库插件文件.....	249
步骤 6.测试自定义函数.....	252
安装自定义函数.....	253
用自定义函数创建表达式.....	253
自定义函数 API 引用.....	254
自定义函数 API 引用概览.....	254
常用 API.....	254
运行时 API.....	258

转换语言参考-版权

本软件和文档仅根据包含使用与披露限制的单独许可协议提供。未事先征得 Informatica LLC 同意，不得以任何形式、通过任何手段（电子、影印、录制或其他手段）复制或传播本文档的任何部分。

Informatica、Informatica 标志和 PowerCenter 是 Informatica LLC 在美国和世界其他许多司法管辖区的商标或注册商标。欲获得 Informatica 商标的最新列表，请访问 <https://www.informatica.com/trademarks.html>。其他公司和产品名称可能是其各自所有者的商业名称或商标。

美国政府权利交付给美国政府客户的程序、软件、数据库及相关文档和技术数据是指适用的联邦采购条例和政府机构特定补充条例中定义的"商业计算机软件"或"商业技术数据"。因此，使用、复制、披露、修改和改编应遵循适用的政府合同中规定的限制和许可条款、政府合同条款的适用范围以及 FAR 52.227-19 商用计算机软件许可中规定的额外权利。

本软件和/或文档的某些部分受第三方版权制约，包括但不限于：版权所有 DataDirect Technologies。保留所有权利。版权所有 (C) Sun Microsystems。保留所有权利。版权所有 (C) RSA Security Inc. 保留所有权利。版权所有 (C) Ordinal Technology Corp. 保留所有权利。版权所有 (C) Aandacht c.v. 保留所有权利。版权所有 Genivia, Inc. 保留所有权利。版权所有 Isomorphic Software。保留所有权利。版权所有 (C) Meta Integration Technology, Inc. 保留所有权利。版权所有 (C) Intalio。保留所有权利。版权所有 (C) Oracle。保留所有权利。版权所有 (C) Adobe Systems Incorporated。保留所有权利。版权所有 (C) DataArt, Inc. 保留所有权利。版权所有 (C) ComponentSource。保留所有权利。版权所有 (C) Microsoft Corporation。保留所有权利。版权所有 (C) Rogue Wave Software, Inc. 保留所有权利。版权所有 (C) Teradata Corporation。保留所有权利。版权所有 (C) Yahoo! Inc. 保留所有权利。版权所有 (C) Glyph & Cog, LLC。保留所有权利。版权所有 (C) Thinkmap, Inc. 保留所有权利。版权所有 (C) Clearpace Software Limited。保留所有权利。版权所有 (C) Information Builders, Inc. 保留所有权利。版权所有 (C) OSS Nokalva, Inc. 保留所有权利。版权所有 Edifecs, Inc. 保留所有权利。版权所有 Cleo Communications, Inc. 保留所有权利。版权所有 (C) International Organization for Standardization 1986。保留所有权利。版权所有 (C) ej-technologies GmbH。保留所有权利。版权所有 (C) Jaspersoft Corporation。保留所有权利。版权所有 (C) International Business Machines Corporation。保留所有权利。版权所有 (C) yWorks GmbH。保留所有权利。版权所有 (C) Lucent Technologies。保留所有权利。版权所有 (c) University of Toronto。保留所有权利。版权所有 (C) Daniel Veillard。保留所有权利。版权所有 (C) Unicode, Inc. 版权所有 IBM Corp. 保留所有权利。版权所有 (C) MicroQuill Software Publishing, Inc. 保留所有权利。版权所有 (C) PassMark Software Pty Ltd. 保留所有权利。版权所有 (C) LogiXML, Inc. 保留所有权利。版权所有 (C) 2003-2010 Lorenzi Davide，保留所有权利。版权所有 (C) Red Hat, Inc. 保留所有权利。版权所有 (C) The Board of Trustees of the Leland Stanford Junior University。保留所有权利。版权所有 (C) EMC Corporation。保留所有权利。版权所有 (C) Flexera Software。保留所有权利。版权所有 (C) Jinfonet Software。保留所有权利。版权所有 (C) Apple Inc. 保留所有权利。版权所有 (C) Telerik Inc. 保留所有权利。

利。版权所有 (C) BEA Systems。保留所有权利。版权所有 (C) PDFlib GmbH。保留所有权利。版权所有 (C) Orientation in Objects GmbH。保留所有权利。版权所有 (C) Tanuki Software, Ltd. 保留所有权利。版权所有 (C) Ricebridge。保留所有权利。版权所有 (C) Sencha, Inc. 保留所有权利。版权所有 (C) Scalable Systems, Inc. 保留所有权利。版权所有 (C) jQWidgets。保留所有权利。版权所有 (C) Tableau Software, Inc. 保留所有权利。版权所有 (C) MaxMind, Inc. 保留所有权利。版权所有 (C) TMat Software s.r.o. 保留所有权利。版权所有 (C) MapR Technologies Inc. 保留所有权利。版权所有 (C) Amazon Corporate LLC。保留所有权利。版权所有 (C) Highsoft。保留所有权利。版权所有 (C) Python Software Foundation。保留所有权利。版权所有 (C) BeOpen.com。保留所有权利。版权所有 (C) CNRI。保留所有权利。

本产品包括由 Apache Software Foundation (<http://www.apache.org/>) 开发的软件和/或在不同 Apache 许可证版本（以下简称“许可证”）下许可的其他软件。您可从 <http://www.apache.org/licenses/> 获取这些许可证的副本。除非适用法律要求或者有相应书面协议，否则依据这些“许可证”分发的软件以“原样”提供，不附带任何明示或暗示的担保或条件。请参阅“许可证”中规定的具体语言管理权限和限制。

本产品包括由 Mozilla (<http://www.mozilla.org/>) 开发的软件、由 JBoss Group, LLC 开发的软件（版权所有 JBoss Group, LLC 保留所有权利）、由 Bruno Lowagie 和 Paulo Soares 开发的软件（版权所有 (C) 1999-2006 Bruno Lowagie 和 Paulo Soares）以及在 <http://www.gnu.org/licenses/lgpl.html> 网站上的不同版本 GNU Lesser General 公共许可协议下许可的软件。这些材料由 Informatica 按“原样”免费提供，不附带任何明示或暗示的担保，包括但不限于适销性和特定用途适用性的暗示担保。

本产品包括 ACE(TM) 和 TAO(TM) 软件，这些软件版权归 Douglas C. Schmidt 及其在华盛顿大学、加利福尼亚大学欧芬分校以及范德堡大学的研发团队所有（版权所有 ((C)) 1993-2006，保留所有权利）。

本产品包括由 OpenSSL Project 开发并在 OpenSSL Toolkit（版权所有 OpenSSL Project。保留所有权利）中使用的软件，该软件的再分发受 <http://www.openssl.org> 和 <http://www.openssl.org/source/license.html> 上规定条款之制约。

本产品包括 Curl 软件，版权所有 1996-2013, Daniel Stenberg <daniel@haxx.se>。保留所有权利。有关该软件的权限和限制受 <http://curl.haxx.se/docs/copyright.html> 上规定条款之制约。允许出于任何目的以免费或收费形式使用、复制、修改和分发该软件，但前提是所有副本均应注明上述版权声明以及本许可声明。

本产品包括由 MetaStuff, Ltd. 开发的软件，版权所有 2001-2005 ((C)) MetaStuff, Ltd. 保留所有权利。有关该软件的权限和限制受 <http://www.dom4j.org/license.html> 上规定条款之制约。

本产品包括由 Dojo Foundation 开发的软件，版权所有 (C) 2004-2007, Dojo Foundation。保留所有权利。有关该软件的权限和限制受 <http://dojotoolkit.org/license> 上规定条款之制约。

本产品包括 ICU 软件，版权所有 International Business Machines Corporation 和其他方。保留所有权利。有关该软件的权限和限制受 <http://source.icu-project.org/repos/icu/icu/trunk/license.html> 上规定条款之制约。

本产品包括由 Per Bothner 开发的软件，版权所有 (C) 1996-2006 Per Bothner。保留所有权利。<http://www.gnu.org/software/kawa/Software-License.html> 上的许可证中规定了您使用这些材料的权利。

本产品包括 OSSP UUID 软件，版权所有 (C) 2002 Ralf S. Engelschall，版权所有 (C) 2002 OSSP Project，版权所有 (C) 2002 Cable & Wireless Deutschland。有关该软件的权限和限制受 <http://www.opensource.org/licenses/mit-license.php> 上规定条款之制约。

本产品包括由 Boost (<http://www.boost.org/>) 开发的软件或在 Boost 软件许可证下许可的软件。有关该软件的权限和限制受 http://www.boost.org/LICENSE_1_0.txt 上规定条款之制约。

本产品包括由 University of Cambridge 开发的软件，版权所有 (C) 1997-2007 University of Cambridge。有关该软件的权限和限制受 [http:// www.pcre.org/license.txt](http://www.pcre.org/license.txt) 上规定条款之制约。

本产品包括由 The Eclipse Foundation 开发的软件，版权所有 (C) 2007 The Eclipse Foundation。保留所有权利。有关该软件的权限和限制受 <http://www.eclipse.org/org/documents/epl-v10.php> 和 <http://www.eclipse.org/org/documents/edl-v10.php> 上规定条款之制约。

本产品包括在 <http://www.tcl.tk/software/tcltk/license.html>、[http://www.bosrup.com/web/overlib/? License](http://www.bosrup.com/web/overlib/?License)、<http://www.stlport.org/doc/license.html>、<http://asm.ow2.org/license.html>、<http://www.cryptix.org/LICENSE.TXT>、<http://hsqldb.org/web/hsqLicense.html>、<http://httpunit.sourceforge.net/doc/license.html>、<http://jung.sourceforge.net/license.txt>、http://www.gzip.org/zlib/zlib_license.html、<http://www.openldap.org/software/release/license.html>、<http://www.libssh2.org>、<http://slf4j.org/license.html>、<http://www.sente.ch/software/OpenSourceLicense.html>、<http://fusesource.com/downloads/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> 和 <https://sourceforge.net/p/xmlunit/code/HEAD/tree/trunk/LICENSE.txt> 下许可的软件。

本产品包括在 Academic 免费许可证 (<http://www.opensource.org/licenses/afl-3.0.php>)、通用开发和分发许可证 (<http://www.opensource.org/licenses/cddl1.php>)、通用公共许可证 (<http://www.opensource.org/licenses/cpl1.0.php>)、Sun Binary Code 许可协议补充许可条款、BSD 许可证 (<http://www.opensource.org/licenses/bsd-license.php>)、新 BSD 许可证 (<http://opensource.org/licenses/BSD-3-Clause>)、MIT 许可证 (<http://www.opensource.org/licenses/mit-license.php>)、Artistic 许可证 (<http://www.opensource.org/licenses/artistic-license-1.0>) 以及原始开发者公共许可证

版本 1.0 (<http://www.firebirdsql.org/en/initial-developer-s-public-license-version-1-0/>) 下许可的软件。

本产品包括由 Joe Walnes 和 XStream Committers 开发的软件，版权所有 (C) 2003-2006 Joe Walnes，2006-2007 XStream Committers。保留所有权利。有关该软件的权限和限制受 <http://xstream.codehaus.org/license.html> 上规定条款之制约。本产品包括由 Indiana University Extreme! Lab 开发的软件。有关详细信息，请访问 <http://www.extreme.indiana.edu/>。

本产品包括软件版权所有 (c) 2013 Frank Balluffi 和 Markus Moeller。保留所有权利。有关此软件的权限和限制受 MIT 许可证上规定条款之制约。

声明

本 Informatica 产品（以下称“软件”）包括由 Progress Software Corporation 的运营公司 DataDirect Technologies（以下称“DataDirect”）提供的某些驱动程序（以下称“DataDirect 驱动程序”），受以下条款和条件制约：

1. DataDirect 驱动程序以“原样”提供，不附带任何明示或暗示的担保，包括但不限于适销性、特定用途适用性以及非侵权的暗示担保。
2. 在任何情况下，DataDirect 或其第三方供应商均不对最终用户客户承担因使用 ODBC 驱动程序而引起的任何直接、间接、偶发、特殊、继发或其他损害赔偿的责任，无论是否已提前告知该种损害的可能性。这些限制适用于所有诉因，包括但不限于违反合同、违反担保、过失、严格责任、虚假陈述以及其他侵权行为。

本文档中的信息如有更改，恕不另行通知。如发现本文档中有什么问题，请通过以下电子邮件地址向我们报告：infa_documentation@informatica.com。

Informatica 产品根据对应协议的条款和条件进行担保。INFORMATICA 按“原样”提供本文档中的信息，无任何明示或暗示的担保，包括但不限于任何适销性和特定用途适用性担保，也没有任何非侵权担保或条件。

Developer 转换语言参考-版权

本软件和文档仅根据包含使用与披露限制的单独许可协议提供。未事先征得 Informatica LLC 同意，不得以任何形式、通过任何手段（电子、影印、录制或其他手段）复制或传播本文档的任何部分。

Informatica 和 Informatica 标志是 Informatica LLC 在美国和世界其他许多司法管辖区的商标或注册商标。欲获得 Informatica 商标的最新列表，请访问 <https://www.informatica.com/trademarks.html>。其他公司和产品名称可能是其各自所有者的商业名称或商标。

美国政府权利交付给美国政府客户的程序、软件、数据库及相关文档和技术数据是指适用的联邦采购条例和政府机构特定补充条例中定义的“商业计算机软件”或“商业技术数据”。因此，使用、复制、披露、修改和改编应遵循适用的政府合同中规定的限制和许可条款、政府合同条款的适用范围以及 FAR 52.227-19 商用计算机软件许可中规定的额外权利。

本软件和/或文档中的若干部分受第三方版权约束。所需的第三方声明随产品一起提供。

本文档中的信息如有更改，恕不另行通知。如发现本文档中有什么问题，请通过以下电子邮件地址向我们报告：infa_documentation@informatica.com。

Informatica 产品根据对应协议的条款和条件进行担保。INFORMATICA 按“原样”提供本文档中的信息，无任何明示或暗示的担保，包括但不限于任何适销性和特定用途适用性担保，也没有任何非侵权担保或条件。

前言

要了解 Developer tool 中的转换语言，请参阅《*Informatica® Developer 转换语言引用*》。您可以学习如何使用常量、运算符、变量、日期和函数来转换源数据。

要了解 PowerCenter 中的转换语言，请参阅《*PowerCenter® 转换语言引用*》。您可以学习如何使用常量、运算符、变量、日期和函数来转换源数据。

Informatica 资源

Informatica 通过 Informatica Network 和其他在线门户为您提供一系列产品资源。使用这些资源，可以充分利用 Informatica 产品和解决方案，并向其他 Informatica 用户和主题专家学习。

Informatica Network

在 Informatica Network 中可以获得许多资源，包括 Informatica 知识库和 Informatica 全球客户支持。要进入 Informatica Network，请访问 <https://network.informatica.com>。

作为 Informatica Network 成员，您可以选择以下服务：

- 在知识库中搜索产品资源。
- 查看产品可用性信息。
- 创建并检查您的支持案例。
- 查找当地的 Informatica 用户组网络并与您的伙伴进行协作。

Informatica 知识库

使用 Informatica 知识库可查找产品资源，例如操作方法文章、最佳实践、视频教程以及常见问题的答案。

要搜索知识库，请访问 <https://search.informatica.com>。如果您对知识库有任何疑问、意见或建议，请与 Informatica 知识库团队联系，电子邮件地址为 KB_Feedback@informatica.com。

Informatica 文档

使用 Informatica 文档门户可浏览大量当前与最近产品版本的文档库。要浏览文档门户，请访问 <https://docs.informatica.com>。

如果您对产品文档有任何疑问、意见或建议，请与 Informatica 文档团队联系，电子邮件地址为 infa_documentation@informatica.com。

Informatica 产品可用性矩阵

产品可用性矩阵 (PAM) 指明了产品版本支持的操作系统版本、数据库以及数据源和目标的类型。您可以在以下网址中浏览 Informatica PAM：

<https://network.informatica.com/community/informatica-network/product-availability-matrices>。

Informatica Velocity

Informatica Velocity 是由 Informatica 专业服务根据数百个数据管理项目的实际经验所开发出来的，其中汇集了大量使用技巧和最佳实践。Informatica Velocity 代表了 Informatica 顾问的集体知识，这些顾问与世界各地的组织合作，共同计划、开发、部署和维护成功的数据管理解决方案。

您可以在以下网址中找到 Informatica Velocity 资源：<http://velocity.informatica.com>。如果您对 Informatica Velocity 有任何疑问、意见或建议，请通过 ips@informatica.com 与 Informatica 专业服务联系。

Informatica Marketplace

Informatica Marketplace 是一个论坛，该论坛中提供的解决方案可扩展和增强您的 Informatica 实施。利用 Informatica 开发人员和合作伙伴在 Marketplace 中提供的数以百计的解决方案，可提高您的工作效率并加快项目实施时间。您可以在以下网址中找到 Informatica Marketplace：

<https://marketplace.informatica.com>。

Informatica 全球客户支持部门

您可以通过电话或 Informatica Network 与全球支持中心联系。

要查找您当地的 Informatica 全球客户支持部门电话号码，请访问 Informatica 网站，链接为：

<https://www.informatica.com/services-and-training/customer-success-services/contact-us.html>。

要在 Informatica Network 上查找在线支持资源，请访问 <https://network.informatica.com>，然后选择 eSupport 选项。

转换语言

转换语言概览

PowerCenter® 提供的转换语言包括类 SQL 函数，这些函数用于转换源数据。这些函数用于编写表达式并创建用户定义的函数。

Informatica Developer 提供的转换语言包括类似于 SQL 的函数，这些函数用于转换源数据。使用这些函数编写表达式。

用户定义的函数重用表达式逻辑并构建复杂表达式。您可将其纳入其他用户定义的函数或表达式中。用户定义的函数与表达式遵循相同的准则。它们使用相同的语法，且可使用相同的转换语言组件。

表达式修改数据或测试数据是否与条件相匹配。例如，您可使用 AVG 函数计算所有员工的平均薪酬，或使用 SUM 函数计算特定部门的总销售额。

您可创建简单的表达式，仅包含一个端口（例如，ORDERS）或一个数值文字（例如，10）。您也可编写包括嵌套式函数的复杂表达式，或使用转换语言运算符组合不同的端口。

转换语言组件

转换语言包括的以下组件用于创建简单或复杂转换表达式：

- **函数。** 超过 100 个类似于 SQL 的函数可供您在映射中更改数据。
- **运算符。** 使用转换运算符创建转换表达式，以执行数学计算、组合数据或比较数据。
- **常量。** 使用内置常量引用保持不变的值，例如 TRUE。
- **映射参数和变量。** 创建将用于映射或 mapplet 中的映射参数，以引用在整个会话中保持不变的值，例如，州营业税率。在 mapplet 或映射中创建映射变量，以使所编写的表达式引用在会话之间变化的值。
- **映射参数。** 创建将用于映射或 mapplet 中的映射参数，以引用在整个映射或 mapplet 运行期间保持不变的值，例如，州营业税率。
- **工作流变量。** 创建将用于工作流中的工作流变量，以使所编写的表达式引用在工作流之间变化的值。

- **内置和局部变量。** 使用内置变量编写引用了可变值（例如，系统日期）的表达式。您也可在转换中创建局部变量。
- **返回值。** 您也可编写包括返回值查找转换的表达式。

国际化和转换语言

转换语言函数可在 ASCII 或 Unicode 数据移动模式中处理字符数据。使用 Unicode 模式处理多字节字符数据。以下函数和转换的返回值取决于 PowerCenter 集成服务的代码页和数据移动模式：

- INITCAP
- LOWER
- UPPER
- MIN (Date)
- MIN (Number)
- MIN (String)
- MAX (Date)
- MAX (Number)
- MAX (String)
- 使用条件语句比较字符串的任何函数，例如，IIF 和 DECODE

MIN 和 MAX 也根据与 PowerCenter 集成服务代码页关联的排序顺序返回值。

在表达式编辑器中验证无效表达式时，对话框将显示表达式及错误指示符“>>>>”。此指示符显示在左侧，指向表达式中包含错误的那一部分。例如，如果表达式 $a = b + c$ 的 c 处有错误，则错误消息显示：

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

计算字符数据的转换语言函数面向字符，不是面向字节。例如，LENGTH 函数返回字符串中字符的数量，而不是字节数量。LOWER 函数根据 PowerCenter 集成服务代码页返回小写字符串。

表达式语法

虽然转换语言基于标准 SQL，但两种语言之间有差异。例如，SQL 支持用于汇总函数的关键字 ALL 和 DISTINCT，但转换语言不支持。另一方面，转换语言支持汇总函数的可选筛选条件，而 SQL 不支持。

可创建如同端口（例如，ORDERS）、预定义工作流变量（例如，\$Start.Status）或数值文字（例如，10）一样简单的表达式。也可编写包括了嵌套式函数的复杂表达式，或使用转换语言运算符组合不同的列。

可创建如同端口（例如，ORDERS）或数值文字（例如，10）一样简单的表达式。也可编写包括了嵌套式函数的复杂表达式，或使用转换语言运算符组合不同的列。

表达式组件

表达式可由以下组件的任何组合组成：

- 端口（输入、输入/输出、变量）
- 字符串文字，数值文字
- 常量

- 函数
- 内置和局部变量
- 映射参数和映射变量
- 映射参数
- 预定义工作流变量
- 用户定义的工作流变量
- 运算符
- 返回值

端口和返回值

当编写的表达式包括端口或未连接转换的返回值时，请使用下表中的引用限定符：

引用限定符	说明
:EXT	当编写的表达式包括来自外部过程转换的返回值时需要使用此限定符。常规语法为： :EXT.external_procedure_transformation(argument1, argument2, ...)
:LKP	当创建的表达式包括未连接查找转换的返回值时所需。常规语法为： :LKP.lookup_transformation(argument1, argument2, ...) 参数为用于查找条件的本地端口。顺序必须与转换中端口的顺序相匹配。本地端口的数据类型必须与查找条件中所用的查找端口的数据类型相匹配。
:SD	可选（仅限于 PowerMart 3.5 表达式）。限定表达式中的源表端口。常规语法为： :SD.source_table.column_name
:SEQ	当创建的表达式包括序列生成器转换中的端口时所需。常规语法为： :SEQ.sequence_generator_transformation.CURRVAL
:SP	当编写的表达式包括未连接存储过程转换的返回值时所需。常规语法为： :SP.stored_procedure_transformation(argument1, argument2, [, PROC_RESULT]) 参数必须与未连接存储过程转换中的参数相匹配。
:TD	当引用 PowerMart 3.5 LOOKUP 函数中的目标表时所需。常规语法为： LOOKUP(:TD.SALES.ITEM_NAME, :TD.SALES.ITEM_ID, 10, :TD.SALES.PRICE, 15.99)

字符串和数值文字

可包括数值文字或字符串文字。

务必用单引号将字符串文字括起来。例如：

'Alice Davis'

字符串文字区分大小写，可包括除单引号之外的任何字符。例如，不允许使用以下字符串：

'Joan's car'

要返回包含单引号的字符串，请使用 CHR 函数：

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

请不要将单引号与数值文字一起使用。只需输入要包括的数字。例如：

```
.05
```

或者

```
$$Sales_Tax
```

表达式语法规则和准则

请在编写表达式时遵循以下规则和准则：

- 不能在汇总器转换中包括单层函数和嵌套式汇总函数。
- 如果需要同时创建单层函数和嵌套式函数，请另行创建汇总器转换。
- 不能在数值表达式中使用字符串。
例如，表达式 `1 + '1'` 无效，因为只能对数值数据类型执行添加。不能添加整数和字符串。
- 不能将字符串用作数值参数。
例如，表达式 `SUBSTR(TEXT_VAL, '1', 10)` 无效，因为 SUBSTR 函数需要整数值而不是字符串作为起始位置。
- 使用比较运算符时不能混用数据类型。
例如，表达式 `123.4 = '123.4'` 无效，因为它将十进制值与字符串相比较。
- 您可传递端口值、文字字符串或数字、变量、查找转换、存储过程转换、外部过程转换或另一个表达式的结果。
- 您可传递端口值、文字字符串或数字、查找转换或另一个表达式的结果。
- 使用表达式编辑器中的端口选项卡将端口名称输入表达式。如果在连接转换中重命名端口，则 Designer 向转换中的表达式传播名称更改。
- 使用表达式编辑器中的端口选项卡将端口名称输入表达式。如果在连接转换中重命名端口，则 Developer 工具向转换中的表达式传播名称更改。
- 用逗号分隔函数中的每个参数。
- 除了文字，转换语言不区分大小写。
- 除了文字，Designer 和 PowerCenter 集成服务忽略空格。
- 除了文字，Developer 工具和数据集成服务忽略空格。
- 冒号 (:)、逗号 (,) 和句号 (.) 有特殊含义，只能用于指定语法。
- PowerCenter 集成服务 将短划线 (-) 当作负运算符处理。
- 如果将文字值传递至函数，请用单引号括住文字字符串。请不要将引号用于文字数值。PowerCenter 集成服务 将用单引号括住的任何字符串值当作字符串处理。
- 向表达式中的函数传递映射参数或变量或工作流量变量时，请不要使用引号指定映射参数或变量或工作流变量。
- 向表达式中的函数传递映射参数时，请不要使用引号指定映射参数。
- 请不要使用引号指定端口。

- 可在表达式中嵌套多个函数，但汇总函数除外，因为只允许一个嵌套式汇总函数。PowerCenter 集成服务从最里面的函数开始计算表达式。

向表达式添加注释

转换语言提供两个注释说明符，用于在表达式中插入注释：

- 两条短划线，如同：
-- These are comments
- 两条斜线，如同：
// These are comments

PowerCenter 集成服务忽略以这两个注释说明符开头的行上的所有文本。例如，如果要连接两个字符串，则可输入以下表达式，注释在表达式中间：

```
-- 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
```

PowerCenter 集成服务忽略注释，并按以下所示计算表达式：

```
FIRST_NAME || LAST_NAME
```

注释不能延续至新行中：

```
-- 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
```

在此情况下，Designer 和 Workflow Manager 不验证表达式，因为最后一行不是有效表达式。

在此情况下，Developer 工具不验证表达式，因为最后一行不是有效表达式。

如果不想嵌入注释，则在表达式编辑器中单击注释，即可添加注释。

保留字

转换语言中的有些关键字，例如，常量、运算符和内置变量，专为特定函数预留。这些包括：

- :EXT
- :INFA
- :LKP
- :MCR
- :SD
- :SEQ
- :SP
- :TD

- :TYPE
- AND
- DD_DELETE
- DD_INSERT
- DD_REJECT
- DD_UPDATE
- FALSE
- NOT
- NULL
- OR
- PROC_RESULT
- SESSSTARTTIME
- SPOUTPUT
- SYSDATE
- TRUE
- WORKFLOWSTARTTIME

以下关键字专为工作流表达式预留：

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

注意：不能使用预留字为端口或局部变量命名。预留字只能用于转换和工作流表达式中。预留字在表达式中有预定义的含义。

注意：不能使用预留字为端口或局部变量命名。预留字只能用于转换表达式中。预留字在表达式中有预定义的含义。

常量

DD_DELETE

标记要在更新策略表达式中删除的记录。DD_DELETE 与整数文字 2 等效。

注意：仅在更新策略转换中使用 DD_DELETE 常量。使用 DD_DELETE 替代整数文字 2，以便对复杂的数值表达式进行故障排除。

运行工作流时，选择数据驱动的更新策略，以根据此标志从目标中删除记录。

示例

以下表达式将 ID 号为 1001 的项目标记为要删除，将所有其他项目标记为要插入：

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

此更新策略表达式使用数值文字生成相同的结果：

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

注意：使用常量的表达式比用数值文字的表达式更易于读取。

DD_INSERT

标记要在更新策略表达式中插入的记录。DD_INSERT 与整数文字 0 等效。

注意：仅在更新策略转换中使用 DD_INSERT 常量。使用 DD_INSERT 替代整数文字 0，以便对复杂的数值表达式进行故障排除。

运行工作流时，选择数据驱动的更新策略，以根据此标志将记录写入目标。

示例

以下示例修改按销售人员计算的月销售额，因此，您可检查仅一个销售员的销售额。

以下更新策略表达式将一位员工的销售额标记为要插入，并拒绝其他一切事物：

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

此更新策略表达式使用数值文字生成相同的结果：

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

提示：使用常量的表达式比用数值文字的表达式更易于读取。

以下更新策略表达式使用 SESSSTARTTIME 查找仅在最后两天装运的订单，并将它们标记为要插入。通过使用 DATE_DIFF，表达式将从系统日期中减去 DATE_SHIPPED，并返回两个日期之间的差值。因为 DATE_DIFF 返回双精度值，所以表达式使用 TRUNC 截断差值。然后，它将结果与整数文字 2 相比较。如果结果大于 2，则表达式将记录标记为要拒绝。如果结果为 2 或更少，则表达式将它们标记为要插入：

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

DD_REJECT

标记要在更新策略表达式中拒绝的记录。DD_REJECT 与整数文字 3 等效。

注意：仅在更新策略转换中使用 DD_REJECT 常量。使用 DD_REJECT 替代整数文字 3，以便对复杂的数值表达式进行故障排除。

运行工作流时，选择数据驱动的更新策略，以根据此标志从目标中拒绝记录。

使用 DD_REJECT 筛选或验证数据。如将记录标记为要拒绝，则 PowerCenter 集成服务跳过记录，并将其写入会话拒绝文件。

示例

以下示例修改计算当前月份销售额的映射，因此，它仅包括正值。

此更新策略表达式将小于 0 的记录标记为要拒绝，将所有其他记录标记为要插入：

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

此表达式使用数值文字生成相同的结果：

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

使用常量的表达式比用数值文字的表达式更易于读取。

以下数据驱动的示例使用 DD_REJECT 和 IS_SPACES 避免将空格写入目标表中的字符列。此表达式将完全由空格组成的记录标记为要拒绝，将所有其他记录标记为要插入：

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

DD_UPDATE

标记要在更新策略表达式中更新的记录。DD_UPDATE 与整数文字 1 等效。

注意：仅在更新策略转换中使用 DD_UPDATE 常量。使用 DD_UPDATE 替代整数文字 1，以便对复杂的数值表达式进行故障排除。

运行工作流时，选择数据驱动的更新策略，以根据此标志将记录写入目标。

示例

以下示例修改了用于计算当前月份销售额的映射。映射加载一位员工的销售额。

此表达式将 Alex 的记录标记为要更新，将所有其他人的记录标记为要拒绝：

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

此表达式使用数值文字产生相同的结果，将 Alex 的销售额标记为要更新 (1)，将所有其他销售额记录标记为要拒绝 (3)：

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

使用常量的表达式比用数值文字的表达式更易于读取。

以下更新策略表达式使用 SYSDATE 查找仅在最后两天装运的订单，并将它们标记为要插入。通过使用 DATE_DIFF，表达式将从系统日期中减去 DATE_SHIPPED，并返回两个日期之间的差值。因为 DATE_DIFF 返回双精度值，所以表达式使用 TRUNC 截断差值。然后，它将结果与整数文字 2 相比较。如果结果大于 2，则表达式将记录标记为要拒绝。如果结果为 2 或更小值，则表达式将记录标记为要更新。否则，表达式将它们标记为要拒绝：

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

FALSE

澄清条件表达式。FALSE 与整数 0 等效。

示例

以下示例在 DECODE 表达式中使用 FALSE，以根据比较结果返回值。当您想要根据单个搜索值执行多个搜索时，这很有用：

```
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

指示值为未知或未定义。NULL 等效于空白或空字符串（对于字符列）或 0（对于数字列）。

虽然您可写入返回空值的表达式，但有 NOT NULL 或 PRIMARY KEY 约束的任何列将不接受空值。因此，如果 PowerCenter 集成服务尝试将空值写入有其中一个这些约束的列，则数据库将拒绝行，PowerCenter 集成服务会将它写入拒绝文件。在创建转换时务必考虑空值。

各函数可能以不同方式处理空值。如果传递空值至函数，则它可能返回 0 或 NULL，或它可能忽略空值。

相关主题：

- [“函数” 页面上 51](#)

处理布尔表达式中的空值

将空值与布尔表达式相结合的表达式产生的结果符合 ANSI。例如，PowerCenter 集成服务产生以下结果：

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

比较表达式中的空值

当您在包含比较运算符的表达式中使用空值时，PowerCenter 集成服务会生成空值。要检查列中是否有空值，必须在比较表达式中使用 ISNULL()。

要返回不包含空值的行，请使用 ISNULL 函数代替常量 !=。例如，使用 NOT ISNULL(Field_A)。

下面这个表达式产生了空值，筛选器转换不返回任何行：Field_A!=NULL。

您也可以配置查找转换，在比较运算中将空值视为高或低。在查找源中使用“空值排序”属性，可配置 PowerCenter 集成服务在查找转换中如何处理比较表达式中的空值。

当您在包含比较运算符的表达式中使用空值时，PowerCenter 集成服务会生成空值。但是，您也可以配置 PowerCenter 集成服务，在比较运算中将空值视为高或低。

使用“在比较运算符中将 Null 视为”属性可配置 PowerCenter 集成服务如何处理比较表达式中的空值。

此 PowerCenter 集成服务配置属性将影响表达式中以下比较运算符的行为：

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

例如，考虑以下表达式：

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

下表介绍了 PowerCenter 集成服务如何计算表达式：

表达式	在比较运算符中将 Null 视为		
	NULL	HIGH	LOW
NULL > 1	NULL	TRUE	FALSE
NULL = NULL	NULL	TRUE	TRUE

汇总函数中的空值

PowerCenter 集成服务将汇总函数中空值当作 NULL 处理。如果传递整个端口或空值组，则函数返回 NULL。

PowerCenter 集成服务将汇总函数中空值当作 NULL 处理。如果传递整个端口或空值组，则函数返回 NULL。然而，在配置 PowerCenter 集成服务时，可选择想要它处理汇总函数中空值的方式。可让 PowerCenter 集成服务将汇总函数中所有空值作为 NULL 或 0 处理。

筛选条件中的空值

如果筛选条件计算结果为 NULL，则函数不选择记录。如果筛选条件对选定端口中所有记录的计算结果为 NULL，则汇总函数返回 NULL（只有 COUNT 返回 0）。可将筛选条件与汇总函数以及 CUME、MOVINGAVG 和 MOVINGSUM 函数组合使用。

运行算空值

使用运算符（除了字符串运算符 ||）且包含空值的任何表达式的计算结果均始终为 NULL。例如，以下表达式计算结果为 NULL：

```
8 * 10 - NULL
```

要测试是否存在空值，请使用 ISNULL 函数。

TRUE

根据比较结果返回值。TRUE 与整数 1 等效。

示例

以下示例在 DECODE 表达式中使用 TRUE，以根据比较结果返回值。当您想要根据单个搜索值执行多个搜索时，这很有用：

```
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.')
```

运算符

运算符优先级

转换语言支持使用多个运算符及在嵌套表达式中使用运算符。

如果编写的表达式包括多个运算符，则 PowerCenter 集成服务按以下顺序计算表达式：

1. 复杂运算符
2. 算术运算符
3. 字符串运算符
4. 比较运算符
5. 逻辑运算符

PowerCenter 集成服务按运算符在下表中的显示顺序计算运算符。它从左到右按相等的优先顺序计算表达式中的所有运算符。

下表列出了所有转换语言运算符的优先顺序：

运算符	含义
[] 和 .	下标和点。
()	圆括号。
+、- 和 NOT	一元加号和减号以及 NOT 逻辑运算符。
*, / 和 %	乘法、除法和取模。
+ 和 -	加法和减法。
	连接。
<, <=, > 和 >=	小于、小于或等于、大于和大于或等于。
=, <> 和 != 和 ^=	等于、不等于、不等于和不等。
AND	AND 逻辑运算符，在指定条件时使用。
OR	OR 逻辑运算符，在指定条件时使用。

运算符	含义
()	圆括号。
+、- 和 NOT	一元加号和减号以及 NOT 逻辑运算符。
*, / 和 %	乘法、除法和取模。
+ 和 -	加法和减法。
	连接。

运算符	含义
<、<=、> 和 >=	小于、小于或等于、大于和大于或等于。
=、<> 和 != 和 ^=	等于、不等于、不等于和不等。
AND	AND 逻辑运算符，在指定条件时使用。
OR	OR 逻辑运算符，在指定条件时使用。

转换语言也支持在嵌套表达式中使用运算符。当表达式包含圆括号时，PowerCenter 集成服务先计算圆括号内的运算符，然后再计算圆括号外的运算符。首先计算最里面圆括号中的运算符。

例如，取决于您如何嵌套运算，方程式 $8 + 5 - 2 * 8$ 返回不同的值：

方程式	返回值
$8 + 5 - 2 * 8$	-3
$8 + (5 - 2) * 8$	32

复杂运算符

使用复杂运算符可访问复杂数据类型中的元素。可以访问数组、映射或结构数据类型中的元素。

对于在 Spark 引擎上运行的映射，您可以在这些映射中使用复杂运算符。

下表列出了转换语言中的复杂运算符：

运算符	含义
[]	下标运算符。 使用下标运算符可访问数组中的一个或多个元素。还可以使用下标运算符访问与映射的键-值对中的给定键相对应的值。
.	点运算符。 使用点运算符可访问结构中的元素。还可以在结构的数组中使用点运算符来访问每个结构中的元素。

在结构的数组中使用点运算符时，它会将每个结构中具有相同名称的元素作为数组返回。要访问嵌套数组或结构中的元素，可以使用复杂运算符的组合。

下标运算符

使用下标运算符可以访问数组或映射中的元素。可以访问数组中的特定元素或一系列元素。可以访问与映射的键-值对中的给定键相对应的值。

语法

要访问数组中的特定元素，请使用以下语法：

```
array[ index ]
```

要访问数组中的一系列元素，请使用以下语法：

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

要访问映射中对应于给定键的值，请使用以下语法：

```
map[ key ]
```

下表介绍了语法中的参数：

参数	说明
array	数组。要从中访问一个或多个元素的数组。 可输入计算结果为数组的任何有效转换表达式。
index	整数。要访问的元素的位置。例如，索引 0 指示数组中的第一个元素。
start_index	整数。要访问的一系列元素中的起始索引。下标运算符包含起始索引所表示的元素。
end_index	整数。要访问的一系列元素的结束索引。下标运算符不包含结束索引所表示的元素。
map	映射。想要从中检索对应于某个键的值的映射。
key	键的数据类型。想要为其检索值的键元素。 可以输入任何有效的转换表达式，其计算结果为映射数据的键值。

可以对索引使用返回整数值的表达式。如果表达式返回负值，则将索引视为 0。

如果指定的索引大于数组大小减 1，该索引将访问数组中的最后一个元素。

返回值

如果指定一个索引，则表达式将返回数组中的元素。返回类型与指定数组中的元素的数据类型相同。

如果指定两个索引并用逗号分隔（例如 [i,j]），则表达式将返回元素 i 到 j-1 的数组。如果 i 大于 j 或大于数组大小，则表达式将返回空数组。表达式返回的子数组的类型配置与指定的数组的类型配置相同。

如果指定键，则表达式将返回与映射中键相关联的值。返回类型与指定映射中的值的数据类型相同。

空值

如果下标中的索引大于数组大小，下标运算符将返回 NULL 值。

如果索引为 NULL，下标运算符将返回 NULL 值。如果指定多个索引（例如 [i,j]）并且 i 或 j 为 NULL，则表达式将返回 NULL。

如果数组为 NULL，下标运算符将返回 NULL 值。

如果映射中不存在键，则下标运算符将返回 NULL 值。

示例

您有以下数组，该数组包含字符串元素：

```
drinks = [ 'milk' , 'coffee' , 'tea' , 'chai' ]
```

以下表达式使用下标运算符访问数组中的字符串元素：

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

您有以下映射，该映射包含字符串-字符串类型的键-值元素：

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

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

点运算符

使用点运算符可访问结构中的元素。还可以在结构的数组中使用点运算符来访问数组中每个结构的元素。

语法

要访问结构中的元素，请使用以下语法：

```
struct.element
```

要访问结构的数组中的元素，请使用以下语法：

```
array_of_structs.element
```

下表介绍了语法中的参数：

参数	说明
struct	结构。要从中访问元素的结构。可输入计算结果为结构的任何有效转换表达式。
array_of_structs	包含结构元素的数组。要从中访问每个结构中的元素的数组。可输入计算结果为数组的任何有效转换表达式。
元素	要访问的结构元素的名称。

返回值

如果对结构使用点运算符，则表达式将返回该结构中的元素。返回类型与指定结构中的元素的数据类型相同。

如果对结构的数组使用点运算符，则表达式将返回一个数组，该数组包含每个结构中的指定元素。

空值

如果结构中的元素具有 NULL 值，则表达式将返回 NULL。

如果结构为 NULL，则表达式将返回 NULL。

示例

您有以下结构：

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

以下表达式使用点运算符访问结构中的元素：

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

还可以使用点运算符来访问结构的数组中的元素。

例如，您有以下数组，该数组包含类型为 struct 的三个元素并且每个结构包含三个元素：

```
employee_info_array = [
  derrick_struct{
    name: 'Derrick'
    city: NULL
    state: 'NY'
  },
  kevin_struct{
    name: 'Kevin'
    city: 'Redwood City'
    state: 'CA'
  },
  lauren_struct{
    name: 'Lauren'
    city: 'Woodcliff Lake'
    state: NULL
  }
]
```

]

以下表达式使用点运算符访问数组的每个结构中的字符串元素：

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

嵌套数据类型的复杂运算符

嵌套数据类型包含复杂数据类型的元素。使用复杂运算符的组合可访问嵌套数据类型中的元素。
当数组或结构包含类型为 array 或 struct 的元素时，请使用复杂运算符的组合来访问元素。可以访问多维数组中的元素、包含结构元素的数组、包含数组元素的结构以及包含结构元素的结构。

多维数组

多维数组是一系列数组，最多可以包含五层嵌套。可以使用下标运算符访问任一层的数组，或者访问最里层的数组中的特定元素。

可以使用下标运算符返回以下值：

- 最里层的数组中的特定元素。
- 任一层的一个或多个数组。
- 任一层的一个或多个数组的子集。

要访问最里层的数组中的特定元素，请使用多个下标运算符。多维数组中的维度数目决定应使用的下标运算符个数。每个下标运算符必须包含一个索引值。返回值的数据类型与数组中的元素的数据类型相同。

例如，在二维数组中，应使用两个下标运算符。第一个下标运算符决定将访问的单维数组。第二个下标运算符将访问该数组中的哪个元素。

以下二维数组包含三个数组，每个数组都包含类型为 string 的元素：

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

以下表达式使用两个下标运算符访问 menu_array 内每个单维数组中的特定元素：

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

以下表达式使用单个下标运算符返回 menu_array 中的单维数组：

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

以下表达式使用两个下标运算符返回 menu_array 内的数组的子集：

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

包含结构元素的数组

包含结构元素的数组是结构数组。使用下标运算符和点运算符可访问数组中的结构的元素。

要访问数组中的结构的元素，请使用下标运算符并后接点运算符。也可以反转运算符的顺序。无论运算符的顺序如何，返回值都是相同的。根据复杂运算符的顺序，按如下访问元素：

使用下标运算符并后接点运算符。

下标运算符首先访问数组中已编制索引的元素并返回一个结构。然后，点运算符访问结构中的元素。

使用点运算符并后接下标运算符。

点运算符将从每个结构中查找具有相同名称的元素并返回一个数组。然后，下标运算符访问数组中的元素。

例如，您有以下数组 employee_info_array：

```
employee_info_array = [  
  derrick_struct{  
    name: 'Derrick'  
    city: NULL  
    state: 'NY'  
  },  
  kevin_struct{  
    name: 'Kevin'  
    city: 'Redwood City'  
  }]
```

```

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

```

以下表达式在数组 `employee_info_array` 中使用下标运算符并后接点运算符：

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

如果先使用点运算符，点运算符将返回每个结构中元素具有相同名称的数组。例如，以下表达式显示了使用点运算符时的返回值：

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

然后，下标运算符访问返回的数组中的元素。以下表达式使用点运算符并后接下标运算符：

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

请注意，无论先使用下标运算符还是先使用点运算符，返回值都是相同的。例如，表达式 `employee_info_array[0].name` 和 `employee_info_array.name[0]` 具有相同的返回值 'Derrick'。

包含数组元素的结构

要访问某个结构内的数组中的元素，请使用点运算符并后接下标运算符。点运算符首先访问结构中的指定数组元素。然后，下标运算符根据索引值访问数组中的元素。

例如，您有以下结构，该结构包含数组元素 `drinks`、`sandwiches` 和 `salads`。

```

menu_struct{
    drinks: ['milk','coffee','tea','chai']

```

```

sandwiches: ['ham', 'turkey', NULL]
salads: ['caesar', 'cobb', 'greek', 'chipotle']
}

```

使用表达式 `menu_struct.drinks[0]` 时，点运算符首先访问数组元素 `drinks`。然后，下标运算符访问数组 `drinks: ['milk', 'coffee', 'tea', 'chai']` 中位置 0 处的元素。返回值为 `milk`。

以下表达式使用点运算符并后接下标运算符来访问结构 `menu_struct` 内的数组中的元素：

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

包含结构元素的结构

包含一层或多层结构的结构是嵌套结构。可以使用点运算符访问任一层的结构，或者访问最里层的结构中的特定元素。

可以使用点运算符返回以下值：

- 最里层的结构中的指定元素。
- 任一层的一个或多个结构。

要访问最里层的结构中的特定元素，请使用多个点运算符。嵌套结构中的层数决定应使用的点运算符个数。返回值的数据类型与结构中的元素的数据类型相同。例如，在包含两层的嵌套结构中，应使用两个点运算符。第一个点运算符访问父结构中的指定子结构元素。然后，第二个点运算符访问子结构中的元素。

以下示例使用结构 `employee_info_struct`，该结构包含两个子结构，即 `home_address_info` 和 `department_info`：

```

employee_info_struct{
    emp_name: 'Derrick'
    home_address_info{
        city: 'New York'
        state: NULL
    }
    department_info{
        NULL
    }
}

```

以下表达式使用点运算符访问结构 `employee_info_struct` 中的元素：

Input Value	RETURN VALUE
<code>employee_info_struct.emp_name</code>	<code>'Derrick'</code>

Input Value	RETURN VALUE
employee_info_struct.home_address_info	{ city: 'New York' state: NULL }
employee_info_struct.department_info	NULL
employee_info_struct.home_address_info.city	'New York'
employee_info_struct.home_address_info.state	NULL

算术运算符

使用算术运算符对数值数据执行算术计算。

下表按转换语言中的优先级顺序列出算术运算符：

运算符	含义
+, -	一元加号和减号。一元加号表示正值。一元减号表示负值。
*, /, %	乘法、除法、取模。模数是两个整数相除之后的余数。例如，13 % 2 = 1，因为 13 除以 2 等于 6，余数为 1。
+, -	加法、减法。 加号运算符 (+) 不连接字符串。要连接字符串，请使用字符串运算符 。要对日期值执行算术，请使用日期函数。

如果对空值执行算术，则函数返回 NULL。

使用表达式中的算术运算符时，表达式中的所有操作数必须为数值。例如，表达式 1 + '1' 无效，因为其向字符串添加整数。表达式 1.23 + 4 / 2 有效，因为所有操作数均为数值。

注意：转换语言提供内置日期函数，以便您对日期/时间值执行算术。

相关主题：

- [“了解日期算术运算” 页面上 50](#)

字符串运算符

使用 || 字符串运算符连接两个字符串。|| 运算符在连接之前将任何数据类型（二进制除外）的操作数转换为字符串数据类型：

输入值	返回值
'alpha' 'betical'	alphabetical
'alpha' 2	alpha2
'alpha' NULL	alpha

|| 运算符包括前导和尾随空白。连接两个字符串之前，使用 LTRIM 和 RTRIM 函数裁减前导和尾随空白。

空值

|| 运算符忽略空值。然而，如果两个值均为 NULL，则 || 运算符返回 NULL。

示例

以下示例显示的表达式连接两列中的员工名字和员工姓氏。此表达式删除名字末尾和姓氏开头的空格，将一个空格连接至每个名字的末尾，然后连接姓氏：

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

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

注意：也可使用 CONCAT 函数连接两个字符串值。然而，|| 运算符在更短的时间内产生相同的结果。

比较运算符

使用比较运算符比较字符或数值字符串、操作数据并返回 TRUE (1) 或 FALSE (0) 值。

下表列出了转换语言中的比较运算符：

运算符	含义
=	等于。
>	大于。
<	小于。
>=	大于或等于。
<=	小于或等于。
<>	不等于。
!=	不等于。
^=	不等于。

使用大于 (>) 和小于 (<) 运算符比较数值或根据特殊端口中主键的排序顺序返回行范围。

使用表达式中的比较运算符时，操作数必须为相同的数据类型。例如，表达式 `123.4 > '123'` 无效，因为此表达式将十进制值与字符串相比较。表达式 `123.4 > 123` 和 `'a' != 'b'` 有效，因为操作数为相同数据类型。

如果将某值与空值相比较，则结果为 NULL。

如果筛选条件计算结果为 NULL，则集成服务返回 NULL。

比较复杂数据类型

可以使用等于 (=) 运算符和不等于 (!=) 运算符来比较诸如数组或结构等复杂数据类型。

要使两个数组等效，必须满足以下条件：

- 数组元素必须属于相同数据类型。
- 数组必须具有相同大小。
- 每个索引中的条目必须相同。

例如，您有以下数组：

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

可以进行以下比较：

```
A = B
```

返回值：TRUE (1)

两个数组具有相同大小，并且每个索引中的条目都相同，例如，`A[0]=B[0]`，`A[1]=B[1]`，and `A[2]=B[2]`。

比较两个结构时，如果满足以下条件，则它们等效：

- 相应的结构元素必须属于相同数据类型。
- 结构必须具有相同的数据。

如果满足这些条件，则即使结构元素具有不同的名称，这两个结构也等效。

例如，您有以下结构：

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

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

可以进行以下比较：

```
struct1 = struct2
```

返回值：TRUE (1)

逻辑运算符

使用逻辑运算符操作数值数据。返回数值的表达式，对于非 0 值，计算结果为 TRUE，对于 0 值，计算结果为 FALSE，对于 NULL，计算结果为 NULL。

下表列出了转换语言中的逻辑运算符：

运算符	含义
NOT	求反表达式结果。例如，如果表达式计算结果为 TRUE，则运算符 NOT 返回 FALSE。如果表达式计算结果为 FALSE，则 NOT 返回 TRUE。
AND	如果两个条件计算结果均为 TRUE，则连接两个条件并返回 TRUE。如果一个条件不为真，则返回 FALSE。
OR	如果任一条件计算结果为 TRUE，则连接两个条件并返回 TRUE。如果两个条件均不为真，则返回 FALSE。

空值

将空值与布尔表达式相结合的表达式产生的结果符合 ANSI。例如，PowerCenter 集成服务产生以下结果：

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

变量

内置变量

转换语言提供内置变量。内置变量返回运行时或系统信息。运行时变量返回以下类似信息：源和目标表名称、文件夹名称、会话运行模式和工作流运行实例名称。系统变量返回会话开始时间、系统日期和工作流开始时间。

转换语言提供可返回系统日期的内置变量 SYSDATE。可在表达式中使用 SYSDATE。例如，可在 DATE_DIFF 函数中使用 SYSDATE。

可在 Designer 或 workflow 管理器的表达式中使用内置变量。例如，可在 DATE_DIFF 函数中使用系统变量 SYSDATE。可在接受映射或 workflow 变量的表达式和输入字段中使用运行时变量。例如，可使用运行时变量 \$PMWorkflowRunInstanceName 作为目标输出文件名称的一部分。PowerCenter 集成服务设置内置变量的值。不能在工作流或会话参数文件中定义内置变量的值。

可在表达式中使用内置变量。例如，可在 DATE_DIFF 函数中使用系统变量 SYSDATE。

以下内置变量提供运行时信息：

- \$PM<SourceName>@TableName, \$PM<TargetName>@TableName
- \$PMFolderName
- \$PMIntegrationServiceName
- \$PMMappingName
- \$PMRepositoryServiceName
- \$PMRepositoryUserName
- \$PMSessionName
- \$PMSessionRunMode
- \$PMWorkflowName
- \$PMWorkflowRunId
- \$PMWorkflowRunInstanceName

以下内置变量提供系统信息：

- \$\$\$SessStartTime
- SESSSTARTTIME
- SYSDATE
- WORKFLOWSTARTTIME

下表描述了可在 Designer 和 workflow 管理器中使用内置变量的位置：

变量名称	Designer	workflow 管理器
\$PM<SourceName>@TableName, \$PM<TargetName>@TableName,	<ul style="list-style-type: none"> - 表达式 - 接受映射变量的输入字段 	<ul style="list-style-type: none"> - 接受映射变量的输入字段
\$PMFolderName	<ul style="list-style-type: none"> - 表达式 - 接受映射变量的输入字段 - 接受 workflow 变量的输入字段 	<ul style="list-style-type: none"> - 表达式 - 接受映射变量的输入字段 - 接受 workflow 变量的输入字段
\$PMIntegrationServiceName	<ul style="list-style-type: none"> - 表达式 - 接受映射变量的输入字段 - 接受 workflow 变量的输入字段 	<ul style="list-style-type: none"> - 表达式 - 接受映射变量的输入字段 - 接受 workflow 变量的输入字段
\$PMMappingName	<ul style="list-style-type: none"> - 表达式 - 接受映射变量的输入字段 	<ul style="list-style-type: none"> - 接受映射变量的输入字段

变量名称	Designer	工作流管理器
\$PMRepositoryServiceName	- 表达式 - 接受映射变量的输入字段 - 接受工作流变量的输入字段	- 表达式 - 接受映射变量的输入字段 - 接受工作流变量的输入字段
\$PMRepositoryUserName	- 表达式 - 接受映射变量的输入字段 - 接受工作流变量的输入字段	- 表达式 - 接受映射变量的输入字段 - 接受工作流变量的输入字段
\$PMSessionName	- 表达式 - 接受映射变量的输入字段	- 接受映射变量的输入字段
\$PMSessionRunMode	- 表达式 - 接受映射变量的输入字段	- 接受映射变量的输入字段
\$PMWorkflowName	- 表达式 - 接受映射变量的输入字段 - 接受工作流变量的输入字段	- 表达式 - 接受映射变量的输入字段 - 接受工作流变量的输入字段
\$PMWorkflowRunId	- 表达式 - 接受映射变量的输入字段 - 接受工作流变量的输入字段	- 表达式 - 接受映射变量的输入字段 - 接受工作流变量的输入字段
\$PMWorkflowRunInstanceName	- 表达式 - 接受映射变量的输入字段 - 接受工作流变量的输入字段	- 表达式 - 接受映射变量的输入字段 - 接受工作流变量的输入字段
\$\$\$SessStartTime	- 映射或 mapplet 筛选条件 - 用户定义的联接 - SQL 替代	- 映射或 mapplet 筛选条件 - 用户定义的联接 - SQL 替代
SESSSTARTTIME	- 表达式	n/a
SYSDATE	- 表达式	- 表达式
WORKFLOWSTARTTIME	n/a	- 表达式

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

\$PM<SourceName>@TableName and \$PM<TargetName>@TableName 返回关系源和目标实例的源和目标表名称作为字符串值。 将这些变量与接受字符串数据类型的任何函数组合使用。

变量名称取决于源或目标实例名称。例如，对于名称为“Customers”的源实例，内置变量名称为 \$PMCustomers@TableName。如果关系源或目标为映射中 mapplet 的一部分，则内置变量名称包括 mapplet 名称：

- \$PM<MappletName>.<SourceName>@TableName
- \$PM<MappletName>.<TargetName>@TableName

在映射或 mapplet 中使用 \$PM<SourceName>@TableName 和 \$PM<TargetName>@TableName。例如，在包含多个关系源的映射中，可使用表达式转换输出端口中的 \$PM<SourceName>@TableName 将每行的源表名称写入目标。也可在接受映射变量的输入字段中使用这些变量。

\$PMFolderName

\$PMFolderName 返回存储库文件夹的名称作为字符串值。将 \$PMFolderName 与接受字符串数据类型的任何函数组合使用。

在映射、mapplet、工作流链接或在工作流任务（例如，分配和决策任务）中使用 \$PMFolderName。也可在接受映射或工作流变量的输入字段中使用 \$PMFolderName。

\$PMIntegrationServiceName

\$PMIntegrationServiceName 返回运行会话的 PowerCenter 集成服务的名称。将 \$PMIntegrationServiceName 与接受字符串数据类型的任何函数组合使用。

\$PMIntegrationServiceName 返回 PowerCenter 集成服务名称作为字符串值。

在映射、mapplet、工作流链接或在工作流任务（例如，分配和决策任务）中使用 \$PMIntegrationServiceName。也可在接受映射或工作流变量的输入字段中使用 \$PMIntegrationServiceName。

\$PMMappingName

\$PMMappingName 返回映射名称作为字符串值。将 \$PMMappingName 与接受字符串数据类型的任何函数组合使用。

可在映射或 Mapplet 中使用 \$PMMappingName。也可在接受映射变量的输入字段中使用 \$PMMappingName。

\$PMRepositoryServiceName

\$PMRepositoryServiceName 返回 PowerCenter 存储库服务的名称作为字符串值。将 \$PMRepositoryServiceName 与接受字符串数据类型的任何函数组合使用。

在映射、mapplet、工作流链接或在工作流任务（例如，分配和决策任务）中使用 \$PMRepositoryServiceName。也可在接受映射或工作流变量的输入字段中使用 \$PMRepositoryServiceName。

\$PMRepositoryUserName

\$PMRepositoryUserName 返回运行会话的存储库用户的名称。将 \$PMRepositoryUserName 与接受字符串数据类型的任何函数组合使用。\$PMRepositoryUserName 返回存储库用户名称作为字符串值。

在映射、mapplet、工作流链接或在工作流任务（例如，分配和决策任务）中使用 \$PMRepositoryUserName。也可在接受映射或工作流变量的输入字段中使用 \$PMRepositoryUserName。

\$PMSessionName

\$PMSessionName 返回会话名称作为字符串值。将 \$PMSessionName 与接受字符串数据类型的任何函数组合使用。

可在映射或 Mapplet 中使用 \$PMSessionName。也可在接受映射变量的输入字段中使用 \$PMSessionName。

\$PMSessionRunMode

\$PMSessionRunMode 返回会话运行模式，*正常*或*恢复*作为字符串值。将 \$PMSessionRunMode 与接受字符串数据类型的任何函数组合使用。

可在映射或 Mapplet 中使用 \$PMSessionRunMode。也可在接受映射变量的输入字段中使用 \$PMSessionRunMode。

\$PMWorkflowName

\$PMWorkflowName 返回工作流的名称作为字符串值。将 \$PMWorkflowName 与接受字符串数据类型的任何函数组合使用。

在映射、mapplet、工作流链接或在工作流任务（例如，分配和决策任务）中使用 \$PMWorkflowName。也可在接受映射或工作流变量的输入字段中使用 \$PMWorkflowName。

\$PMWorkflowRunId

每个工作流运行均有唯一的运行 ID。\$PMWorkflowRunId 返回工作流运行 ID 作为字符串值。将 \$PMWorkflowRunId 与接受字符串数据类型的任何函数组合使用。

在映射、mapplet、工作流链接或在工作流任务（例如，分配和决策任务）中使用 \$PMWorkflowRunId。也可在接受映射或工作流变量的输入字段中使用 \$PMWorkflowRunId。例如，您将工作流配置为与同一实例名称并发运行，且想要使用第三方应用程序跟踪每个工作流运行的状态。在后期会话 shell 命令中使用 \$PMWorkflowRunId 将运行 ID 传递至应用程序。

\$PMWorkflowRunInstanceName

\$PMWorkflowRunInstanceName 返回工作流运行实例名称作为字符串值。将 \$PMWorkflowRunInstanceName 与接受字符串数据类型的任何函数组合使用。

在映射、mapplet、工作流链接或在工作流任务（例如，分配和决策任务）中使用 \$PMWorkflowRunInstanceName。也可在接受映射或工作流变量的输入字段中使用 \$PMWorkflowRunInstanceName。例如，对于实例名称为唯一的并发工作流，您可为每个运行实例创建唯一目标文件，只需在会话属性中将目标输出文件设置为 “OutFile_ \$PMWorkflowRunInstanceName.txt”。

或者，您想要使用后期会话 shell 命令创建供预定义事件等待任务使用的指示符文件。在生成指示符文件的 shell 命令中，使用指示符文件名称中的 \$PMWorkflowRunInstanceName 确保一个工作流运行实例不删除另一个工作流运行实例所需的指示符文件。

SESSSTARTTIME

SESSSTARTTIME 返回集成服务初始化会话后运行会话的节点上的当前日期和时间值。可将 SESSSTARTTIME 与任何接受转换日期/时间数据类型的函数一起使用。SESSSTARTTIME 作为转换日期/时间数据类型值进行存储。

可在映射或 Mapplet 中使用 SESSSTARTTIME。只能在表达式语言中引用 SESSSTARTTIME。

示例

以下表达式在源限定符的源筛选条件中使用 \$\$\$SessStartTime 执行增量提取。在 PowerCenter 集成服务初始化会话之前，此表达式指定一周中所有天数的日期范围。此表达式使用函数 DATE_DIFF 在值

ORDER_DATE 与 \$\$\$SessStartTime 之间查找天数差异。如果两个日期之间的差异小于或等于七天，则 PowerCenter 集成服务从源提取该行：

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

SYSDATE

在运行会话的节点上，SYSDATE 为通过转换的每行返回精确到秒的当前日期和时间。SYSDATE 作为转换 date/time 数据类型值存储。

在为通过转换的每行处理数据的节点上，SYSDATE 返回精确到秒的当前日期和时间。SYSDATE 作为转换 date/time 数据类型值存储。

要捕获静态系统日期，请使用 SESSSTARTTIME 变量替代 SYSDATE。

示例

以下表达式使用 SYSDATE 查找在最后两天已装运的订单，并将其标记为要插入。通过使用 DATE_DIFF，PowerCenter 集成服务从系统日期中减去 DATE_SHIPPED，返回两个日期之间的差异。因为 DATE_DIFF 返回双精度值，所以，表达式截断差异。然后，它将结果与整数文字 2 相比较。如果结果大于 2，则表达式将行标记为要拒绝。如果结果为 2 或更少，则表达式将它们标记为要插入。

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

WORKFLOWSTARTTIME

当 PowerCenter 集成服务初始化工作流时，WORKFLOWSTARTTIME 在托管集成服务的节点上返回当前日期和时间。将 WORKFLOWSTARTTIME 与接受转换 date/time 数据类型的任何函数组合使用。

WORKFLOWSTARTTIME 作为转换 date/time 数据类型值存储。

在工作流链接和任务（例如，分配和决策任务）中使用 WORKFLOWSTARTTIME。只能在表达式语言中引用 WORKFLOWSTARTTIME。

示例

以下表达式使用 WORKFLOWSTARTTIME 显示工作流开始时间与工作流中任务开始时间之间的分钟数。通过使用 SQL 函数 DATE_DIFF，PowerCenter 集成服务从 WORKFLOWSTARTTIME 当中减去任务开始时间，并返回结果作为天数：

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

事务控制变量

事务控制变量定义在处理数据库行期间提交或回滚事务的条件。您将这些变量用于您在表达式编辑器中构建的事务控制表达式中。事务控制表达式使用 IIF 函数根据条件测试每行。视乎条件返回值，PowerCenter 集成服务为行提交、回滚事务或不对其做出更改。

以下示例使用事务控制变量确定在何处理行：

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

如果 NEWTRAN = 1，TC_COMMIT_BEFORE 变量导致在当前行处理之前出现提交。否则，TC_CONTINUE_TRANSACTION 变量强制行在当前事务中处理。

创建事务控制表达式时，在表达式编辑器中使用以下变量：

- **TC_CONTINUE_TRANSACTION.** PowerCenter 集成服务不对当前行执行任何事务更改。这是事务控制变量的默认值。
- **TC_COMMIT_BEFORE.**PowerCenter 集成服务提交事务、开始新事务并将当前行写入目标。当前行在新事务中。
- **TC_COMMIT_AFTER.** PowerCenter 集成服务将当前行写入目标、提交事务并开始新事务。当前行在提交的事务中。
- **TC_ROLLBACK_BEFORE.** PowerCenter 集成服务回滚当前事务、开始新事务并将当前行写入目标。当前行在新事务中。

局部变量

如果在映射中使用局部变量，请将它们用于映射中的任何转换表达式。例如，如果您在整个映射中使用复杂税款计算，则可能想要写入一次表达式并将它指定为变量。这提高了性能，因为 PowerCenter 集成服务仅执行一次此计算。

与存储的过程表达式组合用于计算多个返回值时，局部变量很有用。

日期

日期概览

转换语言提供的一组日期函数和内置日期变量用于对日期执行转换。借助于日期函数，可舍入、截断或比较日期，提取日期的一部分或对日期执行算术运算。可向日期函数传递日期数据类型的任何值。

使用日期变量在托管 PowerCenter 集成服务的节点上捕获当前日期或会话开始时间。

使用日期变量在托管 PowerCenter 集成服务的节点上捕获当前日期。

转换语言也提供以下各组格式字符串：

- **日期格式字符串。** 与日期函数结合使用，以指定日期的各部分。
- **TO_CHAR 格式字符串。** 用于指定返回字符串的格式。
- **TO_DATE and IS_DATE 格式字符串。** 用于指定要转换为日期的字符串的格式或测试其是否是日期格式。

Date/Time 数据类型

Informatica 使用通用数据类型转换不同来源的数据。这些转换数据类型包括的 Date/Time 数据类型支持精确到纳秒的日期时间值。Informatica 以二进制格式在内部存储日期。

日期函数仅接受日期时间值。要向日期函数传递字符串，请首先使用 TO_DATE 将其转换为日期时间值。例如，以下表达式将字符串端口转换为日期时间值，然后向每个日期加上一个月：

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

可使用公历日历系统中 1 A.D. 与 9999 A.D 之间的日期。

儒略日、修改的儒略日和公历日历

只能使用公历日历系统中的日期。儒略日历中的日期称为儒略日期，在 Informatica 中不受支持。此术语不得与儒略日或修改的儒略日相混淆。

可使用 J 格式字符串操作修改的儒略日 (MJD) 格式。给定日期的 MJD 是自 Jan 1 4713 B.C. 00:00:00（午夜）到目前为止的天数。根据定义，MJD 包括以小数表示的时间组件，表示 24 小时的某一分数。J 格式字符串不转换此时间组件。

例如，以下 TO_DATE 表达式将 SHIP_DATE_MJD_STRING 端口中的字符串转换为默认日期格式的日期值：

```
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

因为 J 格式字符串不包括日期的时间部分，因此，返回值的时间设定为 00:00:00.000000000。

也可在 TO_CHAR 表达式中使用 J 格式字符串。例如，在 TO_CHAR 表达式中使用 J 格式字符串将日期值转换为用字符串表示的 MJD 值。例如：

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

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

注意：PowerCenter 集成服务忽略 TO_CHAR 表达式中日期的时间部分。

2000 年中的日期

所有转换语言日期函数均支持 2000 年。PowerCenter 支持 1 A.D. 与 9999 A.D. 之间的日期。

RR 格式字符串

转换语言提供的 RR 格式字符串用于将两位数年份的字符串转换为日期。通过 TO_DATE 和 RR 格式字符串, 可将 MM/DD/RR 格式的字符串转换为日期。RR 格式字符串以不同方式转换数据, 具体取决于当前年份。

- **介于 0 与 49 之间的当前年份。** 如果当前年份介于 0 与 49 之间 (例如, 2003) 且源字符串年份介于 0 与 49 之间, 则 PowerCenter 集成服务返回当前世纪加源字符串的两位数年份。如果源字符串年份介于 50 与 99 之间, 则集成服务返回上一世纪加源字符串的两位数年份。
- **介于 50 与 99 之间的当前年份。** 如果当前年份介于 50 与 99 之间 (例如, 1998), 且源字符串年份介于 0 与 49 之间, 则 PowerCenter 集成服务返回下一世纪加源字符串的两位数年份。如果源字符串介于 50 与 99 之间, 则 PowerCenter 集成服务返回当前世纪加指定的两位数年份。

下表汇总了 RR 格式字符串如何转换为日期:

当前年份	源年份	RR 格式字符串返回
0-49	0-49	当前世纪
0-49	50-99	上一世纪
50-99	0-49	下一世纪
50-99	50-99	当前世纪

示例

以下表达式为 1950 与 2049 之间的任何当前年份产生相同的返回值:

```
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

YY 与 RR 格式字符串之间的差异

PowerCenter 也提供 YY 格式字符串。RR 和 YY 格式字符串均指定两位数年份。与除 TO_DATE 之外的所有日期函数一起使用时, YY 和 RR 格式字符串产生相同的结果。在 TO_DATE 表达式中, RR 和 YY 产生不同的结果。

下表显示了每种格式字符串返回的不同结果:

字符串	当前年份	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

字符串	当前年份	TO_DATE(String, 'MM/DD/RR')	TO_DATE(String, 'MM/DD/YY')
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

对于 2000 年及其以后的日期，YY 格式字符串产生的结果没有 RR 格式字符串产生的结果有意义。将 RR 格式字符串用于二十一世纪中的日期。

关系数据库中的日期

一般而言，关系数据库中存储的日期包含日期和时间值。日期包括年、月和日，而时间可能包括小时、分钟、秒以及次秒。可将日期时间数据传递至任何日期函数。

平面文件中的日期

使用 TO_DATE 函数将字符串转换为日期时间值。用 TO_DATE 转换字符串之前，也可使用 IS_DATE 检查其是否为有效日期。转换语言日期函数仅接受日期值。要将字符串传递至日期函数，必须首先使用 TO_DATE 函数将其转换为转换 Date/Time 数据类型。

默认日期格式

PowerCenter 集成服务使用默认日期格式存储和操作表示日期的字符串。要指定默认日期格式，请为会话或会话配置对象在“Confid 对象”选项卡的“日期时间格式字符串”属性中输入日期格式。默认情况下，日期格式为 MM/DD/YYYY HH24:MI:SS.US。

PowerCenter 集成服务使用默认日期格式存储和操作表示日期的字符串。要指定默认日期格式，请在数据查看器配置的“日期时间格式字符串”属性中输入日期格式。默认情况下，日期格式为 MM/DD/YYYY HH24:MI:SS.US。

因为 Informatica 用二进制格式存储日期，所以，在执行以下操作时，PowerCenter 集成服务使用默认日期格式：

- **将日期/时间端口连接至字符串端口，以将日期转换为字符串。**PowerCenter 集成服务 用会话配置对象中定义的日期格式将日期转换为字符串。
- **将字符串端口连接至日期/时间端口，以将字符串转换为日期。**PowerCenter 集成服务期望字符串值采用会话配置对象定义的日期格式。如果输入值与此字符串不匹配，或如果它是无效日期，则 PowerCenter 集成服务跳过行。如果字符串为此格式，则 PowerCenter 集成服务将此字符串转换为日期值。
- **使用 TO_CHAR(date, [format_string]) 将日期转换为字符串。**如果忽略格式字符串，则 PowerCenter 集成服务返回的字符串采用会话属性中定义的日期格式。如果指定格式字符串，则 PowerCenter 集成服务返回指定格式的字符串。
- **使用 TO_DATE(date, [format_string]) 将字符串转换为日期。**如果忽略格式字符串，则 PowerCenter 集成服务期望会话属性中定义的日期格式的字符串。如果指定格式字符串，则 PowerCenter 集成服务期望指定格式的字符串。
- **将日期/时间端口连接至字符串端口，以将日期转换为字符串。**PowerCenter 集成服务用数据查看器配置中定义的日期格式将日期转换为字符串。

- **将字符串端口连接至日期/时间端口，以将字符串转换为日期。**PowerCenter 集成服务期望字符串值采用数据查看器配置定义的日期格式。如果输入值与此字符串不匹配，或如果它是无效日期，则 PowerCenter 集成服务跳过行。如果字符串为此格式，则 PowerCenter 集成服务将此字符串转换为日期值。
- **使用 TO_CHAR(date, [format_string]) 将日期转换为字符串。**如果忽略格式字符串，则 PowerCenter 集成服务返回的字符串采用数据查看器配置中定义的日期格式。如果指定格式字符串，则 PowerCenter 集成服务返回指定格式的字符串。
- **使用 TO_DATE(date, [format_string]) 将字符串转换为日期。**如果忽略格式字符串，则 PowerCenter 集成服务期望字符串采用数据查看器配置中定义的日期格式。如果指定格式字符串，则 PowerCenter 集成服务期望指定格式的字符串。

MM/DD/YYYY HH24:MI:SS.US 的默认日期格式包括：

- 月份（一月 = 01，九月 = 09）
- 日（一月当中）
- 年份（用四位数表示，例如，1998）
- 小时（采用 24 小时制，例如，12:00:00AM = 0，1:00:00AM = 1，12:00:00PM = 12，11:00:00PM = 23）
- 分钟
- 秒数
- 微秒

日期格式字符串

可使用格式字符串与日期函数的组合计算输入日期。日期格式字符串未国际化，必须按下表中列出的预定义格式输入。

下表汇总了用于指定日期一部分的格式字符串：

格式字符串	说明
D、DD、DDD、DAY、DY、J	天数 (01-31)。使用任何这些格式字符串指定日期的整个日部分。例如，如将 12-APR-1997 传递至日期函数，则请使用任何这些格式字符串指定 12。
HH、HH12、HH24	一天中的几点 (0-23)，其中 0 表示 12 AM（午夜）。使用任何这些格式指定日期的整个小时部分。例如，如果传递日期 12-APR-1997 2:01:32 PM，则请使用 HH、HH12 或 HH24 指定日期的小时部分。
MI	分钟 (0-59)。
MM、MON、MONTH	月份 (01-12)。使用任何这些格式字符串指定日期的整个月份部分。例如，如将 12-APR-1997 传递至日期函数，则请使用 MM、MON 或 MONTH 指定四月。
MS	毫秒 (0-999)。
NS	纳秒 (0-999999999)。
SS、SSSS	秒 (0-59)。

格式字符串	说明
US	微秒 (0-999999)。
Y、YY、YYY、YYYY、RR	日期的年份部分 (0001 至 9999) 。使用任何这些格式字符串指定日期的整个年份部分。例如，如将 12-APR-1997 传递至日期函数，则请使用 Y、YY、YYY 或 YYYY 指定 1997。

注意: 格式字符串不区分大小写。必须始终用单引号将它括起来。

下表描述的日期函数使用日期格式字符串计算输入日期：

函数	说明
ADD_TO_DATE	日期中要更改的那部分。
DATE_DIFF	日期中用于计算两个日期之间差异的那部分。
GET_DATE_PART	日期中要返回的那部分。此函数根据默认日期格式返回整数值。
IS_DATE	要检查的日期。
ROUND	日期中要舍入的那部分。
SET_DATE_PART	日期中要更改的那部分。
SYSTIMESTAMP	时间戳精度。
TO_CHAR (Dates)	字符串。
TO_DATE	字符串。
TRUNC (Dates)	日期中要截断的那部分。

TO_CHAR 格式字符串

TO_CHAR 函数将 Date/Time 数据类型转换为您指定格式的字符串。可将整个日期或其一部分转换为字符串。可使用 TO_CHAR 将日期转换为字符串，为了报告而更改格式。

当目标为不支持 Date/Time 数据类型的平面文件或数据库时，一般使用 TO_CHAR。

下表汇总了函数 TO_CHAR 中日期的格式字符串：

格式字符串	说明
AM、A.M., PM、P.M.	子午线指示符。使用任何这些格式字符串指定 AM 和 PM 小时。AM 和 PM 与 A.M. 和 P.M. 返回相同的值。
D	周日 (1-7)，其中星期日等于 1。
DAY	周日名称，最多为九个字符（例如，Wednesday）。

格式字符串	说明
DD	一个月中的第几天 (01-31)。
DDD	一年中的第几天 (001-366, 包括闰年) 。
DY	周日的三位数缩写 (例如, Wed) 。
HH、 HH12	一天中的几点 (01-12)。
HH24	一天中的几点 (00-23), 其中 00 表示 12AM (午夜) 。
J	修改的儒略日。将日历日期转换为等于其修改儒略日值的字符串, 从 Jan 1, 4713 00:00:00 B.C.计算, 它忽略日期的时间组件。例如, 表达式 TO_CHAR(SHIP_DATE, 'J') 将 Dec 31 1999 23:59:59 转换为字符串 2451544。
MI	分钟 (00-59)。
MM	月份 (01-12)。
MONTH	月份名称, 最多九个字符 (例如, January) 。
MON	月份名称的三位数缩写 (例如, Jan) 。
MS	毫秒 (0-999)。
NS	纳秒 (0-999999999)。
Q	一年中的季节 (1-4), 其中一月至三月等于 1。
RR	年份中的最后两位数。此函数删除前导位数。例如, 如果使用 “RR” 且传递年份 1997, 则 TO_CHAR 返回 97。与 TO_CHAR 一起使用时, ‘RR’ 可与 ‘YY’ 互换, 且两者产生的结果相同。然而, 与 TO_DATE 一起使用时, ‘RR’ 计算最接近的相应世纪, 并提供年份中的前两位数。
SS	秒 (00-59)。
SSSSS	自午夜起的秒数 (00000 - 86399)。使用 TO_CHAR 表达式中的 SSSSS 时, PowerCenter 集成服务仅计算日期的时间部分。例如, 表达式 TO_CHAR(SHIP_DATE, ‘MM/DD/YYYY SSSSS’) 将 12/31/1999 01:02:03 转换为 12/31/1999 03723。
US	微秒 (0-999999)。
Y	年份中的最后一位数。此函数删除前导位数。例如, 如果使用 “Y” 并传递年份 1997, 则 TO_CHAR 返回 7。
YY	年份中的最后两位数。此函数删除前导位数。例如, 如果使用 “YY” 且传递年份 1997, 则 TO_CHAR 返回 97。
YYY	年份中的最后三位数。此函数删除前导位数。例如, 如果使用 “YYY” 且传递年份 1997, 则 TO_CHAR 返回 997。
YYYY	日期中整个年份部分。例如, 如果使用 “YYYY” 且传递年份 1997, 则 TO_CHAR 返回 1997。

格式字符串	说明
W	一个月中的第几周 (1-5)，其中第 1 周从一个月的第一天开始，在第七天结束，第 2 周从第八天开始，在第十四天结束。例如，二月 1 号表示二月的第一周。
WW	一年中的第几周 (01-53)，其中第 01 周从 1 月 1 日开始，在 1 月 7 日结束，第 2 周从 1 月 8 日开始，在 1 月 14 日结束，以此类推。
-/.;:	输出中显示的标点。可使用这些符号分隔日期的各部分。例如，您创建以下表达式，用句号分隔日期的各部分：TO_CHAR(DATES, 'MM.DD.YYYY').
"text"	输出中显示的文本。例如，如果用表达式创建输出端口：TO_CHAR(DATES, 'MM/DD/YYYY "Sales Were Up"') 并传递日期 Apr 1 1997，则此函数返回字符串 '04/01/1997 Sales Were Up'。可输入在存储库代码页中有效的多字节字符。
""	使用双引号分隔有歧义的格式字符串，例如，D "" DDD。空引号不显示在输出中。

注意: 格式字符串不区分大小写。必须始终用单引号将它括起来。

示例

以下示例显示了 J、SSSSS、RR 和 YY 格式字符串。参见单个函数了解更多示例。

注意: PowerCenter 集成服务忽略 TO_CHAR 表达式中日期的时间部分。

J 格式字符串

使用 TO_CHAR 表达式中的 J 格式字符串将日期值转换为用字符串表示的 MJD 值。例如：

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

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

SSSSS 格式字符串

也可使用 TO_CHAR 表达式中的格式字符串 SSSSS。例如，以下表达式将 SHIP_DATE 端口中的日期转换为表示自午夜算起的总秒数的字符串：

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

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

RR 格式字符串

以下表达式将日期转换为 MM/DD/YY 格式的字符串：

```
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 格式字符串

在 TO_CHAR 表达式中，YY 格式字符串与 RR 格式字符串产生相同的结果。以下表达式将日期转换为 MM/DD/YY 格式的字符串：

```
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 和 IS_DATE 格式字符串

TO_DATE 函数将指定格式的字符串转换为日期时间值。TO_DATE 一般用于将平面文件中的字符串转换为日期时间值。TO_DATE 格式字符串不是国际化的，必须按预定义的格式输入。

注意：TO_DATE 和 IS_DATE 使用一组相同的格式字符串。

创建 TO_DATE 表达式时，请使用与源字符串中日期的每个部分相对应的格式字符串。源字符串格式和格式字符串必须匹配。要进行日期验证，日期分隔符不需要匹配。如有任何部分不匹配，则 PowerCenter 集成服务不转换字符串，它将跳过行。如果忽略格式字符串，则源字符串必须采用会话中指定的日期格式。

创建 TO_DATE 表达式时，请使用与源字符串中日期的每个部分相对应的格式字符串。源字符串格式和格式字符串必须匹配。要进行日期验证，日期分隔符不需要匹配。如有任何部分不匹配，PowerCenter 集成服务将不转换字符串，并将跳过行。如果忽略格式字符串，则源字符串必须采用数据查看器配置中指定的日期格式。

IS_DATE 指示一个值是否为有效日期。有效日期是指采用会话中指定日期格式的任何字符串。如果要测试的字符串不采用指定的日期格式，请使用“TO_DATE 和 IS_DATE 格式字符串”表中所列的字符串格式。如果字符串格式与指定格式不匹配，或者字符串不表示有效日期，函数将返回 FALSE (0)。如果字符串格式与指定的字符串格式匹配，并且是有效的日期，函数将返回 TRUE (1)。IS_DATE 格式字符串不是国际化的，必须按照下表中所列的其中一种格式输入。

IS_DATE 指示一个值是否为有效日期。有效日期是指采用数据查看器配置中指定日期格式的任何字符串。如果要测试的字符串不采用指定的日期格式，请使用“TO_DATE 和 IS_DATE 格式字符串”表中所列的字符串

格式。如果字符串格式与指定格式不匹配，或者字符串不表示有效日期，函数将返回 FALSE (0)。如果字符串格式与指定的字符串格式匹配，并且是有效的日期，函数将返回 TRUE (1)。IS_DATE 格式字符串不是国际化的，必须按照下表中所列的其中一种格式输入。

下表列出了函数 TO_DATE 和 IS_DATE 的格式字符串：

表 1. TO_DATE 和 IS_DATE 格式字符串

格式字符串	说明
AM、a.m.、PM、p.m.	子午线指示器。使用任何这些格式字符串指定 AM 和 PM 小时。AM 和 PM 与 a.m. 和 p.m. 返回相同的值。
DAY	周日名称，最多为九个字符（例如，Wednesday）。DAY 格式字符串不区分大小写。
DD	一个月中的第几天 (1-31)。
DDD	一年中的第几天 (001-366，包括闰年)。
DY	周日的三位数缩写（例如，Wed）。DY 格式字符串不区分大小写。
HH、HH12	一天中的几点 (1-12)。
HH24	一天中的几点 (0-23)，其中 0 表示 12AM（午夜）。
J	修改的儒略日。将 MJD 格式的字符串转换为日期值。它会忽略源字符串的时间部分，向所有日期分配时间 00:00:00.000000000。例如，表达式 TO_DATE('2451544', 'J') 将 2451544 转换为 Dec 31 1999 00:00:00.000000000。
MI	分钟 (0-59)。
MM	月份 (1-12)。
MONTH	月份名称，最多九个字符（例如，August）。不区分大小写。
MON	月份名称的三位数缩写（例如，Aug）。不区分大小写。
MS	毫秒 (0-999)。
NS	纳秒 (0-999999999)。
RR	四位数年份（例如，1998、2034）。当源字符串包括两位数年份时使用。与 TO_DATE 一起使用，将两位数年份转换为四位数年份。 <ul style="list-style-type: none"> - 介于 50 与 99 之间的当前年份。如果当前年份介于 50 与 99 之间（例如，1998），且源字符串的年份值介于 0 与 49 之间，则 PowerCenter 集成服务返回下一世纪加上源字符串的两位数年份。如果源字符串的年份值介于 50 与 99 之间，则 PowerCenter 集成服务返回当前世纪加上指定的两位数年份。 - 介于 0 与 49 之间的当前年份。如果当前年份介于 0 与 49 之前（例如，2003）且源字符串年份介于 0 与 49 之间，则 PowerCenter 集成服务返回当前世纪加上源字符串的两位数年份。如果源字符串年份介于 50 与 99 之间，则 PowerCenter 集成服务返回上一世纪加上源字符串的两位数年份。
SS	秒 (0-59)。

格式字符串	说明
SSSSS	自午夜算起的秒数。在 TO_DATE 表达式中使用 SSSSS 时，PowerCenter 集成服务将仅计算日期的时间部分。 例如，表达式 TO_DATE(DATE_STR, 'MM/DD/YYYY SSSSS') 将 12/31/1999 3783 转换为 12/31/1999 01:02:03。
US	微秒 (0-999999)。
Y	运行 PowerCenter 集成服务的节点上的当前年份，年份的最后一位数替换为字符串值。
YY	运行 PowerCenter 集成服务的节点上的当前年份，最后两位数替换为字符串值。
YYY	运行 PowerCenter 集成服务的节点上的当前年份，最后三位数替换为字符串值。
YYYY	四位数年份。在传递两位数年份时不要使用此格式字符串。换用 RR 或 YY 格式字符串。

日期格式字符串的规则和准则

在处理日期格式字符串时遵循以下规则和准则：

- TO_DATE 字符串的格式必须与格式字符串匹配。如果不匹配，PowerCenter 集成服务可能会返回不准确的值或跳过行。例如，如将表示 2020 年 5 月 12 日的字符串 “20200512” 传递至 TO_DATE，则必须包括格式字符串 YYYYMMDD。如果您不包含格式字符串，PowerCenter 集成服务需要字符串采用会话中指定的日期格式。数据查看器配置中指定的日期格式。同样，如果传递的字符串与格式字符串不匹配，PowerCenter 集成服务将返回错误并跳过行。例如，如果您将字符串 2020120 传递至 TO_DATE 并包括格式字符串 YYYYMMDD，PowerCenter 集成服务将返回错误并跳过行，因为该字符串与格式字符串不匹配。
- 格式字符串必须用单引号括起来。
- PowerCenter 集成服务将使用会话中指定的默认日期时间格式。默认值为 MM/DD/YYYY HH24:MI:SS.US。格式字符串不区分大小写。

示例

以下示例说明了 J、RR 和 SSSSS 格式字符串。参见单个函数了解更多示例。

J 格式字符串

以下表达式将 SHIP_DATE_MJD_STRING 端口中的字符串转换为默认日期格式的日期值：

```
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

因为 J 格式字符串不包括日期的时间部分，因此，返回值的时间设定为 00:00:00.000000000。

RR 格式字符串

以下表达式将字符串转换为四位数年份格式。当前年份为 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 格式字符串

以下表达式将字符串转换为四位数年份格式。当前年份为 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

注意：对于第二行，RR 返回年份 2005，但 YY 返回年份 1905。

SSSSS 格式字符串

以下表达式将包括自午夜算起的秒数的字符串转换为日期值：

```
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

了解日期算术运算

转换语言提供内置日期函数，以便您按以下方式对日期时间值执行算术：

- **ADD_TO_DATE.** 加上或减去日期的特定部分。
- **DATE_DIFF.** 减去两个日期。
- **SET_DATE_PART.** 更改日期的一部分。

不能使用数值算术运算符（例如，+ 或 -）加上或减去日期。

转换语言识别闰年，接受 Jan. 1, 0001 00:00:00.000000000 A.D. 与 Dec. 31, 9999 23:59:59.999999999 A.D 之间的日期。

注意：转换语言使用转换 Date/Time 数据类型指定日期值。只能对日期时间值使用日期函数。

函数

本章按字母顺序说明转换语言中的函数。每个函数说明包括：

- 语法
- 返回值
- 示例

函数类别

转换语言提供以下类型的函数：

- 汇总
- 二进制
- 字符
- 复杂
- 转换
- 数据清理
- 日期
- 编码
- 财务
- 数值
- 科学计数
- 特殊
- 字符串
- 测试
- 变量
- 窗口

汇总函数

汇总函数为选定端口中的非空值返回汇总值。通过汇总函数，您可：

- 为一组中所有行计算单值。
- 为汇总器转换中每个组返回单值。
- 应用筛选器，为选定端口中特定行计算值。
- 使用运算符执行函数内算术。
- 一次性计算派生自相同源列中的两个或多个汇总值。

转换语言包括以下汇总函数：

- ANY

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

如果将 PowerCenter 集成服务配置为在 Unicode 模式下运行，则 MIN 和 MAX 根据您在会话属性中指定的代码页排序顺序返回值。

如果将 PowerCenter 集成服务配置为在 Unicode 模式下运行，则 MIN 和 MAX 根据您在映射配置中指定的代码页排序顺序返回值。

可以在汇总器转换中使用汇总函数。可以将一个汇总函数嵌套在另一个汇总函数中。PowerCenter 集成服务将计算最里面的汇总函数表达式，并使用结果计算外面的汇总函数表达式。可按以下方式将汇总器转换设置为按 ID 分组并嵌套两个汇总函数：

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

其中，数据集包含以下值：

ID	EARNINGS
1	32
1	45
1	100
2	65
2	75
2	76

ID	EARNINGS
3	21
3	45
3	99

返回值为 186。PowerCenter 集成服务会按 ID 进行分组，计算 AVG 表达式并返回三个值。然后，它通过 SUM 函数将值相加，以得出结果。

还可以在表达式转换中使用汇总函数作为窗口函数。在 Spark 引擎上运行映射时，要使用汇总函数作为窗口函数，您必须为转换配置窗口化。如果使用汇总函数作为窗口函数，表达式转换将变为活动状态。

汇总函数和空值

在配置 PowerCenter 集成服务时，可选择想要处理汇总函数中空值的方式。可让 PowerCenter 集成服务将汇总函数中所有空值作为 NULL 或 0 处理。

默认情况下，PowerCenter 集成服务将汇总函数中空值作为 NULL 处理。如果传递整个端口或空值组，则函数返回 NULL。如果将空值的整个端口传递至汇总函数以返回 0，则可选择配置 PowerCenter 集成服务。

筛选条件

使用筛选条件限制搜索中返回的行数。

筛选器限制搜索中返回的行数。可向所有汇总函数以及 CUME、MOVINGAVG 和 MOVINGSUM 应用筛选条件。筛选条件必须计算 TRUE、FALSE 或 NULL 的结果。如果筛选条件计算结果为 NULL 或 FALSE，则 PowerCenter 集成服务不选择行。

可输入任何有效的转换表达式。例如，以下表达式计算工资超过 \$50,000 的所有员工的平均薪酬：

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

也可使用其他数值作为筛选条件。例如，可输入以下项作为 MEDIAN 函数的完整语法，包括数值端口：

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

在所有情况下，PowerCenter 集成服务均为筛选条件将小数值舍入为整数（例如，将 1.5 舍入为 2，1.2 舍入为 1，0.35 舍入为 0）。如果值舍入为 0，则筛选条件返回 FALSE。如果不想舍入值，则请使用 TRUNC 函数将值截断为整数：

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

如果忽略筛选条件，则函数选择端口中的所有行。

字符函数

转换语言包括以下字符函数：

- ASCII
- CHR
- CHRCODE

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

字符函数 MAX、MIN、LOWER、UPPER 和 INITCAP 使用 PowerCenter 集成服务代码页计算字符数据。

复杂函数

复杂函数是一种预定义的函数，其中的输入值或返回类型属于复杂数据类型，例如 array、map 或 struct。对于在 Spark 引擎上运行的映射，您可以在这些映射中使用复杂函数。

转换语言包括以下复杂函数：

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

转换函数

转换语言包括以下转换函数：

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

数据清理函数

转换语言包括一组用于清除数据错误的函数。可用数据清理函数完成以下任务：

- 测试输入值。
- 转换输入值的数据类型。
- 截减字符串值。
- 替换字符串中的字符。
- 编码字符串。
- 匹配正则表达式中的模式。

转换语言包括以下数据清理函数：

- GREATEST
- IN
- INSTR
- IS_DATE
- IS_NUMBER
- IS_SPACES
- ISNULL
- LEAST
- LTRIM
- METAPHONE
- REG_EXTRACT
- REG_MATCH
- REG_REPLACE
- REPLACECHR
- REPLACESTR
- RTRIM
- SQL_LIKE

- SOUNDEX
- SUBSTR
- TO_BIGINT
- TO_CHAR
- TO_DATE
- TO_DECIMAL
- TO_FLOAT
- TO_INTEGER

日期函数

转换语言包括的一组日期函数可用于舍入、截断或比较日期，提取日期的一部分或对日期执行算术。

可向任何日期函数传递有日期数据类型的任何值。然而，如果想要向日期函数传递字符串，则必须首先使用 TO_DATE 函数将它转换为转换 Date/Time 数据类型。

转换语言包括以下日期函数：

- ADD_TO_DATE
- DATE_COMPARE
- DATE_DIFF
- GET_DATE_PART
- IS_DATE
- LAST_DAY
- MAKE_DATE_TIME
- MAX
- MIN
- ROUND(Date)
- SET_DATE_PART
- SYSTIMESTAMP
- TO_CHAR(Date)
- TIME_RANGE
- TRUNC(Date)

多个日期函数包括 *格式* 参数。必须为此参数指定一个转换语言格式字符串。日期格式字符串未国际化。

Date/Time 转换数据类型支持精确到纳秒的日期。

相关主题：

- [“日期格式字符串” 页面上 43](#)

编码函数

转换语言包括的以下函数适用于数据加密、压缩、编码和校验和：

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

财务函数

转换语言包括以下财务函数：

- FV
- NPER
- PMT
- PV
- RATE

数值函数

转换语言包括以下数值函数：

- ABS
- CEIL
- CONV
- CUME
- EXP
- FLOOR
- LN
- LOG
- MAX
- MIN
- MOD
- MOVINGAVG

- MOVINGSUM
- POWER
- RAND
- ROUND
- SIGN
- SQRT
- TRUNC

科学函数

转换语言包括以下科学函数：

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

特殊函数

转换语言包括以下特殊函数：

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

一般而言，特殊函数用于“表达式”、“筛选器”和“更新策略”转换中。可在特殊函数中嵌套其他函数。也可在汇总函数嵌套特殊函数。

字符串函数

转换语言包括以下字符串函数：

- CHOOSE
- INDEXOF
- MAX
- MIN
- REVERSE

测试函数

转换语言包括以下测试函数：

- ISNULL
- IS_DATE
- IS_NUMBER
- IS_SPACES

窗口函数

转换语言包括一组窗口函数，这些函数对与当前行相关的一组行执行计算。这些函数为每个输入行计算单个返回值。对于在 Spark 引擎上运行的映射，您可以在这些映射中使用窗口函数。

转换语言包括以下窗口函数：

- LAG
- LEAD

可以在表达式转换中使用窗口函数。如果在表达式转换中使用窗口函数，转换将变为活动状态。

变量函数

转换语言包括一组变量函数，用于更新整个会话中映射变量的当前值。运行工作流时，PowerCenter 集成服务根据最后一次会话运行的最终变量值在会话的开始计算变量的起始和当前值。使用以下变量函数：

- SetCountVariable
- SetMaxVariable
- SetMinVariable
- SetVariable

根据变量的汇总类型将不同变量函数与变量一起使用。

将会话中的映射变量与多个分区一起使用时，请使用变量函数确定每个分区的最终变量值。在会话结束时，PowerCenter 集成服务跨所有分区执行汇总函数，以确定要保存至存储库的一个最终值。当您下次使用此会话时，它使用保存的值作为变量的起始值，除非被替代。

例如，您使用 SetMinVariable 将变量设定为最小计算值。PowerCenter 集成服务为每个分区的变量计算最小当前值。然后，在会话结束时，它跨所有分区查找最小当前值，并将该值保存至存储库。

对于管道中的每个映射变量，只能使用一次 SetVariable。在管道中创建多个分区时，PowerCenter 集成服务使用多个线程处理该管道。如将此函数多次用于同一变量，则映射变量的当前值可能有非确定性结果。

ABORT

将停止会话，并将指定错误消息发布给会话日志文件。当 PowerCenter 集成服务遇到 ABORT 函数时，它在该行停止转换数据。在会话根据为其定义的基于来源或目标的提交间隔和缓冲块大小中止和加载任何行读取之前，它先处理任何行读取。PowerCenter 集成服务将数据写入直到中止行的目标中，然后将所有未提交数据回滚至最后一个提交点。回滚之后可对会话执行恢复。

停止映射运行，并将指定的错误消息发布到日志。当 PowerCenter 集成服务遇到 ABORT 函数时，它在该行停止转换数据。在映射运行中止之前，它处理任何行读取。PowerCenter 集成服务将数据写入直到中止行的目标中，然后将所有未提交数据回滚至最后一个提交点。

使用 ABORT 验证数据。一般而言，可使用 IIF 或 DECODE 函数内的 ABORT 设置会话中止规则。

将 ABORT 函数用于输入和输出端口默认值。可将 ABORT 用于输入端口，以阻止将空值传递至转换。也可使用 ABORT 处理任何类型的转换错误，包括表达式内的 ERROR 函数调用。默认值替代表达式中的 ERROR 函数。如果想要确保会话在出现错误时停止，请分配 ABORT 作为默认值。

如将表达式中的 ABORT 用于未连接端口，则 PowerCenter 集成服务不运行 ABORT 函数。

注意: PowerCenter 集成服务以不同方式处理 ABORT 函数以及从 workflow 管理器发出的 Abort 命令。

语法

ABORT(*string*)

下表介绍了此命令的参数：

参数	必需/ 可选	说明
<i>string</i>	必需	字符串。当会话停止时您想要在会话日志文件中显示的消息。字符串长度不限。可输入任何有效的转换表达式。 字符串。当映射运行停止时您想要在日志中显示的消息。字符串长度不限。可输入任何有效的转换表达式。

返回值

NULL.

ABS

返回数值的绝对值。

语法

ABS(*numeric_value*)

下表介绍了此命令的参数：

参数	必需/ 可选	说明
<i>numeric_value</i>	必需	数值数据类型。传递想要为其返回绝对值的值。可输入任何有效的转换表达式。

返回值

正数值。ABS 返回的数据类型与作为参数传递的数值相同。如果传递 Double，则它返回 Double。同样，如果传递整数，则它返回整数。

如果传递空值至函数，则它返回 NULL。

注意: 如果返回值为精度大于 15 的小数，则可启用高精度，以确保小数精确至 38 位。

示例

以下表达式返回两个数字之间的差异作为正值，不管哪个数字更大：

```
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

添加指定量至日期时间值的一部分，并返回与您传递给函数的日期格式相同的日期。ADD_TO_DATE 接受正负整数值。使用 ADD_TO_DATE 更改日期的以下部分：

- **年份。** 在 *数量* 参数中输入正整数或负整数。使用任何年份格式字符串：Y、YY、YYY 或 YYYY。以下表达式向 SHIP_DATE 端口中的所有日期加上 10 年：

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

- **月份。** 在 *数量* 参数中输入正整数或负整数。使用任何月份格式字符串：MM、MON、MONTH。以下表达式从 SHIP_DATE 端口中的每个日期减去 10 个月：

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

- **日。** 在 *数量* 参数中输入正整数或负整数。使用任何日格式字符串：D、DD、DDD、DY 和 DAY。以下表达式向 SHIP_DATE 端口中的每个日期加上 10 天：

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

- **小时。** 在 *数量* 参数中输入正整数或负整数。使用任何小时格式字符串：HH、HH12、HH24。以下表达式向 SHIP_DATE 端口中的每个日期加上 14 小时：

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

- **分钟。** 在 *数量* 参数中输入正整数或负整数。使用 MI 格式字符串设置分钟。以下表达式向 SHIP_DATE 端口中的每个日期加上 25 分钟：

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

- **秒数。** 在 *数量* 参数中输入正整数或负整数。使用 SS 格式字符串设置秒数。以下表达式向 SHIP_DATE 端口中的每个日期加上 59 秒：

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

- **毫秒。** 在 *数量* 参数中输入正整数或负整数。使用 MS 格式字符串设置毫秒。以下表达式向 SHIP_DATE 端口中的每个日期加上 125 毫秒：

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

- **微秒。** 在 *数量* 参数中输入正整数或负整数。使用 US 格式字符串设置微秒。以下表达式向 SHIP_DATE 端口中的每个日期加上 2,000 微秒：

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

- **纳秒。** 在 *数量* 参数中输入正整数或负整数。使用 NS 格式字符串设置纳秒。以下表达式向 SHIP_DATE 端口中的每个日期加上 3,000,000 纳秒：

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

语法

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

下表介绍了此命令的参数：

参数	必需/ 可选	说明
<i>date</i>	必需	Date/Time 数据类型。传递要更改的值。可输入任何有效的转换表达式。
<i>格式</i>	必需	用于指定日期值中要更改的那一部分的格式字符串。用单引号将格式字符串括起来，例如，'mm'。格式字符串不区分大小写。
<i>数量</i>	必需	此整数值指定年份、月份、日、小时等的数量，将其作为您想要更改日期值的依据。可输入任何有效的转换表达式以计算整数。

返回值

格式与传递至此函数的日期相同的日期。

将空值作为参数传递至函数时，返回 NULL。

示例

以下表达式均向 DATE_SHIPPED 端口中的每个日期添加一个月。如果传递的值所创建的日在特殊月份中不存在，则 PowerCenter 集成服务返回一个月中的最后一天。例如，如果向 Jan 31 1998 加上一个月，则 PowerCenter 集成服务返回 Feb 28 1998。

另请注意，ADD_TO_DATE 识别闰年，并向 Jan 29 2000 加上一个月：

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

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

以下表达式从 DATE_SHIPPED 端口中的每个日期减去 10 天：

```
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 (<i>Leap Year</i>)
Oct 9 1997 2:30:12PM	Sep 30 1997 2:30:12PM
Mar 3 1996 5:12:20AM	Feb 22 1996 5:12:20AM
NULL	NULL

以下表达式从 DATE_SHIPPED 端口中的每个日期减去 15 小时：

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

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

处理日期

使用 ADD_TO_DATE 时要注意以下提示：

- 可加上或减去日期的任何部分，只需指定格式字符串并使数量参数成为正整数或负整数。
- 如果传递的值所创建的日在特殊月份中不存在，则 PowerCenter 集成服务返回一个月中的最后一天。例如，如果向 Jan 31 1998 加上一个月，则 PowerCenter 集成服务返回 Feb 28 1998。
- 可嵌套 TRUNC 和 ROUND 以操作日期。
- 可嵌套 TO_DATE 以将字符串转换为日期。
- ADD_TO_DATE 只更改日期中您所指定的那一部分。如果修改日期以使其从标准时间更改为夏令时，则需要更改该日期的小时部分。

AES_DECRYPT

将加密数据返回字符串格式。PowerCenter 集成服务使用高级加密标准 (AES) 算法进行 128 位编码。AES 算法为 FIPS 批准的加密算法。

语法

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

下表介绍了此命令的参数：

参数	必需/ 可选	说明
<i>值</i>	必需	二进制数据类型。 想要解密的值。
<i>键</i>	必需	字符串数据类型。 精度为 16 个或更少字符。 映射变量可用于键。 使用同一键解密您曾加密过的值。

返回值

解密的二进制值。

当输入值为空值时返回 NULL。

示例

以下示例返回解密的社会保障号。在此示例中，PowerCenter 集成服务使用 SUBSTR 函数从社会保障号的前三位数派生键：

```
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

返回加密格式的数据。PowerCenter 集成服务使用高级加密标准 (AES) 算法进行 128 位编码。AES 算法为 FIPS 批准的加密算法。

使用此函数防止向每个人显示敏感数据。例如，为了在数据仓库中存储社会保障号，请使用 AES_ENCRYPT 函数加密社会保障号以进行保密。

语法

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


下表介绍了此命令的参数：

参数	必需/ 可选	说明
<i>值</i>	必需	字符串数据类型。 想要加密的值。
<i>键</i>	必需	字符串数据类型。 精度为 16 个或更少字符。 映射变量可用于键。

返回值

加密的二进制值。
当输入为空值时返回 NULL。

示例

以下示例返回社会保障号的加密值。 在此示例中，PowerCenter 集成服务使用 SUBSTR 函数从社会保障号的前三位数派生键：

```
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

提示

如果目标不支持二进制数据，则请使用 AES_ENCRYPT 和 ENC_BASE64 函数以兼容于数据库的格式存储数据。

ANY

返回选定端口中的任意行。或者，可应用筛选器限定数据集成服务读取的行数。只能在 ANY 中嵌套一个其他汇总函数。

语法

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

下表介绍了此函数的参数：

参数	必需/ 可选	说明
<i>value</i>	必需	除 Binary 外的任意数据类型。传递要为其返回任意行的值。可输入任何有效的转换表达式。
<i>filter_condition</i>	可选	限制搜索中的行。筛选条件必须为数值，或者计算结果为 TRUE、FALSE 或 NULL。可输入任何有效的转换表达式。

返回值

端口中的任意行。每次返回不同的行。

当传递至函数的所有值均为 NULL 或未选定任何行时返回 NULL。例如，筛选条件对所有行的计算结果均为 FALSE 或 NULL。

示例

以下表达式返回价格大于 \$10.00 的 ITEM_NAME 端口中的任意行：

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

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

RETURN VALUE:Flashlight

ANY 和复杂数据类型

可以使用 ANY 返回类型为 array 或 struct 的复杂端口中的行。

例如，您有以下数组：

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

可以使用以下表达式返回数组端口中的任意行：

```
ANY( emp_phones )
```

返回值：[205-128-6478, 722-515-2889]

ARRAY

生成其元素基于指定的参数的数组。

语法

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

下表介绍了此命令的参数：

参数	必需/可选	说明
array_element1	必需	任何数据类型。要添加到数组中的元素。可输入任何有效的转换表达式。
array_element2	可选	数据类型与 array_element1 相同。

如果在数组端口的输出表达式中使用 ARRAY 函数，则函数参数的数据类型必须与数组端口的类型配置中指定的数组元素数据类型匹配。

返回值

数组。

参数的数据类型决定数组元素的数据类型。例如，如果传递字符串参数，函数将生成字符串元素的数组。

示例

以下表达式将生成字符串元素的数组。

```
ARRAY (work_phone, home_phone)
```

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

ASCII

当 PowerCenter 集成服务使用 ASCII 模式时，ASCII 函数返回与传递至此函数的字符串的第一个字符相对应的 ASCII 数值。

当 PowerCenter 集成服务使用 Unicode 模式时，ASCII 函数返回与传递至此函数的字符串的第一个字符相对应的 Unicode 数值。Unicode 值的范围为 0 至 65,535。

可向 ASCII 传递任何大小的字符串，但它只会计算字符串中的第一个字符。在向 ASCII 传递任何字符串值之前，可解析要转换为 ASCII 或 Unicode 值的特定字符。例如，可使用 RTRIM 或另一个字符串操作函数。如果传递数值，则 ASCII 将其转换为字符串，返回字符串中第一个字符的 ASCII 或 Unicode 值。

此函数的行为与 CHRCODE 函数相同。如果在现有表达式中使用 ASCII，它们仍将正常运行。然而，在创建新表达式时，请使用 CHRCODE 函数替代 ASCII 函数。

语法

ASCII (*string*)

下表介绍了此命令的参数：

参数	必需/ 可选	说明
<i>字符串</i>	必需	字符串。传递要作为 ASCII 值返回的值。可输入任何有效的转换表达式。

返回值

整数。字符串中第一个字符的 ASCII 或 Unicode 值。

传递至函数的值为 NULL 时返回 NULL。

示例

以下表达式为 ITEMS 端口中每个值的第一个字符返回 ASCII 或 Unicode 值：

ASCII(ITEMS)

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

AVG

返回一组行中所有值的平均值。或者，可应用筛选器限定读取的行数以计算平均值。只能在 AVG 内嵌套一个其他汇总函数，嵌套函数必须返回数值数据类型。

语法

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

下表介绍了此命令的参数：

参数	必需/ 可选	说明
<i>numeric_value</i>	必需	数值数据类型。 传递要为其计算平均值的值。 可输入任何有效的转换表达式。
<i>filter_condition</i>	可选	限制搜索中的行数。 筛选条件必须为数值或计算结果为 TRUE、FALSE 或 NULL。 可输入任何有效的转换表达式。

返回值

数值。

当传递至函数的所有值为 NULL 或未选择行时返回 NULL。 例如，筛选条件对所有行的计算结果均为 FALSE 或 NULL。

注意: 如果返回值为精度大于 15 的小数，则可启用高精度，以确保小数精确至 38 位。

空值

如果值为 NULL，则 AVG 忽略行。 然而，如果从端口传递的所有值均为 NULL，则 AVG 返回 NULL。

注意: 默认情况下，PowerCenter 集成服务在汇总函数中将空值当作 NULL 处理。 如果传递整个端口或空值组，则函数返回 NULL。 然而，在配置 PowerCenter 集成服务时，可选择想要处理汇总函数中空值的方式。 可将汇总函数中的空值作为 0 或 NULL 处理。

分组依据

AVG 根据您在转换中定义的分组依据端口对值进行分组，为每组返回一个结果。

如果不存在分组依据端口，则 AVG 将所有行当作一个组处理，返回一个值。

示例

以下表达式返回闪光灯的平均批发成本：

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

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

RETURN VALUE: 31.66

提示

在函数计算平均值之前，可对传递至 AVG 的值执行算术。例如：

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

CAST

重命名给定结构值中的元素，并根据指定的复杂数据类型定义中的数据类型更改每个元素的数据类型。

语法

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

下表介绍了此命令的参数：

参数	必需/可选	说明
:Type.type_definition_library.type_definition	必需	表示结构数据的架构的复杂数据类型定义。使用引用限定符 :Type 可引用包含复杂数据类型定义的类型定义库。
struct_value	必需	要为其更改结构元素的数据类型的结构值。可输入计算结果为结构的任何有效转换表达式。

结构值的数据类型和复杂数据类型定义中的数据类型必须兼容。

返回值

结构。

示例

以下表达式根据复杂数据类型定义 h1_sales_def 中的数据类型更改结构端口 h2_sales 中的元素的数据类型。

```
CAST (:Type.type_definition_library.h1_sales_def, h2_sales)
```

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

CEIL

返回大于或等于已传递至此函数的数值的最小整数。例如，如果传递 3.14 至 CEIL，则函数返回 4。如果传递 3.98 至 CEIL，则函数返回 4。同样，如果传递 -3.17 至 CEIL，则函数返回 -3。

语法

```
CEIL( numeric_value )
```

下表介绍了此命令的参数：

参数	必需/ 可选	说明
<i>numeric_value</i>	必需	数值数据类型。可输入任何有效的转换表达式。

返回值

数值。

如果传递的数值的声明精度大于 38，则返回双精度值。

如果传递的数值的声明精度大于 28，则返回双精度值。

传递至函数的值为 NULL 时返回 NULL。

示例

以下表达式返回舍入为下一个整数的价格：

```
CEIL( PRICE )
```

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

提示: 在 CEIL 返回下一个整数值之前，可对已传递至 CEIL 的值执行算术。 例如，如果想要将数值乘以 10，然后再计算大于修改值的最小整数，可以按以下方式编写函数：

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

CHOOSE

基于给定位置从字符串列表中选择字符串。您指定位置 and 值。 如果值与位置相匹配，则 PowerCenter 集成服务返回值。

语法

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

下表介绍了此命令的参数：

参数	必需/ 可选	说明
<i>索引</i>	必需	数值数据类型。 根据想要匹配的值的位置，输入一个数字。
<i>string</i>	必需	任何字符值。

返回值

与索引值位置匹配的字符串。

当没有字符串与索引值位置相匹配时，返回 NULL。

示例

以下表达式根据索引值 2 返回字符串 “flashlight”：

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

以下表达式根据索引值 4 返回 NULL：

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

CHOOSE 返回 NULL，因为表达式不包含第四个参数。

CHR

当 PowerCenter 集成服务使用 ASCII 模式时，CHR 返回与传递至此函数的数值相对应的 ASCII 字符。ASCII 值的范围为 0 至 255。可传递任何整数至 CHR，但只有 ASCII 代码 32 至 126 为可打印字符。

当 PowerCenter 集成服务使用 Unicode 模式时，CHR 返回与传递至此函数的数值相对应的 Unicode 字符。Unicode 值的范围为 0 至 65,535。

语法

```
CHR( numeric_value )
```

下表介绍了此命令的参数：

参数	必需/ 可选	说明
<i>numeric_value</i>	必需	数值数据类型。要作为 ASCII 或 Unicode 字符返回的值。可输入任何有效的转换表达式。

返回值

ASCII 或 Unicode 字符。包含一个字符的字符串。

传递至函数的值为 NULL 时返回 NULL。

示例

以下表达式为 ITEM_ID 端口中的每个数值返回 ASCII 或 Unicode 字符：

```
CHR( ITEM_ID )
```

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

使用 CHR 函数将单引号连接至字符串。单引号是唯一不能在字符串文字内使用的字符。请考虑以下示例：

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

返回值为：

```
Joan's car
```

CHRCODE

当 PowerCenter 集成服务使用 ASCII 模式时，CHRCODE 返回与传递至此函数的字符串的第一个字符相对应的 ASCII 数值。ASCII 值的范围为 0 至 255。

当 PowerCenter 集成服务使用 Unicode 模式时，CHRCODE 返回与传递至此函数的字符串的第一个字符相对应的 Unicode 数值。Unicode 值的范围为 0 至 65,535。

正常情况下，在向 CHRCODE 传递任何字符串值之前，可解析要转换为 ASCII 或 Unicode 值的特定字符。例如，可使用 RTRIM 或另一个字符串操作函数。如果传递数值，则 CHRCODE 将其转换为字符字符串，返回字符串中第一个字符的 ASCII 或 Unicode 值。

此函数的行为与 ASCII 函数相同。如果当前在表达式中使用 ASCII，它们仍将正常运行。然而，在创建新表达式时，请使用 CHRCODE 函数替代 ASCII 函数。

语法

```
CHRCODE ( string )
```

下表介绍了此命令的参数：

参数	必需/ 可选	说明
字符串	必需	字符字符串。传递要作为 ASCII 或 Unicode 值返回的值。可输入任何有效的转换表达式。

返回值

整数。
传递至函数的值为 NULL 时返回 NULL。

示例

以下表达式为 ITEMS 端口中每个值的第一个字符返回 ASCII 或 Unicode 值：

```
CHRCODE( ITEMS )
```

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

COLLECT_LIST

基于参数返回具有元素的数组。参数的数据类型决定数组的数据类型。COLLECT_LIST 是一个汇总函数。

语法

```
COLLECT_LIST(value as any)
```

下表介绍了此命令的参数：

参数	必需/可选	说明
value	必需	任何数据类型。要汇总到类型为 array 的层次结构数据中的值。可输入任何有效的转换表达式。

返回值

数组。

分组依据

COLLECT_LIST 根据您在转换中定义的分组依据端口对值进行分组，并为每个组返回一个结果。
如果不存在分组依据端口，则 COLLECT_LIST 将所有行当作一个组进行处理，并返回一个值。

示例

以下表达式将返回 PRODUCT_NAME 中包含元素的数组。

```
COLLECT_LIST(PRODUCT_NAME)
```

PRODUCT_NAME

Flashlight

Compass

Pressure Gauge

Vest

返回值: [Flashlight,Compass,Pressure Gauge,Vest]

COLLECT_MAP

根据指定参数返回元素映射。

语法

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

下表介绍了此命令的参数：

参数	必需/可选	说明
map_key	必需	任何原始数据类型。想要作为类型映射的层次结构数据的键而汇总的元素。可输入任何有效的转换表达式。
map_value	必需	任何原始或复杂数据类型。想要作为类型映射的层次结构数据的值而汇总的元素。可输入任何有效的转换表达式。

返回值

映射。

分组依据

COLLECT_MAP 根据您在转换中定义的分组依据端口对值进行分组，并为每个组返回一个结果。

如果不存在分组依据端口，则 COLLECT_MAP 将所有行当作一个组进行处理，并返回一个值。

示例

以下表达式将返回一个映射，将 PRODUCT_ID 中的元素用作键，PRODUCT_NAME 中的元素用作值。

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

PRODUCT_ID

PRODUCT_NAME

34890

Flashlight

PRODUCT_ID	PRODUCT_NAME
12754	Compass
54028	Pressure Gauge
81203	Vest

返回值：

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

COMPRESS

使用 zlib 1.2.1 压缩算法压缩数据。通过广域网发送大量数据之前，请使用 COMPRESS 函数。

语法

COMPRESS(*value*)

下表介绍了此命令的参数：

参数	必需/ 可选	说明
<i>值</i>	必需	字符串数据类型。要压缩的数据。

返回值

输入值的压缩二进制值。

当输入为空值时返回 NULL。

示例

您的组织提供在线订单服务。您想要通过广域网发送客户订单数据。来源包含大小为 10 MB 的一行。可使用 COMPRESS 压缩此行中的数据。压缩数据时，您减少了 PowerCenter 集成服务通过网络写入的数据量。因此，您可提高性能。

CONCAT

连接两个字符串。CONCAT 在连接字符串之前将所有数据转换为文本。或者，使用 || 字符串运算符连接字符串。使用 || 字符串运算符替代 CONCAT 将提升 PowerCenter 集成服务的性能。

语法

CONCAT(*first_string*, *second_string*)

下表介绍了此命令的参数：

参数	必需/ 可选	说明
<i>first_string</i>	必需	除了二进制以外的任何数据类型。 想要连接的字符串的第一部分。 可输入任何有效的转换表达式。
<i>second_string</i>	必需	除了二进制以外的任何数据类型。 想要连接的字符串的第二部分。 可输入任何有效的转换表达式。

返回值

字符串。

两个字符串值均为 NULL 时返回 NULL。

空值

如果其中一个字符串为 NULL，则 CONCAT 忽略它并返回另一个字符串。

如果两个字符串均为 NULL，则 CONCAT 返回 NULL。

示例

以下表达式连接 FIRST_NAME and LAST_NAME 端口中的名称：

```
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 不添加空格以分隔字符串。 如果想要在两个字符串之间添加空格，则可使用两个嵌套式 CONCAT 函数编写表达式。 例如，以下表达式首先连接名字末尾的空格，然后连接姓氏：

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

FIRST_NAME	LAST_NAME	RETURN VALUE
John	Baer	John Baer

FIRST_NAME	LAST_NAME	RETURN VALUE
NULL	Campbell	Campbell <i>(includes leading blank)</i>
Bobbi	Apperley	Bobbi Apperley
Jason	Wood	Jason Wood
Dan	Covington	Dan Covington
Greg	NULL	Greg
NULL	NULL	NULL

使用 CHR 和 CONCAT 函数将单引号连接至字符串。引号是不能在字符串文字内使用的唯一字符。请考虑以下示例：

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

返回值为：

```
Joan's car
```

CONCAT_ARRAY

使用指定的分隔符连接数组中的字符串元素并返回一个字符串。

语法

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

下表介绍了此命令的参数：

参数	必需/可选	说明
' '	必需	每个字符串元素由指定的分隔符分隔。例如，“,”使用逗号分隔值。
array	必需	包含类型为 string 的元素的数组。要连接的数组。

返回值

字符串

空值

如果字符串元素之一为 NULL，CONCAT_ARRAY 将忽略该字符串并返回其他字段。

如果所有字符串元素均为 NULL，CONCAT_ARRAY 将返回空字符串。

示例

以下表达式连接数组中的字符串元素。

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

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

CONVERT_BASE

将一个非负数字字符串从一个基数值转换为另一个基数值。

语法

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

下表介绍了此命令的参数：

参数	必需/ 可选	说明
<i>值</i>	必需	字符串数据类型。要从一个基数转换为另一个基数的值。最大值为 9,233,372,036,854,775,806。
<i>source_base</i>	必需	数值数据类型。要转换的数据的当前基数值。最小基数是 2。最大基数是 36。
<i>dest_base</i>	必需	数值数据类型。要将数据转换为基础的基数值。最小基数是 2。最大基数是 36。

返回值

数值。

示例

以下表达式将 2222 从十进制基数值 10 转换为二进制基数值 2：

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

PowerCenter 集成服务返回 100010101110。

COS

返回数值的余弦(以弧度表示)。

语法

`COS(numeric_value)`

下表介绍了此命令的参数：

参数	必需/ 可选	说明
<i>numeric_value</i>	必需	数值数据类型。以弧度表示的数值数据（度数乘以 pi 除以 180）。传递要为其计算余弦的值。可输入任何有效的转换表达式。

返回值

双精度值。

传递至函数的值为 NULL 时返回 NULL。

示例

以下表达式为 Degrees 端口中的所有值返回余弦：

`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

提示：在函数计算余弦之前，可对传递至 COS 的值执行算术。例如，可将端口中的值转换为弧度，然后再计算余弦，如下所示：

`COS(ARCS * 3.14159265359 / 180)`

COSH

返回数值的双曲余弦(以弧度表示)。

语法

`COSH(numeric_value)`

下表介绍了此命令的参数：

参数	必需/ 可选	说明
<i>numeric_value</i>	必需	数值数据类型。以弧度表示的数值数据（度数乘以 pi 除以 180）。传递要为其计算双曲余弦的值。可输入任何有效的转换表达式。

返回值

双精度值。

传递至函数的值为 NULL 时返回 NULL。

示例

以下表达式为 Angles 端口中的值返回双曲余弦：

```
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

提示：在函数计算双曲余弦之前，可对传递至 COSH 的值执行算术。例如：

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

COUNT

返回组中有非空值的行数。或者，可包括星号 (*) 参数，以对转换中所有输入值进行计数。只能在 COUNT 中嵌套一个其他汇总函数。在计算行数之前可先应用行筛选条件。

语法

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

或者

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

下表介绍了此命令的参数：

参数	必需/ 可选	说明
<i>value</i>	必需	除 Binary 以外的任何数据类型。传递要计数的值。可输入任何有效的转换表达式。
*	可选	用于对转换中的 <i>所有行</i> 进行计数。
<i>filter_condition</i>	可选	限制搜索中的行。筛选条件必须为数值，或者计算结果为 TRUE、FALSE 或 NULL。可输入任何有效的转换表达式。

返回值

整数。

如果传递至此函数的所有值均为 NULL，则返回 0（除非纳入星号参数）。

空值

如果所有值均为 NULL，则函数返回 0。

如果应用星号参数，则此函数对所有行进行计数，不管行中的列是否包含空值。

如果应用 *值* 参数，则此函数忽略有空值的列。

注意：默认情况下，PowerCenter 集成服务在汇总函数中将空值当作 NULL 处理。如果传递整个端口或空值组，则函数返回 NULL。然而，在配置 PowerCenter 集成服务时，可选择想要处理汇总函数中空值的方式。可将汇总函数中的空值作为 0 或 NULL 处理。

分组依据

COUNT 根据您在转换中定义的分组依据端口对值进行分组，为每组返回一个结果。如果不存在分组依据端口，则 COUNT 将所有行当作一个组处理，返回一个值。

示例

以下表达式对库存数量不足 5 个的项目进行计数，不包括空值：

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

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

ITEM_NAME	IN_STOCK
-----------	----------

Halogen Flashlight	1
--------------------	---

RETURN VALUE: 1

在此示例中，函数对卤素闪光灯而不是 NULL 项目进行计数。函数对转换中所有行进行计数，包括空值，如下示例中所示：

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

ITEM_NAME	QTY
-----------	-----

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

RETURN VALUE: 2

在此示例中，函数对 NULL 项目和卤素闪光灯进行计数。如果包括星号参数，但不使用筛选器，则函数对传递至转换的所有行进行计数。例如：

```
COUNT( * )
```

ITEM_NAME	QTY
-----------	-----

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

RETURN VALUE: 6

COUNT 和复杂数据类型

可以使用 COUNT 计算类型为 array 或 struct 的复杂端口中的行数。

例如，您有以下数组：

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

可以使用以下表达式计算数组端口中的行数：

```
COUNT( emp_phones )
```

返回值：4

CRC32

返回 32 位循环冗余检查 (CRC32) 值。使用 CRC32 可查找数据传输错误。如果要验证文件中存储的数据是否已修改，也可使用 CRC32。

如果使用 CRC32 在 ASCII 模式和 Unicode 模式下对数据执行冗余检查，则 PowerCenter 集成服务可能针对同一输入值生成不同的结果。如果使用 CRC32 对不同操作系统上的数据执行冗余检查，则 PowerCenter 集成服务可能针对同一输入值生成不同的结果。

注意：CRC32 可针对不同的输入字符串返回相同的输出。如果要在映射中生成键，请使用序列生成器转换。如果使用 CRC32 在映射中生成键，则可能产生意外结果。

语法

```
CRC32( value )
```

下表介绍了此命令的参数：

参数	必需/ 可选	说明
<i>值</i>	必需	字符串或二进制数据类型。传递要对其执行冗余检查的值。输入值区分大小写。输入值的大小写会影响返回值。例如，CRC32(informatica) 和 CRC32 (Informatica) 返回不同的值。

返回值

32 位整数值。

示例

您想要通过广域网读取源数据。您想要确保数据在传输期间已修改。您可为文件中的数据计算校验和并将其与文件存储在一起。读取源数据时，PowerCenter 集成服务可使用 CRC32 计算校验和并将其与存储的值相比较。如果两个值相同，则表明数据尚未修改。

CREATE_TIMESTAMP_TZ

使用时间戳和时区值构建 Timestamp with Time Zone 数据类型。

CREATE_TIMESTAMP_TZ 表达式的输出端口必须为 timestampWithTZ。

语法

CREATE_TIMESTAMP_TZ (timestamp_value, timezone_value)

下表介绍了此命令的参数：

参数	必需/ 可选	说明
<i>timestamp_value</i>	必需	日期/时间数据类型。可输入任何有效的转换表达式。
<i>timezone_value</i>	必需	必须为 string 数据类型。此字符串必须为字符串。传递要为时区创建的值。您可以按照安装位置处的时区文件中的定义输入任何有效的转换表达式。

返回值

返回 Timestamp with Time Zone 数据类型。

当输入为空值时返回 NULL。

示例

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

CUME

返回累加值。累加值表示 CUME 在每次添加一个值时会返回总值。可以在计算累加值之前添加条件，从行集中筛选出行。

使用 CUME 和相似函数（例如，MOVINGAVG 和 MOVINGSUM）简化报告，只需计算累加值即可。

语法

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

下表介绍了此命令的参数：

参数	必需/ 可选	说明
<i>numeric_value</i>	必需	数值数据类型。传递要为其计算累加值的值。可输入任何有效的转换表达式。可创建嵌套表达式，以根据函数结果计算累加值，只要结果为数值即可。
<i>filter_condition</i>	可选	限制搜索中的行。筛选条件必须为数值，或者计算结果为 TRUE、FALSE 或 NULL。可输入任何有效的转换表达式。

返回值

数值。

当传递至函数的所有值为 NULL 或未选定行时返回 NULL（例如，筛选条件对所有行的计算结果均为 FALSE 或 NULL）。

注意: 如果返回值为精度大于 15 的小数，则可启用高精度，以确保小数精确至 38 位。

空值

如果值为 NULL，则 CUME 返回上一行的累加值。然而，如果选定端口中的所有值均为 NULL，则 CUME 返回 NULL。

示例

以下示例行集可能源自于使用 CUME 函数：

CUME(PERSONAL_SALES)

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

同样，可在计算累加值之前添加值：

CUME(CA_SALES + OR_SALES)

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

DATE_COMPARE

返回指示两个日期中哪一个更早的整数。DATE_COMPARE 返回整数值，而非日期值。

语法

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

下表介绍了此命令的参数：

参数	必需/ 可选	说明
<i>date1</i>	必需	Date/Time 数据类型。要比较的第一个日期。 可输入任何有效转换表达式，只要其计算结果为日期即可。
<i>date2</i>	必需	Date/Time 数据类型。要比较的第二个日期。 可输入任何有效转换表达式，只要其计算结果为日期即可。

返回值

- 第一个日期较早时返回 -1。
- 两个日期相等时返回 0。
- 第二个日期较早时返回 1。
- 其中一个日期值为 NULL 时返回 NULL。

示例

以下表达式比较 DATE_PROMISED 和 DATE_SHIPPED 端口中的每个日期，返回的整数表示较早的日期：

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

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

DATE_DIFF

返回两个日期之间的时间长度。可请求采用年份、月份、日、小时、分钟、秒数、毫秒、微秒或纳秒格式。PowerCenter 集成服务从第一个日期中减去第二个日期，返回差异。

PowerCenter 集成服务根据月数而不是天数计算 DATE_DIFF 函数。它为每个月选定天数的部分月份计算日期差异。为了计算部分月份的日期差异，PowerCenter 集成服务将该月中所用天数相加。然后，将相加所得值除以选定月份的总天数。

对于闰年和平年中的相同期限，PowerCenter 集成服务提供不同的值。当二月是 DATE_DIFF 函数的一部分时，就会出现差异。对于闰年的二月，DATE_DIFF 将所用天数除以 29，对于平年的二月，则除以 28。

例如，您想要计算从 9 月 13 日至 2 月 19 日之间的月数。在闰年，DATE_DIFF 函数将二月计算为 19/29 个月，或 0.655 个月。在平年，DATE_DIFF 函数将二月计算为 19/28 个月，或 0.678 个月。PowerCenter 集成服务以相似方式计算剩余月份中的日期差异，DATE_DIFF 函数返回指定期限的总值。

注意: 某些数据库可能使用不同的算法计算日期差异。

语法

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

下表介绍了此命令的参数：

参数	必需/ 可选	说明
<i>date1</i>	必需	Date/Time 数据类型。传递要比较的第一个日期的值。可输入任何有效的转换表达式。
<i>date2</i>	必需	Date/Time 数据类型。传递要比较的第二个日期的值。可输入任何有效的转换表达式。
<i>格式</i>	必需	指定日期或时间度量的格式字符串。可指定年份、月份、日、小时、分钟、秒数、毫秒、微秒或纳秒。只能指定日期的一个部分，例如，'mm'。用单引号将格式字符串括起来。格式字符串不区分大小写。例如，格式字符串 'mm' 与 'MM'、'Mm' 或 'mM' 相同。

返回值

双精度值。如果 *date1* 晚于 *date2*，则返回值为正数。如果 *date1* 早于 *date2*，则返回值为负数。

日期相同时返回 0。

其中一个（或两个）日期值为 NULL 时返回 NULL。

示例

以下表达式返回 DATE_PROMISED 端口与 DATE_SHIPPED 端口之间的小时数：

```
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

以下表达式返回 DATE_PROMISED 端口与 DATE_SHIPPED 端口之间的天数：

```
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

以下表达式返回 DATE_PROMISED 端口与 DATE_SHIPPED 端口之间的月数：

```
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

以下表达式返回 DATE_PROMISED 端口与 DATE_SHIPPED 端口之间的年数：

```
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

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

以下表达式返回 DATE_PROMISED 端口与 DATE_SHIPPED 端口之间的月数:

```
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

解码 base 64 编码值，并返回使用数据的二进制数据表现形式的字符串。如果使用 ENC_BASE64 编码数据，并且想要使用 DEC_BASE64 解码数据，则必须使用相同的数据移动模式运行解码会话。否则，解码数据的输出可能不同于原始数据。

解码 base 64 编码值，并返回使用数据的二进制数据表现形式的字符串。如果使用 ENC_BASE64 编码数据，并且想要使用 DEC_BASE64 解码数据，则必须使用相同的数据移动模式运行映射。否则，解码数据的输出可能不同于原始数据。

语法

```
DEC_BASE64( value )
```

下表介绍了此命令的参数：

参数	必需/ 可选	说明
<i>值</i>	必需	字符串数据类型。要解码的数据。

返回值

二进制解码值。

当输入为空值时返回 NULL。

在 Unicode 模式而非 ASCII 模式中运行会话时，返回值不同。

在 Unicode 模式与 ASCII 模式中运行映射时的返回值将有所不同。

示例

您已编码 WebSphere MQ 消息 ID，并已在 workflow 期间将其写入平面文件。您想要从平面文件源中读取数据，包括 WebSphere MQ 消息 ID。可使用 DEC_BASE64 解码 ID 并将其转换为其原始二进制值。

DECODE

为指定的值搜索端口。如果函数找到值，则会返回您定义的结果值。可以在 DECODE 函数内构建无限多个搜索。

如果使用 DECODE 在字符串端口中搜索值，则可使用 RTRIM 函数裁减尾随空白，或在搜索字符串中包括空白。

语法

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

下表介绍了此命令的参数：

参数	必需/ 可选	说明
值	必需	除了二进制以外的任何数据类型。传递要搜索的值。可输入任何有效的转换表达式。
搜索	必需	数据类型与值参数相同的任何值。传递要搜索的值。搜索值必须与值参数相匹配。不能搜索值的一部分。此外，搜索值区分大小写。 例如，如果想要在特殊端口中搜索字符串 'Halogen Flashlight'，则必须输入 'Halogen Flashlight'，而不仅是 'Halogen'。如果输入 'Halogen'，则不会搜索到匹配值。可输入任何有效的转换表达式。
result	必需	除了二进制以外的任何数据类型。搜索到匹配值时要返回的值。可输入任何有效的转换表达式。
默认	可选	除了二进制以外的任何数据类型。未搜索到匹配值时要返回的值。可输入任何有效的转换表达式。

返回值

搜索到匹配值时返回 *First_result*。

未搜索到匹配值时返回默认值。

如果忽略默认参数且未搜索到匹配值，则返回 NULL。

即使符合多个条件，PowerCenter 集成服务也返回第一个匹配结果。

如果数据包含多字节字符且 DECODE 表达式比较字符串数据，则返回值取决于 PowerCenter 集成服务的代码页和数据移动模式。

DECODE 和数据类型

使用 DECODE 时，返回值的数据类型始终与最精密结果的数据类型相同。

例如，您有以下表达式：

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

此表达式中的返回值为 5、1.414213562 和 3.141592654。第一个结果为 Integer，其他两个结果为 Decimal。Decimal 数据类型的精度大于 Integer。此表达式始终写入 Decimal 结果。

在高精度模式中运行会话时，如果至少有一个结果为 Double，则返回值的数据类型为 Double。

在高精度模式中运行映射时，如果至少有一个结果为 Double，则返回值的数据类型为 Double。

不能创建同时有字符串和数值返回值的 DECODE 函数。

例如，以下表达式无效：

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

验证以上表达式时，将收到以下错误消息：

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

示例

在搜索特殊 ITEM_ID 并返回 ITEM_NAME 的表达式中，您可使用 DECODE：

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

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

对于项目 17 和 25，DECODE 返回默认值 NONE，因为搜索值与 ITEM_ID 不匹配。此外，对于 NULL ITEM_ID，DECODE 返回 NONE。

以下表达式测试多个列和条件，按从上到下顺序计算 TRUE 或 FALSE 的结果：

```
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

使用 zlib 1.2.1 压缩算法解压缩数据。对通过 COMPRESS 函数或压缩工具(使用 zlib 1.2.1 算法)压缩的数据使用 DECOMPRESS 函数。如果数据解压缩会话使用的数据移动模式不同于数据压缩会话，则解压缩数据的输出可能不同于原始数据。

使用 zlib 1.2.1 压缩算法解压缩数据。对通过 COMPRESS 函数或压缩工具(使用 zlib 1.2.1 算法)压缩的数据使用 DECOMPRESS 函数。如果数据解压缩映射使用的数据移动模式不同于数据压缩映射，则解压缩数据的输出可能不同于原始数据。

语法

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

下表介绍了此命令的参数：

参数	必需/ 可选	说明
<i>值</i>	必需	二进制数据类型。要解压缩的数据。
<i>精度</i>	可选	Integer 数据类型。

返回值

输入值的已解压缩二进制值。

当输入为空值时返回 NULL。

示例

您的组织提供在线订单服务。您通过广域网收到压缩的客户订单数据。您想要使用 PowerCenter 读取数据并将数据加载至数据仓库。可使用 DECOMPRESS 解压缩每行数据。然后，PowerCenter 集成服务 就可将解压缩数据加载至目标。

ENC_BASE64

通过使用多用途 Internet 邮件扩展(MIME)编码方式将二进制数据转换为字符串数据来编码数据。想要在不允许使用二进制数据的数据库或文件中存储数据时编码数据。也可编码数据，以便通过转换以字符串格式传递二进制数据。编码数据比原始数据大约长 33%。它显示为一组随机字符。

语法

ENC_BASE64(*value*)

下表介绍了此命令的参数：

参数	必需/ 可选	说明
<i>值</i>	必需	Binary 或 String 数据类型。要编码的数据。

返回值

编码值。

当输入为空值时返回 NULL。

示例

您想要从 WebSphere MQ 读取消息并将数据写入平面文件目标。您想要包括 WebSphere MQ 消息 ID 作为目标数据的一部分。然而，MsgID 字段为二进制，平面文件目标不支持二进制数据。PowerCenter 集成服务 将数据写入目标前，请先使用 ENC_BASE64 对 MsgID 进行编码。

ERROR

导致 PowerCenter 集成服务跳过行并发出您定义的错误消息。错误消息显示在会话日志中。PowerCenter 集成服务不将这些跳过的行写入会话拒绝文件。

导致 PowerCenter 集成服务跳过行并发出您定义的错误消息。错误消息显示在日志中。PowerCenter 集成服务不将这些跳过的行写入拒绝文件。

使用表达式转换中 ERROR 验证数据。一般而言，可使用 IIF 或 DECODE 函数内的 ERROR 设置行跳过规则。

将 ERROR 函数用于输入和输出端口默认值。可将 ERROR 用于输入端口，以阻止将空值传递至转换。

将 ERROR 用于输出端口以处理任何类型的转换错误，包括表达式内的 ERROR 函数调用。在表达式和输出端口默认值中使用 ERROR 函数时，PowerCenter 集成服务跳过行，并同时记录源自表达式和源自默认值的错误消息。如果想要确保 PowerCenter 集成服务跳过产生错误的行，请分配 ERROR 作为默认值。

如果使用 ERROR 以外的输出默认值，则默认值替代表达式中的 ERROR 函数。例如，您使用表达式中的 ERROR 函数，并将默认值 ‘1234’ 分配至输出端口。每当 PowerCenter 集成服务遇到表达式中的 ERROR 函数时，其均会用值 ‘1234’ 替代错误，并将 ‘1234’ 传递至下一个转换。它不跳过行，也不在会话日志中记录错误。

如果使用 ERROR 以外的输出默认值，则默认值替代表达式中的 ERROR 函数。例如，您使用表达式中的 ERROR 函数，并将默认值 ‘1234’ 分配至输出端口。每当 PowerCenter 集成服务遇到表达式中的 ERROR 函数时，其均会用值 ‘1234’ 替代错误，并将 ‘1234’ 传递至下一个转换。它不跳过行，也不在日志中记录错误。

语法

ERROR(*string*)

下表介绍了此命令的参数：

参数	必需/ 可选	说明
<i>string</i>	必需	字符串值。当集成服务根据包含 ERROR 函数的表达式跳过行时您想要跳过的消息。字符串长度不限。

返回值

字符串。

示例

以下示例显示如何引用映射，以计算某组织所有部门员工的平均薪酬，但跳过负值。以下表达式嵌套 IIF 表达式中的 ERROR 函数，这样，当 PowerCenter 集成服务在薪酬端口中找到负薪酬时，它就会跳过行并显示错误：

```
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

返回 e 的指定的幂(指数)，其中 e = 2.71828183。例如，EXP(2) 返回 7.38905609893065。可使用此函数分析科学和技术数据，而不是业务数据。EXP 是 LN 函数的倒数，返回数值的自然对数。

语法

EXP(*exponent*)

下表介绍了此命令的参数：

参数	必需/ 可选	说明
<i>指数</i>	必需	数值数据类型。要求其指数的值。方程式 $e^{\text{值}}$ 中的指数。可输入任何有效的转换表达式。

返回值

双精度值。
当作为参数传递至函数的值为 NULL 时返回 NULL。

示例

以下表达式使用数字端口中存储的值作为指数值：

```
EXP( NUMBERS )
```

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

FIRST

返回端口或组中找到的第一个值。或者，可应用筛选器限定 PowerCenter 集成服务读取的行数。只能在 FIRST 中嵌套一个其他汇总函数。

语法

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

下表介绍了此命令的参数：

参数	必需/ 可选	说明
值	必需	除了二进制以外的任何数据类型。传递要为其返回第一个值的值。可输入任何有效的转换表达式。
filter_condition	可选	限制搜索中的行数。筛选条件必须为数值或计算结果为 TRUE、FALSE 或 NULL。可输入任何有效的转换表达式。

返回值

组中的第一个值。
当传递至函数的所有值为 NULL 或未选定行时返回 NULL（例如，筛选条件对所有行的计算结果均为 FALSE 或 NULL）。

空值

如果值为 NULL，则 FIRST 忽略行。然而，如果从端口传递的所有值均为 NULL，则 FIRST 返回 NULL。
注意：默认情况下，PowerCenter 集成服务在汇总函数中将空值当作 NULL 处理。如果传递整个端口或空值组，则函数返回 NULL。然而，在配置 PowerCenter 集成服务时，可选择想要处理汇总函数中空值的方式。可将汇总函数中的空值作为 0 或 NULL 处理。

分组依据

FIRST 根据您在转换中定义的分组依据端口对值进行分组，为每组返回一个结果。

如果不存在分组依据端口，则 FIRST 将所有行当作一个组处理，返回一个值。

示例

以下表达式返回价格大于 \$10.00 的 ITEM_NAME 端口中的第一个值：

```
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

以下表达式返回价格大于 \$40.00 的 ITEM_NAME 端口中的第一个值：

```
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

返回小于或等于传递至此函数的数值的最大整数。例如，如果传递 3.14 至 FLOOR，则函数返回 3。如果传递 3.98 至 FLOOR，则函数返回 3。同样，如果传递 -3.17 至 FLOOR，则函数返回 -4。

语法

FLOOR(*numeric_value*)

下表介绍了此命令的参数：

参数	必需/ 可选	说明
<i>numeric_value</i>	必需	数值数据类型。可输入任何有效转换表达式，只要其计算结果为数值数据即可。

返回值

如果传递的数值的声明精度介于 0 与 28 之间，则返回整数。

如果传递的数值的声明精度大于 28，则返回 Double。

传递至函数的值为 NULL 时返回 NULL。

示例

以下表达式返回小于或等于 PRICE 端口中值的最大整数：

FLOOR(PRICE)

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

提示：可对传递至 FLOOR 的值执行算术。例如，为了将数值乘以 10，然后计算小于其积的最大整数，您可编写以下函数：

FLOOR(UNIT_PRICE * 10)

FV

在制定等额分期付款，投资赚取固定利率的情况下，返回投资未来值。

语法

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

下表介绍了此命令的参数：

参数	必需/ 可选	说明
<i>rate</i>	必需	数值。每期所获的利率。用小数表示。用百分率除以 100，即可用小数它。必须大于或等于 0。
<i>terms</i>	必需	数值。期数或付款次数。必须大于 0。
<i>payment</i>	必需	数值。每期应付款金额。必须是负数
<i>现值</i>	可选	数值。投资的当前值。如果忽略此参数，FV 将使用 0。
<i>类型</i>	可选	整数。付款时间。如果是期初付款，请输入 1。如果是期末付款，请输入 0。默认值为 0。如果输入的值不是 0 或 1，则 PowerCenter 集成服务将该值当作 1 处理。

返回值

数值。

示例

您在帐户中存入 \$2,000，按月复利的年利率为 9%（月利率为 9%/12，或 0.75%）。您计划在接下来 12 个月的每个月初存入 \$250。以下表达式返回 \$5,337.96，作为 12 个月结束时的帐户余额：

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

注意事项

要计算每期利率，请将年利率除以每年的付款次数。付款值和现值为负数，因为它们是您支出金额。

GET_DATE_PART

将日期的指定部分返回为整数值。因此，如果您创建的表达式返回日期的月份部分，且传递 Apr 1 1997 00:00:00 之类的日期，则 GET_DATE_PART 返回 4。

语法

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

下表介绍了此命令的参数：

参数	必需/ 可选	说明
<i>date</i>	必需	Date/Time 数据类型。可输入任何有效的转换表达式。
<i>格式</i>	必需	用于指定日期值中要返回的那一部分的格式字符串。用单引号将格式字符串括起来，例如，'mm'。格式字符串不区分大小写。每个格式字符串均根据会话中指定的日期格式返回日期的整个部分。 用于指定日期值中要返回的那一部分的格式字符串。用单引号将格式字符串括起来，例如，'mm'。格式字符串不区分大小写。每个格式字符串均根据映射中指定的日期格式返回日期的整个部分。 例如，如果传递日期 Apr 1 1997 至 GET_DATE_PART，则格式字符串 'Y'、'YY'、'YYY' 或 'YYYY' 全部返回 1997。

返回值

表示日期中指定部分的整数。

传递至函数的值为 NULL 时返回 NULL。

示例

以下表达式为 DATE_SHIPPED 端口中的每个日期返回小时。12:00:00AM 返回 0，因为默认日期格式基于 24 小时间隔：

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

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

以下表达式为 DATE_SHIPPED 端口中的每个日期返回日部分：

```
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

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

以下表达式为 DATE_SHIPPED 端口中的每个日期返回月份：

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

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

以下表达式为 DATE_SHIPPED 端口中的每个日期返回年份：

```
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

返回指定 timestamp with time zone 列中的时区值。

例如：

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

GET_TIMEZONE 表达式的输出端口必须为 String 数据类型。

语法

```
GET_TIMEZONE (timestamp_with_timezone_value)
```

下表介绍了此命令的参数：

参数	必需/ 可选	说明
<i>timestamp_with_timezone_value</i>	必需	必须为 timestamp with time zone 数据类型。可输入任何有效的转换表达式。

返回值

返回包含时区区域名称或时区偏移量的 String 数据类型。

当输入为空值时返回 NULL。

示例

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

GET_TIMESTAMP

返回 timestampWithTZ 输入类型的 date/time 值。返回的 date/time 将位于所请求的时区中，可将该时区作为第二个参数提供。如果未在第二个参数中指定时区值，则该函数将返回输入 timestampWithTZ 值的时间戳部分。

例如：

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

第一个参数 timestamp with time zone 将 (+05:30) 作为时区值。此函数返回以指定为第二个参数 (+08:30) 的时区表示的时间戳。

GET_TIMESTAMP 表达式的输出端口必须为 date/time。

语法

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

下表介绍了此命令的参数：

参数	必需/ 可选	说明
<i>timestamp_with_timezone_value</i>	必需	必须为 timestamp with time zone 数据类型。可输入任何有效的转换表达式。
<i>timezone_value</i>	可选	必须为 string 数据类型。此字符串必须为字符串。传递要为函数可基于其返回时间戳的时区而显示的值。可输入任何有效的转换表达式。如果未指定时区，该函数将返回第一个参数的时间戳部分。

返回值

返回以指定时区偏移量或区域表示的时间戳值。

如果未传递时区值，该函数将返回第一个参数的时间戳部分。

当输入为空值时返回 NULL。

示例

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

GREATEST

返回输入值列表中的最大值。使用此函数返回最大字符串、日期或数字。默认情况下，此匹配区分大小写。

语法

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

下表介绍了此命令的参数：

参数	必需/ 可选	说明
值	必需	除 Binary 外的任意数据类型。数据类型必须与其他值兼容。要与其他值比较的值。必须至少输入一个值参数。 如果值为数值，且其他输入值为数值，则所有值均使用尽可能最高的精度。例如，如果某些值为 Integer 数据类型，但其他值为 Double 数据类型，则 PowerCenter 集成服务会将值转换为 Double。
CaseFlag	可选	必须为整数。当输入值参数为字符串值时，指定一个值。确定函数中的参数是否区分大小写。可输入任何有效的转换表达式。 当 CaseFlag 为 0 以外的数字时，函数区分大小写。 当 CaseFlag 为 0 时，函数不区分大小写。 默认区分大小写。

返回值

如果是输入值中最大值，则返回 value1，如果是输入值中最大值，则返回 value2，以此类推。

当任何参数均为 NULL 时返回 NULL。

示例

以下表达式返回所订购项目的最大数量：

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

QUANTITY1	QUANTITY2	QUANTITY3	RETURN VALUE
150	756	27	756
			NULL
5000	97	17	5000
120	1724	965	1724

IIF

根据条件的结果返回指定的两个值中的一个。

语法

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

下表介绍了此命令的参数：

参数	必需/ 可选	说明
<i>condition</i>	必需	要计算的条件。可输入计算结果为 TRUE 或 FALSE 的任何有效转换表达式。
<i>value1</i>	必需	除 Binary 以外的任何数据类型。条件为 TRUE 时要返回的值。返回值始终具有此参数指定的数据类型。可输入任何有效的转换表达式，包括另一个 IIF 表达式。
<i>value2</i>	可选	除 Binary 以外的任何数据类型。条件为 FALSE 时要返回的值。可输入任何有效的转换表达式，包括另一个 IIF 表达式。

不同于某些系统中的条件函数，IIF 函数中的 FALSE (*value2*) 条件不是必需项。如果忽略 *value2*，当条件为 FALSE 时，函数返回以下值：

- 当 *value1* 为 Numeric 数据类型时，返回 0。
- 当 *value1* 为 String 数据类型时返回空字符串。
- 当 *value1* 为 Date/Time 数据类型时返回 NULL。

例如，以下表达式不包括 FALSE 条件，并且 *value1* 为 string 数据类型，因此 PowerCenter 集成服务为计算结果为 FALSE 的每一行返回空字符串：

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

SALES	EMP_NAME	RETURN VALUE
150	John Smith	John Smith
50	Pierre Bleu	' ' (<i>empty string</i>)
120	Sally Green	Sally Green
NULL	Greg Jones	' ' (<i>empty string</i>)

返回值

条件为 TRUE 时返回 *value1*。

条件为 FALSE 时，返回 *value2*。

例如，以下表达式包括 FALSE 条件 NULL，因此 PowerCenter 集成服务为计算结果为 FALSE 的每一行返回 NULL：

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

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

如果数据包含多字节字符且条件参数比较字符串数据，则返回值取决于 PowerCenter 集成服务的代码页和数据移动模式。

IIF 和数据类型

使用 IIF 时，返回值的数据类型与最精密结果的数据类型相同。

例如，您有以下表达式：

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

TRUE 结果 (1) 为整数，FALSE 结果 (.3333) 为小数。Decimal 数据类型的精度大于 Integer，因此返回值的数据类型始终为 Decimal。

在高精度模式中运行会话且至少有一个结果为 Double 时，返回值的数据类型为 Double。

在高精度模式中运行映射且至少有一个结果为 Double 时，返回值的数据类型为 Double。

IIF 和复杂数据类型

可以使用 IIF 返回数组或结构，或者返回数组或结构中的元素。

例如，您有以下数组：

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

可以使用以下表达式返回数组中的值之一：

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

返回值: 'Laura'

IIF 的特殊用途

使用嵌套式 IIF 语句测试多个条件。以下示例测试不同的条件，当销售额为 0 或负数时返回 0：

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

可提高此逻辑的可读性，只需添加注释：

```
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)
```

在更新策略中使用 IIF。例如：

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

IIF 的替代项

在许多情况下，使用 [“DECODE” 页面上 91](#) 替代 IIF。DECODE 可提高可读性。以下显示如何通过上节的第一个示例使用 DECODE 替代 IIF：

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

通常可使用筛选器转换替代 IIF 以尽量提高会话性能。

通常可使用筛选器转换替代 IIF 以尽量提高性能。

IN

将输入数据与值的列表匹配。默认情况下，此匹配区分大小写。

语法

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

下表介绍了此命令的参数：

参数	必需/ 可选	说明
<i>valueToSearch</i>	必需	可以是字符串、日期或数值。想要与逗号分隔的值列表匹配的输入值。
<i>值</i>	必需	可以是字符串、日期或数值，具体取决于为 valueToSearch 参数指定的数据类型。要搜索的值的逗号分隔列表。值可以是转换中的端口。可以列出的值没有数量上限。
<i>CaseFlag</i>	可选	必须为整数。 必须是整数或 NULL。 当 valueToSearch 参数是字符串值时，指定一个值。 确定函数中的参数是否区分大小写。可输入任何有效的转换表达式。 当 CaseFlag 为 0 以外的数字时，函数区分大小写。 当 CaseFlag 为 0 时，函数不区分大小写。 当 CaseFlag 为 0 时，函数不区分大小写。 当 CaseFlag 为空值时，该函数将在不匹配函数中的参数时返回 NULL。 否则，CaseFlag 将在匹配函数中的参数时返回 1。 默认区分大小写。

返回值

输入值与值列表匹配时返回 TRUE (1)。

输入值与值列表不匹配时返回 FALSE (0)。

当输入为空值时返回 NULL。

示例

以下表达式确定输入值是安全刀、凿刀还是中等钛刀。输入值与逗号分隔列表中值的大小写不匹配：

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

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

INDEXOF

在值的列表中查找值的索引。默认情况下，此匹配区分大小写。

语法

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

下表介绍了此命令的参数：

参数	必需/ 可选	说明
<i>valueToSearch</i>	必需	字符串数据类型。要在字符串列表中搜索的值。
字符串	必需	字符串数据类型。要在其中执行搜索的逗号分隔值列表。值可以为字符串格式。可以列出的值没有数量上限。此值区分大小写，除非将 CaseFlag 设置为 0。
<i>CaseFlag</i>	可选	必须为整数。当 valueToSearch 参数是字符串值时，指定一个值。确定函数中的参数是否区分大小写。可输入任何有效的转换表达式。 当 CaseFlag 为 0 以外的数字时，函数区分大小写。 当 CaseFlag 为 0 时，函数不区分大小写。

返回值

当输入值与 *string1* 匹配时返回 1，当输入值与 *string2* 匹配时返回 2，以此类推。

未找到输入值时返回 0。

当输入为空值时返回 NULL。

示例

以下表达式确定 ITEM_NAME 端口的值是与第一个、第二个还是第三个字符串相匹配：

```
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

安全刀返回 0 值，因为它与输入值大小写不匹配。

INITCAP

将字符串中每个单词的首字母大写，并将其他所有字母转换为小写。可通过空格（空格键，换页键，换行键，回车键，制表键或垂直制表键）和非字母字符删除单词。例如，如果传递字符串 ‘...THOMAS’，函数返回 Thomas。

语法

INITCAP(*string*)

下表介绍了此命令的参数：

参数	必需/ 可选	说明
字符串	必需	除了二进制以外的任何数据类型。可输入任何有效的转换表达式。

返回值

字符串。如果数据包含多字节字符，则返回值取决于 PowerCenter 集成服务的代码页和数据移动模式。

传递至函数的值为 NULL 时返回 NULL。

示例

以下表达式将 FIRST_NAME 端口中的所有名称设为大写：

INITCAP(FIRST_NAME)

FIRST_NAME	RETURN VALUE
ramona	Ramona
18-albert	18-Albert
NULL	NULL
?!SAM	?!Sam
THOMAS	Thomas
PierRe	Pierre

INSTR

按照从左到右计数，返回字符串中字符集的位置。

语法

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

下表介绍了此命令的参数：

参数	必填/ 可选	说明
<i>string</i>	必需	此字符串必须为字符串。传递要计算的值。可输入任何有效的转换表达式。表达式结果必须为字符串。如果不是，INSTR 将值转换为字符串后再计算它。
<i>search_value</i>	必需	任何值。搜索值区分大小写。要搜索的字符集。search_value 必须与字符串的一部分相匹配。例如，如果编写 INSTR('Alfred Pope', 'Alfred Smith') ，则函数返回 0。 可输入任何有效的转换表达式。如果要搜索字符串，请用单引号将要搜索的字符括起来，例如，'abc'。
<i>启动</i>	可选	必须为整数值。要在字符串中开始搜索的位置。可输入任何有效的转换表达式。 默认值为 1，这意味着 INSTR 从字符串的第一个字符开始搜索。 如果起始位置为 0，则 INSTR 从字符串的第一个字符搜索。如果起始位置为正数，则 INSTR 从字符串的开头计数以查找起始位置。如果起始位置为负数，则 INSTR 从字符串的末尾计数以查找起始位置。如果忽略此参数，则函数使用默认值 1。
<i>occurrence</i>	可选	大于 0 的正整数。可输入任何有效的转换表达式。如果搜索值在字符串中出现多次，则可指定要搜索第几次出现。例如，如果要从起始位置搜索第二次出现，则可输入 2。 如果忽略此参数，则函数使用默认值 1，这意味着 INSTR 搜索第一次出现的搜索值。如果传递小数，则 PowerCenter 集成服务会将其四舍五入到最接近的整数。如果传递负整数或 0，则会话失败。
<i>comparison_type</i>	可选	当 PowerCenter 集成服务在 Unicode 模式中运行时，字符串比较类型为语言比较或二进制比较。当 PowerCenter 集成服务在 ASCII 模式中运行时，比较类型始终为二进制。 语言比较考虑语言特定排序规则，而二进制比较执行按位匹配。例如，在语言比较中，German sharp s 字符与字符串“ss”相匹配，但在二进制比较中却不匹配。二进制比较运行速度快于语言比较。 必须为整数值，0 或 1： - 0: INSTR 执行语言字符串比较。 - 1: INSTR 执行二进制字符串比较。 默认值为 0。 如果输入 0，则会话排序顺序不能是二进制。

返回值

搜索成功时返回整数。整数表示 *search_value* 中第一个字符的位置，从左到右计数。

搜索不成功时返回 0。

传递至函数的值为 NULL 时返回 NULL。

示例

以下表达式返回字母 ‘a’ 的第一次出现位置，从每家公司名称开头开始。因为 *search_value* 参数区分大小写，所以，它跳过 ‘Blue Fin Aqua Center’ 中的 ‘A’，返回 ‘a’ 在 ‘Aqua’ 中的位置：

```
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

以下表达式返回字母 ‘a’ 的第二次出现位置，从每家公司名称开头开始。因为 *search_value* 参数区分大小写，所以，它跳过 ‘Blue Fin Aqua Center’ 中的 ‘A’，返回 0：

```
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

以下表达式返回字母 ‘a’ 在每家公司名称中的第二次出现位置，从公司名称的最后一个字符开始。因为 *search_value* 参数区分大小写，所以，它跳过 ‘Blue Fin Aqua Center’ 中的 ‘A’，返回 0：

```
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

以下表达式返回字符串 ‘Blue Fin Aqua Center’ 中第一个字符的位置（从公司名称的最后一字符开始）：

```
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

使用嵌套式 INSTR

可将 INSTR 函数嵌套于其他函数中，以完成更为复杂的任务。

以下表达式计算字符串，从字符串的末尾开始。此表达式查找字符串中的最后（最右）空格，然后返回其左侧的所有字符：

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

CUST_NAME	RETURN VALUE
PATRICIA JONES	PATRICIA
MARY ELLEN SHAH	MARY ELLEN

以下表达式删除字符串中的字符 '#'：

```
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

返回值是否为 NULL。ISNULL 将空字符串计算为 FALSE。

注意：要测试是否有空字符串，请使用 LENGTH。

语法

```
ISNULL( value )
```


下表介绍了此命令的参数：

参数	必需/ 可选	说明
<i>value</i>	必需	除 Binary 以外的任何数据类型。传递要计算的行。可输入任何有效的转换表达式。

返回值

当值为 NULL 时返回 TRUE (1)。

当值不为 NULL 时返回 FALSE (0)。

示例

以下示例检查项目表中是否有空值：

```
ISNULL( ITEM_NAME )
```

ITEM_NAME	RETURN VALUE
Flashlight	0 (FALSE)
NULL	1 (TRUE)
Regulator system	0 (FALSE)
' '	0 (FALSE) <i>Empty string is not NULL</i>

ISNULL 和复杂数据类型

可以使用 ISNULL 检查数组或结构是否具有空值。

以下表达式检查下列复杂数据类型中的空值：

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

IS_DATE

返回字符串值是否为有效日期。有效日期是会话中指定的日期时间格式的日期部分中的任何字符串。如果要测试的字符串不是此日期格式，请使用 TO_DATE 格式字符串指定日期格式。如果传递至 IS_DATE 的字符串与指定的格式字符串不匹配，则函数返回 FALSE (0)。如果此字符串与格式字符串匹配，则函数返回 TRUE (1)。

IS_DATE 计算字符串并返回整数值。

IS_DATE 表达式的输出端口必须为 String 或 Numeric 数据类型。

可使用 IS_DATE 测试或筛选平面文件中的数据，然后再将其写入目标。

RR 格式字符串与 IS_DATE 一起使用，而不是与 YY 格式字符串一起使用。在大多数情况下，两个格式字符串均返回相同的值，但在一些独特情况下 YY 返回不正确的结果。例如，表达式 IS_DATE('02/29/00' , 'YY') 在内部作为 IS_DATE(02/29/1900 00:00:00) 计算，返回 false。然而，PowerCenter 集成服务 将表达式 IS_DATE('02/29/00' , 'RR') 作为 IS_DATE(02/29/2000 00:00:00) 计算，返回 TRUE。在第一种情况下，年份 1900 不是闰年，因此，没有二月 29 日。

注意: IS_DATE 与 TO_DATE 使用相同的格式字符串。

语法

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

下表介绍了此命令的参数：

参数	必需/ 可选	说明
值	必需	必须是 string 数据类型。传递要计算的行。可输入任何有效的转换表达式。
格式	可选	输入有效的 TO_DATE 格式字符串。此格式字符串必须与 <i>string</i> 参数的各部分相匹配。例如，如果传递字符串 'Mar 15 1997 12:43:10AM'，则必须使用格式字符串 'MON DD YYYY HH12:MI:SSAM'。如果忽略格式字符串，则字符串值必须采用会话中指定的日期格式。 输入有效的 TO_DATE 格式字符串。此格式字符串必须与 <i>string</i> 参数的各部分相匹配。例如，如果传递字符串 'Mar 15 1997 12:43:10AM'，则必须使用格式字符串 'MON DD YYYY HH12:MI:SSAM'。如果忽略格式字符串，则字符串值必须采用映射配置中指定的日期格式。

返回值

当行是有效日期时返回 TRUE (1)。

当行不是有效日期时返回 FALSE (0)。

当表达式中的值为 NULL 或格式字符串为 NULL 时返回 NULL。

警告: IS_DATE 字符串的格式必须与格式字符串相匹配，包括任何日期分隔符。如果不匹配，则 PowerCenter 集成服务可能返回不准确的值或跳过记录。

示例

以下表达式检查 INVOICE_DATE 端口是否存在有效日期：

IS_DATE(INVOICE_DATE)

此表达式返回的数据类似于：

INVOICE_DATE	RETURN VALUE
NULL	NULL
'180'	0 (FALSE)
'04/01/98'	0 (FALSE)
'04/01/1998 00:12:15.7008'	1 (TRUE)
'02/31/1998 12:13:55.9204'	0 (FALSE) (<i>February does not have 31 days</i>)
'John Smith'	0 (FALSE)

以下 IS_DATE 表达式指定格式字符串 ‘YYYY/MM/DD’：

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

如果字符串值与此格式不匹配，则 IS_DATE 返回 FALSE：

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

以下示例显示在使用 TO_DATE 将字符串转换为日期之前如何使用 IS_DATE 测试数据。此表达式检查 INVOICE_DATE 端口中的值，并将每个有效日期转换为日期值。如果值不是有效日期，则 PowerCenter 集成服务返回 ERROR 并跳过行。

此示例返回 Date/Time 值。因此，表达式的输出端口需要为 Date/Time：

```
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'

INVOICE_DATE	RETURN VALUE
'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

返回字符串是否为有效数字。有效数字由以下部分组成：

- 数字之前的可选空格
- 可选符号 (+/-)
- 带有可选小数点的一位或多位数
- 可选科学表示法，例如，字母 ‘e’ 或 ‘E’（Windows 上的字母 ‘d’ 或 ‘D’），之后相继是可选符号 (+/-) 以及一个或多个位数
- 数字之后的可选空格

以下数字均有效：

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

IS_NUMBER 表达式的输出端口必须为 String 或 Numeric 数据类型。

可使用 IS_NUMBER 测试或筛选平面文件中的数据，然后再将其写入目标。

语法

IS_NUMBER(*value*)

下表介绍了此命令的参数：

参数	必需/ 可选	说明
<i>值</i>	必需	必须是 String 数据类型。传递要计算的行。可输入任何有效的转换表达式。

返回值

当行是有效数字时返回 TRUE (1)。

当行不是有效数字时返回 FALSE (0)。

当表达式中的值为 NULL 时返回 NULL。

示例

以下表达式检查 ITEM_PRICE 端口是否存在有效数字：

```
IS_NUMBER( ITEM_PRICE )
```

ITEM_PRICE	RETURN VALUE
'123.00'	1 (True)
'-3.45e+3'	1 (True)
'-3.45D-3'	1 (True - Windows only)
'-3.45d-3'	0 (False - UNIX only)
'3.45E- '	0 (False) <i>Incomplete number</i>
' '	0 (False) <i>Consists entirely of blanks</i>
''	0 (False) <i>Empty string</i>
' +123abc'	0 (False)
' 123'	1 (True) <i>Leading white blanks</i>
'123 '	1 (True) <i>Trailing white blanks</i>
'ABC'	0 (False)
'-ABC'	0 (False)
NULL	NULL

使用 IS_NUMBER 测试数据，然后再使用其中一个数值转换函数，例如，TO_FLOAT。例如，以下表达式检查 ITEM_PRICE 端口中的值，并将每个有效数字转换为双精度浮点值。如果值不是有效数字，则 PowerCenter 集成服务返回 0.00：

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

ITEM_PRICE	RETURN VALUE
'123.00'	123
'-3.45e+3'	-3450
'3.45E-3'	0.00345
' '	0.00 <i>Consists entirely of blanks</i>
''	0.00 <i>Empty string</i>
' +123abc'	0.00

ITEM_PRICE	RETURN VALUE
' ' 123ABC'	0.00
'ABC'	0.00
'-ABC'	0.00
NULL	NULL

IS_SPACES

返回字符串值是否完全由空格组成。空格可以运用空格键，换页键，换行键，回车键，制表键或垂直制表键输入。

IS_SPACES 将空字符串计算为 FALSE，因为没有空格。要测试是否有空字符串，请使用 LENGTH。

语法

IS_SPACES(*value*)

下表介绍了此命令的参数：

参数	必需/ 可选	说明
<i>值</i>	必需	必须是 string 数据类型。传递要计算的行。可输入任何有效的转换表达式。

返回值

行完全由空格组成时返回 TRUE (1)。

行包含数据时返回 FALSE (0)。

当表达式中的值为 NULL 时返回 NULL。

示例

以下表达式检查 ITEM_NAME 端口是否有完全由空格组成的行：

IS_SPACES(ITEM_NAME)

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

提示: 使用 IS_SPACES 避免将空格写入目标表中的字符列。例如，如果您的转换将客户名称写入目标表中固定长度 CHAR(5) 列，则您可能想要写入 '00000' 而不是空格。您将创建以下类似表达式：

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

LAG

在表达式转换中返回一个值，该值表示在当前行之前偏移的行数。在 Hadoop 环境中使用 Spark 引擎运行映射时，使用此函数可将当前行中的值与前一行中的值相比较。

在数据集中的当前行之前，将出现滞后值。

在转换中使用 LAG 时，必须为转换配置窗口化。窗口化属性定义了数据的分区方式和排序方式。

语法

```
LAG ( column_name, offset, default )
```

下表介绍了此命令的参数：

参数	必需/ 可选	说明
<i>column_name</i>	必需	函数对其执行运算的目标列或表达式。
<i>offset</i>	必需	Integer 数据类型。要从当前行之前的行获取值的行数。
<i>default</i>	可选	在偏移量超出分区或表的边界时返回的默认值。默认值为 NULL。

返回值

指定的 *column_name* 的数据类型。

Default，如果返回值超出指定分区的边界。

如果 *default* 被忽略或设置为 NULL，则返回 NULL。

示例

以下表达式将返回已下的上一个订单的日期：

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

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

ORDER_DATE	ORDER_ID	RETURN VALUE
2017/09/30	6	2017/09/29

第一行的滞后值超出分区，因此函数返回了默认值 NULL。

在以下示例中，您的组织接收车辆发出的 GPS 脉冲信号，包括行程 ID 和事件 ID 以及时间戳。您要计算每次脉冲信号之间的时间差。

以下表达式为两段单独行程计算当前行和上一行之间的时间差：

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

可以按行程 ID 对数据进行分区并按事件 ID 进行排序。

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

第一行和第四行的滞后值超出指定的分区，因此函数返回了两个默认值 NULL。

LAST

返回选定端口中最后一行。或者，可应用筛选器限定 PowerCenter 集成服务读取的行数。只能在 LAST 中嵌套一个其他汇总函数。

语法

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

下表介绍了此命令的参数：

参数	必需/ 可选	说明
<i>值</i>	必需	除了二进制以外的任何数据类型。传递要为其返回最后一行的值。可输入任何有效的转换表达式。
<i>filter_condition</i>	可选	限制搜索中的行数。筛选条件必须为数值或计算结果为 TRUE、FALSE 或 NULL。可输入任何有效的转换表达式。

返回值

端口中的最后一行。

当传递至函数的所有值为 NULL 或未选定行时返回 NULL（例如，筛选条件对所有行的计算结果均为 FALSE 或 NULL）。

注意: 默认情况下，PowerCenter 集成服务在汇总函数中将空值当作 NULL 处理。如果传递整个端口或空值组，则函数返回 NULL。然而，在配置 PowerCenter 集成服务时，可选择想要处理汇总函数中空值的方式。可将汇总函数中的空值作为 0 或 NULL 处理。

示例

以下表达式返回价格大于 \$10.00 的 ITEMS_NAME 端口中的最后一行：

```
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

为端口中每个日期返回一个月中最后一天的日期。

语法

```
LAST_DAY( date )
```

下表介绍了此命令的参数：

参数	必需/ 可选	说明
<i>date</i>	必需	Date/Time 数据类型。传递想要为其返回一个月中最后一天的日期。可输入任何有效的转换表达式以计算日期。

返回值

日期。与传递至此函数的该日期值对应的一个月中最后一天。

当选定端口中的值为 NULL 时返回 NULL。

空值

如果值为 NULL，则 LAST_DAY 忽略行。然而，如果从端口传递的所有值均为 NULL，则 LAST_DAY 返回 NULL。

分组依据

LAST_DAY 根据您在转换中定义的分组依据端口对值进行分组，为每组返回一个结果。如果不存在分组依据端口，则 LAST_DAY 将所有行当作一个组处理，返回一个值。

示例

以下表达式为 ORDER_DATE 端口中的每个日期返回一个月中的最后一天：

```
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 (<i>Leap year</i>)
NULL	NULL
Jul 31 1998 12:00:00AM	Jul 31 1998 12:00:00AM

可嵌套 TO_DATE 以将字符串值转换为日期。TO_DATE 始终包括时间信息。如果传递的字符串没有时间值，则返回的日期将包括时间 00:00:00。

以下示例以与字符串相同的格式为每个订单日期返回一个月的最后一天：

```
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 (<i>Leap year</i>)

LEAD

在表达式转换中返回一个值，该值表示在当前行之后偏移的行数。在 Hadoop 环境中使用 Spark 引擎运行映射时，使用此函数可将当前行中的值与后一行中的值相比较。

在数据集中的当前行之后，将出现前导值。

注意：在转换中使用 LEAD 时，必须为转换配置窗口化。窗口化属性定义了数据的分区方式和排序方式。

语法

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

下表介绍了此命令的参数：

参数	必需/ 可选	说明
<i>column_name</i>	必需	函数对其执行运算的目标列或表达式。
<i>offset</i>	必需	Integer 数据类型。要从当前行之后的行获取值的行数。
<i>default</i>	可选	在偏移量超出分区或表的边界时返回的默认值。默认值为 NULL。

返回值

指定的 *column_name* 的数据类型。

Default，如果返回值超出指定分区的边界。

如果 *default* 被忽略或设置为 NULL，则返回 NULL。

示例

以下表达式返回每位员工之后的下一位员工的录用日期：

LEAD (HIRE_DATE, 1, NULL)

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

最后一行没有前导值可用，因此函数返回了默认值 NULL。

以下表达式返回两个日历年的第一季度和第三季度之间的销售配额差异：

LEAD (Sales_Quota, 2, 0) - Sales_Quota

可以按年份对数据进行分区并按季度进行排序。

YEAR	QUARTER	SALES_QUOTA	QUOTA_DIFF
2016	1	300	7700

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

第二季度和第三季度的前导值超出指定的分区，因此函数返回了值“0”。

LEAST

返回输入值的列表中的最小值。默认情况下，此匹配区分大小写。

语法

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

LEAST(*value1*, [*value2*, ..., *valueN*])

下表介绍了此命令的参数：

参数	必需/ 可选	说明
<i>值</i>	必需	除了二进制以外的任何数据类型。数据类型必须与其他值兼容。要与其他值比较的值。必须至少输入一个值参数。 如果值为 Numeric，且其他输入值为其他数值数据类型，则所有值均使用尽可能最高的精度。例如，如果某些值为 Integer 数据类型，但其他值为 Double 数据类型，则 PowerCenter 集成服务会将值转换为 Double。
<i>CaseFlag</i>	可选	必须为整数。当输入值参数为字符串值时，指定一个值。确定函数中的参数是否区分大小写。可输入任何有效的转换表达式。 当 CaseFlag 为 0 以外的数字时，函数区分大小写。 当 CaseFlag 为 0 时，函数不区分大小写。 默认区分大小写。

返回值

如果是输入值中最小值，则返回 *value1*，如果是输入值中最小值，则返回 *value2*，以此类推。

当任何参数均为 NULL 时返回 NULL。

示例

以下表达式返回所订购项目的最小数量：

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

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

LENGTH

返回字符串中的字符数，其中包括尾随空白。

语法

```
LENGTH( string )
```

下表介绍了此命令的参数：

参数	必需/ 可选	说明
<i>string</i>	必需	字符串数据类型。 要计算的字符串。 可输入任何有效的转换表达式。

返回值

表示字符串长度的整数。

传递至函数的值为 NULL 时返回 NULL。

示例

以下表达式返回每个客户名称的长度：

```
LENGTH( CUSTOMER_NAME )
```

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

LENGTH 提示

使用 LENGTH 测试是否存在空字符串条件。如果想要查找客户名称为空的字段，请使用以下类似表达式：

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

要测试是否有空字段，请使用 ISNULL。要测试是否有空格，请使用 IS_SPACES。

LN

返回数值的自然对数。例如，LN(3) 返回 1.098612。通常使用此函数分析科学数据，而不是业务数据。
此函数是函数 EXP 的倒数。

语法

```
LN( numeric_value )
```

下表介绍了此命令的参数：

参数	必需/ 可选	说明
numeric_value	必需	数值数据类型。它必须是大于 0 的正数。传递要为其计算自然对数的值。可输入任何有效的转换表达式。

返回值

双精度值。
传递至函数的值为 NULL 时返回 NULL。

示例

以下表达式为 NUMBERS 端口中的所有值返回自然对数：

```
LN( NUMBERS )
```

NUMBERS	RETURN VALUE
10	2.302585092994
125	4.828313737302
0.96	-0.04082199452026
NULL	NULL
-90	Error. (The Integration Service does not write row.)
0	Error. (The Integration Service does not write row.)

注意：传递负数或 0 时，PowerCenter 集成服务显示错误，且不写入行。numeric_value 必须为大于 0 的正数。

LOG

返回数值的对数。 此函数更常用于分析科学数据，而不是业务数据。

语法

LOG(base, exponent)

下表介绍了此命令的参数：

参数	必需/ 可选	说明
底数	必需	对数的底数。 必须为 0 或 1 之外的正数值。 计算结果为 0 或 1 之外的正数的任何有效转换表达式。
指数	必需	对数的指数。 必须是大于 0 的正数值。 计算结果为大于 0 的正数的任何有效转换表达式。

返回值

双精度值。

传递至函数的值为 NULL 时返回 NULL。

示例

以下表达式返回 NUMBERS 端口中所有值的对数：

LOG(BASE, EXPONENT)

BASE	EXPONENT	RETURN VALUE
15	1	0
.09	10	-0.956244644696599
NULL	18	NULL
35.78	NULL	NULL
-9	18	Error. (PowerCenter 集成服务 does not write the row.)
0	5	Error. (PowerCenter 集成服务 does not write the row.)
10	-2	Error. (PowerCenter 集成服务 does not write the row.)

如果传递负数、0 或 1 作为基数值或如果传递指数的负值，则 PowerCenter 集成服务显示错误且不写入行。

LOOKUP

在查找源列中搜索值。

LOOKUP 函数将查找源中的数据与您指定的值相比较。当 PowerCenter 集成服务在查找表中查找搜索值时，它返回查找表中同一行指定列中的值。

根据使用 LOOKUP 函数的映射创建会话时，必须为会话属性中的 \$Source Connection Value 和 \$Target Connection Value 指定数据库连接。为了验证表达式转换中的 LOOKUP 函数，请确保查找定义在映射中。

注意: 此函数在 mapplet 中不受支持。

使用查找转换或 LOOKUP 函数

使用查找转换而不是 LOOKUP 函数在 PowerCenter 映射中查找值。如果使用映射中的 LOOKUP 函数，则需要为会话属性中的 3.5 兼容性启用查找缓存选项。对于想要继续使用 LOOKUP 函数而不是创建查找转换的 PowerMart 3.5 用户，此选项明确可供其使用。有关更多信息，请参阅《PowerCenter 转换指南》中的“查找转换”。

可为 LOOKUP 函数中的一个查找表定义多个搜索。然而，每个搜索必须找到一个匹配值才能返回查找值。

语法

```
LOOKUP( result, search1, value1 [, search2, value2]... )
```

下表介绍了此命令的参数：

参数	必需/ 可选	说明
result	必需	除了二进制以外的任何数据类型。必须为与搜索相同的查找表中的输出端口。当搜索与值匹配时指定返回值。始终在此参数之前放置引用限定符 :TD。
search1	必需	与 value1 相匹配的数据类型。必须为与结果相同的查找表中的输出端口。指定要与某值相匹配的值。始终在此参数之前放置引用限定符 :TD。
value1	必需	除了二进制以外的任何数据类型。必须与 search1 数据类型相匹配。要在 search1 中指定的查找源列中搜索的值。可输入任何有效的转换表达式。

返回值

当所有搜索均找到匹配值时，返回 Result。如果 PowerCenter 集成服务找到匹配值，则它返回与 search1 参数相同的行中的结果。

搜索未找到任何匹配值时返回 NULL。

搜索找到多个匹配值时返回 Error。

示例

以下表达式搜索在查找源 :TD.SALES 中搜索特定项目 ID 和价格，当两项搜索均找到匹配值时返回项目名称：

```
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

ITEM_NAME	ITEM_ID	PRICE
Halogen Flashlight	15	15.99
NULL	20	15.99

RETURN VALUE: Flashlight

LOOKUP 提示

如将 char 值与 varchar 值相比较，则 LOOKUP 函数只有在两行匹配时才返回结果。这意味着每行的值与长度均必须相匹配。如果查找源为填充的指定长度的 char 值，且查找搜索为 varchar 值，则需要使用 RTRIM 函数裁减查找源的尾随空白，以使值与查找搜索相匹配：

```
LOOKUP(:TD.ORDERS.PRICE, :TD.ORDERS.ITEM, RTRIM( ORDERS.ITEM, ' '))
```

在 LOOKUP 函数的 *result* 和 *search* 参数中使用 :TD 引用限定符：

```
LOOKUP(:TD.ORDERS.ITEM, :TD.ORDERS.PRICE, ORDERS.PRICE, :TD.ORDERS.QTY, ORDERS.QTY)
```

LOWER

将大写字符串字符转换为小写。

语法

```
LOWER( string )
```

下表介绍了此命令的参数：

参数	必需/ 可选	说明
<i>string</i>	必需	任何字符串值。参数传递要作为小写返回的字符串值。可输入任何有效的转换表达式以计算字符串。

返回值

小写字符串。如果数据包含多字节字符，则返回值取决于集成服务的代码页和数据移动模式。

当选定端口中的值为 NULL 时返回 NULL。

示例

以下表达式将所有名字转换为小写：

```
LOWER( FIRST_NAME )
```

FIRST_NAME	RETURN VALUE
antonia	antonia
NULL	NULL

FIRST_NAME	RETURN VALUE
THOMAS	thomas
PierRe	pierre
BERNICE	bernice

LPAD

将一组空白或字符添加到字符串的开头，以将该字符串设置为指定长度。

语法

```
LPAD( first_string, length [,second_string] )
```

下表介绍了此命令的参数：

参数	必需/ 可选	说明
<i>first_string</i>	必需	可以是字符串。要更改的字符串。可输入任何有效的转换表达式。
长度	必需	必须为正整数文字。此参数指定所需的每个字符串长度。
<i>second_string</i>	可选	可以是任何字符串值。要附加至 <i>first_string</i> 值左侧的字符。可输入任何有效的转换表达式。可输入特定字符串文字。然而，请用单引号将要添加至字符串开头的字符括起来，例如，'abc'。此参数区分大小写。如果忽略 <i>second_string</i> ，函数将在第一个字符串的开头填充空白。

返回值

指定长度的字符串。

当传递至函数的值为 NULL 或 长度为负数时返回 NULL。

示例

以下表达式将数字标准化为六位数，方法是向其填充前导零：

```
LPAD( PART_NUM, 6, '0')
```

PART_NUM	RETURN VALUE
702	000702
1	000001
0553	000553
484834	484834

LPAD 从左到右计算长度。如果第一个字符串长于指定长度，则 LPAD 从右至左截断字符串。例如，LPAD('alphabetical' , 5, 'x') 返回字符串 'alpha' 。

如果第二个字符串长于返回指定长度所需的总字符数，则 LPAD 使用第二个字符串的一部分：

```
LPAD( ITEM_NAME, 16, '*' )
```

ITEM_NAME	RETURN VALUE
Flashlight	*..**.Flashlight
Compass	*..**.*Compass
Regulator System	Regulator System
Safety Knife	*..*Safety Knife

LTRIM

从字符串的开头删除空白或字符。可将 LTRIM 与表达式或更新策略转换中的 IIF 或 DECODE 一起使用，以避免目标表中出现空格。

如不在表达式中指定 *trim_set* 参数：

- 在 UNICODE 模式中，LTRIM 删除字符串开头的单字节和双字节空格。
- 在 ASCII 模式中，LTRIM 只删除单字节空格。

如果使用 LTRIM 删除字符串字符，则 LTRIM 将 *trim_set* 与 *string* 参数中的每个字符逐一作比较，从字符串的左侧开始。如果字符串中的字符与 *trim_set* 中的任何字符相匹配，则 LTRIM 将删除它。LTRIM 继续比较和删除字符，直到它未能在 *trim_set* 中找到匹配字符。然后，它返回不包括匹配字符的字符串。

语法

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

下表介绍了此命令的参数：

参数	必需/ 可选	说明
<i>string</i>	必需	任何字符串值。传递要修改的字符串。可输入任何有效的转换表达式。使用运算符执行比较或连接字符串，然后再删除字符串开头的字符。
<i>trim_set</i>	可选	任何字符串值。传递要从第一个字符串开头删除的字符。可输入任何有效的转换表达式。也可输入字符串。然而，请用单引号将要从字符串开头删除的字符括起来，例如，'abc'。如果忽略第二个字符串，则函数删除字符串开头的任何空白。 LTRIM 区分大小写。例如，如果要删除字符串 'Alfredo' 中的 'A'，则将输入 'A' 而不是 'a'。

返回值

字符串。已将 *trim_set* 参数中指定字符删除的字符串值。

传递至函数的值为 NULL 时返回 NULL。当 *trim_set* 为空时，函数返回 NULL。

示例

以下表达式从 LAST_NAME 端口中的字符串删除字符 ‘S’ 和 ‘.’ ：

```
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 删除 S. MacDonald 中的 ‘S.’ 以及 Sawyer 和 Steadman 中的 ‘S’ ，但不删除 H. Bender 中的句号。这是因为 LTRIM 逐字符搜索您在 *trim_set* 参数中指定的字符集。如果字符串中的第一个字符与 *trim_set* 中的第一个字符相匹配，则 LTRIM 将删除它。然后，LTRIM 查看字符串中的第二个字符。如果它与 *trim_set* 中的第二个字符相匹配，则 LTRIM 将删除它，以此类推。当字符串中的第一个字符与 *trim_set* 中的对应字符不匹配时，LTRIM 返回字符串并计算下一行。

在 H. Bender 示例中，H 与 *trim_set* 参数中的任一字符均不匹配，因此，LTRIM 返回 LAST_NAME 端口中的字符串，并移至下一行。

LTRIM 提示

将 RTRIM 和 LTRIM 与 || 或 CONCAT 一起使用，以便在连接两个字符串后删除前导和尾随空白。

也可删除多个字符集，只需嵌套 LTRIM。例如，如果要删除名称列中的前导空白和字符 'T'，则可创建以下类似表达式：

```
LTRIM( LTRIM( NAMES ), 'T' )
```

MAKE_DATE_TIME

基于输入值返回日期和时间。

语法

```
MAKE_DATE_TIME( year, month, day, hour, minute, second, nanosecond )
```

下表介绍了此命令的参数：

参数	必需/ 可选	说明
<i>year</i>	必需	数值数据类型。4 位正整数。如果向此函数传递一个 2 位数年份，则 PowerCenter 集成服务返回 “00” 作为年份的头两位数。
<i>month</i>	必需	数值数据类型。1 与 12 之间的正整数（一月 = 1，十二月 = 12）。
<i>day</i>	必需	数值数据类型。1 与 31 之间的正整数（但不足 31 天的月份除外：二月、四月、六月、九月和十一月）。
<i>hour</i>	可选	数值数据类型。0 与 24 之间的正整数（其中 0 = 12AM，12 = 12PM，24 = 12AM）。
<i>minute</i>	可选	数值数据类型。0 与 59 之间的正整数。
<i>second</i>	可选	数值数据类型。0 与 59 之间的正整数。
<i>nanosecond</i>	可选	数值数据类型。0 与 999,999,999 之间的正整数。

返回值

MM/DD/YYYY HH24:MI:SS 格式的日期。不向函数传递年份、月份或日时，返回空值。

示例

以下表达式从输入端口创建日期和时间：

```
MAKE_DATE_TIME( SALE_YEAR, SALE_MONTH, SALE_DAY, SALE_HOUR, SALE_MIN, SALE_SEC )
```

SALE_YR	SALE_MTH	SALE_DAY	SALE_HR	SALE_MIN	SALE_SEC	RETURN VALUE
2002	10	27	8	36	22	10/27/2002 08:36:22
2000	6	15	15	17		06/15/2000 15:17:00
2003	1	3		22	45	01/03/2003 00:22:45
04	3	30	12	5	10	03/30/0004 12:05:10
99	12	12	5		16	12/12/0099 05:00:16

MAP

基于指定的键-值对生成元素映射。

语法

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

下表介绍了此命令的参数：

参数	必需/可选	说明
map_key1	必需	任何原始数据类型。想要添加为映射数据的键的元素。可输入任何有效的转换表达式。
map_value1	必需	任何原始或复杂数据类型。想要添加为映射数据键的值的元素。可输入任何有效的转换表达式。

如果在映射端口的输出表达式中使用 MAP 函数，则函数参数的数据类型必须与映射端口的类型配置中指定的映射元素数据类型匹配。map_key 不能为空。

返回值

映射。

参数的数据类型决定映射元素的数据类型。例如，如果将整数参数作为键传递，将结构参数作为值传递，此函数将使用整数类型和结构类型的键-值对生成映射数据。

示例

以下表达式将生成整数元素和字符串元素的映射。

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

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

MAP_FROM_ARRAYS

从指定的键和值数组生成映射。

语法

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

下表介绍了此命令的参数：

参数	必需/可选	说明
map_keys	必需	元素为原始数据类型的数组类型。想要添加为映射数据的键的元素数组。可输入任何有效的转换表达式。
map_values	必需	元素为任意数据类型的数组类型。想要添加为映射数据的值的元素数组。可输入任何有效的转换表达式。

map_keys 数组中的元素数与 map_values 数组中的元素数必须匹配。如果在映射端口的输出表达式中使用 MAP 函数，则函数参数的数据类型必须与映射端口的类型配置中指定的映射元素数据类型匹配。

返回值

映射。

参数的数据类型决定映射元素的数据类型。例如，如果将整数参数作为键传递，将结构参数作为值传递，此函数将使用整数类型和结构类型的键-值对生成映射数据。

示例

以下表达式将用字符串类型的数组元素生成一个映射。

```
MAP_FROM_ARRAYS(cust_name, cust_phone)
```

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

MAP_KEYS

返回指定映射的键元素数组。

语法

```
MAP_KEYS(map as MAP)
```

下表介绍了此命令的参数：

参数	必需/可选	说明
map	必需	映射数据类型。想要从其中检索键的键-值对元素的映射。可输入任何有效的转换表达式。

返回值

数组。

键的数据类型决定数组元素的数据类型。例如，如果键属于字符串类型，此函数将使用字符串元素生成数组数据。

如果映射键为空，则返回 -1。

如果映射为空，则返回 NULL。

示例

以下表达式将使用字符串类型元素生成一个数组。

```
MAP_KEYS(stock_sellprice)
```

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

MAP_VALUES

返回指定映射的值元素数组。

语法

```
MAP_VALUES(map as MAP)
```

下表介绍了此命令的参数：

参数	必需/可选	说明
map	必需	映射数据类型。想要从其中检索值的键-值对元素的映射。可输入任何有效的转换表达式。

返回值

数组。

映射中值的数据类型决定数组元素的数据类型。例如，如果值属于整数类型，此函数将使用整数元素生成数组数据。

如果映射值为空，则返回 -1。

如果映射为空，则返回 NULL。

示例

以下表达式将使用字符串类型元素生成一个数组。

```
MAP_VALUES(stock_sellprice)
```

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

MAX (Dates)

返回端口或组中发现的最晚日期。可应用筛选器以限制搜索中的行数。只能在 MAX 中嵌套一个其他汇总函数。

也可使用 MAX 返回端口或组中的最大数值或最大字符串值。

语法

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

下表介绍了此命令的参数：

参数	必需/ 可选	说明
<i>date</i>	必需	Date/Time 数据类型。传递要为其返回最大值的日期。可输入任何有效的转换表达式。
<i>filter_condition</i>	可选	限制搜索中的行数。筛选条件必须为数值或计算结果为 TRUE、FALSE 或 NULL。可输入任何有效的转换表达式。

返回值

日期。

当传递至函数的所有值为 NULL 或未选定行时返回 NULL（例如，筛选条件对所有行的计算结果均为 FALSE 或 NULL）。

示例

可返回端口或组的最大日期。以下表达式返回闪光灯的最大订购日期：

```
MAX( ORDERDATE, ITEM_NAME='Flashlight' )
```

ITEM_NAME	ORDER_DATE
Flashlight	Apr 20 1998
Regulator System	May 15 1998
Flashlight	Sep 21 1998
Diving Hood	Aug 18 1998
Flashlight	NULL

MAX (Numbers)

返回端口或组中找到的最大数值。可应用筛选器以限制搜索中的行数。只能在 MAX 中嵌套一个其他汇总函数。也可使用 MAX 返回端口或组中的最晚日期或最大字符串值。

语法

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

下表介绍了此命令的参数：

参数	必需/ 可选	说明
<i>numeric_value</i>	必需	数值数据类型。传递要为其返回最大数值的数值。可输入任何有效的转换表达式。
<i>filter_condition</i>	可选	限制搜索中的行数。筛选条件必须为数值或计算结果为 TRUE、FALSE 或 NULL。可输入任何有效的转换表达式。

返回值

数值。

当传递至函数的所有值为 NULL 或未选定行时返回 NULL（例如，筛选条件对所有行的计算结果均为 FALSE 或 NULL）。

注意: 如果返回值为精度大于 15 的小数，则可启用高精度，以确保小数精确至 38 位。

空值

如果值为 NULL，则 MAX 忽略它。然而，如果从端口传递的所有值均为 NULL，则 MAX 返回 NULL。

注意: 默认情况下，PowerCenter 集成服务在汇总函数中将空值当作 NULL 处理。如果传递整个端口或空值组，则函数返回 NULL。然而，在配置 PowerCenter 集成服务时，可选择想要处理汇总函数中空值的方式。可将汇总函数中的空值作为 0 或 NULL 处理。

分组依据

MAX 根据您在转换中定义的分组依据端口对值进行分组，为每组返回一个结果。

如果不存在分组依据端口，则 MAX 将所有行当作一个组处理，返回一个值。

示例

以下表达式返回闪光灯的最高价格：

```
MAX( PRICE, ITEM_NAME='Flashlight' )
```

ITEM_NAME	PRICE
Flashlight	10.00
Regulator System	360.00
Flashlight	55.00
Diving Hood	79.00
Halogen Flashlight	162.00
Flashlight	85.00
Flashlight	NULL

RETURN VALUE: 85.00

MAX (String)

返回端口或组中找到的最大字符串值。可应用筛选器以限制搜索中的行数。只能在 MAX 中嵌套一个其他汇总函数。

注意: MAX 函数与排序器转换使用相同的排序顺序。然而，MAX 函数区分大小写，排序器转换可能不区分大小写。

也可使用 MAX 返回端口或组中的最晚日期或最大数值。

语法

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

下表介绍了此命令的参数：

参数	必需/ 可选	说明
<i>string</i>	必需	字符串数据类型。传递要为其返回最大字符串值的字符串值。可输入任何有效的转换表达式。
<i>filter_condition</i>	可选	限制搜索中的行数。筛选条件必须为数值或计算结果为 TRUE、FALSE 或 NULL。可输入任何有效的转换表达式。

返回值

字符串。

当传递至函数的所有值为 NULL 或未选定行时返回 NULL（例如，筛选条件对所有行的计算结果均为 FALSE 或 NULL）。

空值

如果值为 NULL，则 MAX 忽略它。然而，如果从端口传递的所有值均为 NULL，则 MAX 返回 NULL。

注意: 默认情况下，PowerCenter 集成服务在汇总函数中将空值当作 NULL 处理。如果传递整个端口或空值组，则函数返回 NULL。然而，在配置 PowerCenter 集成服务时，可选择想要处理汇总函数中空值的方式。可将汇总函数中的空值作为 0 或 NULL 处理。

分组依据

MAX 根据您在转换中定义的分组依据端口对值进行分组，为每组返回一个结果。

如果不存在分组依据端口，则 MAX 将所有行当作一个组处理，返回一个值。

示例

以下表达式返回制造商 ID 104 的最大项目名称：

```
MAX( ITEM_NAME, MANUFACTURER_ID='104' )
```

MANUFACTURER_ID	ITEM_NAME
101	First Stage Regulator
102	Electronic Console

MANUFACTURER_ID	ITEM_NAME
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

计算输入值的校验和。函数使用消息摘要算法 5(MD5)。MD5 是单向加密哈希函数，有一个 128 位哈希值。可得出结论，当输入值校验和不同时，输入值就不同。使用 MD5 验证数据完整性。

语法

MD5(*value*)

下表介绍了此命令的参数：

参数	必需/ 可选	说明
<i>值</i>	必需	String 或 Binary 数据类型。想要为其计算校验和的值。输入值的大小写影响返回值。例如，MD5(informatica) 和 MD5 (Informatica) 返回不同的值。

返回值

十六进制位数 0-9 和 a-f 的唯一 32 位字符串。

当输入为空值时返回 NULL。

示例

您想要将更改的数据写入数据库。使用 MD5 为从源读取的数据行生成校验和值。运行会话时，将以前生成的校验和值与新的校验和值进行比较。然后，将有更新校验和值的行写入目标。可得出结论：更新校验和值表示数据已更改。

您想要将更改的数据写入数据库。使用 MD5 为从源读取的数据行生成校验和值。运行映射时，将以前生成的校验和值与新的校验和值进行比较。然后，将有更新校验和值的行写入目标。可得出结论：更新校验和值表示数据已更改。

提示

可使用返回值作为哈希键。

MEDIAN

返回选定端口中所有值的中间值。

如果端口中有偶数个值，当将所有值依序放在一个数字行上时，中间值就是中间两个值的平均值。如果端口中有奇数个值，则中间值为中间数字。

只能在 MEDIAN 内嵌套一个其他汇总函数，嵌套函数必须返回 Numeric 数据类型。

PowerCenter 集成服务读取所有数据行，以执行中间值计算。为了执行计算而读取数据行这一进程可能影响性能。或者，可应用筛选器限定读取的行数以计算中间值。

语法

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

下表介绍了此命令的参数：

参数	必需/ 可选	说明
<i>numeric_value</i>	必需	数值数据类型。传递要为其计算中间值的值。可输入任何有效的转换表达式。
<i>filter_condition</i>	可选	限制搜索中的行数。筛选条件必须为数值或计算结果为 TRUE、FALSE 或 NULL。可输入任何有效的转换表达式。

返回值

数值。

当传递至函数的所有值为 NULL 或未选定行时返回 NULL。例如，筛选条件对所有行的计算结果均为 FALSE 或 NULL。

注意：如果返回值为精度大于 15 的小数，则可启用高精度，以确保小数精确至 38 位。

空值

如果值为 NULL，则 MEDIAN 忽略行。然而，如果从端口传递的所有值均为 NULL，则 MEDIAN 返回 NULL。

注意：默认情况下，PowerCenter 集成服务在汇总函数中将空值当作 NULL 处理。如果传递整个端口或空值组，则函数返回 NULL。然而，在配置 PowerCenter 集成服务时，可选择想要处理汇总函数中空值的方式。可将汇总函数中的空值作为 0 或 NULL 处理。

分组依据

MEDIAN 根据您在转换中定义的分组依据端口对值进行分组，为每组返回一个结果。

如果不存在分组依据端口，则 MEDIAN 将所有行当作一个组处理，返回一个值。

示例

为了计算所有部门的平均薪酬，您创建了一个按部门分组的汇总器转换，用一个端口指定以下表达式：

```
MEDIAN( SALARY )
```

以下表达式返回稳定马甲订单的中间值：

```
MEDIAN( SALES, ITEM = 'Stabilizing Vest' )
```

ITEM	SALES
Flashlight	85
Stabilizing Vest	504
Stabilizing Vest	36
Safety Knife	5
Medium Titanium Knife	150
Tank	NULL
Stabilizing Vest	441
Chisel Point Knife	60
Stabilizing Vest	NULL
Stabilizing Vest	1044
Wrist Band Thermometer	110

RETURN VALUE: 472.5

METAPHONE

对字符串值进行编码。可以指定要进行编码的字符串的长度。

METAPHONE 编码英语字母 (A-Z) 字符。它用大写编码大写和小写字母。

METAPHONE 根据以下规则列表编码字符：

- 跳过元音 (A、E、I、O 和 U)，除非其中一个是输入字符串的第一个字符。METAPHONE('CAR') 返回 'KR'，METAPHONE('AAR') 返回 'AR'。
- 使用特殊编码准则。

下表列出了 METAPHONE 编码准则：

输入	返回	条件	示例
B	- n/a	- 当它前接 M 时	- METAPHONE ('Lamb') 返回 LM。
B	- B	- 在所有其他情况下	- METAPHONE ('Box') 返回 BKS。
C	- X	- 后跟 IA 或 H 时	- METAPHONE ('Facial') 返回 FXL。

输入	返回	条件	示例
C	- S	- 后跟 I、E 或 Y 时	- METAPHONE (‘Fence’) 返回 FNS。
C	- n/a	- 当它前接 S 且后跟 I、E 或 Y 时	- METAPHONE (‘Scene’) 返回 SN。
C	- K	- 在所有其他情况下	- METAPHONE (‘Cool’) 返回 KL。
D	- J	- 后跟 GE、GY 或 GI 时	- METAPHONE (‘Dodge’) 返回 TJ。
D	- T	- 在所有其他情况下	- METAPHONE (‘David’) 返回 TFT。
F	- F	- 在所有情况下	- METAPHONE (‘FOX’) 返回 FKS。
G	- F	- 后跟 H 且输入字符串中的第一个字符不是 B、D 或 H 时	- METAPHONE (‘Tough’) 返回 TF。
G	- n/a	- 后跟 H 且输入字符串中的第一个字符是 B、D 或 H 时	- METAPHONE (‘Hugh’) 返回 HF。
G	- J	- 后跟 I、E 或 Y 且不重复时	- METAPHONE (‘Magic’) 返回 MJK。
G	- K	- 在所有其他情况下	- METAPHONE (‘GUN’) 返回 KN。
H	- H	- 当它未前接 C、G、P、S 或 T 且后跟 A、E、I 或 U 时	- METAPHONE (‘DHAT’) 返回 THT。
H	- n/a	- 在所有其他情况下	- METAPHONE (‘Chain’) 返回 XN。
J	- J	- 在所有情况下	- METAPHONE (‘Jen’) 返回 JN。
K	- n/a - K	- 当它前接 C - 在所有其他情况下	- METAPHONE (‘Ckim’) 返回 KM。 - METAPHONE (‘Kim’) 返回 KM。
L	- L	- 在所有情况下	- METAPHONE (‘Laura’) 返回 LR。
M	- M	- 在所有情况下	- METAPHONE (‘Maggi’) 返回 MK。
N	- N	- 在所有情况下	- METAPHONE (‘Nancy’) 返回 NNS。
P	- F	- 当后跟 H 时	- METAPHONE (‘Phone’) 返回 FN。

输入	返回	条件	示例
P	- P	- 在所有其他情况下	- METAPHONE (‘Pip’) 返回 PP。
Q	- K	- 在所有情况下	- METAPHONE (‘Queen’) 返回 KN。
R	- R	- 在所有情况下	- METAPHONE (‘Ray’) 返回 R。
S	- X	- 后跟 H、IO、IA 或 CHW 时	- METAPHONE (‘Cash’) 返回 KX。
S	- S	- 在所有其他情况下	- METAPHONE (‘Sing’) 返回 SNK。
T	- X	- 后跟 IA 或 IO 时	- METAPHONE (‘Patio’) 返回 PX。
T	- 0 ¹	- 当后跟 H 时	- METAPHONE (‘Thor’) 返回 0R。
T	- n/a	- 当后跟 CH 时	- METAPHONE (‘Glitch’) 返回 KLTX。
T	- T	- 在所有其他情况下	- METAPHONE (‘Tim’) 返回 TM。
V	- F	- 在所有情况下	- METAPHONE (‘Vin’) 返回 FN。
W	- W	- 后跟 A、E、I、O 或 U 时	- METAPHONE (‘Wang’) 返回 WNK。
W	- n/a	- 在所有其他情况下	- METAPHONE (‘When’) 返回 HN。
X	- KS	- 在所有情况下	- METAPHONE (‘Six’) 返回 SKS。
Y	- Y	- 后跟 A、E、I、O 或 U 时	- METAPHONE (‘Yang’) 返回 YNK。
Y	- n/a	- 在所有其他情况下	- METAPHONE (‘Bobby’) 返回 BB。
Z	- S	- 在所有情况下	- METAPHONE (‘Zack’) 返回 SK。

¹. 整数 0。

- 如果输入字符串的头两个字符有以下其中一个值，则跳过初始字符并编码剩余字符串：
 - **KN**. 例如，METAPHONE(‘KNOT’) 返回 ‘NT’ 。
 - **GN**. 例如，METAPHONE(‘GNOB’) 返回 ‘NB’ 。
 - **PN**. 例如，METAPHONE(‘PNRX’) 返回 ‘NRKS’ 。
 - **AE**. 例如，METAPHONE(‘AERL’) 返回 ‘ERL’ 。

- 如果 “C” 以外的字符在输入字符串中出现多次，则仅编码第一次出现的 “C”。例如，`METAPHONE('BBOX')` 返回 ‘BKS’，`METAPHONE('CCOX')` 返回 ‘KKKS’。

语法

`METAPHONE(string [, length])`

下表介绍了此命令的参数：

参数	必需/ 可选	说明
<i>string</i>	必需	必须是字符串。传递要编码的值。第一个字符必须是英语字母 (A-Z) 中的字符。可输入任何有效的转换表达式。 跳过 <i>string</i> 中的任何非字母字符。
<i>length</i>	可选	必须是大于 0 的整数。指定要编码的 <i>string</i> 中的字符数。可输入任何有效的转换表达式。 当 <i>length</i> 为 0 或大于 <i>string</i> 长度的值时，编码整个输入字符串。 默认值为 0。

返回值

字符串。

当其中一个以下条件为真时返回 NULL：

- 传递至函数的所有值均为 NULL。
- 字符串中没有字符是英语字母。
- 字符串为空。

示例

以下表达式将 `EMPLOYEE_NAME` 端口中的头两个字符编码为字符串：

`METAPHONE(EMPLOYEE_NAME, 2)`

Employee_Name	Return Value
John	JH
*@#\$	NULL
P\$%%oc&&KMNL	PK

以下表达式将 `EMPLOYEE_NAME` 端口中的头四个字符编码为字符串：

`METAPHONE(EMPLOYEE_NAME, 4)`

Employee_Name	Return Value
John	JHN
1ABC	ABK

Employee_Name	Return Value
*@#	NULL
P\$%%oc&&KMNL	PKKM

MIN (Dates)

返回端口或组中找到的最早日期。可应用筛选器以限制搜索中的行数。只能在 MIN 内嵌套一个其他汇总函数，嵌套函数必须返回日期数据类型。

也可使用 MIN 返回端口或组中的最小数值或最小字符串值。

语法

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

下表介绍了此命令的参数：

参数	必需/ 可选	说明
<i>date</i>	必需	Date/Time 数据类型。传递要为其返回最小值的值。可输入任何有效的转换表达式。
<i>filter_condition</i>	可选	限制搜索中的行数。筛选条件必须为数值或计算结果为 TRUE、FALSE 或 NULL。可输入任何有效的转换表达式。

返回值

当 *date* 参数为日期时返回 Date。

当传递至函数的所有值为 NULL 或未选定行时返回 NULL（例如，筛选条件对所有行的计算结果均为 FALSE 或 NULL）。

空值

如果单值为 NULL，则 MIN 忽略它。然而，如果从端口传递的所有值均为 NULL，则 MIN 返回 NULL。

分组依据

MIN 根据您在转换中定义的分组依据端口对值进行分组，为每组返回一个结果。

如果不存在分组依据端口，则 MIN 将所有行当作一个组处理，返回一个值。

示例

以下表达式返回闪光灯的最早订购日期：

```
MIN( ORDER_DATE, ITEM_NAME='Flashlight' )
```

ITEM_NAME	ORDER_DATE
Flashlight	Apr 20 1998

ITEM_NAME	ORDER_DATE
Regulator System	May 15 1998
Flashlight	Sep 21 1998
Diving Hood	Aug 18 1998
Halogen Flashlight	Feb 1 1998
Flashlight	Oct 10 1998
Flashlight	NULL

RETURN VALUE: Feb 1 1998

MIN (Numbers)

返回端口或组中找到的最小数值。可应用筛选器以限制搜索中的行数。只能在 MIN 内嵌套一个其他汇总函数，嵌套函数必须返回数值数据类型。

也可使用 MIN 返回端口或组中的最晚日期或最低字符串值。

语法

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

下表介绍了此命令的参数：

参数	必需/ 可选	说明
<i>numeric_value</i>	必需	Numeric 数据类型。传递要为其返回最小值的值。可输入任何有效的转换表达式。
<i>filter_condition</i>	可选	限制搜索中的行数。筛选条件必须为数值或计算结果为 TRUE、FALSE 或 NULL。可输入任何有效的转换表达式。

返回值

数值。

当传递至函数的所有值为 NULL 或未选定行时返回 NULL（例如，筛选条件对所有行的计算结果均为 FALSE 或 NULL）。

注意: 如果返回值为精度大于 15 的小数，则可启用高精度，以确保小数精确至 38 位。

空值

如果单值为 NULL，则 MIN 忽略它。然而，如果从端口传递的所有值均为 NULL，则 MIN 返回 NULL。

注意: 默认情况下，PowerCenter 集成服务在汇总函数中将空值当作 NULL 处理。如果传递整个端口或空值组，则函数返回 NULL。然而，在配置 PowerCenter 集成服务时，可选择想要处理汇总函数中空值的方式。可将汇总函数中的空值作为 0 或 NULL 处理。

分组依据

MIN 根据您在转换中定义的分组依据端口对值进行分组，为每组返回一个结果。

如果不存在分组依据端口，则 MIN 将所有行当作一个组处理，返回一个值。

示例

以下表达式返回闪光灯的最低价格：

```
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)

返回端口或组中找到的最小字符串值。可应用筛选器以限制搜索中的行数。只能在 MIN 内嵌套一个其他汇总函数，嵌套函数必须返回字符串数据类型。

注意：MIN 函数与排序器转换使用相同的排序顺序。然而，MIN 函数区分大小写，但排序器转换可能不区分大小写。

也可使用 MIN 返回端口或组中的最晚日期或最小数值。

语法

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

下表介绍了此命令的参数：

参数	必需/ 可选	说明
<i>string</i>	必需	字符串数据类型。传递要为其返回最小值的值。可输入任何有效的转换表达式。
<i>filter_condition</i>	可选	限制搜索中的行数。筛选条件必须为数值或计算结果为 TRUE、FALSE 或 NULL。可输入任何有效的转换表达式。

返回值

字符串值。

当传递至函数的所有值为 NULL 或未选定行时返回 NULL（例如，筛选条件对所有行的计算结果均为 FALSE 或 NULL）。

空值

如果单值为 NULL，则 MIN 忽略它。然而，如果从端口传递的所有值均为 NULL，则 MIN 返回 NULL。

注意: 默认情况下，PowerCenter 集成服务在汇总函数中将空值当作 NULL 处理。如果传递整个端口或空值组，则函数返回 NULL。然而，在配置 PowerCenter 集成服务时，可选择想要处理汇总函数中空值的方式。可将汇总函数中的空值作为 0 或 NULL 处理。

分组依据

MIN 根据您在转换中定义的分组依据端口对值进行分组，为每组返回一个结果。

如果不存在分组依据端口，则 MIN 将所有行当作一个组处理，返回一个值。

示例

以下表达式返回制造商 ID 104 的最小项目名称：

```
MIN ( ITEM_NAME, MANUFACTURER_ID='104' )
```

MANUFACTURER_ID	ITEM_NAME
101	First Stage Regulator
102	Electronic Console
104	Flashlight
104	Battery (9 volt)
104	Rope (20 ft)
104	60.6 cu ft Tank
107	75.4 cu ft Tank
108	Wristband Thermometer

RETURN VALUE: 60.6 cu ft Tank

MOD

返回除法计算的余数。例如，MOD(8,5) 返回 3。

语法

MOD(*numeric_value*, *divisor*)

下表介绍了此命令的参数：

参数	必需/ 可选	说明
<i>numeric_value</i>	必需	数值数据类型。 要除以的值。 可输入任何有效的转换表达式。
除数	必需	要除以的数值。 除数不能为 0。

返回值

要传递至函数的数据类型数值。 数值除以除数后的余数。

传递至函数的值为 NULL 时返回 NULL。

示例

以下表达式返回 PRICE 端口中值除以 QTY 端口中值所得的模数：

MOD(PRICE, QTY)

PRICE	QTY	RETURN VALUE
10.00	2	0
12.00	5	2
9.00	2	1
15.00	3	0
NULL	3	NULL
20.00	NULL	NULL
25.00	0	<i>Error. Integration Service does not write row.</i>

最后一行 (25, 0) 产生错误，因为不能除以 0。 为了避免除以 0，可创建以下类似表达式，只有数量不是 0 时，该表达式才返回价格除以数量所得模数。 如果数量为 0，则函数返回 NULL：

MOD(PRICE, IIF(QTY = 0, NULL, QTY))

PRICE	QTY	RETURN VALUE
10.00	2	0
12.00	5	2
9.00	2	1
15.00	3	0

PRICE	QTY	RETURN VALUE
NULL	3	NULL
20.00	NULL	NULL
25.00	0	NULL

最后一行 (25, 0) 产生 NULL 而不是错误，因为 IIF 函数用 QTY 端口中的 0 替代了 NULL。

MOVINGAVG

返回指定行集的平均值(逐行)。或者，可以在计算移动平均值之前，应用条件来筛选行。

语法

`MOVINGAVG(numeric_value, rowset [, filter_condition])`

下表介绍了此命令的参数：

参数	必需/ 可选	说明
<i>numeric_value</i>	必需	数值数据类型。要为其计算移动平均值的值。可输入任何有效的转换表达式。
行集	必需	必须是大于 0 的正数值文字。定义要为其计算移动平均值的行集。例如，如果想要为数据列计算移动平均值，一次计算五行，则可写入以下类似表达式： <code>MOVINGAVG(SALES, 5)</code> 。
<i>filter_condition</i>	可选	限制搜索中的行数。筛选条件必须为数值或计算结果为 TRUE、FALSE 或 NULL。可输入任何有效的转换表达式。

返回值

数值。

当传递至函数的所有值为 NULL 或未选定行时返回 NULL（例如，筛选条件对所有行的计算结果均为 FALSE 或 NULL）。

注意: 如果返回值为精度大于 15 的小数，则可启用高精度，以确保小数精确至 38 位。

空值

在计算移动平均值时 MOVINGAVG 忽略空值。然而，如果所有值均为 NULL，则函数返回 NULL。

示例

以下表达式根据销售额端口中的头五行返回稳定马甲的平均订单，此后，返回最后五行读数的平均值：

```
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

函数返回一组五行的平均值：1 至 5 行基于 358，2 至 6 行基于 245.8，3 至 7 行基于 243。

MOVINGSUM

返回指定行集的和值(逐行)。

或者，可以在计算移动和值之前，应用条件来筛选行。

语法

```
MOVINGSUM( numeric_value, rowset [, filter_condition] )
```

下表介绍了此命令的参数：

参数	必需/ 可选	说明
<i>numeric_value</i>	必需	数值数据类型。 要为其计算移动和值的值。 可输入任何有效的转换表达式。
行集	必需	必须是大于 0 的正数值文字。 定义要为其计算移动和值的行集。 例如，如果想要为数据列计算移动和值，一次计算五行，则可写入以下类似表达式： MOVINGSUM(SALES, 5)
<i>filter_condition</i>	可选	限制搜索中的行数。 筛选条件必须为数值或计算结果为 TRUE、FALSE 或 NULL。 可输入任何有效的转换表达式。

返回值

数值。

当传递至函数的所有值均为 NULL 或函数未选定任何行时返回 NULL（例如，筛选条件对所有行的计算结果均为 FALSE 或 NULL）。

注意: 如果返回值为精度大于 15 的小数，则可启用高精度，以确保小数精确至 38 位。

空值

在计算移动和值时 MOVINGSUM 忽略空值。然而，如果所有值均为 NULL，则函数返回 NULL。

示例

以下表达式根据销售额端口中的前五五行返回稳定马甲的订单总和，此后，返回最后五行读数的平均值：

```
MOVINGSUM( SALES, 5 )
```

ROW_NO	SALES	RETURN VALUE
1	600	NULL
2	504	NULL
3	36	NULL
4	100	NULL
5	550	1790
6	39	1229
7	490	1215

函数返回一组五行的和值：1 至 5 行基于 1790，2 至 6 行基于 1229，3 至 7 行基于 1215。

NPER

依据固定利率和等额分期付款返回投资期数。

语法

```
NPER( rate, present value, payment [, future value, type] )
```

下表介绍了此命令的参数：

参数	必需/ 可选	说明
<i>rate</i>	必需	数值。每期所获的利率。用小数表示。用利率除以 100，即可用小数表示利率。必须大于或等于 0。
<i>现值</i>	必需	数值。一系列未来付款现在所值的总金额。
<i>payment</i>	必需	数值。每期应付款金额。必须是负数。

参数	必需/ 可选	说明
未来值	可选	数值。在最后一次付款后可以获得的现金余额。如果忽略此值，NPER 将使用 0。
类型	可选	布尔型。付款时间。如果是期初付款，请输入 1。如果是期末付款，请输入 0。默认值为 0。如果输入的值不是 0 或 1，则 PowerCenter 集成服务将该值当作 1 处理。

返回值

数值。

示例

投资现值为 \$500。每笔付款为 \$2000，投资未来值为 \$20,000。以下表达式返回 9 作为您需要付款的期数：

```
NPER ( 0.015, -500, -2000, 20000, TRUE )
```

注意事项

要计算每期利率，请将年利率除以每年的付款次数。例如，如果每月按 15% 的年利率付款，则利率参数的值为 15% 除以 12。如果做出年付款，则利率参数的值为 15%。

付款值和现值为负数，因为它们是您支出金额。

PERCENTILE

计算位于一组数字的给定百分位处的值。只能在 PERCENTILE 内嵌套一个其他汇总函数，嵌套函数必须返回数值数据类型。

PowerCenter 集成服务读取所有数据行，以执行百分比计算。为了执行计算而读取数据行这一进程可能影响性能。或者，可应用筛选器限定读取的行数以计算百分比。

语法

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

下表介绍了此命令的参数：

参数	必需/ 可选	说明
<i>numeric_value</i>	必需	数值数据类型。传递要为其计算百分比的值。可输入任何有效的转换表达式。
<i>percentile</i>	必需	介于 0 和 100 之间（包括 0 和 100）的整数。传递要计算的百分比。可输入任何有效的转换表达式。如果传递的数字超出 0 至 100 范围，则 PowerCenter 集成服务显示错误，且不写入行。
<i>filter_condition</i>	可选	限制搜索中的行数。筛选条件必须为数值或计算结果为 TRUE、FALSE 或 NULL。可输入任何有效的转换表达式。

返回值

数值。

当传递至函数的所有值为 NULL 或未选定行时返回 NULL（例如，筛选条件对所有行的计算结果均为 FALSE 或 NULL）。

注意: 如果返回值为精度大于 15 的小数，则可启用高精度，以确保小数精确至 38 位。

空值

如果值为 NULL，则 PERCENTILE 忽略行。然而，如果组中所有值均为 NULL，则 PERCENTILE 返回 NULL。

注意: 默认情况下，PowerCenter 集成服务在汇总函数中将空值当作 NULL 处理。如果传递整个端口或空值组，则函数返回 NULL。然而，在配置 PowerCenter 集成服务时，可选择想要处理汇总函数中空值的方式。可将汇总函数中的空值作为 0 或 NULL 处理。

分组依据

PERCENTILE 根据您在转换中定义的分组依据端口对值进行分组，为每组返回一个结果。

如果不存在分组依据端口，则 PERCENTILE 将所有行当作一个组处理，返回一个值。

示例

PowerCenter 集成服务使用以下逻辑计算百分比：

$$i = \frac{(x + 1) \times \text{percentile}}{100}$$

使用此方程式的以下准则：

- x 是在为其计算百分比的一组值中的元素数量。
- 如果 $i < 1$ ，则 PERCENTILE 返回列表中第一个元素的值。
- 如果 i 是整数值，则 PERCENTILE 返回列表中第 i 个元素的值。
- 否则，PERCENTILE 返回 n 的值：

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

以下表达式返回高于 \$50,000 的薪酬的 75% 的薪酬：

```
PERCENTILE( SALARY, 75, SALARY > 50000 )
```

SALARY

125000.0

27900.0

100000.0

NULL

SALARY

55000.0

9000.0

85000.0

86000.0

48000.0

99000.0

RETURN VALUE: 106250.0

PMT

依据等额付款和固定利率返回贷款的付款。

语法

`PMT(rate, terms, present value[, future value, type])`

下表介绍了此命令的参数：

参数	必需/ 可选	说明
<i>rate</i>	必需	数值。每期贷款利率。用小数表示。用利率除以 100，即可用小数表示利率。必须大于或等于 0。
<i>terms</i>	必需	数值。期数或付款次数。必须大于 0。
现值	必需	数值。贷款原则。
未来值	可选	数值。要在最后一次付款后获得的现金余额。如果忽略此值，PMT 将使用 0。
类型	可选	布尔型。付款时间。如果是期初付款，请输入 1。如果是期末付款，请输入 0。默认值为 0。如果输入的值不是 0 或 1，则 PowerCenter 集成服务将该值当作 1 处理。

返回值

数值。

示例

以下表达式返回 -2111.64 作为贷款的每月偿还金额：

`PMT(0.01, 10, 20000)`

注意事项

要计算每期利率，请将年利率除以每年的付款次数。例如，如果按 15% 的年利率做出每月付款，则利率为 15%/12。如果做出年付款，则利率为 15%。

付款值为负数，因为这些是您的支出金额。

POWER

返回已自乘到传递至函数的指数的值。

语法

```
POWER( base, exponent )
```

下表介绍了此命令的参数：

参数	必需/ 可选	说明
底数	必需	数值。此参数是基数值。可输入任何有效的转换表达式。如果基数值为负数，则指数必须为整数。
指数	必需	数值。此参数是指数值。可输入任何有效的转换表达式。如果基数值为负数，则指数必须为整数。在此情况下，函数将任何小数值舍入为最接近的整数，然后再返回值。

返回值

双精度值。

如果传递空值至函数，则它返回 NULL。

示例

以下表达式返回已自乘到指数端口中值的数值端口值：

```
POWER( NUMBERS, EXPONENT )
```

NUMBERS	EXPONENT	RETURN VALUE
10.0	2.0	100
3.5	6.0	1838.265625
3.5	5.5	982.594307804838
NULL	2.0	NULL
10.0	NULL	NULL
-3.0	-6.0	0.00137174211248285
3.0	-6.0	0.00137174211248285

NUMBERS	EXPONENT	RETURN VALUE
-3.0	6.0	729.0
-3.0	5.5	729.0

自乘到 6 的 -3.0 与自乘到 5.5 的 -3.0 返回相同结果。如果基数为负数，则指数必须为整数。否则，PowerCenter 集成服务将指数舍入为最接近的整数值。

PV

返回投资的现值。

语法

`PV(rate, terms, payment [, future value, type])`

下表介绍了此命令的参数：

参数	必需/ 可选	说明
rate	必需	数值。每期所获的利率。用小数表示。用利率除以 100，即可用小数表示利率。必须大于或等于 0。
terms	必需	数值。期数或付款次数。必须大于 0。
付款	必需	数值。每期应付款金额。必须是负数。
未来值	可选	数值。最后一次付款后的现金余额。如果忽略此值，PV 将使用 0。
类型	可选	布尔型。付款时间。如果是期初付款，请输入 1。如果是期末付款，请输入 0。默认值为 0。如果输入的值不是 0 或 1，则 PowerCenter 集成服务将该值当作 1 处理。

返回值

数值。

示例

以下表达式返回 12,524.43 作为今天必须在帐户中存入的金额，只有这样，当您也在每个期初存入 \$500 时，一年的未来值就为 \$20,000：

`PV(0.0075, 12, -500, 20000, TRUE)`

RAND

返回 0 到 1 之间的一个随机数。这对概率方案有用。

语法

`RAND(seed)`

下表介绍了此命令的参数：

参数	必需/ 可选	说明
种子	可选	数值。集成服务的起始值，用于生成随机数。值必须是常量。如未输入种子，则 PowerCenter 集成服务使用当前系统时间派生自 1971 年 1 月 1 日算起的秒数。它使用此值作为种子。

返回值

数值。

对于相同的种子，PowerCenter 集成服务生成相同的数字序列。

示例

以下表达式可能返回值 0.417022004702574：

```
RAND (1)
```

RATE

返回债券每期所获的利率。

语法

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

下表介绍了此命令的参数：

参数	必需/ 可选	说明
terms	必需	数值。期数或付款次数。必须大于 0。
付款	必需	数值。每期应付款金额。必须是负数。
现值	必需	数值。一系列未来付款现在所值的总金额。
未来值	可选	数值。要在最后一次付款后获得的现金余额。例如，一笔贷款的未来值为 0。如果忽略此参数，RATE 将使用 0。
类型	可选	布尔型。付款时间。如果是期初付款，请输入 1。如果是期末付款，请输入 0。默认值为 0。如果输入的值不是 0 或 1，则 PowerCenter 集成服务将该值当作 1 处理。

返回值

数值。

示例

以下表达式返回 0.0077 作为贷款的每月利率：

```
RATE( 48, -500, 20000 )
```

要计算贷款年利率，请将 0.0077 乘以 12。年利率为 0.0924 或 9.24%。

REG_EXTRACT

在输入值内提取正则表达式的子模式。例如，可从全名的正则表达式模式提取名字或姓氏。

注意: 使用 REG_REPLACE 函数用另一个字符模式替代字符串中的字符模式。

语法

```
REG_EXTRACT( subject, 'pattern', subPatternNum, match_from_start )
```

下表介绍了此命令的参数：

参数	必需/ 可选	说明
主题	必需	字符串数据类型。传递要与正则表达式模式比较的值。
模式	必需	字符串数据类型。要匹配的正则表达式模式。必须使用与 perl 兼容的正则表达式语法。用单引号将模式引起来。用圆括号将每个子模式括起来。
subPatternNum	可选	整数值。要匹配的正则表达式的子模式编号。使用以下准则确定子模式编号： <ul style="list-style-type: none">- 无值或 1。提取第一个正则表达式子模式。- 2. 提取第二个正则表达式子模式。- n. 提取第 <i>n</i> 个正则表达式子模式。 默认值为 1。
match_from_start	可选	数值。从字符串的开头找到匹配项时返回子字符串。使用以下准则确定起始值的匹配项： <ul style="list-style-type: none">- 0. 将模式与启动索引或任何索引的主题字符串相匹配。- 非零。将模式与启动索引的主题字符串相匹配。

使用 perl 兼容正则表达式语法

必须将 perl 兼容正则表达式语法与 REG_EXTRACT、REG_MATCH 和 REG_REPLACE 函数组合使用。

下表提供 perl 兼容正则表达式语法准则：

语法	说明
. (句点)	匹配任何一个字符。
[a-z]	匹配小写字母的一个实例。例如，[a-z] 匹配 ab。使用 [A-Z] 匹配大写字母。
\d	匹配从 0-9 的任何数字的一个实例。
\s	匹配空格字符。
\w	匹配一个字母字符，包括下划线 (_)
()	对表达式分组。例如，(\d-\d-\d\d) 中的圆括号对表达式 \d-\d-\d\d 进行分组，该表达式查找后跟连字符和任何两位数的任何两位数，如同 12-34。
{ }	匹配字符数。例如，\d{3} 匹配任何三位数，例如，650 或 510。或，[a-z]{2} 匹配任何两个字母，例如，CA 或 NY。

语法	说明
?	匹配前导字符或字符组零次或一次。例如， <code>\d{3}(-\d{4})?</code> 匹配后跟一个连字符和任何四位数的任何三位数。
* (星号)	匹配星号后面的值的零个或更多实例。例如， <code>*0</code> 是 0 之前的任何值。
+	匹配加号后面的值的一个或更多实例。例如， <code>\w+</code> 是字母字符后面的任何值。

例如，以下正则表达式查找 5 位数美国邮编（例如，93930）和 9 位数邮编，例如（93930-5407）：

`\d{5}(-\d{4})?`

`\d{5}` 指任何五位数，例如，93930。`-\d{4}` 周围的圆括号对这段表达式分组。连字符表示 9 位数邮编的连字符，如同 93930-5407。`\d{4}` 指任何四位数，例如，5407。问号表明连字符和最后四位数是可选的，或可出现一次。

将 COBOL 语法转换为 perl 兼容正则表达式语法

如果您熟悉 COBOL 语法，则可使用以下信息编写 perl 兼容正则表达式。

下表显示 COBOL 语法及其 perl 等效值的示例：

COBOL 语法	perl 语法	说明
9	<code>\d</code>	匹配从 0-9 的任何数字的一个实例。
9999	<code>\d\d\d\d</code> 或 <code>\d{4}</code>	匹配由 0-9 组成的任何四位数，如同 1234 或 5936。
x	<code>[a-z]</code>	匹配字母的一个实例。
9xx9	<code>\d[a-z][a-z]\d</code>	匹配任何数字后跟两个字母及另一个数字，如同 1ab2。

将 SQL 语法转换为 perl 兼容正则表达式语法

如果您熟悉 SQL 语法，则可使用以下信息编写 perl 兼容正则表达式。

下表显示 SQL 语法及其 perl 等效值的示例：

SQL 语法	perl 语法	说明
%	<code>. *</code>	匹配任何字符串。
A%	<code>A. *</code>	匹配字母“A”后跟任何字符串，如同 Area。
_	<code>. (句点)</code>	匹配任何一个字符。
A_	<code>A.</code>	匹配“A”后跟任何一个字符，例如，AZ。

返回值

返回属于输入值一部分的第 *n* 个子模式的值。第 *n* 个子模式基于为 `subPatternNum` 指定的值。

当输入为空值或模式为 NULL 时返回 NULL。

示例

可在表达式中使用 REG_EXTRACT 从与名字、中间名和姓氏匹配的正则表达式提取中间名。例如，以下表达式返回正则表达式的中间名：

```
REG_EXTRACT( Employee_Name, '(\w+)\s+(\w+)\s+(\w+)',2)
```

Employee_Name	Return Value
Stephen Graham Smith	Graham
Juan Carlos Fernando	Carlos

REG_MATCH

返回值是否匹配正则表达式模式。这使您可验证日期模式，例如，ID、电话号码、邮编和州名称。

注意：使用 REG_REPLACE 函数用一个新字符模式替代字符串中的字符模式。

语法

```
REG_MATCH( subject, pattern )
```

下表介绍了此命令的参数：

参数	必需/ 可选	说明
<i>subject</i>	必需	字符串数据类型。传递要与正则表达式模式匹配的值。
<i>pattern</i>	必需	字符串数据类型。要匹配的正则表达式模式。必须使用 perl 兼容正则表达式语法。用单引号将模式引起来。有关详细信息，请参阅 “REG_EXTRACT” 页面上 160。

返回值

数据与模式匹配时返回 TRUE。

数据与模式不匹配时返回 FALSE。

当输入为空值或模式为 NULL 时返回 NULL。

示例

可在表达式中使用 REG_MATCH 验证电话号码。例如，以下表达式将一个 10 位数电话号码与模式相匹配，根据匹配结果返回布尔值：

```
REG_MATCH (Phone_Number, '(\d\d\d\d-\d\d\d\d-\d\d\d\d)')
```

Phone_Number	Return Value
408-555-1212	TRUE

Phone_Number	Return Value
	NULL
510-555-1212	TRUE
92 555 51212	FALSE
650-555-1212	TRUE
415-555-1212	TRUE
831 555 12123	FALSE

提示

也可使用 REG_MATCH 完成以下任务：

- 验证值是否与模式相匹配。此用途类似于 SQL LIKE 函数。
- 验证值是否为字符。此用途类似于 SQL IS_CHAR 函数。

要验证值是否与模式相匹配，请在表达式中组合使用句点 (.) 和星号 (*) 与 REG_MATCH 函数。句点匹配任何一个字符。星号匹配其后面的值的 0 个或更多实例。

例如，使用以下表达式查找以 1835 开头的帐号：

```
REG_MATCH(ACCOUNT_NUMBER, '1835.*')
```

要验证值是否为字符，请将 REG_MATCH 函数与正则表达式 [a-zA-Z]+ 组合使用。a-z 匹配所有小写字符。A-Z 匹配所有大写字符。加号 (+) 指示应至少存在一个字符。

例如，使用以下表达式验证姓氏列表是否仅包含字符：

```
REG_MATCH(LAST_NAME, '[a-zA-Z]+')
```

REG_REPLACE

用其他字符模式替换字符串中的字符。默认情况下，REG_REPLACE 在输入字符串中搜索您指定的字符模式，并用替换模式替换出现的所有字符模式。也可指示要在字符串中替换的字符模式出现次数。

语法

```
REG_REPLACE( subject, pattern, replace, numReplacements )
```

下表介绍了此命令的参数：

参数	必需/ 可选	说明
<i>subject</i>	必需	字符串数据类型。传递要搜索的字符串。
<i>pattern</i>	必需	字符串数据类型。传递要替换的字符串。必须使用 perl 兼容正则表达式语法。用单引号将模式引起来。有关详细信息，请参阅 “REG_EXTRACT” 页面上 160 。

参数	必需/ 可选	说明
<i>replace</i>	必需	字符串数据类型。 传递新字符串。
<i>numReplacements</i>	可选	数值数据类型。 指定要替换的出现次数。 如果忽略此选项，则 REG_REPLACE 将替换字符串的所有出现。

返回值

字符串

示例

以下表达式为 Employee_name 端口的每行删除员工姓名数据中的附加空格:

```
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

将字符串中的字符替换为单个字符或无字符。REPLACECHR 在输入字符串中搜索您指定的字符，用您指定的新字符替换所有字符的所有出现。

语法

```
REPLACECHR( CaseFlag, InputString, OldCharSet, NewChar )
```

下表介绍了此命令的参数:

参数	必需/ 可选	说明
<i>CaseFlag</i>	必需	必须为整数。确定函数中的参数是否区分大小写。可输入任何有效的转换表达式。 当 <i>CaseFlag</i> 为 0 以外的数字时，函数区分大小写。 当 <i>CaseFlag</i> 为空值或 0 时，函数不区分大小写。
<i>InputString</i>	必需	必须是字符串。传递要搜索的字符串。可输入任何有效的转换表达式。如果传递数值，则函数将其转换为字符串。 如果 <i>InputString</i> 为 NULL，则 REPLACECHR 返回 NULL。

参数	必需/ 可选	说明
<i>OldCharSet</i>	必需	必须是字符串。要替换的字符。可输入一个或多个字符。可输入任何有效的转换表达式。也可输入用单引号引住的文本文字，例如，'abc'。 如果传递数值，则函数将其转换为字符串。 如果 <i>OldCharSet</i> 为 NULL 或空，则 REPLACECHR 返回 <i>InputString</i> 。
<i>NewChar</i>	必需	必须是字符串。可输入一个字符、空字符串或 NULL。可输入任何有效的转换表达式。 如果 <i>NewChar</i> 为 NULL 或为空，REPLACECHR 将删除 <i>OldCharSet</i> （在 <i>InputString</i> 中）中出现的所有字符。 如果 <i>NewChar</i> 包含多个字符，REPLACECHR 将使用第一个字符替换 <i>OldCharSet</i> 。

返回值

字符串。

当 REPLACECHR 删除 *InputString* 中所有字符时，返回空字符串。

当 *InputString* 为 NULL 时返回 NULL。

当 *OldCharSet* 为 NULL 或空时，返回 *InputString*。

示例

以下表达式为 WEBLOG 端口中的每行删除 Web 日志数据的双引号：

```
REPLACECHR( 0, WEBLOG, '"', NULL )
```

WEBLOG	RETURN VALUE
"GET /news/index.html HTTP/1.1"	GET /news/index.html HTTP/1.1
"GET /companyinfo/index.html HTTP/1.1"	GET /companyinfo/index.html HTTP/1.1
GET /companyinfo/index.html HTTP/1.1	GET /companyinfo/index.html HTTP/1.1
NULL	NULL

以下表达式为 WEBLOG 端口的每行删除多个字符：

```
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

WEBLOG	RETURN VALUE
[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

以下表达式为 CUSTOMER_CODE 端口中的每行更改客户代码值的一部分：

```
REPLACECHR ( 1, CUSTOMER_CODE, 'A', 'M' )
```

CUSTOMER_CODE	RETURN VALUE
ABA	MBM
abA	abM
BBC	BBC
ACC	MCC
NULL	NULL

以下表达式为 CUSTOMER_CODE 端口中的每行更改客户代码值的一部分：

```
REPLACECHR ( 0, CUSTOMER_CODE, 'A', 'M' )
```

CUSTOMER_CODE	RETURN VALUE
ABA	MBM
abA	MbM
BBC	BBC
ACC	MCC

以下表达式为 CUSTOMER_CODE 端口中的每行更改客户代码值的一部分：

```
REPLACECHR ( 1, CUSTOMER_CODE, 'A', NULL )
```

CUSTOMER_CODE	RETURN VALUE
ABA	B
BBC	BBC
ACC	CC
AAA	<i>[empty string]</i>

CUSTOMER_CODE	RETURN VALUE
aaa	aaa
NULL	NULL

以下表达式为 INPUT 端口的每行删除多个数字：

```
REPLACECHR ( 1, INPUT, '14', NULL )
```

INPUT	RETURN VALUE
12345	235
4141	NULL
111115	5
NULL	NULL

当您想要在 *OldCharSet* 或 *NewChar* 中使用单引号 (') 时，必须使用 CHR 函数。引号是不能在字符串文字内使用的唯一字符。

以下表达式为 INPUT 端口的每行删除多个字符，包括单引号：

```
REPLACECHR (1, INPUT, CHR(39), NULL )
```

INPUT	RETURN VALUE
'Tom Smith' 'Laura Jones'	Tom Smith Laura Jones
Tom's	Toms
NULL	NULL

REPLACESTR

将字符串中的字符替换为单个字符、多个字符或无字符。REPLACESTR 在输入字符串中搜索您指定的所有字符串，并用您指定的新字符串替换它们。

语法

```
REPLACESTR ( CaseFlag, InputString, OldString1, [OldString2, ... OldStringN,] NewString )
```

下表介绍了此命令的参数：

参数	必需/ 可选	说明
<i>CaseFlag</i>	必需	必须为整数。确定此函数中的参数是否区分大小写。可输入任何有效的转换表达式。 当 <i>CaseFlag</i> 为 0 以外的数字时，函数区分大小写。 当 <i>CaseFlag</i> 为空值或 0 时，函数不区分大小写。
<i>InputString</i>	必需	必须是字符串。传递要搜索的字符串。可输入任何有效的转换表达式。如果传递数值，则函数将其转换为字符串。 如果 <i>InputString</i> 为 NULL，则 REPLACESTR 返回 NULL。
<i>OldString</i>	必需	必须是字符串。要替换的字符串。必须至少输入一个 <i>OldString</i> 参数。可为每个 <i>OldString</i> 参数输入一个或多个字符。可输入任何有效的转换表达式。也可输入用单引号引住的文本文字，例如，'abc'。 如果传递数值，则函数将其转换为字符串。 当 REPLACESTR 包含多个 <i>OldString</i> 参数且一个或多个 <i>OldString</i> 参数为 NULL 或空时，REPLACESTR 忽略 <i>OldString</i> 参数。当所有 <i>OldString</i> 参数均为 NULL 或空时，REPLACESTR 返回 <i>InputString</i> 。 此函数替换 <i>OldString</i> 参数中的字符，按其在函数中出现的顺序。例如，如果输入多个 <i>OldString</i> 参数，则第一个 <i>OldString</i> 参数优先于第二个 <i>OldString</i> 参数，第二个 <i>OldString</i> 参数优先于第三个 <i>OldString</i> 参数。当 REPLACESTR 替换字符串时，它将光标放在 <i>InputString</i> 中已替换字符的后面，然后再搜索下一个匹配项。
<i>NewString</i>	必需	必须是字符串。可输入一个字符、多个字符，空字符串或 NULL。可输入任何有效的转换表达式。 如果 <i>NewString</i> 为 NULL 或空，则 REPLACESTR 删除 <i>OldString</i> 在 <i>InputString</i> 中的所有出现。

返回值

字符串。

当 REPLACESTR 删除 *InputString* 中所有字符时，返回空字符串。

当 *InputString* 为 NULL 时返回 NULL。

当所有 *OldString* 参数为 NULL 或空时，返回 *InputString*。

示例

以下表达式为 WEBLOG 端口中的每行删除 Web 日志数据的双引号和两个不同的文本字符串：

```
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

WEBLOG	RETURN VALUE
GET	<i>[empty string]</i>
NULL	NULL

以下表达式为 TITLE 端口中的每行更改某些值的标题：

```
REPLACESTR ( 1, TITLE, 'rs.', 'iss', 's.' )
```

TITLE	RETURN VALUE
Mrs.	Ms.
Miss	Ms.
Mr.	Mr.
MRS.	MRS.

以下表达式为 TITLE 端口中的每行更改某些值的标题：

```
REPLACESTR ( 0, TITLE, 'rs.', 'iss', 's.' )
```

TITLE	RETURN VALUE
Mrs.	Ms.
MRS.	Ms.

以下表达式显示 REPLACESTR 函数如何为 INPUT 端口中的每行替换多个 OldString 参数：

```
REPLACESTR ( 1, INPUT, 'ab', 'bc', '*' )
```

INPUT	RETURN VALUE
abc	*c
abbc	**
abbbbc	*bb*
bc	*

以下表达式显示 REPLACESTR 函数如何为 INPUT 端口中的每行替换多个 OldString 参数：

```
REPLACESTR ( 1, INPUT, 'ab', 'bc', 'b' )
```

INPUT	RETURN VALUE
ab	b

INPUT	RETURN VALUE
bc	b
abc	bc
abbc	bb
abbcc	bbc

当您想要在 *OldString* 或 *NewString* 中使用单引号 (') 时，必须使用 CHR 函数。使用 CHR 和 CONCAT 函数将单引号连接至字符串。引号是不能在字符串文字内使用的唯一字符。请考虑以下示例：

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

返回值为：

```
Joan's car
```

以下表达式为 INPUT 端口的每行更改包括单引号的字符串：

```
REPLACESTR ( 1, INPUT, CONCAT('it', CONCAT(CHR(39), 's' )), 'its' )
```

INPUT	RETURN VALUE
it's	its
mit's	mits
mits	mits
mits'	mits'

RESPEC

根据指定的复杂数据类型定义中的元素名称重命名给定结构值的每个元素。

语法

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

下表介绍了此命令的参数：

参数	必需/可选	说明
:Type.type_definition_library.type_definition	必需	表示结构数据的架构的复杂数据类型定义。使用引用限定符 :Type 可引用包含复杂数据类型定义的类型定义库。
struct_value	必需	要为其更改元素名称的结构值。可输入计算结果为结构的任何有效转换表达式。

复杂数据类型定义中每个元素的数据类型必须与结构的相应元素的数据类型匹配。

返回值

结构。

示例

以下表达式根据复杂数据类型定义 h1_sales_def 中的名称更改结构端口 h2_sales 中的元素名称。

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

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

REVERSE

反转输入字符串。

语法

```
REVERSE( string )
```

下表介绍了此命令的参数：

参数	必需/ 可选	说明
<i>string</i>	必需	任何字符值。 要反转的值。

返回值

字符串。输入值的反转。

示例

以下表达式反转客户代码的数量：

```
REVERSE( CUSTOMER_CODE )
```

CUSTOMER_CODE	RETURN VALUE
0001	1000
0002	2000
0003	3000
0004	4000

ROUND (Dates)

舍入日期的一部分。 也可使用 ROUND 舍入数字。

此函数可舍入日期的以下部分：

年

根据月份舍入日期的年份部分。

月

根据一个月中的日舍入日期的月份部分。

日

根据时间舍入日期的日部分。

小时

根据小时中的分钟舍入日期的小时部分。

分钟

根据秒数舍入日期中的分钟部分。

秒

根据毫秒舍入日期的秒部分。

毫秒

根据微秒舍入日期的毫秒部分。

微秒

根据纳秒舍入日期的微秒部分。

下表显示 ROUND 表达式使用的条件及返回值：

条件	表达式	返回值
对于一月与六月之间的月份，函数返回同一年份的 1 月 1 日，将时间设定为 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
对于七月与十二月之间的月份，函数返回下一年份的 1 月 1 日，将时间设定为 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
对于一个月中的第 1 至第 15 日，函数返回输入月份的第一天，将时间设定为 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
对于一个月中的第 16 日至最后一天，函数返回下一个月的第一天，将时间设定为 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

条件	表达式	返回值
对于 00:00:00 (12 a.m.) 与 11:59:59 a.m. 之间的时间, 函数返回当前日期, 将时间设定为 00:00:00.000000000 (12 a.m.)。	<code>ROUND(TO_DATE('06/13/1998 2:30:45', 'MM/DD/YYYY HH24:MI:SS'), 'DD')</code>	06/13/1998 00:00:00.000000000
对于 12:00:00 (12 p.m.) 或以后的时间, 函数将日期舍入为下一日, 将时间设定为 00:00:00.000000000 (12 a.m.)。	<code>ROUND(TO_DATE('06/13/1998 22:30:45', 'MM/DD/YYYY HH24:MI:SS'), 'DD')</code>	06/14/1998 00:00:00.000000000
对于 0 至 29 分钟之间的分钟部分, 函数返回当前小时, 将分钟、秒、毫秒和纳秒设定为 0。	<code>ROUND(TO_DATE('04/01/1998 11:29:35', 'MM/DD/YYYY HH24:MI:SS'), 'HH')</code>	04/01/1998 11:00:00.000000000
对于 30 或更大的分钟部分, 函数返回下一小时, 将分钟、秒、毫秒和纳秒设定为 0。	<code>ROUND(TO_DATE('04/01/1998 13:39:00', 'MM/DD/YYYY HH24:MI:SS'), 'HH')</code>	04/01/1998 14:00:00.000000000
对于 0 至 29 秒之间的时间, 函数返回当前分钟, 将秒、毫秒和纳秒设定为 0。	<code>ROUND(TO_DATE('05/22/1998 10:15:29', 'MM/DD/YYYY HH24:MI:SS'), 'MI')</code>	05/22/1998 10:15:00.000000000
对于 30 至 59 秒之间的时间, 函数返回下一分钟, 将秒、毫秒和纳秒设定为 0。	<code>ROUND(TO_DATE('05/22/1998 10:15:30', 'MM/DD/YYYY HH24:MI:SS'), 'MI')</code>	05/22/1998 10:16:00.000000000
对于 0 至 499 毫秒之间的时间, 函数返回当前秒, 将毫秒设定为 0。	<code>ROUND(TO_DATE('05/22/1998 10:15:29.499', 'MM/DD/YYYY HH24:MI:SS.MS'), 'SS')</code>	05/22/1998 10:15:29.000000000
对于 500 至 999 毫秒之间的时间, 函数返回下一秒, 将毫秒设定为 0。	<code>ROUND(TO_DATE('05/22/1998 10:15:29.500', 'MM/DD/YYYY HH24:MI:SS.MS'), 'SS')</code>	05/22/1998 10:15:30.000000000
对于 0 至 499 微秒之间的时间, 函数返回当前毫秒, 将微秒设定为 0。	<code>ROUND(TO_DATE('05/22/1998 10:15:29.498125', 'MM/DD/YYYY HH24:MI:SS.US'), 'MS')</code>	05/22/1998 10:15:29.498000000
对于 500 至 999 微秒之间的时间, 函数返回下一毫秒, 将微秒设定为 0。	<code>ROUND(TO_DATE('05/22/1998 10:15:29.498785', 'MM/DD/YYYY HH24:MI:SS.US'), 'MS')</code>	05/22/1998 10:15:29.499000000
对于 0 至 499 纳秒之间的时间, 函数返回当前微秒, 将纳秒设定为 0。	<code>ROUND(TO_DATE('05/22/1998 10:15:29.498125345', 'MM/DD/YYYY HH24:MI:SS.NS'), 'US')</code>	05/22/1998 10:15:29.498125000
对于 500 至 999 纳秒之间的时间, 函数返回下一微秒, 将纳秒设定为 0。	<code>ROUND(TO_DATE('05/22/1998 10:15:29.498125876', 'MM/DD/YYYY HH24:MI:SS.NS'), 'US')</code>	05/22/1998 10:15:29.498126000

语法

`ROUND(date [,format])`

下表介绍了此命令的参数：

参数	必需/ 可选	说明
日期	必需	日期/时间数据类型。在舍入之前，可嵌套 TO_DATE 以将字符串转换为日期。
格式	可选	输入有效的格式字符串。这是日期中要舍入的那部分。只能舍入日期的一个部分。如果忽略格式字符串，函数将日期舍入为最接近的日期。

返回值

指定部分已舍入的日期。ROUND 返回与源日期格式相同的日期。可将此函数的结果链接至数据类型为日期/时间的任何端口。

如果传递空值至函数，则它返回 NULL。

示例

以下表达式舍入 DATE_SHIPPED 端口中日期的年份部分：

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

DATE_SHIPPED	RETURN VALUE
Jan 15 1998 2:10:30AM	Jan 1 1998 12:00:00.000000000AM
Apr 19 1998 1:31:20PM	Jan 1 1998 12:00:00.000000000AM
Dec 20 1998 3:29:55PM	Jan 1 1999 12:00:00.000000000AM
NULL	NULL

以下表达式舍入 DATE_SHIPPED 端口中每个日期的月份部分：

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

DATE_SHIPPED	RETURN VALUE
Jan 15 1998 2:10:30AM	Jan 1 1998 12:00:00.000000000AM
Apr 19 1998 1:31:20PM	May 1 1998 12:00:00.000000000AM
Dec 20 1998 3:29:55PM	Jan 1 1999 12:00:00.000000000AM
NULL	NULL

以下表达式舍入 DATE_SHIPPED 端口中每个日期的日部分：

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

DATE_SHIPPED	RETURN VALUE
Jan 15 1998 2:10:30AM	Jan 15 1998 12:00:00.000000000AM
Apr 19 1998 1:31:20PM	Apr 20 1998 12:00:00.000000000AM
Dec 20 1998 3:29:55PM	Dec 21 1998 12:00:00.000000000AM
Dec 31 1998 11:59:59PM	Jan 1 1999 12:00:00.000000000AM
NULL	NULL

以下表达式舍入 DATE_SHIPPED 端口中每个日期的小时部分：

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

DATE_SHIPPED	RETURN VALUE
Jan 15 1998 2:10:31AM	Jan 15 1998 2:00:00.000000000AM
Apr 19 1998 1:31:20PM	Apr 19 1998 2:00:00.000000000PM
Dec 20 1998 3:29:55PM	Dec 20 1998 3:00:00.000000000PM
Dec 31 1998 11:59:59PM	Jan 1 1999 12:00:00.000000000AM
NULL	NULL

以下表达式舍入 DATE_SHIPPED 端口中每个日期的分钟部分：

```
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 (Numbers)

将数字舍入为指定位数或小数位。也可使用 ROUND 舍入日期。

语法

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

下表介绍了此命令的参数：

参数	必需/ 可选	说明
<i>numeric_value</i>	必需	数值数据类型。可输入任何有效的转换表达式。在舍入值之前使用运算符执行算术运算。
<i>精度</i>	可选	正整数或负整数。如果输入正 <i>精度</i> ，则函数舍入至此数字所表示的第几位小数。例如，ROUND(12.99, 1) 返回 13.0，ROUND(15.44, 1) 返回 15.4。 如果输入负 <i>精度</i> ，则函数舍入小数点左侧此数字所表示的位数，返回整数。例如，ROUND(12.99, -1) 返回 10，ROUND(15.99, -1) 返回 20。 如果输入小数 <i>精度</i> ，则函数舍入为最接近的整数，然后再计算表达式。例如，ROUND(12.99, 0.8) 返回 13.0，因为函数将 0.8 舍入为 1，然后再计算表达式。 如果忽略 <i>精度</i> 参数，则函数舍入为最接近的整数，截断数字的小数部分。例如，ROUND(12.99) 返回 13。

返回值

数值。

如果其中一个参数为 NULL，则 ROUND 返回 NULL。

注意: 如果返回值为精度大于 15 的小数，则可启用高精度，以确保小数精确至 38 位。

示例

以下表达式返回 PRICE 端口中已舍入为三位小数的值：

```
ROUND( PRICE, 3 )
```

PRICE	RETURN VALUE
12.9936	12.994
15.9949	15.995
-18.8678	-18.868
56.9561	56.956
NULL	NULL

可舍入小数点左侧的位数，只需在 *精度* 参数中传递负整数：

`ROUND(PRICE, -2)`

PRICE	RETURN VALUE
13242.99	13200.0
1435.99	1400.0
-108.95	-100.0
NULL	NULL

如果在 *精度* 参数中传递小数，则 PowerCenter 集成服务将其舍入为最接近的整数，然后再计算表达式：

`ROUND(PRICE, 0.8)`

PRICE	RETURN VALUE
12.99	13.0
56.34	56.3
NULL	NULL

如果忽略 *精度* 参数，则函数舍入为最接近的整数：

`ROUND(PRICE)`

PRICE	RETURN VALUE
12.99	13.0
-15.99	-16.0
-18.99	-19.0
56.95	57.0
NULL	NULL

提示

也可使用 `ROUND` 明确设置已计算值的精度，并得出期望的结果。当 PowerCenter 集成服务在低精度模式中运行时，如果值的精度超过 15 位数，则它截断计算结果。例如，您可能想要在低精度模式中处理以下表达式：

`7/3 * 3 = 7`

在此情况下，PowerCenter 集成服务对表达式左侧的计算结果为 6.999999999999999，因为它截断了第一个除法运算符的结果。PowerCenter 集成服务对整个表达式的计算结果为 FALSE。这可能不是您期望的结果。

为了获取期望的结果，请使用 ROUND 将表达式左侧的截断结果舍入为期望的结果。PowerCenter 集成服务对以下表达式的计算结果为 TRUE：

```
ROUND(7/3 * 3) = 7
```

RPAD

通过将空白或字符添加到字符串的末尾，将字符串转换为指定长度。

语法

```
RPAD( first_string, length [,second_string] )
```

下表介绍了此命令的参数：

参数	必需/ 可选	说明
<i>first_string</i>	必需	任何字符串值。要更改的字符串。可输入任何有效的转换表达式。
<i>length</i>	必需	必须为正整数文字。指定所需的每个字符串长度。
<i>second_string</i>	可选	任何字符串值。传递要附加至 <i>first_string</i> 值右侧的字符串。请用单引号将要添加至字符串末尾的字符引起来，例如，'abc'。此参数区分大小写。 如果忽略第二个字符串，则函数用空白填充第一个字符串的末尾。

返回值

指定长度的字符串。

当传递至函数的值为 NULL 或长度为负数时返回 NULL。

示例

以下表达式返回长度为 16 个字符的项目名称，附加字符串 '.' 至每个项目名称的末尾：

```
RPAD( ITEM_NAME, 16, '.' )
```

ITEM_NAME	RETURN VALUE
Flashlight	Flashlight.....
Compass	Compass.....
Regulator System	Regulator System
Safety Knife	Safety Knife....

RPAD 从左到右计算长度。因此，如果第一个字符串长于指定长度，则 RPAD 从右至左截断字符串。例如，RPAD('alphabetical' , 5, 'x') 将返回字符串 'alpha' 。必要时，RPAD 使用 *second_string* 的一部分。

以下表达式返回长度为 16 个字符的项目名称，附加字符串 ‘*.*’ 至每个项目名称的末尾：
RPAD(ITEM_NAME, 16, '*.*')

ITEM_NAME	RETURN VALUE
Flashlight	Flashlight*.*.
Compass	Compass*.*.*.*
Regulator System	Regulator System
Safety Knife	Safety Knife*.*

RTRIM

从字符串的末尾删除空白或字符。

如不在表达式中指定 *trim_set* 参数：

- 在 UNICODE 模式中，RTRIM 删除字符串末尾的单字节和双字节空格。
- 在 ASCII 模式中，RTRIM 只删除单字节空格。

如果使用 RTRIM 删除字符串中的字符，则 RTRIM 将 *trim_set* 与字符串参数中的每个字符逐一作比较，从字符串的右侧开始。如果字符串中的字符与 *trim_set* 中的任何字符相匹配，则 RTRIM 将删除它。RTRIM 继续比较和删除字符，直到它未能在 *trim_set* 中找到匹配字符。它返回没有匹配字符的字符串。

语法

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

下表介绍了此命令的参数：

参数	必需/ 可选	说明
<i>string</i>	必需	任何字符串值。传递要裁减的值。可输入任何有效的转换表达式。使用运算符执行比较或连接字符串，然后再删除字符串末尾的空白。
<i>trim_set</i>	可选	任何字符串值。传递要从字符串末尾删除的字符。也可输入文本。然而，必须用单引号将要从字符串末尾删除的字符引起来，例如，'abc'。如果忽略第二个字符串，则函数删除第一个字符串末尾的空白。 RTRIM 区分大小写。

返回值

字符串。已将 *trim_set* 参数中指定字符删除的字符串值。

传递至函数的值为 NULL 时返回 NULL。

示例

以下表达式从 LAST_NAME 端口中字符串删除字符 're'：

```
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 从 Page 中删除 'e'，即使 'r' 是 *trim_set* 中的第一个字符。这是因为 RTRIM 逐一字符搜索您在 *trim_set* 参数中指定的字符集。如果字符串中的最后一个字符与 *trim_set* 中的第一个字符相匹配，则 RTRIM 将删除它。然而，如果字符串中的最后一个字符不匹配，则 RTRIM 比较 *trim_set* 中的第二个字符。如果字符串中的倒数第二个字符与 *trim_set* 中的第二个字符相匹配，则 RTRIM 将删除它，以此类推。当字符串中的字符未能与 *trim_set* 匹配时，RTRIM 返回字符串，并计算下一行。

在最后一个示例中，Nelson 的最后一个字符与 *trim_set* 参数中的任何字符均不匹配，因此，RTRIM 返回字符串 'Nelson'，并计算下一行。

RTRIM 提示

将 RTRIM 和 LTRIM 与 || 或 CONCAT 一起使用，以便在连接两个字符串后删除前导和尾随空白。

也可删除多个字符集，只需嵌套 RTRIM。例如，如果要删除名称列中每个字符串末尾的尾随空白和字符 't'，则可创建以下类似表达式：

```
RTRIM( RTRIM( NAMES ), 't' )
```

SETCOUNTVARIABLE

计数由函数计算的行，并根据计数增加映射变量的当前值。将标记为插入的每一行的当前值增加一。将标记为删除的每一行的当前值减少一。将标记为更新或拒绝的每一行的当前值保持不变。返回新当前值。

在成功会话结束时，PowerCenter 集成服务将最后一个当前值保存至存储库。与包含多个分区的会话组合使用时，PowerCenter 集成服务为每个分区生成不同的当前值。在会话结束时，它确定所有分区的总计数，并将总数保存至存储库。当您下次使用此会话时，它使用保存的值作为变量的初始值，除非被替代。

对于管道中的每个映射变量，只能使用一次 SETCOUNTVARIABLE 函数。PowerCenter 集成服务处理在映射中遇到的变量函数。对于每次会话运行，PowerCenter 集成服务在映射中遇到变量函数的顺序可能各不相同。当您在映射中多次使用相同的变量函数时，这可能导致不一致的结果。

将 SETCOUNTVARIABLE 与汇总类型为计数的映射变量组合使用。在以下转换中使用 SETCOUNTVARIABLE：

- 表达式
- 过滤器
- 路由器
- 更新策略

当以下任一项为真时，PowerCenter 集成服务不将映射变量的最终值保存至存储库：

- 会话未能完成。
- 已为测试负载配置会话。
- 会话为调试会话。
- 会话在调试模式中运行，且配置为丢弃会话输出。

语法

```
SETCOUNTVARIABLE( $$Variable )
```

下表介绍了此命令的参数：

参数	必需/ 可选	说明
\$\$ 变量	必需	要设置的映射变量的名称。使用汇总类型为计数的映射变量。

返回值

变量当前值。

示例

您有一个映射可更新包含发行商信息的缓慢变化的维度表。以下表达式对映射变量为 \$CurrentDistributors 的当前发行商的数量进行计数，将当前值返回至 CUR_DIST 端口。它为每个插入行增加一个计数，为每个删除行减少一个计数，且使所有已更新或拒绝行的计数均保持不变。源自上次会话运行的 \$\$CurrentDistributors 的初始值为 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

在会话结束时，PowerCenter 集成服务将 ‘25’ 保存至存储库，作为 \$\$CurrentDistributors 的当前值。当会话下次运行时，集成服务计算的 \$\$CurrentDistributors 初始值为 ‘25’。

PowerCenter 集成服务为多个分区会话将相同的 \$\$CurrentDistributors 值保存至存储库，就如同为单个分区会话保存值。

SET_DATE_PART

将日期/时间值的一部分设置为指定值。借助于 SET_DATE_PART，您可更改日期的以下部分：

- **年份。** 更改年份，只需在 *值* 参数中输入正整数。使用任何年份格式字符串：Y、YY、YYY 或 YYYY 以设置年份。例如，以下表达式为 SHIP_DATE 端口中的所有日期将年份更改为 2001：

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

- **月份。** 更改月份，只需在 *值* 参数中输入介于 1 与 12 之间的正整数（一月 = 1，十二月 = 12）。使用任何月份格式字符串：MM、MON、MONTH 以设置月份。例如，以下表达式为 SHIP_DATE 端口中的所有日期将月份更改为 10 月：

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

- **日。** 更改日，只需将 1 与 31 之间的正整数（但没有 31 天的月份除外：二月、四月、六月、九月和十一月）输入 *值* 参数。使用任何月份格式字符串（DD、DDD、DY 和 DAY）设置日。例如，以下表达式为 SHIP_DATE 端口中的所有日期将日更改为 10：

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

- **小时。** 更改小时，只需在 *值* 参数中输入介于 0 与 24 之间的正整数（其中 0 = 12AM，12 = 12PM，24 = 12AM）。使用任何小时格式字符串（HH、HH12、HH24）以设置小时。例如，以下表达式为 SHIP_DATE 端口中的所有日期将小时更改为 14:00:00（或 2:00:00PM）：

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

- **分钟。** 更改分钟，只需在 *值* 参数中输入 0 与 59 之间的正整数。使用 MI 格式字符串设置分钟。例如，以下表达式为 SHIP_DATE 端口中的所有日期将分钟更改为 25：

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

- **秒数。** 更改秒数，只需在 *值* 参数中输入 0 与 59 之间的正整数。使用 SS 格式字符串设置秒数。例如，以下表达式为 SHIP_DATE 端口中的所有日期将秒数更改为 59：

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

- **毫秒。** 更改毫秒，只需在 *值* 参数中输入 0 与 999 之间的正整数。使用 MS 格式字符串设置毫秒。例如，以下表达式为 SHIP_DATE 端口中的所有日期将毫秒更改为 125：

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

- **微秒。** 更改微秒，只需在 *值* 参数中输入 1000 与 999999 之间的正整数。使用 US 格式字符串设置微秒。例如，以下表达式为 SHIP_DATE 端口中的所有日期将微秒更改为 12555：

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

- **纳秒。** 更改纳秒，只需在 *值* 参数中输入 1000000 与 999999999 之间的正整数。使用 NS 格式字符串设置纳秒。例如，以下表达式为 SHIP_DATE 端口中的所有日期将纳秒更改为 12555555：

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

语法

```
SET_DATE_PART( date, format, value )
```

下表介绍了此命令的参数：

参数	必需/ 可选	说明
<i>date</i>	必需	Date/Time 数据类型。要修改的日期。可输入任何有效的转换表达式。
<i>格式</i>	必需	格式字符串，用于指定要更改的日期部分。格式字符串不区分大小写。
<i>值</i>	必需	要分配给日期的指定部分的正整数值。对于要更改的日期部分而言，整数必须为有效值。如果输入不正确的值，例如，2 月 30 日，则会失败。

返回值

与指定部分已更改的源日期格式相同的日期。

传递至函数的值为 NULL 时返回 NULL。

示例

以下表达式为 DATE_PROMISED 端口中的每个日期将小时更改 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

以下表达式为 DATE_PROMISED 端口中的日期将月份更改为 6 月。当您尝试创建的日期不存在时，例如，将 3 月 31 日更改为 6 月 31 日，则 PowerCenter 集成服务显示错误：

```
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	<i>Error. Integration Service doesn't write row.</i>

DATE_PROMISED	RETURN VALUE
Dec 12 1997 8:07:33AM	Jun 12 1997 8:07:33AM
NULL	NULL

以下表达式为 DATE_PROMISED 端口中的日期将年份更改为 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
Dec 12 1997 8:07:33AM	Dec 12 2000 4:07:33PM
NULL	NULL

提示

如果想要一次更改日期的多个部分，则可在 *日期* 参数中嵌套多个 SET_DATE_PART 函数。例如，您可写入以下表达式，将 DATE_ENTERED 端口中的所有日期更改为 1998 年 7 月 1 日：

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

SETMAXVARIABLE

将映射变量的当前值设置为两个值中的较大值：变量的当前值或您指定的值。返回新当前值。只有在有行被标记为插入时，函数才会执行。SETMAXVARIABLE 忽略其他所有行类型，当前值保持不变。

在成功会话结束时，PowerCenter 集成服务将最终当前值保存至存储库。与包含多个分区的会话组合使用时，PowerCenter 集成服务为每个分区生成不同的当前值。在会话结束时，它将所有分区之间的最大当前值保存至存储库。当会话下次运行时，它使用保存的值作为变量的初始值，除非被替代。

与字符串映射变量组合使用时，SETMAXVARIABLE 根据为会话选定的排序顺序返回较大的字符串。

对于管道中的每个映射变量，只能使用一次 SETMAXVARIABLE 函数。PowerCenter 集成服务处理在映射中遇到的变量函数。对于每次会话运行，PowerCenter 集成服务在映射中遇到变量函数的顺序可能各不相同。当您在映射中多次使用相同的变量函数时，这可能导致不一致的结果。

将 SETMAXVARIABLE 与汇总类型为 Max 的映射变量组合使用。在以下转换中使用 SETMAXVARIABLE：

- 表达式
- 过滤器
- 路由器

- 更新策略

当以下任一条件为真时，PowerCenter 集成服务不将映射变量的最终值保存至存储库：

- 会话未能完成。
- 已为测试负载配置会话。
- 会话为调试会话。
- 会话在调试模式中运行，且配置为丢弃会话输出。

语法

```
SETMAXVARIABLE( $$Variable, value )
```

下表介绍了此命令的参数：

参数	必需/ 可选	说明
<code>\$\$变量</code>	必需	要设置的映射变量的名称。 使用汇总类型为 Max 的映射变量。
<code>值</code>	必需	想要 PowerCenter 集成服务与变量的当前值进行比较的值。 可输入计算结果为兼容于变量数据类型的数据类型的任何有效转换表达式。

返回值

两个值中较大值：您指定的变量或值的当前值。 返回值为变量的新当前值。

当 `值` 为 NULL 时，PowerCenter 集成服务返回 `$$` 的当前值。

示例

以下表达式将每个事务中购买的项目数量与映射变量 `$$MaxItems` 相比较。 它将 `$$MaxItems` 设置为两个值中的较大值，将单个事务中购买的历史上最大项目数量返回至 `MAX_ITEMS` 端口。 源自上次会话运行的 `$$MaxItems` 的初始值为 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

在会话结束时，PowerCenter 集成服务 将 ‘35’ 保存至存储库，作为 `$$MaxItems` 的最大当前值。 当会话下次运行时，PowerCenter 集成服务计算的 `$$MaxItems` 初始值为 ‘35’ 。

如果同一会话包含三个分区，则 PowerCenter 集成服务为每个分区计算 \$\$MaxItems。然后，它将最大值保存至存储库。例如，最后计算的每个分区中的 \$\$MaxItems 值如下：

Partition	Final Current Value for \$\$MaxItems
Partition 1	35
Partition 2	23
Partition 3	22

SETMINVARIABLE

将映射变量的当前值设置为两个值中的较小值: 变量的当前值或您指定的值。返回新当前值。只有在有行被标记为插入时，SETMINVARIABLE 函数才会执行。SETMINVARIABLE 忽略其他所有行类型，当前值保持不变。

在成功会话结束时，PowerCenter 集成服务将最终当前值保存至存储库。与包含多个分区的会话组合使用时，PowerCenter 集成服务为每个分区生成不同的当前值。在会话结束时，它将所有分区之间的最小当前值保存至存储库。当会话下次运行时，它使用保存的值作为变量的初始值，除非被替代。

与字符串映射变量组合使用时，SETMINVARIABLE 根据为会话选定的排序顺序返回较小的字符串。

对于管道中的每个映射变量，只能使用一次 SETMINVARIABLE 函数。PowerCenter 集成服务处理在映射中遇到的变量函数。对于每次会话运行，PowerCenter 集成服务在映射中遇到变量函数的顺序可能各不相同。当您在映射中多次使用相同的变量函数时，这可能导致不一致的结果。

将 SETMINVARIABLE 与汇总类型为 Max 的映射变量组合使用。在以下转换中使用 SETMINVARIABLE：

- 表达式
- 过滤器
- 路由器
- 更新策略

当以下任一条件为真时，PowerCenter 集成服务不将映射变量的最终值保存至存储库：

- 会话未能完成。
- 已为测试负载配置会话。
- 会话为调试会话。
- 会话在调试模式中运行，且配置为丢弃会话输出。

语法

```
SETMINVARIABLE( $$Variable, value )
```

下表介绍了此命令的参数：

参数	必需/ 可选	说明
<code>\$\$变量</code>	必需	要设置的映射变量的名称。与汇总类型为 Min 的映射变量组合使用。
<code>值</code>	必需	想要 PowerCenter 集成服务与变量的当前值进行比较的值。可输入计算结果为兼容于变量数据类型的数据类型的任何有效转换表达式。

返回值

两个值中较小值：您指定的变量或值的当前值。返回值为变量的新当前值。

当 `值` 为 NULL 时，PowerCenter 集成服务返回 `$$Variable` 的当前值。

示例

以下表达式将项目价格与映射变量 `$$MinPrice` 相比较。它将 `$$MinPrice` 设置为两个值中较小值，将历史上最低项目价格返回至 `MIN_PRICE` 端口。源自上次会话运行的 `$$MinPrice` 的初始值为 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

在会话结束时，PowerCenter 集成服务 将 22.00 保存至存储库，作为 `$$MinPrice` 的最小当前值。当会话下次运行时，PowerCenter 集成服务计算的 `$$MinPrice` 初始值为 22.00。

如果同一会话包含三个分区，则 PowerCenter 集成服务为每个分区计算 `$$MinPrice`。然后，它将最小值保存至存储库。例如，最后计算的每个分区中的 `$$MinPrice` 值如下：

Partition	Final Current Value for \$\$MinPrice
Partition 1	22.00

Partition	Final Current Value for \$\$MinPrice
Partition 2	22.50
Partition 3	22.50

SETVARIABLE

将映射变量的当前值设置为您指定的值。返回指定值。只有在有行被标记为插入或更新时，SETVARIABLE 函数才会执行。SETVARIABLE 忽略其他所有行类型，当前值保持不变。

在成功会话结束时，PowerCenter 集成服务将变量的最终当前值与变量的起始值进行比较。根据变量的汇总类型，它将最终当前值保存至存储库。当会话下次运行时，它使用保存的值作为变量的初始值，除非被替代。

对于管道中的每个映射变量，只能使用一次 SETVARIABLE 函数。PowerCenter 集成服务处理在映射中遇到的变量函数。对于每次会话运行，PowerCenter 集成服务在映射中遇到变量函数的顺序可能各不相同。当您在映射中多次使用相同的变量函数时，这可能导致不一致的结果。

在以下转换中使用 SETVARIABLE：

- 表达式
- 过滤器
- 路由器
- 更新策略

当以下任一条件为真时，PowerCenter 集成服务不将映射变量的最终值保存至存储库：

- 会话未能完成。
- 已为测试负载配置会话。
- 会话为调试会话。
- 会话在调试模式中运行，且配置为丢弃会话输出。

语法

```
SETVARIABLE( $$Variable, value )
```

下表介绍了此命令的参数：

参数	必需/ 可选	说明
<code>\$\$变量</code>	必需	要设置的映射变量的名称。与汇总类型为 Max/Min 的映射变量组合使用。
<code>值</code>	必需	要设置为变量的当前值的值。可输入计算结果为兼容于变量数据类型的数据类型的任何有效转换表达式。

返回值

变量的当前值。

当值为 NULL 时，PowerCenter 集成服务返回 \$\$Variable 的当前值。

示例

以下表达式将映射变量 \$\$Time 设置为 PowerCenter 集成服务计算行时的系统日期，并将系统日期返回至 SET_\$\$TIME 端口：

```
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

在会话结束时，PowerCenter 集成服务 将 10/10/2000 01:34:37 保存至存储库，作为最后计算的 \$\$Time 当前值。当会话下次运行时，PowerCenter 集成服务计算的 \$\$Time 所有引用均为 10/10/2000 01:34:37。

以下表达式将映射变量 \$\$Timestamp 设置为与行关联的时间戳，并将该时间戳返回至 SET_\$\$TIMESTAMP 端口：

```
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

在会话结束时，PowerCenter 集成服务 将 10/01/2000 12:40:31 保存至存储库，作为最后计算的 \$Timestamp 当前值。

当会话下次运行时，PowerCenter 集成服务计算的 \$\$Timestamp 初始值为 10/01/2000 12:40:31。

SIGN

返回数值是否为正、负或 0。

语法

```
SIGN( numeric_value )
```

下表介绍了此命令的参数：

参数	必需/ 可选	说明
<i>numeric_value</i>	必需	数值。 传递要计算的值。 可输入任何有效的转换表达式。

返回值

对于负值，返回 -1。

对于 0，返回 0。

对于正值，返回 1。

值为 NULL 时返回 NULL。

示例

以下表达式确定 SALES 端口是否包括任何负值：

SIGN(SALES)

SALES	RETURN VALUE
100	1
-25.99	-1
0	0
NULL	NULL

SIN

返回数值的正弦(以弧度表示)。

语法

SIN(*numeric_value*)

下表介绍了此命令的参数：

参数	必需/ 可选	说明
<i>numeric_value</i>	必需	数值数据类型。 以弧度表示的数值数据（度数乘以 pi 除以 180）。 传递要为其计算正弦的值。 可输入任何有效的转换表达式。 也可使用运算符将数值转换为弧度或在 SIN 计算中执行算术运算。

返回值

双精度值。

传递至函数的值为 NULL 时返回 NULL。

示例

以下表达式将度数端口中的值转换为弧度，然后计算每个弧度的正弦：

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

DEGREES	RETURN VALUE
0	0
90	1
70	0.939692620785936
30	0.500000000000003
5	0.0871557427476639
89	0.999847695156393
NULL	NULL

在函数计算正弦之前，可对传递至 SIN 的值执行算术运算。 例如：

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

SINH

返回数值的双曲正弦。

语法

```
SINH( numeric_value )
```

下表介绍了此命令的参数：

参数	必需/ 可选	说明
<i>numeric_value</i>	必需	数值数据类型。 以弧度表示的数值数据（度数乘以 pi 除以 180）。 传递要为其计算双曲正弦的值。 可输入任何有效的转换表达式。

返回值

双精度值。

传递至函数的值为 NULL 时返回 NULL。

示例

以下表达式为角度端口中的值返回双曲正弦：

```
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

提示

在函数计算正弦之前，可对传递至 SINH 的值执行算术运算。例如：

```
SINH( MEASURES.ARCS / 180 )
```

SIZE

返回指定数组或映射的大小。

语法

```
SIZE(value)
```

下表介绍了此命令的参数：

参数	必需/可选	说明
value	必需	数组或映射数据类型。想要确定其大小的数组或映射。 如果指定数组，则函数将返回数组中的元素数。 如果指定映射，则函数将返回映射中的键-值对数。

返回值

整数。

如果数组或映射为 NULL，则返回 -1。

示例

以下表达式将返回数组 ITEM_NAME 的大小。

```
SIZE(ITEM_NAME)
```

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

以下表达式将返回映射 SHIPMENT_DETAILS 的大小。

```
SIZE(SHIPMENT_DETAILS)
```

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

SOUNDEX

将字符串值编码为四个字符的字符串。

SOUNDEX 对英语字母字符 (A-Z) 起作用。它使用输入字符串的第一个字符作为返回值的第一个字符，并将剩余三个唯一辅音编码为数字。

SOUNDEX 根据以下规则列表编码字符：

- 使用 *字符串* 中第一个字符作为返回值的第一个字符，并用大写对其进行编码。例如，SOUNDEX('John') 和 SOUNDEX('john') 均返回 'J500' 。
- 对 *字符串* 中第一个字符后面的头三个唯一辅助音进行编码，并忽略剩余辅音。例如，SOUNDEX('JohnRB') 和 SOUNDEX('JohnRBCD') 均返回 'J561' 。
- 向听起来相似的辅音分配单个代码。

下表列出了辅音的 SOUNDEX 编码准则：

表 2. 辅音的 SOUNDEX 编码准则

代码	辅音字母
1	B、P、F、V
2	C、S、G、J、K、Q、X、Z
3	D、T
4	L
5	M、N
6	R

- 跳过字符 A、E、I、O、U、H 和 W，除非其中一个是字符串中的第一个字符。例如，SOUNDEX('A123') 返回 “A000”，SOUNDEX('MAeiouhWC') 返回 “M200”。
- 如果字符串产生的字符少于四个，则 SOUNDEX 用零填充生成的字符串。例如，SOUNDEX('J') 返回 ‘J000’。
- 如果字符串包含的连续辅音集使用了 [“SOUNDEX” 页面上 193](#) 中所列的相同代码，则 SOUNDEX 对第一次出现的辅音进行编码，跳过集中出现的剩余辅音。例如，SOUNDEX('AbbpdMN') 返回 ‘A135’。
- 跳过字符串中的数字。例如，SOUNDEX('Joh12n') 和 SOUNDEX('1John') 均返回 ‘J500’。
- 当字符串为 NULL 或当字符串中所有字符均不是英语 alphabet 字母时，返回 NULL。

语法

SOUNDEX(*string*)

下表介绍了此命令的参数：

参数	必需/ 可选	说明
<i>string</i>	必需	字符串。传递要编码的字符串值。可输入任何有效的转换表达式。

返回值

字符串。

当其中一个以下条件为真时返回 NULL：

- 传递至函数的值为 NULL 时。
- 字符串中没有字符是英语字母。
- 字符串为空。

示例

以下表达式对 EMPLOYEE_NAME 端口中的值进行编码：

```
SOUNDEX( EMPLOYEE_NAME )
```

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

SQL_LIKE

返回值是否匹配正则表达式模式。这使您可验证日期模式，例如，ID、电话号码、邮政编码和州名称。

语法

```
SQL_LIKE(subject, pattern, escape character)
```

下表介绍了此命令的参数：

参数	必需/可选	说明
主题	必需	String 数据类型。传递要与正则表达式匹配的值。用单引号将值引起来。
模式	必需	String 数据类型。要匹配的正则表达式。用单引号将模式引起来。
转义符	可选	String 数据类型。SQL_LIKE 函数支持百分比符号 (%) 和下划线 (_) 作为转义符。用单引号将转义符引起来。

返回值

数据与模式匹配时返回 TRUE。

数据与模式不匹配时返回 FALSE。

当输入为空值或模式为 NULL 时返回 NULL。

示例

可以在表达式中使用 SQL_LIKE 来查找与某个模式相匹配的名称。例如，以下表达式将名称与具有转义符 '#' 的模式 "A_#%" 进行匹配：

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

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

SQRT

返回非负数值的平方根。

语法

```
SQRT( numeric_value )
```

下表介绍了此命令的参数：

参数	必需/ 可选	说明
<i>numeric_value</i>	必需	正数值。 传递要为其计算平方根的值。 可输入任何有效的转换表达式。

返回值

双精度值。

传递至函数的值为 NULL 时返回 NULL。

示例

以下表达式为 Numbers 端口中的值返回平方根：

```
SQRT( NUMBERS )
```

NUMBERS	RETURN VALUE
100	10
-100	<i>Error. PowerCenter 集成服务 does not write row.</i>

NUMBERS	RETURN VALUE
NULL	NULL
60.54	7.78074546557076

值 -100 导致错误，因为函数 SQRT 仅计算正数值。如果传递负值或字符值，则 PowerCenter 集成服务显示转换计算错误且不写入行。

在函数计算平方根之前，可对传递至 SQRT 的值执行算术运算。

STDDEV

返回传递给此函数的数值的标准偏差。STDDEV 用于分析统计数据。只能在 STDDEV 内嵌套一个其他汇总函数，嵌套函数必须返回 Numeric 数据类型。

语法

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

下表介绍了此命令的参数：

参数	必需/ 可选	说明
<i>numeric_value</i>	必需	Numeric 数据类型。此函数传递要为其计算标准偏差的值或函数的结果。可输入任何有效的转换表达式。可使用运算符求取不同端口中值的平均值。
<i>filter_condition</i>	可选	限制搜索中的行数。筛选条件必须为数值或计算结果为 TRUE、FALSE 或 NULL。可输入任何有效的转换表达式。

返回值

数值。

当传递至函数的所有值为 NULL 或未选定行时返回 NULL（例如，筛选条件对所有行的计算结果均为 FALSE 或 NULL）。

注意：如果返回值为精度大于 15 的小数，则可启用高精度，以确保小数精确至 38 位。

空值

如果单值为 NULL，则 STDDEV 忽略它。然而，如果所有值均为 NULL，则 STDDEV 返回 NULL。

注意：默认情况下，PowerCenter 集成服务在汇总函数中将空值当作 NULL 处理。如果传递整个端口或空值组，则函数返回 NULL。然而，在配置 PowerCenter 集成服务时，可选择想要处理汇总函数中空值的方式。可将汇总函数中的空值作为 0 或 NULL 处理。

分组依据

STDDEV 根据您在转换中定义的分组依据端口对值进行分组，为每组返回一个结果。

如果不存在分组依据端口，则 STDDEV 将所有行当作一个组处理，返回一个值。

示例

以下表达式为 TOTAL_SALES 端口中所有行计算大于 \$2000.00 的标准偏差：

```
STDDEV( SALES, SALES > 2000.00 )
```

SALES

2198.0

1010.90

2256.0

153.88

3001.0

NULL

8953.0

RETURN VALUE: 3254.60361129688

此函数不将值 1010.90 和 153.88 纳入计算中，因为 *filter_condition* 指定大于 \$2,000 的销售额。

以下表达式计算 SALES 端口中所有行的标准偏差：

```
STDDEV(SALES)
```

SALES

2198.0

2198.0

2198.0

2198.0

RETURN VALUE: 0

返回值为 0，因为每行均包含相同的数字（不存在标准偏差）。如果没有标准偏差，则返回值为 0。

STRUCT

生成其元素名称和数据类型基于指定的参数的结构。

语法

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

下表介绍了此命令的参数：

参数	必需/可选	说明
element_name1	必需	结构元素的名称。
value1	必需	任何数据类型。结构元素的值。

如果在结构端口的输出表达式中使用 STRUCT 函数，则函数参数必须与复杂数据类型定义中的元素数据类型匹配。

返回值

结构。

示例

以下表达式将生成一个结构。

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

RETURN VALUE

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

以下表达式使用复杂数据类型定义 **adrs_typedef** 为结构输出端口生成一个结构：

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

复杂数据类型定义 **adrs_typedef** 在类型定义库中按如下进行定义：

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

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

STRUCT_AS

生成其架构基于指定的复杂数据类型定义和作为参数传递的值的结构。

语法

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

下表介绍了此命令的参数：

参数	必需/可选	说明
:Type.type_definition_library.type_definition	必需	表示结构数据的架构的复杂数据类型定义。使用引用限定符 :Type 可引用包含复杂数据类型定义的类型定义库。
struct_value	必需	复杂数据类型定义中每个元素的值，用逗号分隔。

返回值

结构。

示例

以下表达式根据指定的复杂数据类型定义 `h1_address_def` 并使用所传递的值作为结构元素的参数来生成一个结构。

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

复杂数据类型定义 **h1_address_def** 在类型定义库中按如下进行定义：

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

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

SUBSTR

返回字符串的一部分。SUBSTR 从字符串的开头开始计数所有字符，其中包括空白。

语法

SUBSTR(*string*, *start* [, *length*])

下表介绍了此命令的参数：

参数	必需/ 可选	说明
<i>string</i>	必需	必须是字符串。传递要搜索的字符串。可输入任何有效的转换表达式。如果传递数值，则函数将其转换为字符串。
<i>启动</i>	必需	必须为整数。想要在字符串中开始计数的位置。可输入任何有效的转换表达式。如果起始位置为正数，则 SUBSTR 从字符串的开头计数以查找起始位置。如果起始位置为负数，则 SUBSTR 从字符串的末尾计数以查找起始位置。如果起始位置为 0，则 SUBSTR 从字符串的第一个字符搜索。
<i>length</i>	可选	必须是大于 0 的整数。想要 SUBSTR 返回的字符数量。可输入任何有效的转换表达式。如果忽略长度参数，则 SUBSTR 返回从字符串的起始位置至末尾的所有字符。如果传递负整数或 0，则函数返回空字符串。如果传递小数，则函数将其舍入为最接近的整数。

返回值

字符串。

如果传递负数或 0 长度值，则返回空字符串。

传递至函数的值为 NULL 时返回 NULL。

示例

以下表达式为电话端口中的每行返回区号：

SUBSTR(PHONE, 0, 3)

PHONE	RETURN VALUE
809-555-0269	809
357-687-6708	357
NULL	NULL

SUBSTR(PHONE, 1, 3)

PHONE	RETURN VALUE
809-555-3915	809
357-687-6708	357
NULL	NULL

以下表达式为电话端口中的每行返回无区号的电话号码：

```
SUBSTR( PHONE, 5, 8 )
```

PHONE	RETURN VALUE
808-555-0269	555-0269
809-555-3915	555-3915
357-687-6708	687-6708
NULL	NULL

也可传递负起始值，为电话端口中的每行返回电话号码。当返回 *长度* 参数的结果时，表达式仍从左到右读取源字符串：

```
SUBSTR( PHONE, -8, 3 )
```

PHONE	RETURN VALUE
808-555-0269	555
809-555-3915	555
357-687-6708	687
NULL	NULL

可在 *起始* 或 *长度* 参数中嵌套 INSTR，以搜索特定字符串并返回其位置。

以下表达式计算字符串，从字符串的末尾开始。此表达式查找字符串中的最后（最右）空格，然后返回其前面的所有字符：

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

CUST_NAME	RETURN VALUE
PATRICIA JONES	PATRICIA
MARY ELLEN SHAH	MARY ELLEN

以下表达式删除字符串中的字符 '#'：

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

当 *长度* 参数长于字符串时，SUBSTR 返回从字符串的起始位置至末尾的所有字符。请考虑以下示例：

```
SUBSTR('abcd', 2, 8)
```

返回值为 'bcd'。将此结果与以下示例相比较：

```
SUBSTR('abcd', -2, 8)
```

返回值为 'cd'。

SUM

返回选定端口中所有值的和值。或者，可应用筛选器限定读取的行数以计算和值。只能在 SUM 内嵌套一个其他汇总函数，嵌套函数必须返回 Numeric 数据类型。

语法

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

下表介绍了此命令的参数：

参数	必需/ 可选	说明
<i>numeric_value</i>	必需	数值数据类型。传递要加上的值。可输入任何有效的转换表达式。可使用运算符加上不同端口中的值。
<i>filter_condition</i>	可选	限制搜索中的行数。筛选条件必须为数值或计算结果为 TRUE、FALSE 或 NULL。可输入任何有效的转换表达式。

返回值

数值。

当传递至函数的所有值为 NULL 或未选定行时返回 NULL（例如，筛选条件对所有行的计算结果均为 FALSE 或 NULL）。

注意: 如果返回值为精度大于 15 的小数，则可启用高精度，以确保小数精确至 38 位。

空值

如果单值为 NULL，则 SUM 忽略它。然而，如果从端口传递的所有值均为 NULL，则 SUM 返回 NULL。

注意: 默认情况下，PowerCenter 集成服务在汇总函数中将空值当作 NULL 处理。如果传递整个端口或空值组，则函数返回 NULL。然而，在配置 PowerCenter 集成服务时，可选择想要处理汇总函数中空值的方式。可将汇总函数中的空值作为 0 或 NULL 处理。

分组依据

SUM 根据您在转换中定义的分组依据端口对值进行分组，为每组返回一个结果。

如果不存在分组依据端口，则 SUM 将所有行当作一个组处理，返回一个值。

示例

以下表达式返回销售额端口中大于 2000 的所有值的和值：

```
SUM( SALES, SALES > 2000 )
```

SALES
2500.0
1900.0
1200.0

SALES

NULL

3458.0

4519.0

RETURN VALUE: 10477.0

提示

在函数计算和值之前，可对传递至 SUM 的值执行算术运算。例如：

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

SYSTIMESTAMP

返回 PowerCenter 集成服务上托管的节点的当前日期和时间，精确至纳秒。显示日期和时间的精度取决于平台。

函数的返回值因您如何配置参数而异：

- 当您将参数 SYSTIMESTAMP 配置为变量时，PowerCenter 集成服务为转换中的每行计算函数。
- 当您将参数 SYSTIMESTAMP 配置为常量时，PowerCenter 集成服务计算函数一次并保留转换中每行的值。

语法

```
SYSTIMESTAMP( [format] )
```

下表介绍了此命令的参数：

参数	必需/ 可选	说明
格式	可选	要将时间戳检索到的精度。可将精度指定为秒 (SS)、毫秒 (MS)、微秒 (US) 或纳秒 (NS)。用单引号将格式字符串引起来。格式字符串不区分大小写。例如，要以毫秒精度显示日期和时间，请使用以下语法： SYSTIMESTAMP('MS')。默认精度为微秒 (US)。

返回值

时间戳。将日期和时间返回指定精度。

示例

贵组织提供在线订单服务，并处理实时数据。您可使用 SYSTIMESTAMP 函数为目标数据库中每个事务生成主键。

创建有以下端口和值的表达式转换：

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)

在运行时，PowerCenter 集成服务为每行生成精度为微秒的系统时间：

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

返回数值的正切(以弧度表示)。

语法

TAN(*numeric_value*)

下表介绍了此命令的参数：

参数	必需/ 可选	说明
<i>numeric_value</i>	必需	数值数据类型。以弧度表示的数值数据（度数乘以 pi 除以 180）。传递要为其计算正切的数值。可输入任何有效的转换表达式。

返回值

双精度值。

传递至函数的值为 NULL 时返回 NULL。

示例

以下表达式为度数端口中的所有值返回正切：

```
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

返回传递至此函数的数值的双曲正切。

语法

```
TANH( numeric_value )
```

下表介绍了此命令的参数：

参数	必需/ 可选	说明
<i>numeric_value</i>	必需	数值数据类型。以弧度表示的数值数据（度数乘以 pi 除以 180）。传递要为其计算双曲正切的数值。可输入任何有效的转换表达式。

返回值

双精度值。

传递至函数的值为 NULL 时返回 NULL。

示例

以下表达式为角度端口中的值返回双曲正切：

```
TANH( ANGLES )
```

ANGLES	RETURN VALUE
1.0	0.761594155955765

ANGLES	RETURN VALUE
2.897	0.993926947790665
3.66	0.998676551914886
5.45	0.999963084213409
0	0.0
0.345	0.331933853503641
NULL	NULL

提示

在函数计算双曲正切之前，可对传递至 TANH 的值执行算术运算。例如：

```
TANH( ARCS / 360 )
```

TIME_RANGE

确定要联接的流事件的时间范围。

TIME_RANGE 函数只适用于流映射中的联接器转换。

语法

```
TIME_RANGE(EventTime1,EventTime2,Format,Interval)
```

下表介绍了此命令的参数：

参数	必需/可选	说明
EventTime1	必需	Date 数据类型。在联接器转换的主端口生成流事件的时间。
EventTime2	必需	Date 数据类型。在联接器转换的详细端口生成流事件的时间。
Format	必需	用于指定要更改的事件时间值部分的格式字符串。用单引号将格式字符串引起来。例如，'Seconds'。格式字符串不区分大小写。 格式参数接受以下值： <ul style="list-style-type: none"> - 年 - 月 - 周 - 日 - 小时 - 分钟 - 秒 - 毫秒 - 微秒
Interval	必需	要根据格式更改事件时间值的整数值。

返回值

如果传递空值至函数，则返回 NULL。

示例

以下示例返回联接器转换的时间范围表达式：

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

返回值：

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

TO_BIGINT

将字符串或数值转换为长整型值。TO_BIGINT 语法包含可选参数，您可以选择该参数将数值舍入为最接近的整数或截断小数部分。TO_BIGINT 忽略前导空白。

语法

```
TO_BIGINT( value [, flag] )
```

下表介绍了此命令的参数：

参数	必需/ 可选	说明
值	必需	String 或 numeric 数据类型。 传递要转换为长整型值的值。 可输入任何有效的转换表达式。
flag	可选	指定截断还是舍入小数部分。 标记必须为整数文字或常量 TRUE 或 FALSE。 当标志为 TRUE 或非 0 数字时，TO_BIGINT 截断小数部分。 当标志为 FALSE 或 0 或当您忽略此参数时，TO_BIGINT 将值舍入为最接近的整数。 默认情况下不设置此标志。

返回值

长整型。

传递至函数的值为 NULL 时返回 NULL。

当传递至函数的值包含字母字符时返回 0。

如果传递至函数的值包含对长整型值无效的数据，则数据集成服务将行标记为错误行或映射失败。

示例

以下表达式使用源自端口 IN_TAX 的值：

```
TO_BIGINT( IN_TAX, TRUE )
```

IN_TAX	RETURN VALUE
'7245176201123435.6789'	7245176201123435

IN_TAX	RETURN VALUE
'7245176201123435.2'	7245176201123435
'7245176201123435.2.48'	7245176201123435
NULL	NULL
'A12.3Grove'	0
'A12.3Grove'	<i>Error. Integration Service skips this row.</i>
'176201123435.87'	176201123435
'-7245176201123435.2'	-7245176201123435
'-7245176201123435.23'	-7245176201123435
-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'	<i>Error. Integration Service skips this row.</i>
'176201123435.87'	176201123436
'-7245176201123435.6789'	-7245176201123436
'-7245176201123435.23'	-7245176201123435
-9223372036854775806.9	-9223372036854775807
9223372036854775806.9	9223372036854775807

TO_CHAR (Dates)

将日期转换为字符串。TO_CHAR 也将数值转换为字符串。也可使用 TO_CHAR 格式字符串将日期转换为任何格式。

TO_CHAR (date [,format]) 将 date、Timestamp、Timestamp with Time Zone 或 Timestamp with Local Time Zone 数据类型的数据类型或内部值转换为通过格式字符串指定的 string 数据类型的值。

语法

```
TO_CHAR( date [,format] )
```

下表介绍了此命令的参数：

参数	必需/ 可选	说明
日期	必需	日期/时间数据类型。传递要转换为字符串的日期值。可输入任何有效的转换表达式。
格式	可选	输入有效的 TO_CHAR 格式字符串。格式字符串定义返回值的格式，而不定义日期参数值的格式。如果忽略格式字符串，则函数将根据会话中指定的日期格式返回字符串。 输入有效的 TO_CHAR 格式字符串。格式字符串定义返回值的格式，而不定义日期参数值的格式。如果忽略格式字符串，则函数将根据映射配置中指定的日期格式返回字符串。

返回值

字符串。

传递至函数的值为 NULL 时返回 NULL。

示例

以下表达式将 DATE_PROMISED 端口中的日期转换为 MON DD YYYY 格式的文本：

```
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

如果忽略 格式参数，则 TO_CHAR 返回会话中指定日期格式的字符串，该格式默认为 MM/DD/YYYY HH24:MI:SS.US：

如果忽略 格式参数，则 TO_CHAR 返回映射配置中指定日期格式的字符串，该格式默认为 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'

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

以下表达式为端口中的每个日期返回周日：

`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

以下表达式为端口中的每个日期返回一个月中的一天：

`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

以下表达式为端口中的每个日期返回一年中的一天：

```
TO_CHAR( DATE_PROMISED, 'DDD' )
```

DATE_PROMISED	RETURN VALUE
04-01-1997 12:00:10AM	'091'
02-22-1997 01:31:10PM	'053'
10-24-1997 02:12:30PM	'297'
NULL	NULL

以下表达式为端口中的每个日期返回一天中的几点：

```
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

以下表达式将日期值转换为用字符串表示的 MJD 值：

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

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

以下表达式将日期转换为 MM/DD/YY 格式的字符串：

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

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

也可在 TO_CHAR 表达式中使用格式字符串 SSSSS。例如，以下表达式将 SHIP_DATE 端口中的日期转换为表示自午夜算起的总秒数的字符串：

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

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

在 TO_CHAR 表达式中，YY 格式字符串与 RR 格式字符串产生相同的结果。

以下表达式将日期转换为 MM/DD/YY 格式的字符串：

```
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

以下表达式为端口中的每个日期返回一个月中第几周：

```
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

以下表达式为端口中的每个日期返回一年中的第几周：

```
TO_CHAR( DATE_PROMISED, 'WW' )
```

DATE_PROMISED	RETURN VALUE
04-01-1997 12:00:10PM	'18'
02-22-1997 01:31:10AM	'08'
10-24-1997 02:12:30AM	'43'
NULL	NULL

提示

可组合使用 TO_CHAR 和 TO_DATE，以使用以下类似函数将一个月的数值转换为该月的文本值：

```
TO_CHAR( TO_DATE( numeric_month, 'MM' ), 'MONTH' )
```

TO_CHAR (Numbers)

将数值转换为文本字符串。TO_CHAR 也将日期转换为字符串。

TO_CHAR 将双精度值转换为文本字符串，如下所示：

- 将长达 16 位的双精度值转换为字符串，并精确到 15 位。如果传递超过 15 位的数值，则 TO_CHAR 根据第十六位舍入该数值，并返回以科学表示法表示该数值的字符串。例如，1234567890123456 双精度值转换为 “1.23456789012346e+015” 字符串值。
- 为范围内的数值返回小数表示法（-1e16、-1e-16] 和 [1e-16, 1e16）。TO_CHAR 为这些范围之外的数值返回科学表示法。例如，10842764968208837340 双精度值转换为 “1.08427649682088e+019” 字符串值。

TO_CHAR 将小数值转换为文本字符串，如下所示：

- 在高精度模式中，TO_CHAR 将长达 38 位的小数值转换为字符串。如果传递超过 38 位的小数值，TO_CHAR 为超过 38 位的数值返回科学表示法。
- 在高精度模式中，TO_CHAR 将长达 28 位的小数值转换为字符串。如果传递超过 28 位的小数值，TO_CHAR 为超过 28 位的数值返回科学表示法。
- 在低精度模式中，TO_CHAR 将小数值视为双精度值。

语法

```
TO_CHAR( numeric_value )
```

下表介绍了此命令的参数：

参数	必需/ 可选	说明
<i>numeric_value</i>	必需	数值数据类型。要转换为字符串的数值。可输入任何有效的转换表达式。

返回值

字符串。

传递至函数的值为 NULL 时返回 NULL。

双精度转换示例

以下表达式将 SALES 端口中的双精度值转换为字符串：

```
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

小数转换示例

以下表达式在高精度模式中将 SALES 端口中的小数值转换为字符串：

```
TO_CHAR( SALES )
```

SALES	RETURN VALUE
2378964536789761	'2378964536789761'
1234567890123456789012345679	'1234567890123456789012345679'
1.234578945469649345876123456	'1.234578945469649345876123456'
0.999999999999999999999999999999	'0.999999999999999999999999999999'
12345678901234567890123456799	'12345678901234567890123456799'
23456788992233456678458934567123465239	'23456788992233456678458934567123465239'
423456789012345678901234567991234567899 (大于 38)	'4.23456789012346e+038'

SALES	RETURN VALUE
2378964536789761	'2378964536789761'

SALES	RETURN VALUE
1234567890123456789012345679	'1234567890123456789012345679'
1.234578945469649345876123456	'1.234578945469649345876123456'
0.9999999999999999999999999999	'0.9999999999999999999999999999'
12345678901234567890123456799 (大于 28)	'1.23456789012346e+028'

TO_DATE

将字符串转换为 Date/Time 数据类型。您使用 TO_DATE 格式字符串指定源字符串的格式。

输出端口必须为 TO_DATE 表达式的 Date/Time。

如果通过 TO_DATE 转换两位数年份，请使用 RR 或 YY 格式字符串。不要使用 YYYY 格式字符串。

语法

TO_DATE(*string* [, *format*])

下表介绍了此命令的参数：

参数	必需/ 可选	说明
<i>string</i>	必需	必须是 string 数据类型。传递想要转换为日期的值。可输入任何有效的转换表达式。
<i>格式</i>	可选	输入有效的 TO_DATE 格式字符串。此格式字符串必须与 <i>string</i> 参数的各部分相匹配。例如，如果传递字符串 'Mar 15 1998 12:43:10AM'，则必须使用格式字符串 'MON DD YYYY HH12:MI:SSAM'。如果忽略格式字符串，则字符串值必须采用会话中指定的日期格式。

返回值

日期。

TO_DATE 始终返回日期和时间。如果传递没有时间值的字符串，则返回的日期始终包括时间 00:00:00.000000000。可将此函数的结果映射至数据类型为日期时间的任何目标列。如果目标列精度低于纳秒，则 PowerCenter 集成服务截断日期时间值，以便在将日期时间值写入目标时匹配目标列的精度。

传递空值至此函数时返回 NULL。

警告: TO_DATE 字符串的格式必须与格式字符串相匹配，包括任何日期分隔符。如果不匹配，则 PowerCenter 集成服务可能返回不准确的值或跳过记录。

示例

以下表达式为 DATE_PROMISED 端口中的字符串返回日期值。TO_DATE 始终返回日期和时间。如果传递没有时间值的字符串，则返回的日期始终包括时间 00:00:00.000000000。如果在第二十七世纪运行会话，则世纪将为 19。在此示例中，运行 PowerCenter 集成服务的节点上的当前年份为 1998。目标列的日期时间值为 MON DD YY HH24:MI SS，因此 PowerCenter 集成服务在写入目标时将日期时间值截断为秒数：

以下表达式为 DATE_PROMISED 端口中的字符串返回日期值。TO_DATE 始终返回日期和时间。如果传递没有时间值的字符串，则返回的日期始终包括时间 00:00:00.000000000。如果在二十世纪运行映射，则世纪将为 19。在此示例中，运行 PowerCenter 集成服务的节点上的当前年份为 1998。目标列的日期时间值为 MON DD YY HH24:MI SS，因此 PowerCenter 集成服务在写入目标时将日期时间值截断为秒数：

```
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

以下表达式为 DATE_PROMISED 端口中的字符串返回日期和时间值。如果传递没有时间值的字符串，则 PowerCenter 集成服务返回错误。如果在二十世纪运行会话，则世纪将为 19。运行 PowerCenter 集成服务的节点上的当前年份为 1998：

以下表达式为 DATE_PROMISED 端口中的字符串返回日期和时间值。如果传递没有时间值的字符串，则 PowerCenter 集成服务返回错误。如果在二十世纪运行映射，则世纪将为 19。运行 PowerCenter 集成服务的节点上的当前年份为 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'	<i>Error. Integration Service skips this row.</i>
NULL	NULL

以下表达式将 SHIP_DATE_MJD_STRING 端口中的字符串转换为日期值：

```
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

因为 J 格式字符串不包括日期的时间部分，因此，返回值的时间设定为 00:00:00.000000000。

以下表达式将字符串转换为四位数年份格式。当前年份为 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

以下表达式将字符串转换为四位数年份格式。当前年份为 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

注意: 对于第二行, RR 返回年份 2005, 但 YY 返回年份 1905。

以下表达式将字符串转换为四位数年份格式。当前年份为 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

以下表达式将字符串转换为四位数年份格式。当前年份为 1998：

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

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

以下表达式将包括自午夜算起的秒数的字符串转换为日期值：

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

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

如果目标接受不同的日期格式，请将 TO_DATE 和 IS_DATE 与 DECODE 函数组合，以测试是否存在可接受的字符串。例如：

```
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' ) )
```

可组合使用 TO_CHAR 和 TO_DATE，以使用以下类似函数将一个数的数值转换为该月的文本值：

```
TO_CHAR( TO_DATE( numeric_month, 'MM' ), 'MONTH' )
```

相关主题：

- [“日期格式字符串的规则和准则” 页面上 49](#)

TO_DECIMAL

将字符串或数值转换为十进制值。TO_DECIMAL 忽略前导空白。

语法

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

下表介绍了此命令的参数：

参数	必需/ 可选	说明
值	必需	必须是字符串或数值数据类型。传递要转换为小数值的值。可输入任何有效的转换表达式。
小数位数	可选	必须是介于 0 与 28 之间（包括 0 和 28）的整数文字。指定小数点后允许的位数。如果忽略此参数，则函数返回小数位数与输入值相同的值。

返回值

精度和小数位数介于 0 与 28 之间（包括 0 和 28）的小数。

传递至函数的值为 NULL 时返回 NULL。

如果字符串包含非数字字符，请转换字符串中直到第一个非数字字符的数字部分。

当第一个数值字符为非数值时返回 0。

如果传递至函数的值包含对小数值无效的数据，则数据集成服务会将行标记为错误行。

注意: 如果返回值为精度大于 15 的小数，则可启用高精度，以确保小数精确至 28 位。

示例

此表达式使用源自端口 IN_TAX 的值。IN_TAX 是一个精度为 44 位的 String 数据类型。RETURN VALUE 是一个精度为 28、小数位数为 3 的 Decimal 数据类型：

```
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.
'1234567890123456789012345678901234567890.123'	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
'123456789012345678901234567890.123'	Error. Integration Service skips this row.
'1234567890123456789012345678901234567890.123'	Error. Integration Service skips this row.

TO_DECIMAL38

将字符串或数值转换为十进制值。TO_DECIMAL38 忽略前导空格。

语法

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

下表介绍了此命令的参数：

参数	必需/ 可选	说明
值	必需	必须为 string 或 numeric 数据类型。传递要转换为小数值的值。可输入任何有效的转换表达式。
小数位数	可选	必须是介于 0 与 38 之间（包括 0 和 38）的整数文字。指定小数点后允许的位数。如果忽略此参数，则函数返回小数位数与输入值相同的值。

返回值

精度和小数位数介于 0 与 38 之间（包括 0 和 38）的小数。

传递至函数的值为 NULL 时返回 NULL。

如果传递至函数的值包含对小数值无效的数据，则数据集成服务会将行标记为错误行。例如，如果传递 TO_DECIMAL38("1234567890123456789012345678901234567890.12")，则数据集成服务将拒绝该行。

注意: 如果返回值为精度大于 15 的小数，则可启用高精度，以确保小数精确至 38 位。

示例

此表达式使用源自端口 IN_TAX 的值。IN_TAX 是一个精度为 44 位的 String 数据类型。RETURN VALUE 是一个精度为 38、小数位数为 3 的 Decimal 数据类型：

```
TO_DECIMAL38( IN_TAX, 3 )
```

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

IN_TAX	RETURN VALUE
'711A1'	<i>Error. Integration Service skips this row.</i>

TO_FLOAT

将字符串或数值转换为双精度浮点数字（Double 数据类型）。TO_FLOAT 忽略前导空白。

语法

TO_FLOAT(*value*)

下表介绍了此命令的参数：

参数	必需/ 可选	说明
<i>值</i>	必需	必须是 string 或 numeric 数据类型。传递要转换为双精度值的值。可输入任何有效的转换表达式。

返回值

双精度值。

传递至此函数的值为 NULL 时返回 NULL。

端口中的值为空白或非数字字符时返回 0。

如果传递至函数的值包含对浮点值无效的数据，则数据集成服务将行标记为错误行或映射失败。

示例

此表达式使用源自端口 IN_TAX 的值：

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

TO_INTEGER

将数值或字符串值转换为整数。TO_INTEGER 语法包含可选参数，您可以选择该参数将数值舍入为最接近的整数或截断小数部分。TO_INTEGER 忽略前导空白。

语法

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

下表介绍了此命令的参数：

参数	必需/ 可选	说明
值	必需	String 或 numeric 数据类型。传递要转换为整数的值。可输入任何有效的转换表达式。
flag	可选	指定截断还是舍入小数部分。标记必须为整数文字或常量 TRUE 或 FALSE。 当标志为 TRUE 或非 0 数字时，TO_INTEGER 截断小数部分。 当标志为 FALSE 或 0 或忽略此参数时，TO_INTEGER 将值舍入为最接近的整数。

返回值

整数。

传递至函数的值为 NULL 时返回 NULL。

当传递至函数的值包含字母字符时返回 0。

如果传递至函数的值包含对整数值无效的数据，则数据集成服务将行标记为错误行或映射失败。

示例

以下表达式使用源自端口 IN_TAX 的值。当转换导致数值溢出时，PowerCenter 集成服务显示错误：

```
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'	<i>Error. Integration Service skips this row.</i>
NULL	NULL
'A12.3Grove'	0
'A12.3Grove'	<i>Error. Integration Service skips this row.</i>
' 123.87'	124
'-15.6789'	-16
'-15.23'	-15

TO_TIMESTAMP_TZ

将字符串转换为 Timestamp with Time Zone 值。该函数返回 Timestamp with Time Zone 数据类型。使用 TO_TIMESTAMP_TZ 格式字符串指定源字符串的格式。

语法

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


下表介绍了此命令的参数：

参数	必需/ 可选	说明
<i>String</i>	必需	必须为 string 数据类型。传递要转换为 Timestamp with Time Zone 的值。 可输入任何有效的转换表达式。 此字符串必须为字符串。
格式	可选	输入有效的 TO_TIMESTAMP_TZ 格式字符串。此格式字符串必须与 string 参数的各部分相匹配。例如，如果传递字符串 'Mar 15 1997 12:43:10AM ASIA/CALCUTTA'，则必须使用格式字符串 'MON DD YYYY HH12:MI:SSAM TZR'。 如果未指定格式字符串，该函数将使用“运行配置”对话框中的默认日期时间格式。

返回值

返回 timestamp with time zone 数据类型。

当输入为空值时返回 NULL。

如果传递至函数的值包含对 timestamp with time zone 值无效的数据，则数据集成服务会将行标记为错误行或使映射失败。

示例

INPUT VALUE	RETURN VALUE
'1947-08-05 10:45:00.221111000 AM America/Los_Angeles' , 'YYYY-MM-DD HH:MI:SS.NS AM TZR'	返回具有以下数据的 timestamp with time zone 数据类型： '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'	即使未以时区区域格式指定时区区域，也会返回 timestamp with time zone 数据类型。 '1947-08-05 10:45:00.221111000 AM AMERICA/LOS_ANGELES'
'1947-08-05 10:45:00.221111000 AM America/Los_Angeles'	即使未以时区格式指定时间戳，也会返回 timestamp with time zone 数据类型。 '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'	当未在函数级别指定日期时间格式时，将使用“运行配置”对话框中的默认格式。 默认日期时间格式：'YYYY-MM-DD HH:MI:SS.NS AM TZR' 如果 timestamp with time zone 数据与指定格式不匹配，则将出现以下错误： Process row failed for function [TO_TIMESTAMP_TZ]: Failed to convert the string to timestamp with time zone value. Verify that the specified date format string is valid. Verify that the timestamp with time zone string used in the first argument is compatible with the specified date format.

TRUNC (Dates)

将日期截断为特定年份、月份、日、小时、分种、秒、毫秒或微秒。也可使用 TRUNC 截断数字。

可截断以下日期部分：

- **年份**。如果截断日期的年份部分，则函数返回输入年份 1 月 1 日，将时间设定为 00:00:00.000000000。例如，以下表达式返回 1/1/1997 00:00:00.000000000：
`TRUNC(12/1/1997 3:10:15, 'YY')`
- **月份**。如果截断日期的月份部分，则函数返回月份的第一天，将时间设定为 00:00:00.000000000。例如，以下表达式返回 4/1/1997 00:00:00.000000000：
`TRUNC(4/15/1997 12:15:00, 'MM')`
- **日**。如果截断日期的日部分，则函数返回将时间设定为 00:00:00.000000000 的日期。例如，以下表达式返回 6/13/1997 00:00:00.000000000：
`TRUNC(6/13/1997 2:30:45, 'DD')`
- **小时**。如果截断日期的小时部分，则函数返回将分钟、秒数和子秒设定为 0 的日期。例如，以下表达式返回 4/1/1997 11:00:00.000000000：
`TRUNC(4/1/1997 11:29:35, 'HH')`
- **分钟**。如果截断日期的分钟部分，则函数返回将秒数和子秒设定为 0 的日期。例如，以下表达式返回 5/22/1997 10:15:00.000000000：
`TRUNC(5/22/1997 10:15:29, 'MI')`
- **秒**。如果截断日期的秒数部分，则函数返回将毫秒设定为 0 的日期。例如，以下表达式返回 5/22/1997 10:15:29.000000000：
`TRUNC(5/22/1997 10:15:29.135, 'SS')`
- **毫秒**。如果截断日期的毫秒部分，则函数返回将微秒设定为 0 的日期。例如，以下表达式返回 5/22/1997 10:15:30.135000000：
`TRUNC(5/22/1997 10:15:30.135235, 'MS')`
- **微秒**。如果截断日期的微秒部分，则函数返回将纳秒设定为 0 的日期。例如，以下表达式返回 5/22/1997 10:15:30.135235000：
`TRUNC(5/22/1997 10:15:29.135235478, 'US')`

语法

`TRUNC(date [,format])`

下表介绍了此命令的参数：

参数	必需/ 可选	说明
<i>date</i>	必需	Date/Time 数据类型。要截断的日期值。可输入任何有效的转换表达式以计算日期。
<i>格式</i>	可选	输入有效的格式字符串。格式字符串不区分大小写。如果忽略格式字符串，则函数截断日期的时间部分，将其设置为 00:00:00.000000000。

返回值

日期。

传递至函数的值为 NULL 时返回 NULL。

示例

以下表达式截断 DATE_SHIPPED 端口中日期的年份部分：

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

DATE_SHIPPED	RETURN VALUE
Jan 15 1998 2:10:30AM	Jan 1 1998 12:00:00.000000000
Apr 19 1998 1:31:20PM	Jan 1 1998 12:00:00.000000000
Jun 20 1998 3:50:04AM	Jan 1 1998 12:00:00.000000000
Dec 20 1998 3:29:55PM	Jan 1 1998 12:00:00.000000000
NULL	NULL

以下表达式截断 DATE_SHIPPED 端口中每个日期的月份部分：

```
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
Dec 20 1998 3:29:55PM	Dec 1 1998 12:00:00.000000000AM
NULL	NULL

以下表达式截断 DATE_SHIPPED 端口中每个日期的日部分：

```
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

DATE_SHIPPED	RETURN VALUE
Dec 20 1998 3:29:55PM	Dec 20 1998 12:00:00.000000000AM
Dec 31 1998 11:59:59PM	Dec 31 1998 12:00:00.000000000AM
NULL	NULL

以下表达式截断 DATE_SHIPPED 端口中每个日期的小时部分：

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

DATE_SHIPPED	RETURN VALUE
Jan 15 1998 2:10:31AM	Jan 15 1998 2:00:00.000000000AM
Apr 19 1998 1:31:20PM	Apr 19 1998 1:00:00.000000000PM
Jun 20 1998 3:50:04AM	Jun 20 1998 3:00:00.000000000AM
Dec 20 1998 3:29:55PM	Dec 20 1998 3:00:00.000000000PM
Dec 31 1998 11:59:59PM	Dec 31 1998 11:00:00.000000000AM
NULL	NULL

以下表达式截断 DATE_SHIPPED 端口中每个日期的分钟部分：

```
TRUNC( DATE_SHIPPED, 'MI' )
```

DATE_SHIPPED	RETURN VALUE
Jan 15 1998 2:10:30AM	Jan 15 1998 2:10:00.000000000AM
Apr 19 1998 1:31:20PM	Apr 19 1998 1:31:00.000000000PM
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 (Numbers)

将数字截断为特定位数。也可使用 TRUNC 截断日期。

语法

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

下表介绍了此命令的参数：

参数	必需/ 可选	说明
<i>numeric_value</i>	必需	数值数据类型。 传递要截断的值。 可输入计算结果为 Numeric 数据类型的任何有效转换表达式。
<i>精度</i>	可选	可能为正整数或负整数。 可输入任何有效的转换表达式以计算整数。 整数指定要截断的位数。

如果 *精度* 为正整数，则 TRUNC 返回 *numeric_value*，其小数位由 *精度* 指定。 如果 *精度* 为负整数，则 TRUNC 将小数点左侧的指定位数更改为零。 如果忽略 *精度* 参数，则 TRUNC 截断 *numeric_value* 的小数部分，并返回整数。

如果传递小数 *精度* 值，则 PowerCenter 集成服务将 *numeric_value* 舍入为最接近的整数，然后再计算表达式。

在高精度模式中运行会话时，请在截断之前使用 ROUND 函数。

在高精度模式中运行映射时，请在截断之前使用 ROUND 函数。

例如，假定以下表达式用于截断 QTY 端口中的值：

```
TRUNC ( QTY / 15 )
```

当 QTY 的值 = 15000000，会话返回值 999999。 期望的结果为 1000000。

在运行时，PowerCenter 集成服务先计算表达式的常量部分，然后计算变量部分。

在上述表达式中，QTY 为变量值，(1/15) 为常量值。

当 QTY = 15000000，表达式按以下所示截断：

```
TRUNC ( 15000000 * (1/15)
TRUNC ( 15000000 * (1/15)
= TRUNC ( 15000000 * 0.0666666666666666)
= TRUNC ( 15000000 * 0.0666666666666666)
= TRUNC ( 999999.99999999)
= 999999
```

如果在截断之前使用 ROUND 函数，则按以下所示对表达式求值：

```
TRUNC (ROUND (QTY/15, .999999999999999999999999999999)).
```

返回值

数值。

当其中一个参数均为 NULL 时返回 NULL。

注意：如果返回值为精度大于 15 的小数，则可启用高精度，以确保小数精确至 38 位。

示例

以下表达式截断价格端口中的值：

`TRUNC(PRICE, 3)`

PRICE	RETURN VALUE
12.9995	12.999
-18.8652	-18.865
56.9563	56.956
15.9928	15.992
NULL	NULL

`TRUNC(PRICE, -1)`

PRICE	RETURN VALUE
12.99	10.0
-187.86	-180.0
56.95	50.0
1235.99	1230.0
NULL	NULL

`TRUNC(PRICE)`

PRICE	RETURN VALUE
12.99	12.0
-18.99	-18.0
56.95	56.0
15.99	15.0
NULL	NULL

UPPER

将小写字符串字符转换为大写。

语法

`UPPER(string)`

下表介绍了此命令的参数：

参数	必需/ 可选	说明
<i>string</i>	必需	字符串数据类型。 传递要更改为大写文本的值。 可输入任何有效的转换表达式。

返回值

大写字符串。 如果数据包含多字节字符，则返回值取决于 PowerCenter 集成服务的代码页和数据移动模式。

传递至函数的值为 NULL 时返回 NULL。

示例

以下表达式将 FIRST_NAME 端口中的所有名称更改为大写：

```
UPPER( FIRST_NAME )
```

FIRST_NAME	RETURN VALUE
Ramona	RAMONA
NULL	NULL
THOMAS	THOMAS
PierRe	PIERRE
Bernice	BERNICE

UUID4

返回符合 RFC 4122 中所述 UUID 规范的变量 4 的随机生成 16 字节二进制值。 UUID4 不获取参数。

语法

```
UUID4()
```

返回值

二进制。

UUID4 从不返回空值或错误。

UUID_UNPARSE

将 16 字节二进制值转换为 RFC 4122 中指定的 36 字符字符串表示形式。

语法

```
UUID_UNPARSE( binary )
```

下表介绍了此命令的参数：

参数	必需/ 可选	说明
<i>binary</i>	必需	二进制数据类型。要转换为 36 个字符字符串的任何 16 字节二进制值。

返回值

36 个字符字符串。

当参数为 NULL 时返回 NULL，当参数不是 16 字节二进制值时返回错误。

示例

以下表达式可能返回 6948DF80-14BD-4E04-8842-7668D9C001F5 的值：

```
UUID_UNPARSE(UUID4())
```

VARIANCE

返回向其传递的值的差额。VARIANCE 用于分析统计数据。只能在 VARIANCE 内嵌套一个其他汇总函数，嵌套函数必须返回 Numeric 数据类型。

语法

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

下表介绍了此命令的参数：

参数	必需/ 可选	说明
<i>numeric_value</i>	必需	数值数据类型。传递要为其计算差额的值。可输入任何有效的转换表达式。
<i>filter_condition</i>	可选	限制搜索中的行数。筛选条件必须为数值或计算结果为 TRUE、FALSE 或 NULL。可输入任何有效的转换表达式。

返回值

双精度值。

当传递至函数的所有值均为 NULL 或未选定行时返回 NULL（例如，*filter_condition* 对所有行的计算结果均为 FALSE 或 NULL）。

空值

如果单值为 NULL，则 VARIANCE 忽略它。然而，当传递至函数的所有值均为 NULL 或未选择行时，VARIANCE 返回 NULL。

注意：默认情况下，PowerCenter 集成服务在汇总函数中将空值当作 NULL 处理。如果传递整个端口或空值组，则函数返回 NULL。然而，在配置 PowerCenter 集成服务时，可选择想要处理汇总函数中空值的方式。可将汇总函数中的空值作为 0 或 NULL 处理。

分组依据

VARIANCE 根据您在转换中定义的分组依据端口对值进行分组，为每组返回一个结果。

如果不存在分组依据端口，则 VARIANCE 将所有行当作一个组处理，返回一个值。

示例

以下表达式计算 TOTAL_SALES 端口中所有行的差额：

```
VARIANCE( TOTAL_SALES )
```

TOTAL_SALES

2198.0

2256.0

3001.0

NULL

8953.0

RETURN VALUE: 10592444.6666667

创建自定义函数

创建自定义函数概览

自定义函数扩展 PowerCenter 函数库。它们是您创建的用于转换和工作流表达式的函数。您在 PowerCenter 之外用自定义函数 API 创建了自定义函数。要使用自定义函数 API，您必须从 Informatica Developer Community 网站安装 Informatica Development Platform。

自定义函数 API 使用 C 编程语言。您可与他人共享自定义函数。用户可向其存储库添加函数，并将这些函数当作 PowerCenter 语言转换函数一样使用。

本章包括的示例函数展示了如何创建自定义函数并将其与下推优化组合使用。本章中的步骤创建 ECHO 函数。此函数获取参数作为输入，并向用户返回输入值。ECHO 函数的示例代码位于 Informatica Development Platform 安装的 \CustomFunctionAPI\samples\echo 目录中。

您也可查看更复杂的示例自定义函数。SampleLoanPayment 自定义函数包括不能通过使用 C 获取的函数。SampleLoanPayment 位于 Informatica Development Platform 安装的 \CustomFunctionAPI\samples\loanpayment 目录中。

自定义函数创建步骤

完成以下步骤以创建自定义函数：

1. **获取存储库 ID 属性。** 获取存储库 ID 属性以包括存储库插件。
2. **创建表头文件。** 在表头文件中定义一个或更多自定义函数。
3. **创建实现文件。** 在实现文件中定义一个或更多自定义函数。

- 4. **构建模块。** 构建模块以创建 DLL 和共享库。
- 5. **创建存储库插件文件。** 为自定义函数定义元数据。
- 6. **测试自定义函数。** 安装自定义函数并将其用于映射和工作流中进行验证。

安装自定义函数

要使用自定义函数，必须将其添加至 PowerCenter 环境。

相关主题：

- [“安装自定义函数” 页面上 253](#)

步骤 1.获取存储库 ID 属性

在开发自定义函数之前，您必须为自定义函数存储插件确定存储库 ID 属性。定义插件元数据时使用存储库 ID 属性标识插件。

要获取存储库 ID 属性，请执行以下任务之一：

- 如果您在贵组织之外分发自定义函数，则请联系 Informatica。Informatica 向每个插件分配唯一存储库 ID 属性。与其他供应商的存储库 ID 属性有冲突的存储库 ID 属性无效。要获取存储库 ID 属性，请访问 <https://community.informatica.com/community/marketplace/repositoryidattributes> 并单击提交。
- 如果您仅在贵组织之内使用自定义函数，则请定义存储库 ID 属性，无需联系 Informatica。如果您为贵组织开发将与 PowerCenter 中其他插件组合使用的插件，则请为每个插件的存储库 ID 属性分配唯一值。

下表显示了需要唯一值才能定义插件的 XML 属性：

存储库 ID 属性	说明
插件 ID	标识插件的 ID。此值与 PLUGIN 元素的 ID 属性相对应。
供应商 ID	标识开发插件的供应商。此值与 PLUGIN 元素的 VENDORID 属性相对应。
函数组 ID	标识函数组的 ID。此值与 FUNCTION_GROUP 元素的 ID 属性相对应。
函数 ID	标识函数的 ID。此值与 FUNCTION 元素的 ID 属性相对应。

注意: 互相有冲突的存储库 ID 属性无效。

相关主题：

- [“PLUGIN 元素” 页面上 249](#)
- [“FUNCTION_GROUP 元素” 页面上 250](#)
- [“FUNCTION 元素” 页面上 251](#)

步骤 2.创建表头文件

使用 C 创建表头文件，以声明所有函数。将一个表头文件用于一个或多个自定义函数。

以下示例显示 ECHO 自定义函数的 echo.h 表头文件：

```
#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

```

步骤 3.创建实现文件

实现文件包含用于创建自定义函数的函数定义。使用 C 创建实现文件。可将一个实现文件用于一个或多个自定义函数。也可使用一个实现文件定义自定义函数的验证和运行时函数。

以下示例显示 ECHO 自定义函数的 echo.c 实现文件：

```

/*****
 * 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 validity. 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 plug-in 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;
}

```

步骤 4.构建模块

可在 Windows 或 UNIX 上构建模块。为 PowerCenter 运行所在的每个平台构建模块。必须在 Windows 上构建模块，因为 PowerCenter Client 驻留在 Windows 上。您可能需要在 UNIX 或 Linux 上构建模块，具体取决于托管集成服务的节点。

下表列出了在构建模块时每个平台的库文件名称：

平台	模块文件名称
Windows	<module_identifier>.dll
AIX	lib<module_identifier>.a
Linux	lib<module_identifier>.so
Solaris	lib<module_identifier>.so

您在存储库插件 XML 文件中声明这些模块。

相关主题：

- [“步骤 5.创建存储库插件文件” 页面上 249](#)

在 Windows 上构建模块

在 Windows 上，您可使用 Microsoft Visual C++ 构建模块。

要在 Windows 上构建模块：

1. 启动 Visual C++。
2. 单击“文件”>“新建”。
3. 在“新建”对话框中，单击“项目”选项卡，选择“Win32 动态链接库”选项。
4. 输入其位置。

在 Echo 示例中，输入以下目录：

<Informatica Development Platform installation directory>\CustomFunctionAPI\samples\echo

5. 输入项目的名称。
必须使用为自定义函数指定的模块名称作为项目名称。在 Echo 示例中，输入 EchoDemo。
6. 单击“确定”。
此时，Visual C++ c 创建一个向导以定义项目组件。
7. 在向导中，选择一个空 DLL 项目，单击“完成”。在“新项目信息”对话框中单击“确定”。
此时，Visual C++ 在您指定的目录中创建项目文件。
8. 单击“项目”>“添加至项目”>“文件”。
9. 向上导航一个目录级别。此目录包含您创建的过程文件。选择所有 .c 文件，并单击“确定”。
在 Echo 示例中，添加 Echo.c 文件。
10. 单击“项目”>“设置”。
11. 单击 C/C++ 选项卡，从“类别”字段中选择“预处理器”。

12. 在“附加包括目录”字段中，输入以下路径，并单击“确定”：
..; <Informatica Development Platform installation directory>\CustomFunctionAPI\samples\echo; ...
13. 单击“构建” > “构建 <module_name>.dll” 或按 F7 以构建项目。
Visual C++ 创建 DLL，并将其放在项目目录下的调试或发行目录中。

在 UNIX 上构建模块

在 UNIX 上，您可使用任何 C 编译器构建模块。

要在 UNIX 上构建模块：

1. 将环境变量 INFA_HOME 设置为 PowerCenter 集成服务安装目录。
注意: 如果您为 INFA_HOME 环境变量指定了不正确的目录路径，则 PowerCenter 集成服务无法启动。
重新启动节点以应用更改。
2. 输入下表中的命令以创建项目：

UNIX 版本	命令
AIX (32 位)	make -f makefile.aix
AIX (64 位)	make -f makefile.aix64
Linux	make -f makefile.linux
Solaris	make -f makefile.sol

步骤 5.创建存储库插件文件

创建一个 XML 文件，为更多的一个自定义函数定义函数元数据。 在创建或修改插件 XML 文件时使用插件 DTD 文件的结构。 插件 DTD 文件 plugin.dtd 存储在 PowerCenter Client 目录中。 使用可创建 XML 文件的任何工具。 创建存储库插件文件时，为文件提供一个新名称。

下图显示了 plugin.dtd 的结构：

```
POWERMART
├── REPOSITORY
│   ├── PLUGIN
│   │   ├── FUNCTION_GROUP
│   │   │   ├── FUNCTION
│   │   │   └── LIBRARY
```

PLUGIN 元素

使用 PLUGIN 元素为想要创建的插件定义元数据。 PLUGIN 元素的属性唯一标识插件元数据。

下表显示了 PLUGIN 元素的属性：

属性	必需/ 可选	说明
名称	必需	插件的名称。插件名称显示在 PowerCenter 存储库服务的“插件”选项卡上。
ID	必需	插件的 ID。用于区分使用同一 VENDORID 开发的插件。
VENDORNAME	必需	供应商的名称。供应商名称显示在 PowerCenter 存储库服务的“插件”选项卡上。
VENDORID	必需	供应商 ID。如果您在开发要在贵组织之外分发的自定义函数，请从 Informatica 获取一个供应商 ID。有关更多信息，请参阅 “步骤 1. 获取存储库 ID 属性” 页面上 234。
DESCRIPTION	可选	插件的说明。插件说明显示在 PowerCenter 存储库服务的“插件”选项卡上。
VERSION	必需	插件的版本。用于跟踪插件元数据的更新。

FUNCTION_GROUP 元素

使用 FUNCTION_GROUP 元素定义自定义函数所属的组。

下表显示了 FUNCTION_GROUP 元素的属性：

属性	必需/ 可选	说明
名称	必需	要定义的自定义函数组的名称。函数组名称显示在 PowerCenter 存储库服务的“插件”选项卡上。
ID	必需	函数组的 ID。如果您在开发要在贵组织之外分发的自定义函数，请从 Informatica 获取一个函数组 ID。有关更多信息，请参阅 “步骤 1. 获取存储库 ID 属性” 页面上 234。 函数组 ID 显示在 PowerCenter 存储库服务的“插件”选项卡上。
COMPONENTVERSION	必需	函数组的版本号。这跟踪 FUNCTION_GROUP 元素的更新。
DESCRIPTION	可选	函数组的说明。函数组说明显示在 PowerCenter 存储库服务的“插件”选项卡上。
NAMESPACE	必需	函数组的命名空间。表达式编辑器在“函数”选项卡的单独文件夹中显示有命名空间的自定义函数。 命名空间不区分大小写。不能使用命名空间“infa”。它已保留。此外，命名空间不得为空。

确定命名空间

您可为自己创建的所有函数选择一个命名空间。然而，命名空间不能与其他供应商开发的自定义函数的命名空间发生冲突。因此，请选择一个唯一的命名空间。例如，您可选择与贵公司的名称相关的命名空间，例如，其股票符号。

FUNCTION 元素

使用 FUNCTION 元素定义自定义函数的属性。

下表显示了 FUNCTION 元素的属性：

属性	必需/ 可选	说明
名称	必需	要定义的第三方函数的名称。
ID	必需	FUNCTION 元素的 ID。标识函数。 如果您在开发要在贵组织之外分发的自定义函数，请从 Informatica 获取一个函数 ID。有关更多信息，请参阅 步骤 1. 获取存储库 ID 属性 页面上 234。
FUNCTION_CATEGORY	可选	要定义的函数的类别。使用以下类别之一： <ul style="list-style-type: none">- 字符- 转换- 数据清理- 日期- 数值- 科学计数- 特殊- 测试 表达式编辑器显示此类别下的自定义函数。

LIBRARY 元素

使用 LIBRARY 元素为自定义函数指定编译的共享库。

下表显示了 LIBRARY 元素的属性：

属性	必需/ 可选	说明
名称	必需	已编译共享库的名称。
OSTYPE	必需	要为其编译共享库的操作系统。
类型	必需	共享库的类型。请指定以下类型之一： <ul style="list-style-type: none">- VALIDATION. PowerCenter Client 用于检索自定义函数说明并验证函数调用的库，例如，返回类型和参数数量。- SERVER. PowerCenter 集成服务用于执行函数调用的库。

示例插件 XML 文件

以下示例显示定义了 ECHO 自定义函数的存储库插件文件：

```
<?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>

```

步骤 6.测试自定义函数

您可在开发期间测试自定义函数。完成以下任务以便在 PowerCenter 中测试自定义函数：

- 验证存储库插件 XML 文件。
- 验证表达式中的自定义函数是否能产生准确的数据。

要测试自定义函数，您必须在 PowerCenter 环境中安装自定义函数。

验证存储库插件文件

您可验证存储库插件文件，只需在 PowerCenter 存储库中注册它。注册插件文件时，称为 plugin.dtd 的关联 DTD 文件验证文件的结构。文件必须符合关联 plugin.dtd 的结构。plugin.dtd 位于 PowerCenter Client 目录中。

开发自定义函数时，可创建存储库插件文件并注册它，然后再完成函数创建表头文件和实现文件。注册文件时，可添加自定义函数元数据（例如，插件 ID）、命名空间和函数名称。这将此信息保留在存储库中。

注册存储库插件文件后，即可继续开发自定义函数。开发函数完毕，请再次注册存储库插件文件，以更新存储库的自定义函数元数据。

注册存储库插件之后，即可查看插件元数据。

查看插件元数据详细信息

下表显示了 Informatica Administrator 在“插件”选项卡上显示的元数据：

存储库服务属性	XML 元素和属性
名称	PLUGIN NAME
供应商名称	PLUGIN VENDORNAME

存储库服务属性	XML 元素和属性
说明	PLUGIN DESCRIPTION
组名称	FUNCTION_GROUP NAME
组 ID	FUNCTION_GROUP ID
组说明	FUNCTION_GROUP DESCRIPTION

验证函数准确性

要验证自定义函数的准确性，请用函数创建一个表达式，并将其包括在映射和工作流中。完成以下步骤，以验证自定义函数的准确性：

1. 创建测试数据。
2. 创建映射。
3. 将自定义函数添加至映射中的表达式。
4. 创建映射。
5. 运行调试器（可选）。或者，为映射创建会话和工作流。
6. 运行工作流。
7. 查看结果。

安装自定义函数

完成以下步骤以安装自定义函数：

1. 将自定义函数库复制至 PowerCenter 环境。
2. 注册存储库插件。

安装自定义函数之后，请将其用于转换表达式和工作流表达式。

步骤 1.将自定义函数库复制至 PowerCenter

根据自定义函数开发人员的指示将自定义函数库和存储库插件 XML 文件复制至 PowerCenter Client 和集成服务目录。

如果您需要经常使用这些库或在网格上运行会话，请将库放在单一位置，并将此位置定义为必需资源。

步骤 2.注册插件

从管理员工具注册存储库插件 XML 文件。

用自定义函数创建表达式

您可将自定义函数添加至表达式。如果在手动创建表达式时输入自定义函数，则必须将自定义函数开发人员提供的命名空间作为用户定义函数的前缀。当您使用表达式编辑器创建表达式时，自定义函数显示在所有函数及其函数类型的列表中。如同使用任何其他函数一样使用自定义函数。

验证表达式时，Designer 或工作流管理器不验证自定义函数。它们仅验证表达式。插件验证自定义函数。

自定义函数 API 引用

自定义函数 API 引用概览

使用自定义函数 API 开发可包括在转换或工作流表达式中的自定义函数。自定义函数 API 是用于创建自定义函数的框架。它包括常用和运行时 API。API 可使 PowerCenter 验证带有自定义函数的表达式，并在工作流中使用表达式。

在表头和实现文件中使用 API 开发自定义函数。必须使用表头和实现文件构建共享库。您在 PowerCenter 中注册了存储库插件文件，在该文件中指定了共享库。您也将共享库复制到 PowerCenter 环境。

常用 API

PowerCenter 客户端、集成服务和存储库服务调用常用 API 验证表达式、在使用之后从内存删除函数返回并在使用之后从内存删除函数说明和原型。

常用 API 包含以下结构：

```
INFA_EXPR_VALIDATE_METHODS
├── INFA_EXPR_ValidateGetUserInterface()
│   ├── validateFunction
│   ├── getFunctionDescription
│   └── getFunctionPrototype
INFA_EXPR_OPD_METADATA
INFA_EXPR_GetPluginVersion
INFA_EXPR_DestroyString
```

验证句柄

INFA_EXPR_VALIDATE_METHODS 句柄是验证句柄。PowerCenter 调用 INFA_EXPR_ValidateGetUserInterface 以在此验证句柄中获取函数指针。

用户界面验证函数

当 PowerCenter 调用 INFA_EXPR_ValidateGetUserInterface 时，插件返回验证函数的函数指针。

请使用以下语法：

```
INFA_EXPR_STATUS INFA_EXPR_ValidateGetUserInterface( IUNICHAR* sNamespace, IUNICHAR* sFuncName,  
INFA_EXPR_VALIDATE_METHODS* functions);
```

参数	数据类型	输入/ 输出	说明
sNamespace	IUNICHAR	输入	自定义函数的命名空间。
sFuncName	IUNICHAR	输入	自定义函数的名称。
函数	INFA_EXPR_VALIDATE_METHODS	输出	在验证和报告期间调用的不同函数的指针。

返回数据类型为 INFA_EXPR_STATUS。使用 ISUCCESS 和 IFailure 作为返回值。当函数返回 IFailure 时，插件不实现函数或出现的另一个错误。

INFA_EXPR_ValidateGetUserInterface 返回以下函数：

- **validateFunction.** 验证自定义函数。
- **getFunctionDescription.** 描述自定义函数。
- **getFunctionPrototype.** 提供自定义函数的原型。
- **pushdownFunction.** 为下推优化生成 SQL 代码。

自定义函数验证函数

PowerCenter 调用 validateFunction 以验证自定义函数中的参数。它使用此函数为自定义函数中的参数提供名称、数据类型、精度和小数位。它也使用此函数提供自定义函数返回值的数据类型。

PowerCenter 为映射或工作流中所用的自定义函数的每个实例调用一次此函数。

请使用以下语法：

```
INFA_EXPR_STATUS *(validateFunction)(IUNICHAR* sNamespace, IUNICHAR* sFuncName, IUINT32 numArgs,  
INFA_EXPR_OPD_METADATA** inputArgList, INFA_EXPR_OPD_METADATA* retValue);
```

参数	数据类型	输入/ 输出	说明
sNamespace	IUNICHAR	输入	函数的命名空间。
sFuncName	IUNICHAR	输入	要验证的自定义函数的名称。
numArgs	IUINT32	输入	自定义函数中参数的名称。
inputArgList	INFA_EXPR_OPD_METADATA	输入	自定义函数的输入参数。
retValue	INFA_EXPR_OPD_METADATA	输出	自定义函数返回端口的元数据。设置此参数中返回值的数据类型、精度和小数位数。

返回数据类型为 INFA_EXPR_STATUS。使用 ISUCCESS 和 IFailure 作为返回值。当函数返回 IFailure 时，PowerCenter 显示一条错误消息。

自定义函数说明函数

PowerCenter 调用 `getFunctionDescription` 以获取自定义函数的说明。它调用 `destroyString` 以便在使用完毕从内存中删除说明。

请使用以下语法：

```
IUNICHAR* *(getFunctionDescription) (IUNICHAR* sNamespace, IUNICHAR* sFuncName);
```

参数	数据类型	输入/输出	说明
sNamespace	IUNICHAR	输入	函数的命名空间。
sFuncName	IUNICHAR	输入	插件应描述的自定义函数的名称。

返回数据类型为 IUNICHAR。返回值必须为空值终止的字符串。

自定义函数原型函数

PowerCenter 调用 `getFunctionPrototype`，以便在表达式编辑器中获取自定义函数的参数。它调用 `destroyString`，以便在完成使用时从内存中删除参数。

请使用以下语法：

```
IUNICHAR* *(getFunctionPrototype) (IUNICHAR* sNamespace, IUNICHAR* sFuncName);
```

参数	数据类型	输入/输出	说明
sNamespace	IUNICHAR	输入	函数的命名空间。
sFuncName	IUNICHAR	输入	插件应描述的自定义函数的名称。

返回数据类型为 IUNICHAR。返回值必须为空值终止的字符串。当参数没有值时，函数返回 NULL。

自定义函数下推函数

PowerCenter 调用 `pushdownFunction`，以便为下推优化生成 SQL 代码。

请使用以下语法：

```
INFA_EXPR_STATUS pushdownFunctionEcho(IUNICHAR* sNameSpace,  
                                       IUNICHAR* sFuncName,  
                                       IUINT32 numArgs,  
                                       INFA_EXPR_OPD_METADATA** inputArgList,  
                                       EDatabaseType dbType,  
                                       EPushdownMode pushdownMode,  
                                       IUNICHAR** sGenSql)
```

参数	数据类型	输入/输出	说明
sNameSpace	IUNICHAR	输入	函数的命名空间。
sFuncName	IUNICHAR	输入	要验证的自定义函数的名称。

参数	数据类型	输入/输出	说明
numArgs	IUNINT32	输入	自定义函数中参数的名称。
inputArgList	INFA_EXPR_OPD_METADATA	输入	自定义函数的输入参数。
dbType	EDatabaseType	输入	数据库类型。
pushdownMode	EPushdownMode	输入	下推优化的类型。
sGenSql	IUNICHAR	输出	自定义函数生成的 SQL。

返回数据类型为 INFA_EXPR_STATUS。使用 ISUCCESS 和 IFAILURE 作为返回值。当函数返回 IFAILURE 时，PowerCenter 显示一条错误消息。

INFA_EXPR_OPD_METADATA 结构

此函数定义表达式操作数的元数据，包括传递至函数和返回值的参数。

结构包含以下元数据：

- **数据类型。** 参数的数据类型。
- **精度。** 参数的精度。
- **小数位数。** 参数的小数位数。
- **isValueConstant.** 指示参数是否为常量。如果是，框架为每次函数调用计算一次参数。框架使用 isValueConstant 优化性能。对于为常量的输入参数，插件可在函数实体初始化期间获取参数以优化性能。对于输出值，插件将 isValueConstant 设置为 TRUE。

获取插件版本函数

此函数定义插件的版本。它必须与自定义函数 API 版本 1.0.0 相同。

请使用以下语法：

```
INFA_EXPR_STATUS INFA_EXPR_GetPluginVersion(INFA_VERSION *sdkVersion, INFA_VERSION *pluginVersion);
```

参数	数据类型	输入/输出	说明
sdkVersion	INFA_VERSION	输入	自定义函数 API 的版本。使用 1.0.0。
pluginVersion	INFA_VERSION	输出	要创建的插件版本。

返回数据类型为 INFA_EXPR_STATUS。使用 ISUCCESS 和 IFAILURE 作为返回值。如果函数返回 IFAILURE，会话或工作流失败。

销毁字符串函数

此函数销毁插件返回的所有字符串。例如，它销毁错误消息或其他函数调用的返回值，例如，getFunctionDescription。

请使用以下语法：

```
void *(DestroyString)(IUNICHAR* str);
```

参数	数据类型	输入/输出	说明
str	IUNICHAR	输入	此函数删除的输入字符串。

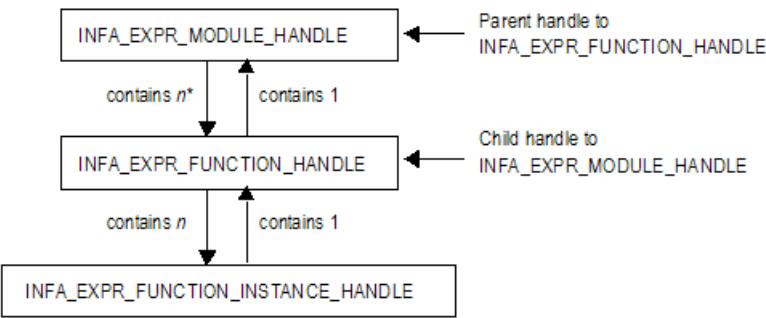
函数不返回值。

运行时 API

PowerCenter 集成服务在会话期间调用运行时，以计算包含自定义函数的表达式。它初始化模块、函数和函数实例级别的插件。

每个级别均包含一个函数集。这些函数与句柄相关联，例如，INFA_EXPR_MODULE_HANDLE。这些函数的第一个参数是函数影响的句柄。自定义函数 API 句柄互相有层次结构关系。父句柄与其子句柄的有 1:n 关系。

下图显示了自定义函数 API 句柄：



下表描述了运行时句柄：

句柄名称	说明
INFA_EXPR_MODULE_HANDLE	代表共享库或 DLL。插件只能访问其自己共享库或 DLL 中的模块句柄。它不能访问任何其他共享库或 DLL 中的模块句柄。
INFA_EXPR_FUNCTION_HANDLE	代表共享库或 DLL 内的自定义函数。
INFA_EXPR_FUNCTION_INSTANCE_HANDLE	代表特定自定义函数实例。

模块级别函数

PowerCenter 为每个共享库或 DLL 调用一次模块级别函数。

获取用户界面模块级别函数

此函数为模块级别交互设置函数指针。

请使用以下语法：

```
INFA_EXPR_STATUS INFA_EXPR_ModuleGetUserInterface(INFA_EXPR_LIB_METHODS* functions);
```

参数	数据类型	输入/ 输出	说明
函数	INFA_EXPR_LIB_METHODS	输出	模块获取用户界面函数。

返回数据类型为 INFA_EXPR_STATUS。使用 ISUCCESS 和 IFailure 作为返回值。如果函数返回 IFailure，会话或工作流失败。

此函数返回以下函数：

- **function_init**. 初始化函数。
- **function_deinit**. 取消初始化函数。

模块级别初始化函数

PowerCenter 为每个模块调用一次此 module_init，以初始化函数中的任何全局数据结构。它先调用此函数，然后再调用任何函数级别函数。

请使用以下语法：

```
INFA_EXPR_STATUS (*module_init) (INFA_EXPR_MODULE_HANDLE module);
```

参数	数据类型	输入/ 输出	说明
模块	INFA_EXPR_MODULE_HANDLE	输入	存储它可在函数级别检索的数据。

返回数据类型为 INFA_EXPR_STATUS。使用 ISUCCESS 和 IFailure 作为返回值。如果函数返回 IFailure，会话或工作流失败。

模块级别取消初始化函数

PowerCenter 为每个模块调用一次 module_deinit，以取消初始化此函数中的任何数据结构。它在所有函数级别交互完成之后，调用此函数。

请使用以下语法：

```
INFA_EXPR_STATUS (*module_deinit) (INFA_EXPR_MODULE_HANDLE module);
```

参数	数据类型	输入/ 输出	说明
模块	INFA_EXPR_MODULE_HANDLE	输入	调用模块初始化函数时框架向插件传递的模块级别句柄。

返回数据类型为 INFA_EXPR_STATUS。使用 ISUCCESS 和 IFailure 作为返回值。如果函数返回 IFailure，会话或工作流失败。

函数级别函数

PowerCenter 为每个自定义函数调用函数级别函数一次，为提供自定义函数参数的每个共享库或 DLL 调用一次。

获取用户界面函数级别函数

此函数为函数级别交互设置函数指针。PowerCenter 为此库实现的每个自定义函数调用此函数。

请使用以下语法：

```
INFA_EXPR_STATUS INFA_EXPR_FunctionGetUserInterface (IUNICHAR* nameSpaceName, IUNICHAR* functionName,
INFA_EXPR_FUNCTION_METHODS* functions);
```

参数	数据类型	输入/输出	说明
nameSpaceName	IUNICHAR	输入	函数的命名空间。
functionName	IUNICHAR	输入	插件应描述的自定义函数的名称。
函数	INFA_EXPR_FUNCTION_METHODS	输入	要在函数实例级别调用的函数指针的占位符。

返回数据类型为 INFA_EXPR_STATUS。使用 ISUCCESS 和 IFailure 作为返回值。如果函数返回 IFailure，会话或工作流失败。

此函数返回以下函数：

- **function_init**. 初始化函数。
- **function_deinit**. 取消初始化函数。

函数级别初始化函数

PowerCenter 为每个自定义函数调用一次 function_init，以初始化与自定义函数相关的任何结构。它先调用模块级别初始化函数，然后再调用此函数。

请使用以下语法：

```
INFA_EXPR_STATUS (*function_init) (INFA_EXPR_FUNCTION_HANDLE fnInstance);
```

参数	数据类型	输入/输出	说明
fnInstance	INFA_EXPR_FUNCTION_HANDLE	输入	执行以下任务： <ul style="list-style-type: none">- 存储用户定义的指针，以供框架在运行时或取消初始化期间检索。- 为函数实例级别初始化数据结构。- 如果输入参数为常量，则插件检索此常量值，并执行任何所需的重新处理。

返回数据类型为 INFA_EXPR_STATUS。使用 ISUCCESS 和 IFailure 作为返回值。如果函数返回 IFailure，会话或工作流失败。

函数级别取消初始化函数

PowerCenter 为每个函数级别调用此函数一次，以取消初始化与自定义函数相关的任何结构。

请使用以下语法：

```
INFA_EXPR_STATUS (*function_deinit) (INFA_EXPR_FUNCTION_HANDLE function);
```

参数	数据类型	输入/输出	说明
fnInstance	INFA_EXPR_FUNCTION_HANDLE	输入	调用函数实例级别初始化函数时框架向插件传递的函数级别句柄。

返回数据类型为 INFA_EXPR_STATUS。使用 ISUCCESS 和 IFailure 作为返回值。如果函数返回 IFailure，会话或工作流失败。

函数实例级别函数

每当自定义函数用于映射或工作流时，PowerCenter 均调用这些函数。

获取用户界面函数级别函数

此函数为函数级别交互设置函数指针。PowerCenter 为此库实现的每个自定义函数调用此函数。

请使用以下语法：

```
INFA_EXPR_STATUS INFA_EXPR_FunctionInstanceGetUserInterface(IUNICHAR* functionName,  
INFA_EXPR_FUNCTION_INSTANCE_METHODS* functions)
```

参数	数据类型	输入/输出	说明
functionName	IUNICHAR	输入	函数的命名空间。
函数	INFA_EXPR_FUNCTION_INSTANCE_METHODS	输入	要在函数实例级别调用的函数指针的占位符。

此函数返回以下函数：

- **fnInstance_init.** 初始化自定义函数的实例。
- **fnInstance_processRow.** 为自定义函数的实例处理数据。
- **fnInstance_deinit.** 取消初始化自定义函数的实例。

函数实例级别初始化函数

PowerCenter 为每个自定义函数实例调用一次 fnInstance_init，以初始化与自定义函数实例相关的任何结构。如果映射或工作流中在自定义函数的两个实例，则 PowerCenter 调用此函数两次。PowerCenter 先调用模块级别初始化函数，然后再调用此函数。

请使用以下语法：

```
INFA_EXPR_STATUS (*fnInstance_init)(INFA_EXPR_FUNCTION_INSTANCE_HANDLE fnInstance);
```

参数	数据类型	输入/输出	说明
fnInstance	INFA_EXPR_FUNCTION_HANDLE	输入	执行以下任务： <ul style="list-style-type: none">- 存储用户定义的指针，以供框架在运行时或取消初始化期间检索。- 为函数实例级别初始化数据结构。- 如果输入参数为常量，则插件检索此常量值，并执行任何所需的重新处理。

返回数据类型为 INFA_EXPR_STATUS。使用 ISUCCESS 和 IFailure 作为返回值。如果函数返回 IFailure，会话或工作流失败。

函数实例行处理函数

当输入流可用于自定义函数实例时，PowerCenter 调用此 fnInstance_processRow。自定义函数输入参数的数据通过 fnInstance-inputOPDHandles 绑字和访问。为 fnInstance->retHandle 中的输出和返回端口设置数据、长度和指示符。PowerCenter 先调用函数级别初始化函数，然后再调用此函数。

请使用以下语法：

```
INFA_EXPR_ROWSTATUS (*fnInstance_processRow) (INFA_EXPR_FUNCTION_INSTANCE_HANDLE fnInstance);
```

参数	数据类型	输入/输出	说明
fnInstance	INFA_EXPR_FUNCTION_HANDLE	输入	数据可用于的函数级别句柄。

返回值的数据类型为 INFA_EXPR_ROWSTATUS。将以下值用于返回值：

- **INFA_ROWSUCCESS.** 指示成功处理数据行的函数。
- **INFA_ROWERROR.** 指示遇到数据行错误的函数。PowerCenter 集成服务增加内部错误计数。当数据访问模式为行时仅返回此值。
- **INFA_FATALERROR.** 指示遇到数据行或数据块致命错误的函数。PowerCenter 集成服务处理会话失败。

函数实例级别取消初始化函数

PowerCenter 在取消初始化期间为每个函数级别调用一次 fnInstance_deinit。它可调用此函数，以取消初始化与自定义函数相关的任何结构。

请使用以下语法：

```
INFA_EXPR_STATUS (*fnInstance_deinit)(INFA_EXPR_FUNCTION_INSTANCE_HANDLE fnInstance);
```

参数	数据类型	输入/输出	说明
fnInstance	INFA_EXPR_FUNCTION_INSTANCE_HANDLE	输入	调用函数实例级别初始化函数时框架向插件传递的函数级别句柄。

返回数据类型为 INFA_EXPR_STATUS。使用 ISUCCESS 和 IFailure 作为返回值。如果函数返回 IFailure，会话或工作流失败。