Hewlett Packard Enterprise

SQL/MX 3.6 SQL/MX 手続き型言語 (PL/MX) リファレンスマニュアル

部品番号: 875223-193
発行: 2018 年 4 月
版数: L18.02 およびそれ以降のすべての L シリーズ RVU

ご注意

本書の内容は、将来予告なしに変更されることがあります。Hewlett Packard Enterprise 製品およびサービスに対する保証については、当該製品およびサービスの保証規定書に記載されています。本書のいかなる内容も、新たな保証を追加するものではありません。本書の内容につきましては万全を期しておりますが、本書中の技術的あるいは校正上の誤り、脱落に対して、責任を負いかねますのでご了承ください。

本書で取り扱っているコンピューターソフトウェアは秘密情報であり、その保有、使用、または複製には、Hewlett Packard Enterprise から使用許諾を得る必要があります。FAR 12.211 および 12.212 に従って、商業用コンピューターソフトウェア、コンピューターソフトウェアドキュメンテーション、および商業用製品の技術データ（Commercial Computer Software, Computer Software Documentation, and Technical Data for Commercial Items）は、ベンダー標準の商業用使用許諾のもとで、米国政府に使用許諾が付与されます。

他社の Web サイトへのリンクは、Hewlett Packard Enterprise の Web サイトの外に移動します。Hewlett Packard Enterprise は、Hewlett Packard Enterprise の Web サイト以外にある情報を管理する権限を持たず、また責任を負いません。

商標

Microsoft®および Windows®は、米国および/またはその他の国における Microsoft Corporation の登録商標または商標です。

Intel®、インテル、Itanium®、Pentium®、Intel Inside®、および Intel Inside ロゴは、インテルコーポレーションまたはその子会社のアメリカ合衆国およびその他の国における商標または登録商標です。

Adobe®および Acrobat®は、米国 Adobe Systems Incorporated の登録商標です。

UNIX®は、The Open Group の登録商標です。

Java®および Oracle®は、Oracle および/またはその関連会社の登録商標です。

Open Software Foundation、OSF、OSF ロゴ、OSF/1、OSF/Motif、および Motif は、Open Software Foundation, Inc.の商標です。

保証

Open Software Foundation、OSF、OSF ロゴ、OSF/1、OSF/Motif、および Motif は、Open Software Foundation, Inc.の商標です。

OSF MAKES NO WARRANTY OF ANY KIND WITH REGARD TO THE OSF MATERIAL PROVIDED HEREIN, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

OSF shall not be liable for errors contained herein or for incidental consequential damages in connection with the furnishing, performance, or use of this material.

© 1990, 1991, 1992, 1993 Open Software Foundation, Inc. This documentation and the software to which it relates are derived in part from materials supplied by the following:

© 1987, 1988, 1989 Carnegie-Mellon University. © 1989, 1990, 1991 Digital Equipment Corporation. © 1985, 1988, 1989, 1990 Encore Computer Corporation. © 1988 Free Software Foundation, Inc. © 1987, 1988, 1989, 1990, 1991 Hewlett Packard Enterprise. © 1985, 1987, 1988, 1989, 1990, 1991, 1992 International Business Machines Corporation. © 1988, 1989 Massachusetts Institute of Technology. © 1988, 1989, 1990 Mentat Inc. © 1988 Microsoft Corporation. © 1987, 1988, 1989, 1990, 1991, 1992 SecureWare, Inc. © 1990, 1991 Siemens Nixdorf Informationssysteme AG. © 1986, 1989, 1996, 1997 Sun Microsystems, Inc. © 1989, 1990, 1991 Transarc Corporation.

目次

このドキュメントについて

このマニュアルでは PL/MX について説明します。PL/MX は、Oracle PL/SQL や ANSI SQL/PSM と多くの点で類似した NonStop SQL/MX の手続き型言語です。これは NonStop SQL/MX でのユーザー定義ルーチン (UDR) 向けの実装言語です。

サポートされているリリースバージョンの更新 (RVU)

本書では、L18.02 以降の L シリーズ RVU をすべてサポートしています。

対象読者

このマニュアルは、NonStop のアプリケーション開発者および一般ユーザーを対象としています。

新規情報と更新内容

「875223-003 マニュアル」に対する変更点

- 「**組み込み関数**」、「**例外処理**」、および「**PL/MX 内部の SQL 文**」の章を追加しました。
- 現在の内容を更新して、手順、ラベル、カーソル、および「**宣言文**」内の可視性ルールを追加しました。
- 現在の内容を更新して、**文**内の DBMS_OUTPUT、GOTO、RAISE、および RETURN に関する情報を追加しました。
- 現在の内容を更新して、**式**内の CASE、連結、および制約に関する情報を追加しました。
- **スタンドアロンコンパイラ**、**Datatypes**、および**言語エレメント**を更新しました。

関連ドキュメント

入門ガイド

SQL/MX Comparison Guide for SQL/MP Users

NonStop SQL/MP と NonStop SQL/MX の SQL の違いについて説明しています。

SQL/MX Quick Start

SQL/MX 会話型インターフェイス（MXCI）で SQL を使用するための基本的な方法について説明しています。サンプルデータベースのインストールについても説明しています。

リファレンスマニュアル

SQL/MX Reference Manual

SQL/MX 文、MXCI コマンド、関数、その他の SQL/MX 言語要素の構文について説明しています。

SQL/MX Messages Manual

SQL/MX のメッセージについて説明しています。

SQL/MX Database Services Manual

マルチテナント環境でのユーザーデータベースのプロビジョニングについて説明しています。mxdbs コマンドラインユーティリティを使用してユーザーデータベースを作成および管理する方法についても説明しています。

MXDM User Guide for SQL/MX

HPE NonStop SQL/MX Database Manager を使用して SQL/MX データベースを監視および管理する方法について説明しています。

SQL/MX Workload Management Services (WMS) Reference Manual

SQL/MX 3.6 ワークロード管理サービス（WMS）の設定と構成、および NonStop システムのワークロードの監視方法について説明しています。

SQL/MX Glossary

SQL/MX の用語を定義しています。

インストールおよび移行ガイド

SQL/MX Installation and Upgrade Guide

SQL/MX データベースのインストールとアップグレードの計画方法について説明しています。

NonStop NS-Series Database Migration Guide

NonStop SQL/MX、NonStop SQL/MP、Enscribe のデータベースとアプリケーションを HPE Integrity NonStop NS シリーズのシステムに移行する方法について説明しています。

NonStop SQL/MP to SQL/MX Database and Application Migration Guide

データベースとアプリケーションを SQL/MP から SQL/MX に移行する方法について説明しています。

接続に関するマニュアル

SQL/MX Connectivity Service Manual

HPE NonStop SQL/MX 接続サービス（MXCS）をインストールして管理する方法について説明しています。MXCS を使用すると、Microsoft Open Database Connectivity（ODBC）アプリケーションプログラミングインターフェイス（API）やその他の接続 API 向けに開発されたアプリケーションで NonStop SQL/MX を利用できるようになります。

SQL/MX Connectivity Service Administrative Command Reference

SQL/MX 会話型インターフェイス（MXCI）で使用できる SQL/MX 管理コマンドライブラリ（MACL）について説明しています。

ODBC/MX Driver for Windows

Microsoft Windows 用に HPE NonStop ODBC/MX をインストールして構成する方法について説明しています。ODBC/MX を使用すると、ODBC API 向けに開発されたアプリケーションで NonStop SQL/MX を利用できるようになります。

ODBC/MX Client Drivers User Guide for SQL/MX

ODBC/MX クライアントドライバーをインストール、構成、使用する方法について説明しています。

JDBC Type 2 Driver Programmer's Reference for SQL/MX

NonStop SQL/MX 用の JDBC タイプ 2 ドライバーを使用する方法について説明しています。

JDBC Type 4 Driver Programmer's Reference for SQL/MX

NonStop SQL/MX 用の JDBC タイプ 4 ドライバーを使用する方法について説明しています。

データ管理ガイド

SQL/MX Management Manual

SQL/MX データベースの管理方法について説明しています。

SQL/MX Data Mining Guide

SQL/MX のデータ構造と、ナレッジ発見のプロセスを実行するための操作について説明しています。

SQL/MX Report Writer Guide

SQL/MX データベースからのデータを使用して書式設定されたレポートを作成する方法について説明しています。

DataLoader/MX Reference Manual

SQL/MX データベースをロードするためのツールである DataLoader/MX 製品の特徴と機能について説明しています。

アプリケーション開発ガイド

SQL/MX プログラミングマニュアル C および COBOL 言語用

SQL/MX 文を ANSI C および COBOL プログラムに埋め込む方法について説明しています。

SQL/MX Query Guide

クエリ実行プランを理解し、SQL/MX データベースに最適なクエリを記述する方法について説明しています。

SQL/MX Queuing and Publish/Subscribe Services

NonStop SQL/MX で、トランザクションキューイングおよびパブリッシュ/サブスクライブサービスをそのデータベースインフラストラクチャに統合するしくみについて説明しています。

SQL/MX Guide to Stored Procedures in Java

Java によって記述されるストアドプロシージャーを NonStop SQL/MX 内で使用する方法について説明しています。

オンラインヘルプ

SQL/MX Messages Online Help

SQL/MX Messages Manual の個々のメッセージをソース別に分類しています。

SQL/MX Glossary Online Help

SQL/MX Glossary の用語および定義で構成されています。

SQL/MX Database Manager Help

MXDM User Guide for SQL/MX のオンラインヘルプバージョンです。

MXCI Online Help

`SQL/MX Reference Manual` の SQL/MX 文と MXCI コマンドの構文について説明しています。

発行履歴

部品番号	製品のバージョン	発行
875223-003	NonStop SQL/MX 3.6	2018 年 3 月

PL/MX について

PL/MX は、Oracle PL/SQL や ANSI SQL/PSM と多くの点で類似した NonStop SQL/MX の手続き型言語です。これは NonStop SQL/MX でのユーザー定義ルーチン (UDR) 向けの実装言語です。

PL/MX 言語

PL/MX 言語には、現代の多くのプログラミング言語と同じように、変数や定数の宣言、文、条件付きコンパイル、および関数ための構造が含まれています。

変数宣言と定数宣言

PL/MX の変数宣言と定数宣言は、文字長で最大 30 半角文字までの識別子により、ストレージおよび値を記号的に関連付ける 1 つの手段を提供します。定数は特定の値に初期化されてから、そのスコープ内部で不変のままになります。

文

PL/MX 文には割り当て文、制御文、および出力文が含まれています。

サブプログラム

用語、サブプログラムは、PL/MX 言語で記述された 手順または関数に対して参照します。PL/MX 関数は、どのユーザー定義関数 (UDF) が定義されるかのための手段です。その他のプログラミング言語と同様に、サポートされているデータ型のパラメータを受け取ることができます。加えて、PL/MX 関数は値を返すことができます。NonStop SQL/MX データベースは PL/MX をユーザー定義ルーチン (UDR)、UDF、またはストアドプロシージャとして参照することができます。

注記: PL/MX の目標とは Oracle PL/SQL との互換性確保です。しかしながら、一部のケースで両者の間には偏差が存在しています。これらの相違点については、「**PL/MX と Oracle PL/SQL の違い**」を参照してください。

PL/MX 環境

PL/MX 開発環境

PL/MX 関数はテキストエディタを使用して作成され、CREATE FUNCTION 文を使用して、コンパイルおよび NonStop SQL/MX データベースへの格納のために送信されます。また、PL/MX コンパイラ (PLMXCMP) をスタンドアロンモードで実行して、PL/MX モジュールの構文をチェックすることもできます。スタンドアロンのコンパイルモードについて詳しくは、**スタンドアロンコンパイラー**を参照してください。

PL/MX 実行環境

PL/MX UDR が正常に作成された後、これらは SQL クエリで使用されることができます。UDR サーバーは、クエリの実行中にそれが最初に検出した各 UDR の実行可能ファイル (オブジェクトコード) を読み込みます。この UDR の初回およびそれ以降の呼び出し時に、NonStop SQL/MX エグゼキューターは、SQL 仮想マシン (SVM) 内の UDR コードを実行する UDR サーバーに対して必要な任意のパラメータを渡します。各 UDR 呼び出しが完了するにつれ、UDR サーバーは、任意の関数戻り値や出力値をクエリ データフローに取り込む NonStop SQL/MX エグゼキューターに返します。

言語要素

文字セット

PL/MX は、大文字と小文字のアルファベット文字、0〜9 の数字、および特殊文字を含む ASCII 文字セットをサポートしています。

予約語

次の表に、PL/MX の予約語を示します。予約語は、二重引用符で囲まれていないかぎり、PL/MX の識別子には使用できません。

ACTION	FOR	PUBLIC
ADD	FOREIGN	READ
ADMIN	FOUND	READS
AFTER	FRACTION	REAL
AGGREGATE	FREE	RECURSIVE
ALIAS	FROM	REF
ALL	FULL	REFERENCES
ALLOCATE	FUNCTION	REFERENCING
ALTER	GENERAL	RELATIVE
AND	GET	REPLACE
ANY	GLOBAL	RESIGNAL
ARE	GO	RESOURCE
ARRAY	GOTO	RESTRICT
AS	GRANT	RESULT
ASC	GROUP	RETURN
ASSERTION	GROUPING	RETURNS
ASYNC	HAVING	REVOKE
AT	HOST	RIGHT
AUTHORIZATION	HOUR	ROLE
AVG	IDENTIFIED	ROLLBACK
BEFORE	IDENTITY	ROLLUP
BEGIN	IF	ROUTINE
BETWEEN	IGNORE	ROW
BINARY	IMMEDIATE	ROWS
BIT	IN	SAVEPOINT

表は続く

BIT_LENGTH	INDEX	SCHEMA
BLOB	INDEXES	SCOPE
BOOLEAN	INDICATOR	SCROLL
BOTH	INITIALLY	SEARCH
BREADTH	INNER	SECOND
BY	INOUT	SECTION
CALL	INPUT	SELECT
CASE	INSENSITIVE	SENSITIVE
CASCADE	INSERT	SESSION
CASCADED	INT	SESSION_USER
CAST	INTEGER	SET
CATALOG	INTERSECT	SETS
CHAR	INTERVAL	SHARE
CHAR_LENGTH	INTO	SIGNAL
CHARACTER	IS	SIMILAR
CHARACTER_LENGTH	ISOLATION	SIZE
CHECK	ITERATE	SMALLINT
CLASS	JOIN	SOME
CLOB	KEY	SPECIFIC
CLOSE	LANGUAGE	SPECIFICTYPE
CLUSTER	LARGE	SQL
CLUSTERS	LAST	SQL_CHAR
COALESCE	LATERAL	SQL_DATE
COLAUTH	LEADING	SQL_DECIMAL
COLLATE	LEAVE	SQL_DOUBLE
COLLATION	LEFT	SQL_FLOAT
COLUMN	LESS	SQL_INT
COLUMNS	LEVEL	SQL_INTEGER
COMMIT	LIKE	SQL_REAL
COMPLETION	LIMIT	SQL_SMALLINT
COMPRESS	LOCAL	SQL_TIME
CONNECT	LOCALTIME	SQL_TIMESTAMP
CONNECTION	LOCALTIMESTAMP	SQL_VARCHAR
CONSTRAINT	LOCATOR	SQLCODE
CONSTRAINTS	LOCK	SQLERROR

表は続く

CONSTRUCTOR	LOOP	SQLEXCEPTION
CONTINUE	LOWER	SQLSTATE
CONVERT	MAP	SQLWARNING
CORRESPONDING	MATCH	START
COUNT	MAX	STRUCTURE
CRASH	MIN	SUBSTRING
CREATE	MINUS	SUBTYPE
CROSS	MINUTE	SUM
CUBE	MODE	SYSTEM_USER
CURRENT	MODIFIES	TABAUTH
CURRENT_DATE	MODIFY	TABLE
CURRENT_PATH	MODULE	TEMPORARY
CURRENT_ROLE	MONTH	TERMINATE
CURRENT_TIME	NAMES	TEST
CURRENT_TIMESTAMP	NATIONAL	THAN
CURRENT_USER	NATURAL	THEN
CURRVAL	NCHAR	THERE
CURSOR	NCLOB	TIME
CYCLE	NEW	TIMESTAMP
DATE	NEXT	TIMEZONE_HOUR
DATETIME	NEXTVAL	TIMEZONE_MINUTE
DAY	NO	TO
DEALLOCATE	NOCOMPRESS	TRAILING
DEC	NONE	TRANSACTION
DECIMAL	NOT	TRANSLATE
DECLARE	NOWAIT	TRANSLATION
DEFAULT	NULL	TRANSPOSE
DEFERRABLE	NULLIF	TREAT
DEFERRED	NUMERIC	TRIGGER
DELETE	OBJECT	TRIM
DEPTH	OCTET_LENGTH	TRUE
DEREF	OF	TYPE
DESC	OFF	UNDER
DESCRIBE	OID	UNION
DESCRIPTOR	OLD	UNIQUE

表は続く

DESTROY	ON	UNKNOWN
DESTRUCTOR	ONLY	UNNEST
DETERMINISTIC	OPEN	UPDATE
DIAGNOSTICS	OPERATORS	UPPER
DISTINCT	OPTION	UPSHIFT
DICTIONARY	OR	USAGE
DISCONNECT	ORDER	USER
DOMAIN	ORDINALITY	USING
DOUBLE	OTHERS	VALUE
DROP	OUT	VALUES
DYNAMIC	OUTER	VARCHAR
EACH	OUTPUT	VARIABLE
ELSE	OVERLAPS	VARYING
ELSEIF	PAD	VIEW
ELSIF	PARAMETER	VIEWS
END	PARAMETERS	VIRTUAL
ENDIS	PARTIAL	VISIBLE
END-EXEC	PENDANT	WAIT
EQUALS	POSITION	WHEN
ESCAPE	POSTFIX	WHENEVER
EXCEPT	PRECISION	WHERE
EXCEPTION	PREFIX	WHILE
EXEC	PREORDER	WITH
EXCLUSIVE	PREPARE	WITHOUT
EXECUTE	PRESERVE	WORK
EXISTS	PRIMARY	WRITE
EXTERNAL	PRIOR	YEAR
EXTRACT	PRIVATE	ZONE
FALSE	PRIVILEGES	
FETCH	PROCEDURE	
FIRST	PROTECTED	
FLOAT	PROTOTYPE	

区切り文字

PL/MX 内の区切り文字は、対象言語の構文要素を開始、終了、または分離する、特殊文字または特殊文字の組み合わせです。以下の表は、PL/MX の区切り文字をリストします。

注記: さらに、半角の左右角カッコなどの、その他の文字は PL/MX 内の SQL 文でも出現する場合があります。

表 1: PL/MX の区切り文字

区切り文字	意味
'(アポストロフィ)	文字列を開始または終了します
(	式を開始します
<<	ラベルを開始します
/*	複数行コメントを開始します
"	引用符付き識別子を開始または終了します
--	一行コメントを開始します
)	式を終了します
>>	ラベルを終了します
*/	複数行コメントを終了します
,	リスト項目セパレーター
..	範囲セパレーター
;	文のターミネータ

オペレーター

PL/MX 内のオペレーターは、算術または関係オペレーターを記述する特殊文字、特殊文字の組み合わせまたは予約語です。以下の表に、PL/MX オペレーターをリストします。

表 2: PL/MX オペレーター

オペレーター	意味
+	加算、算術同一性
:=	Cesión
=	イコール
/	除算
>	大なり
>=	大なりイコール
<	小なり
<=	小なりイコール
*	乗算
^=	ノットイコール
!=	ノットイコール
<>	ノットイコール

表は続く

~=	ノットイコール
||	文字列連結
-	減算/算術否定
AND	論理積
NOT	論理否定
OR	論理和
BETWEEN	X BETWEEN A AND B. Equivalent to `(X >= A) AND (X <= B)`
ISNULL	単項オペレーターは、オペランドの後方に指定します。その結果は BOOLEAN であり、そのオペランドが NULL である場合は、TRUE となる。
IS NOT NULL	単項オペレーターは、オペランドの後方に指定します。その結果は BOOLEAN であり、そのオペランドが NULL でない場合は、TRUE となる。

リテラル

PL/MX リテラルは、いくつかのタイプの定数値です。さらに、PL/MX には、すべてのタイプに対して適用される NULL リテラルもサポートしています。リテラルは、変数、およびその他の対象などに割り当てられた、式で使用されることができます。リテラルは、正確性に関してコンパイル時にチェックされます。文字列リテラルが、非文字列タイプに対して割り当てられるために使用される場合、実行時のその変換が完了するまでは、その文字列の内容はチェックされません。

表 3: PL/MX のリテラルの例

リテラルの型	リテラルの例
ブール値	FALSE TRUE
文字	'abc' '半角スペースを含む半角文字列'
整数値	654 2147483647
NUMERIC	65.4 6.02
DATE	DATE '2017-08-21'
TIME	TIME '14:02:02'
TIMESTAMP	TIMESTAMP '2017-08-21 14:02:02.001234'
任意のタイプ	NULL

宣言

宣言は、ストレージを割り当て、PL/MX 内で宣言可能なオブジェクトに識別子を関連付けます。

- **識別子**
- **変数**
- **定数**
- **関数**
- **手順**(18 ページ)
- **ラベル**
- **カーソル**(20 ページ)
- **可視性ルール**(20 ページ)

識別子

識別子は、最大 30 文字の英数字、ドル記号（$)、アンダースコア（_)、または数字記号（#）で構成され、文字で始まる必要があります。識別子は、二重引用符で囲まれていないかぎり、大文字と小文字を区別しません。たとえば、識別子 LASTNAME と LastName は同一ですが、識別子"LASTNAME"と"LastName"は同一ではありません。予約語は、二重引用符で囲まれていないかぎり、識別子として使用できません。たとえば、"AND"は有効な識別子であり、AND はそうではありません。二重引用符で囲まれた識別子には、ヌルと改行以外のあらゆる有効な ASCII 文字を含めることができます。

変数

PL/MX 変数は、ストレージと関連付けられる識別子です。PL/MX コンパイラは、特定のデータ型に対して正しいストレージを自動的に割り当てて管理します。変数識別子が割り当てオペレーターの左側に現れる場合、その割り当てオペレーターの右側の式の値が、それに関連付けられたメモリ位置に格納されます。変数識別子が任意のその他のコンテキストで使用される場合、その値はメモリからフェッチされ、識別子の代わりに使用されます。

Syntax

```
{identifier} {datatype} [ SIGNED | UNSIGNED ]  [ NOT  NULL ]
[:=initial-value-expression]
```

identifier

datatype によって指定された型のデータに関連付けるための名前を指定する有効な PL/MX 識別子。

datatype

データ型で説明されている PL/MX データ型を指定します。

SIGNED | UNSIGNED

「**Datatypes**」で説明されるように、いくつかのデータタイプに関してのみ許可されます。

NOT NULL

この変数には NULL 値を持たないことを示します。値 NULL がこの変数に割り当てられる場合は、VALUE_ERROR 例外が発生します。

initial-value-expression

宣言時に、対象変数に対して計算され、割り当てられる PL/MX 式です。可能な場合は、式はその変数のタイプに変換されます。式の値が変数の制約を満たさない場合は、VALUE_ERROR 例外が発生します。*initial-value-expression* が省略される場合は、その変数のデフォルト値は NULL です。*initial-value-expression* が省略される場合は、その変数に関して NOT NULL を指定することができません。

例

NULL (デフォルトの初期化値) 対してに初期化されることになる正の整数変数を宣言します。

```
a_variable POSITIVE;
```

初期値を持つ整数変数を宣言します。

```
loop_counter INTEGER := 5;
```

定数

PL/MX 定数は識別子を不変の値に関連付けます。

Syntax

identifier CONSTANT *datatype* [SIGNED | UNSIGNED] := *initial-value-expression*

ここで、

identifier

これは、datatype により指定されたタイプの initial-value-expression に関連付けるために名前を指定する有効な PL/MX 識別子です。

datatype

データ型で説明されている PL/MX データ型を指定します。

SIGNED | UNSIGNED

「**Datatypes**」で説明されるように、いくつかのデータタイプに関してのみ許可されます。

initial-value-expression

指定されたデータタイプに関して有効な PL/MX 式であるか。initial-value-expression が、宣言時に、対象定数に対して計算され、割り当てられます。可能な場合は、式はその変数のタイプに変換されます。式の値が変数の制約を満たさない場合は、VALUE_ERROR 例外が発生します。

例

値 4096 を持つ 2 進整数定数を宣言します。

```
    fourK CONSTANT BINARY_INTEGER := 4096;
```

関数

Syntax

```
FUNCTION function [(parameter-list)] RETURN mx-datatype
IS
    block
parameter-list :=
    parameter-spec parameter-specs

parameter-spec :=
    identifier [parameter-mode] mx-datatype
```

```
parameter-specs :=
    [ , parameter-spec parameter-specs ]

parameter-mode :=
      IN
    | IN OUT
    | OUT
```

ここで、

function

は、この関数に関する、ANSI 論理名 (ピリオドにより区切られた最大 3 つの部分) です。3 つの部分のそれぞれは、最大 128 文字までで指定できます。

mx-datatype

PL/MX のデータ型を**データ型**に説明されているように指定します。これが NonStop SQL/MX のデータ型にもなります。データ型は、INT、INTEGER、LARGEINT、SMALLINT、CHAR、CHARACTER、VARCHAR、VARCHAR2、NUMBER、NUMERIC、DATE、TIME、TIMESTAMP、および INTERVAL です。

parameter-mode

パラメータのモードを指定します。IN または IN OUT は、値が関数に指定されることを示します。OUT 設定の場合、パラメータは NULL とみなされます。IN は関数内部で定数として動作し、その関数内部では変更できません。IN OUT と OUT の設定では、関数は値を読み書きでき、最終値が関数から返されます。指定しない場合、そのデフォルト値は IN となります。

1 セットの空のカッコを使用することによるか、または *parameter_list* 関数にパラメータを除外することによるかして、その関数には何もパラメータがないことを示すことができます。

例

整数の階乗を計算する関数を宣言します。

```
FUNCTION factorial( n in INTEGER ) RETURN INTEGER
IS
  fact     INTEGER;
  inner_n  INTEGER;
 BEGIN
  fact  := 1;
  FOR inner_n IN REVERSE 2..n LOOP
    fact := fact * inner_n;
  END LOOP;
        RETURN fact;
 END factorial;
```

手順

Syntax

```
PROCEDURE procedure  [ ( parameter-list ) ]  IS block

    parameter-list :=        parameter-spec  parameter-specs

    parameter-spec ::=    identifier [ parameter-mode ] mx-datatype

    parameter-specs ::=    [ , parameter-spec  parameter-specs ]

    parameter-mode ::=    IN  |  IN OUT  |  OUT
```

ここで、

procedure

手順については、ANSI の論理名 (ピリオドで区切られた最大 3 つの部分) です。3 つの部分のそれぞれは、最大 128 文字までで指定できます。

mx-datatype

PL/MX のデータ型を**データ型**に説明されているように指定します。これが NonStop SQL/MX のデータ型にもなります。データ型は、INT、INTEGER、LARGEINT、SMALLINT、CHAR、CHARACTER、VARCHAR、VARCHAR2、NUMBER、NUMERIC、DATE、TIME、TIMESTAMP、および INTERVAL です。

Parameter-mode

パラメータのモードを指定します。IN または IN OUT は値が手順に指定されることを示します。OUT 設定の場合、パラメータは NULL とみなされます。IN は手順内部で定数として動作し、その手順内部では変更できません。IN OUT と OUT の設定では、手順は値を読み書きでき、最終値が手順から返されます。指定しない場合、そのデフォルト値は IN となります。サブクエリは IN パラメータ内で許可されます。サブクエリは、対象パラメータ向けに単一スカラー値を返す必要があります。タイプは、それがサポートするタイプの内の 1 つである必要があります。

1 セットの空のカッコを使用することによるか、または *parameter_list* 関数にパラメータを除外することによるかして、その手順には何もパラメータがないことを示すことができます。

例

```
PROCEDURE factorial (n in INTEGER, result out INTEGER) IS
BEGIN
    result := 1;
    FOR inner_n in 2 .. n LOOP
        result := result * inner_n;
    END LOOP;
END factorial;
```

以下に示す例では、IN パラメータ内でのサブクエリの使用を示してます。

```
CREATE PROCEDURE CAT.SCH.SIMPLTIMESTAMP2
  (
    TS IN TIMESTAMP
  , TS1 OUT TIMESTAMP
  )
  LANGUAGE PLMX
  NO SQL
  LOCATION \NSK.$DATA4.ZSDLDQXH.K4XPKD00
  PARAMETER STYLE PLMX
  NOT DETERMINISTIC
  ISOLATE
  IS
 begin
    ts1:=TO_TIMESTAMP(''2003/12/13 10:13:18'', ''YYYY/MM/DD HH:MI:SS'');
    ts1:=dateadd(yy, 2, ts1);
 end simpltimestamp2;

>>invoke t2;

-- Definition of table CAT.SCH.T2
-- Definition current  Mon Feb 05 14:25:19 2018

  (
    TS1                              TIMESTAMP(6) DEFAULT NULL
```

```
  , TS2                                TIMESTAMP(6) DEFAULT NULL
  )

--- SQL operation complete.

call simpltimestamp2((select ts1 from t2), ?a);

TS1
--------------------------

2005-12-13 10:13:18.000000
```

サブクエリが 1 つ以上の行が返す場合は、エラーが返されます。

```
call simpltimestamp2((select ts1 from t2), ?a);

*** ERROR[8401] A row subquery or SELECT...INTO statement cannot return more than one row.

--- SQL operation failed with errors.
```

ラベル

Syntax

```
<< label_name >>  statement
```

FOR、LOOP、および WHILE 文に配置されたラベルは、CONTINUE 文と EXIT 文により参照されることができます。任意の文上に配置されたラベルは、GOTO 文により参照されることができます。

カーソル

カーソルが宣言され、PL/MX 内の SQL 文で使用されている。SQL 文の説明については、カーソルの例を参照してください。PL/MX カーソル変数はサポートされません。

可視性ルール

可視性ルールは、変数、定数、ラベル、およびカーソルに対して同じルールが適用されます。サブプログラム内部にはさまざまな「可視性レベル」が存在しています。最も外部レベルが、その最も外部の「BEGIN」の前に宣言される、パラメータおよびすべての変数および定数が含んでいます。このレベル内部の入れ子レベルは以下のような 2 つの理由により発生します。

- FOR 文
- DECLARE ブロック

変数と定数が DECLARE を跨いで宣言されると、対応する BEGIN 間は、それら自身のレベルを取ります。同様に、FOR 文内部で宣言される FOR ループ インデックス変数は、それ自身のレベルを取ります。

さらに、ラベルとカーソルはレベルにも属しています。ラベルに関して、ラベルは、そのラベルを含んでいる文の場所に対応します。カーソルに関して、ラベルは、そのカーソルを宣言する SQL 文の場所に対応します。対象の位置が、任意の FOR 文または DECLARE ブロックの外部である場合は、そのラベルまたはカーソルは最も外側の可視性レベルに属することになります。対象の位置が、任意の FOR 文または DECLARE ブロック内部である場合は、そのラベルまたはカーソルはその文を含むこれらの構文の最も内部に属することになります。

変数、定数、ラベル、またはカーソルは名前 (「識別子」) があります。同じ識別子は同じ可視性レベル内で 1 回以上宣言されることはできません。同じ識別子は異なる可視性レベル内で宣言されることができます。このようなそれぞれの識別子はそのレベル内部でのみ可視化され、そのレベルの外部のレベルで宣言されている同じ名前の識別子は非可視となります。

例えば、次に示すものは、識別子 x が同じ (最も外部) レベルで、1 回は変数として、もう 1 回は ラベルとして、2 回出現するために、無効となります。

```
PROCEDURE p IS
        x INT;
  BEGIN
         << x >> NULL;
     END;
```

けれども、以下に示す例にある X の 全ての 3 回の出現はすべて異なるレベルにあるため、リーガルとなります。

```
PROCEDURE p IS
        x INT:= 1;
        y1 INT;
        y2 INT;
        y3 INT;
    BEGIN
        FOR x in 2 .. 5 LOOP
            DECLARE
                x INT := 3;
            BEGIN
                y3 := x;
            END;
            y2 := x;
        END;
        y1 := x;
    END;
```

上記のコードは、1 を y1 に、5 を y2 に、および 3 を y3 に割り当てます。

文

PL/MX 内の文は、ストレージとの間でデータを移動し、式を計算し、実行フローを制御し、プログラムロジックを実装します。詳しくは、**PL/MX 文**を参照してください。

ブロック

ブロックには、変数宣言と定数宣言、および関数の本体を構成する文が含まれています。ブロックの構造は次のとおりです。

```
--- 変数と定数のローカル宣言がここに入ります。
BEGIN
--- 1 つ以上の PL/MX 文がここに入ります。
END;
```

コンパイラーの条件文

PL/MX ソースの選択された部分は、コンパイラーの条件文および PLMX_CCFLAGS コマンドライン引数を使用して条件付きでコンパイルできます。

構文

```
$IF conditional-boolean-expression $THEN
  source
[$ELSIF conditional-boolean-expression $THEN source]
[$ELSE source]
$END
```

```
conditional-boolean-expression :=
   [(] [unary-boolean-operator] conditional-boolean-operand
   [ binary-boolean-operator conditional-boolean-expression ] [)]

conditional-boolean-operand :=
    $$ccflags-variable
  | FALSE
  | TRUE
  | NULL
  | conditional-boolean-expression
  | conditional-comparison-expression

unary-boolean--operator := NOT

binary-boolean-operator :=
    AND
  | OR

conditional-comparison-expression :=
  conditional-arithmetic-expression relational-operator
conditional-arithmetic-expression
  | conditional-boolean-expression relational-operator
  conditional-boolean-expression

relational-operator :=
    =
  | <
  | <=
  | >
  | >=
  | ^=
  | !=
  | <>
  | ~=
```

上記において、

source

文、コメント、または追加のコンパイラー条件文などの PL/MX ソース。

conditional-boolean-expression

構文で説明されているブール式。

ccflags-variable

-plmx_ccflags コマンドラインオプションで定義される変数の名前。plmx_ccflags オプションについて詳しくは、**スタンドアロンコンパイラー**を参照してください。

conditional-arithmetic-expression

-2147483647 から 2147483647 までの範囲内の数値リテラル 。

例

リテラル定数 TRUE を使用したシンプルな$IF 文。

```
$IF TRUE $THEN
  ---このソースをコンパイルに含めます。
$END
```

$ELSIF と$ELSE を使用した$IF 文。

```
    $IF $$compileThis $THEN
   ---このソースをコンパイルに含めます。
            built_for_flag := 1;
 $ELSIF $$compileThat $THEN
  ---あのソースをコンパイルに含めます。
   built_for_flag := 2;
 $ELSE
   ---このソースでもあのソースでもない場合にここから含めます。
   built_for_flag := 3;
 $END
```

データ型

変数または定数の宣言は、データ型に関連付けられています。データ型は、受け入れ可能な値の範囲およびその変数または定数上で実行することができる操作に影響します。PL/MX は、以下のような datatype をサポートしています。

BOOLEAN Datatype

サポートされている値は TRUE または FALSE です。論理型および比較型の操作は BOOLEAN データ型に関してサポートされています。

Integer Datatypes

算術と比較オペレーションはすべての整数データ型に関してサポートされています。INT、PLS_INTEGER、および SIMPLE_INTEGER は INTEGER の同義語です。

Datatype	Range
BINARY_INTEGER [SIGNED]	$-2^{31} .. 2^{31} - 1$
BINARY_INTEGER UNSIGNED	$0 .. 2^{32} - 1$
INTEGER [SIGNED]	$-2^{31} .. 2^{31} - 1$ および NULL
INTEGER UNSIGNED	$0 .. 2^{32} - 1$ および NULL
LARGEINT	$-2^{63} .. 2^{63} - 1$ および NULL
NATURAL	$0 .. 2^{31} - 1$ および NULL
NATURALN	$0 .. 2^{31} - 1$
POSITIVE	$1 .. 2^{31} - 1$ および NULL
POSITIVEN	$1 .. 2^{31} - 1$
SIGNTYPE	-1 .. +1 および NULL
SMALLINT [SIGNED]	$-2^{15} .. 2^{15} - 1$ および NULL
SMALLINT UNSIGNED	$0 .. 2^{16} - 1$ および NULL

注記: そのテーブル内の指定された NULL を持たない整数 datatype は、暗黙的に NOT NULL 属性を持ちます。

String Datatypes

連結型および比較型のオペレーターはすべての以下のような文字列データ型に関してサポートされています。

- CHAR [(N [CHAR | BYTE])]
- VARCHAR (N [CHAR | BYTE])
- VARCHAR2 (N [CHAR | BYTE])

CHAR () とは、固定長 N の文字列を意味します。(N) が省略される場合、デフォルト値は 1 となります。このような変数に不正な文字長の文字列を割り当てないでください。VARCHAR (N) または VARCHAR2 (N) とは、最大長 N の文字長を意味します。最大で N までの文字長の任意の文字列は、このような変数に対して割り当てられる可能性があります。すべての文字列に関して、N に関して許可される最大値は 32767 です。この N の後方のオプション CHAR または BYTE は、意味を持っていません。VARCHAR と VARCHAR2 の差異は、VARCHAR2 データ型では、空の文字列が NULL と同等であることです。

CHARACTER は CHAR の同義語です。STRING は VARCHAR の同義語です。

Exact Numeric Datatypes

算術と比較オペレーションは NUMERIC [(P [, S])] exact numeric datatype に関してサポートされています。

NUMERIC (P, S) は、P の精度 (合計の小数点以下桁数) を持つ固定少数点値、および S のスケール (小数点以下の桁数) を意味しています。S が省略されるような場合、それは 0 になります。P が省略されるような場合、それは 9 になります。この精度に関して許可される最大値は 128 です。

NUMBER は NUMERIC の同義語です。

Datetime Datatypes

算術と比較オペレーションは、以下に示すような datetime datatype に関してサポートされています。

- DATE
- TIME [(S)]
- TIMESTAMP [(S)]

SQL/MX では、DATE datatype の 2 つのタイプ、DATE、および DATE type2 をサポートしています。デフォルトでは、スタンドアロンコンパイラーはそのタイプに DATE を取ります。-DC_TYPE オプションが指定される場合は、スタンドアロンコンパイラーは DATE を DATE type2 と見なします。2 つのタイプの日付は、同じサブプログラム内で混在できません。DATE には 3 つの整数値、即ち、1 ～ 9999 までの「年」、1 ～ 12 までの「月」、および 1 ～ 31 までの「日」が含まれています。

TIME には 2 つの整数値、即ち、0 ～ 23 までの 1 時間、0 ～ 59 までの 1 分が含まれています。そして、そのスケールに関して S 桁の小数点以下を付与する、0 ～ 59 までの秒に関する NUMERIC が含まれています。S に関して許可される値は 0 ～ 6 であり、そのデフォルト値は 0 となります。

TIMESTAMP には、5 つの整数および NUMERIC、即ち、DATE および TIME の組み合わせが含まれています。しかしながら、この場合、そのスケールのデフォルト値は 0 ではなく、6 となります。

DATE type2 には、TIMESTAMP のような、6 つの整数を含んでいますが、小数の秒数はありません。

Interval Datatypes

算術と比較オペレーションは、以下に示すような interval datatype をサポートします。

- INTERVAL F [(M)] TO F [(S)]
- INTERVAL F [(M [, S])]

間隔タイプに関して、F とは、2 つのカテゴリ、年-月のフィールドおよび日-時間のフィールド、に分割されるフィールドの名前です。年月フィールドは、YEAR および MONTH です。この日付時刻フィールドは、DAY、HOUR、MINUTE、および SECOND です。

INTERVAL F [(M)] TO F [(S)] 形式で宣言する場合、両方のフィールドは、同じカテゴリからのものである必要があります。最初のフィールドは、そのカテゴリに関するフィールドのリスト内の第 2 フィールドに、等しいか、またはそれに先行する必要があります。それから、このペアはそのカテゴリ内部のフィールドのリストを代表することになり、指定済みの最初のものから開始され、指定済みの最後のものにより終了されることになります。INTERVAL F [(M [, S])] 形式で 1 つのフィールドが指定されます。

INTERVAL には、現在のフィールドのそれぞれの値が含まれています。それぞれのフィールドに関して、その SECOND フィールドが存在する場合には、それが NUMERIC であることを除外し、その値は整数となります。それ故、全てのフィールドは、整数型部分、および最後のフィールドを持ちます。さらに、SECOND フィールドには、小数部分も持ちます。

SECOND フィールドは、S (スケール) により指定されるように、0 ～ 6 の小数点下桁数を持つことができます。最後のフィールドの INTERVAL が SECOND である場合にのみ、S が許可されます。指定しない場合、そのデフォルト値は 6 となります。間隔を宣言するための INTERVAL F [(M [, S])] 構文では、さらに M を指定することなく、S を指定することはできません。その SECOND フィールドが存在する場合は、その小数部分は S 桁の「場所」を「占有」すると表現されます。

最初のフィールドは、整数型であり、その整数部分は、2 つの場所を占有しているデフォルトですが、占有されている場所の桁数は、M とその大きさにより指定でき、全体の INTERVAL 値が 18 桁の「場所」を占有するようになるまで、可能な限りどんな大きな値でも取ることができます。例えば、フィールドは、INTERVAL MINUTE (4) TO SECOND (2) と指定することができます。この場合、分数は 0000 ～ 9999 までを取り、100 分の 1 秒を表示することができます。INTERVAL 全体は、その分数に関しては 4 桁の場所、その秒数の整数型部分に関しては 2 桁、およびその秒数の小数分に関しては 2 桁を占有します。

Datatype に関する留意事項

変数を宣言する場合、キーワード、SIGNED および UNSIGNED は 、NUMERIC タイプや、ほとんどの整数型タイプに対して追加することができますが、LARGEINT または SIGNTYPE に対してはできません。
SIGNED または UNSIGNED を指定しない場合、そのデフォルト値は SIGNED となります。式内で使用される整数型タイプに関して、その符号は、その整数型の変数はまたは符号なし整数コンテナーにロードされているかどうかを決定します。整数型タイプの符号は、それに割り当てできる値の許容範囲に時折影響することがあります。

次のデータタイプは、パラメータタイプとして指定することも、関数の戻り値タイプとしても指定することができます。

- CHAR または CHARACTER
- INT または INTEGER
- LARGEINT
- NUMERIC または NUMBER
- SMALLINT
- VARCHAR または VARCHAR2
- DATE
- TIME
- TIMESTAMP
- INTERVAL

パラメータの種類として、または戻り値タイプとして文字列を使用する場合のサイズは指定しません。パラメータの種類として、または戻り値タイプとして numeric または datetime を使用する場合は、何の精度またはスケールも指定することはできません。INTERVAL がパラメータまたは戻り値タイプを返す場合は、INTERVAL 内に存在するフィールドを指定する必要があります。精度またはスケールを指定しないでください。これらのすべての場合で、実行時にその呼び出し元から渡される属性は、IN パラメータに関する式の値としてか、または IN OUT パラメータ、OUT パラメータ、もしくはその戻り値からか、のいずれかとなります。

式

式は、評価時に単一値を生成するオペランドとオペレーターのシーケンスです。文内のオペレーターは、そのオペランドの datatype と互換性がある必要があります。

ブールオペレーター

BOOLEAN オペレーターのオペランドは BOOLEAN である必要があり、その結果は BOOLEAN です。オペレーションの結果は以下に示すとおりです。

A	B	An OR B	A AND B	NOT A
TRUE	TRUE	TRUE	TRUE	FALSE
	FALSE	(A が TRUE となるような場合、B は計算されません)	FALSE	
	NULL		NULL	
FALSE	TRUE	TRUE	FALSE	TRUE
	FALSE	FALSE	(A が FALSE となるような場合、B は計算されません)	
	NULL	NULL		
NULL	TRUE	TRUE	NULL	NULL
	FALSE	NULL	FALSE	
	NULL	NULL	NULL	

注記: ルールは、任意のオペランドが NULL の場合、結果のオペレーションは例外を持つ NULL です。(A が NULL である場合、B が TRUE である場合、A OR B の結果は TRUE です) (A が NULL である場合、B が FALSE である場合、A AND B の結果は FALSE です)

(A が TRUE となるような場合、A OR B は TRUE であり、B は計算されません) (A が FALSE となるような場合、A AND B は FALSE であり、B は計算されません)

比較オペレーター

以下に示すのは、バイナリ形式比較オペレーターです。

- <
- <=
- >
- >=
- =
- <> != ~= ^= (「not equal」を示すための 4 つの方法)

以下に示す、オペランドの組み合わせはオペレーターに関して許可します。

- 両方の BOOLEAN
- 整数、NUMERIC、および文字列の任意の組み合わせ
- 両方の DATE、または 1 つの DATE および 1 つの文字列
- 両方の TIME、または 1 つの TIME および 1 つの文字列
- 両方の TIMESTAMP、または 1 つの TIMESTAMP および 1 つの文字列
- INTERVAL の同等のカテゴリの両方、または 1 つの INTERVAL および 1 つの文字列

BOOLEAN 比較に関して、TRUE は FALSE より大きい値です。DATE、TIME、TIMESTAMP、および INTERVAL に関して 、その比較は年代順となります。すべての場合で、その結果は BOOLEAN であり、いずれかのオペランドが NULL となるような場合は、その結果は NULL となるようになります。

両方のオペランドが文字列となるような場合は、文字列比較が実行されることになります。1 つのパラメータのみが文字列となるような場合で、もう一方のオペランドが NUMERIC である場合には、このパラメータは NUMERIC に変換されます。または、もう一方のオペランドが DATE、TIME、TIMESTAMP、または INTERVAL であるようになる場合は、その文字列は比較するためにそのタイプに変換されることになります。

他の比較オペレーターは、BETWEEN、IS NULL および IS NOT NULL があります。IS NULL と IS NOT NULL は単項オペレーターで、そのオペランドの後に来るもので、どの型でも使用できます。その結果は BOOLEAN です。IS NULL に関して、そのオペランドが NULL である場合は、その結果は TRUE となり、そのオペランドがそれ以外のものである場合は、その結果は FALSE となります。IS NOT NULL に関して、そのオペランドが NULL である場合は、その結果は FALSE となり、そのオペランドがそれ以外のものである場合は、その結果は TRUE となります。

BETWEEN オペレーターの構文は次のとおりです。

X BETWEEN A AND B

X が A と B のそれぞれと比較できる限り、X、A、および B は、さまざまなタイプを取ることができます。この意味は、X が 1 回に限定して計算されることを除外し、以下に対して同等です:

(X >= A) AND (X <= B)

算術オペレーター

単項算術オペレーター + および - 、バイナリ算術オペレーター +、 - 、*、および / は、整数、NUMERIC、および文字列オペランドの任意の組み合わせに適用されることができます。文字列は、常に、そのオペレーションに関する NUMERIC に変換されます。32 ビット整数は、64 ビットの整数に変換されることができるか、または任意のタイプの整数は、両方のオペランドを操作に関して同等とするために、NUMERIC に変換されることができます。例えば、5 < '6' が TRUE と評価され、5 < 'six' が VALUE_ERROR 例外を発生させ、'six' は NUMERIC に変換できません。

以下に示すオペレーションは、datetime および interval datatype に関して許可されます。

- datetime および INTERVAL を加算することは、datetime と同じタイプを返します
- DATE type2 および NUMERIC (即ち、多数の日数) を加算することは、DATE type2 と同じタイプを返します
- INTERVAL を datetime から減算することは、datetime と同じタイプを返します
- NUMERIC (即ち、多数の日数) を DATE type2 から減算することは、DATE type2 を返します
- 2 つの DATE 値を減算することは、INTERVAL を返します
- 2 つの TIME、または 2 つの TIMESTAMP 値を減算することは、INTERVAL を返します
- 2 つの DATE type2 値を減算することは、その NUMERIC を返し、それらの間の日数を表示します。
- 2 つの INTERVAL 値を加算または減算することは、同じカテゴリの INTERVAL を返します

- INTERVAL を NUMERIC により乗算または除算することは、同じタイプの INTERVAL を返します
- 単項オペレーター + または - を INTERVAL に適用することは、同じタイプの INTERVAL を返します。

任意のオペランドが NULL である場合、算術オペレーターの結果は NULL です。

整数型タイプには、1 つのコンテナーサイズの概念、およびそれらが符号付きまたは符号なしであるかどうかの 1 つの概念があります。これらの概念は、計算を実行する方法に影響します。可能なコンテナーサイズは 32 ビットおよび 64 ビットです。コンテナーサイズは、LARGEINT に関しては 64 ビットであり、その他の整数型タイプに関しては 32 ビットです。それ故、そのコンテナーサイズに関して合計で 3 つの可能な組み合わせが存在しており、整数が符号付きまたは符号なしであるかどうかで、即ち、以下に示すようになります。

- 32 ビット (符号付き)
- 32 ビット (符号なし)
- 64 ビット (符号付き)

変数やパラメータの名前が式内部で発生する場合、その変数やパラメータのコンテナーサイズは、その基本アイテムのサイズであると見なされます。変数やパラメータが符号付きまたは符号なしであるかどうかは、その基本アイテムが符号付きまたは符号なしの整数型として処理されるかどうかで、決定されます。

整数型リテラルは、符号付きとして処理されます。これらが 32 ビットに収まる場合、これらは 32 ビットであるとみなされ、それ以外の場合では、これらは、64 ビットであるとみなされます。

バイナリ算術オペレーターの両方のオペランドが、同じタイプのコンテナーを持つ整数型となるような場合、さらに、その結果はそのコンテナーと同じタイプとなることになります。その結果がそのコンテナーに適合しない場合、VALUE_ERROR 例外が発生します。バイナリの算術または比較オペレーションの 2 つのオペランドが異なるタイプのコンテナーの整数のオペランドである場合、これらは LARGEINT に変換されます。そしてそれから、操作は実行されません。オペランドが整数であり、もう一方は NUMERIC である場合、この整数は、NUMERIC に変換されます。例えば、2000000000 + 2000000000 はオーバーフローとなり、2000000000 + '2000000000' は 4000000000 と評価されます。最初の例では、両方のオペランドが INTEGER のタイプとみなされるリテラルです。これらのタイプが一致すると、加算されます。けれども、結果が INTEGER に適合しない場合、オーバーフローを発生させます。この第 2 の例では、このリテラル '2000000000' は文字列です。それは NUMERIC に変換され、従って、2000000000 もまた NUMERIC に変換されます。両方のタイプは同じであり、何のオーバーフローも存在しません。

単項減算は、値「-1」の符合付きの整数による乗算と同等です。それ故、単項減算が符号なし整数に対して適用される場合は、その整数は 64 ビットの整数に変換されてから、無効とされるようになります。単項減算を NUMERIC に適用することは、NUMERIC (1 の精度、0 のスケール、および 値による) による乗算と同等です。

単項加算は、それが整数または NUMERIC に対して適用される場合は、NO-OP 命令 (なにもしない) として扱われます。符号なし整数に対して適用される場合は、その符号なし整数は 64 ビットの整数に変換されません。

単項加算又は単項減算の文字列への適用は、その文字列を NUMERIC に変換させます。

ブール式

```
boolean-expression :=
  [(] [unary-boolean-operator] boolean-operand
  [ binary-boolean-operator boolean-expression ] [)]

boolean-operand :=
    variable
  | FALSE
  | TRUE
  | boolean-expression
```

```
  | comparison-expression

unary-boolean-operator := NOT

binary-boolean-operator :=
    AND
  | OR

comparison-expression :=
    arithmetic-expression relational-operator arithmetic-expression
  | boolean-expression relational-operator boolean-expression

relational-operator :=
    =
  | <
  | <
  | >
  | >=
  | ^=
  | !=
  | <>
  | ~=
```

算術式

```
arithmetic-expression :=
  [() [unary-arithmeticoperator] numeric-operand
  [ binary-arithmetic-operator arithmetic-expression ] [)]

numeric-operand :=
    variable
  | literal
  | arithmetic-expression

unary-arithmetic-operator :=
    +
  | -

binary-arithmetic-operator :
    +
  | -
  | *
  | /
```

CASE 式

CASE 式は、式の計算結果として返す対象の値を決定する式です。この式は、代替文の中からの選択肢に基づき計算されます。さらに、CASE 文には、「シンプル CASE 式」および「検索済み CASE 式」と呼ばれる、2 つのタイプの CASE 式が存在しています。

シンプル CASE 式

シンプル CASE 式に関する構文は以下のようになります。

```
CASE case-expression
   {WHEN  when-expression THEN
      then-expression }...
```

```
   [ELSE
      else-expression]
```

この *case-expression* は 1 回計算されます。比較オペレーションが *case-expression* のタイプの組み合わせおよび それぞれの *when-expression* に対して適用できることが要求されます。それぞれの *when-expression* が計算され、比較オペレーションの 1 回が TRUE を返すまで *case-expression* に対して等価性に関して比較されます。それから、対応する *then-expression* が評価されます。比較が TRUE を返し、ELSE が存在するようになる場合は、その *else-expression* が評価されることになります。比較が TRUE を返し、何の ELSE も存在しないようになる場合は、その *case-expression* が NULL になることになります。

CASE 式の結果タイプは、すべての *then-expression* および *else-expression* が評価されることができるタイプです。

検索済み CASE 式

検索済み CASE 式に関する構文は以下のようになります。

```
CASE
{WHEN  when-expression THEN
       then-expression }...
[ELSE
       else-expression]
```

それぞれの *when-expression* は BOOLEAN 式である必要があります。これらの式の内の 1 つが TRUE となるまで、*when-expression* は計算されます。対応する *then-expression* がこの文に関して評価されます。いずれの *when-expression* も TRUE にはならないような場合で、ELSE も存在するようになる場合は、その *else-expression* は評価されることになります。いずれの *when-expression* も TRUE にはならないような場合で、何の ELSE も存在しないような場合は、そのケース文の結果は NULL となります。

連結

文字列は、2 本の垂直の棒 ('||') である、連結オペレーターにより連結されます。オペランドが文字列にはならないような場合、これらのオペランドは文字列に変換されてから、連結されることになります。例えば、5 || 6 は「56」と評価されます。これらのオペランドの内の 1 つが NULL である場合は、例外が発生することになります。

制約

変数には制約の概念があります。制約は、計算の実行方法には影響を与えません。値はどこかに配置されている場合、1 回の計算最後にその制約が満たされる必要があります。例えば、値が変数またはパラメータに対して割り当てられる場合。

制約のタイプは以下のようになります。

- 整数型タイプは範囲の制約を持ちます。
- NUMERIC は大きさの制約を持ち、その精度はそのスケールを減算します。
- 文字列タイプは文字列長の制約を持ちます。
- INTERVAL タイプは存在している対象フィールドに従い、制約されます。
- 整数型タイプでもそれが NOT NULL の制約を持っ可能性があります。

その式の値がその変数の制約での範囲制約を満たさない場合は、整数に対する割り当てが失敗します。

その値がその NUMERIC 変数によりサポートされる大きさを持たない場合は、NUMERIC に対する割り当てが失敗します。その変数がそれに割り当てられている式よりもより小さいスケールを持つ場合は、その式の出力は丸められることになります。

割り当てられている値が N よりも大きい場合、文字列に対する割り当ては失敗します。

その INTERVAL が持たないユニット内に非ゼロ値が存在している場合は、INTERVAL に対する割り当ては失敗します。例えば、5 時間と 3 分間は、対応する HOUR と MINUTE の単位なしでは、変数に割り当てることはできません。

注記: 整数の制約は、値が整数に割り当てられる場合に関するルールを提供します。整数が符合付きまたは符合なしであっても、計算の実行方法には影響を与えません。例えば、NATURAL が符合付きタイプであっても、その制約が、非負数のみがそれに割り当てるできると定義していれば、それに割り当てることができます。NATURAL に関連する計算は、符号付きのコンテナー内から起動します。NATURAL 上の制約の上限は、32 ビットに収容できる最大の符号付きの値です。NATURAL は PL/SQL に由来し、INTEGER UNSIGNED は SQL/MX に由来し、および両方はサポートされますが、同様には動作しません。

演算子の優先順位

式内の演算子は、優先順位の高い順に左から右に評価されます。次の表に、演算子とその優先順位を示します。

表 4: PL/MX の演算子の優先順位

演算子	意味	優先順位
単項+、単項 -	算術恒等、算術否定	1
*, /	乗算、除算	2
バイナリ+、バイナリ- 、||	加算、減算、連結	3
=、<、<=、>、> =、^ =、 ! =、<>、 ~=、IS NULL	比較	4
NOT	論理否定	5
AND	論理積	6
OR	論理和	7

評価の順序を変更するには括弧を使用します。括弧内の最も深くネストされた式が最初に評価され、次に演算子の優先順位が使用され、最後に左から右の順序で評価されます。

文

Assignment

Assignment 文は、式の値を 変数またはパラメータに代入します。式は変数のタイプに変換されます。式の値が変数の制約を満たさない場合は、VALUE_ERROR 例外が発生します。

Syntax

```
variable-identifier := {arithmetic-expression | boolean-expression}
```

ここで、

variable-identifier

は、以前に宣言された変数の識別子です。

例

算術式を変数に割り当てます。

```
percent_shipped := (quantity_shipped / quantity_in_stock) * 100;
```

ブール式を変数に割り当てます。

```
no_item_supplied := item == 0;
```

CASE

CASE 文は、代替文の中から選択されたものに基づき実行される入れ子になった文のリストを決定する文の一種です。CASE 文は、「検索済み case 文」または「単純な case 文」である可能性があります。

検索済み CASE 文

Syntax

```
CASE when-clauses [ else-clause ] END CASE;
when-clauses := when-clause [when-clauses]
when-clause := WHEN boolean-value THEN statements
statements := statement [statements]
```

ここで、

boolean-value

任意の式のタイプのブール値。

when-clauses 内で *boolean-value* は、それが TRUE であるものが見つかるまで、1 つずつ計算されます。それから、対応する *statement* が実行されます。*boolean-value* のどれもが TRUE ではなく、 *else-clause* が存在する場合、その *else-clause* 内の *statements* が実行されます。*boolean-value* のどれもが TRUE ではなく、*else-clause* も存在しない場合、CASE_NOT_FOUND 例外が発生します。

例

```
CASE
    WHEN (type = alumni_family) or (income < value1) THEN
        DBMS_OUTPUT.PUT_LINE ('award scholarship');
    WHEN (income < value2) THEN
        DMBS_OUTPUT.PUT_LINE ('award loan');
    ELSE
```

```
                    DMBS_OUTPUT.PUT_LINE ('no award');
END CASE;
```

単純な CASE 文

Syntax

```
CASE case-value when-clauses [ else-clause ] END CASE;
when-clauses := when-clause [when-clauses]
when-clause := WHEN when-value THEN statements
statements := statement [statements]
```

ここで、

case-value

任意のタイプの式です。

when-value

case-value のタイプに対して比較できるタイプの任意の式です。

case-value は 1 回計算されます。それから、*when-clauses* 内で *when-value* は、それが case-value と同等であるものが見つかるまで、1 つずつ計算されます。それから、対応する *statement* が実行されます。*when-value* のどれもが case-value に等しくなく、 *else-clause* が存在する場合、その *else-clause* 内の *statements* が実行されます。*when-value* のどれもが case-value に等しくなく、 *else-clause* も存在しない場合、CASE_NOT_FOUND 例外が発生します。

例

```
CASE grade
    WHEN 'a' THEN
        raw_gpa = raw_gpa + 4.0;
    WHEN 'b' THEN
        raw_gpa = raw_gpa + 3.0;
    WHEN 'c' THEN
        raw_gpa = raw_gpa + 2.0;
    ELSE
        need_to_repeat = TRUE;
END CASE;
```

CONTINUE

CONTINUE 文は、囲みループの現在の反復処理の実行を終了し、次の反復処理による実行を再開します。最後の反復処理内のループの実行中の場合は、CONTINUE はそのループの実行を完了します。ループとは、FOR 文、LOOP 文、または WHILE 文を表します。

Syntax

```
CONTINUE [label] [when-clause];
when-clause := WHEN boolean-expression
```

ここで、

label

これは複数の囲みループ上にあるラベルです。

boolean-expression

これは任意のブール式です。

ラベルが指定されている場合、ラベルは囲みループ内のラベルである必要があり、これはこの CONTINUE 文が適用されるループです。ラベルが指定されていない場合、その最も内側の囲みループはこの CONTINUE 文

が適用されるループです。*when-clause* が指定されている場合で、*boolean-expression* が真である場合にのみ、CONTINUE 文が実行されます。

例

```
<< outer >> FOR loop_counter in 1..10 LOOP
    << inner >> WHILE x LOOP
CONTINUE outer WHEN some_value = 5;
    END LOOP;
END LOOP;
```

通常、外側のループは 10 回実行され、内側のループの先頭に到達するたびに x が真である限り、内側のループが実行されます。しかしながら、この場合の CONTINUE は、内側ループを完全に終了し、外側ループの次の反復にジャンプします。

DBMS_OUTPUT

DBMS_OUTPUT 文は、DBMS_OUTPUT システムパッケージ内のサポートされている 2 つの手順の内の 1 つのに対する呼び出しです。これらの手順は、実行中に変数の値を出力するのに便利です。

Syntax

```
DBMS_OUTPUT.PUT( expression );
DBMS_OUTPUT.PUT_LINE( expression );
```

ここで、

expression

これは文字列式です。

PUT が文字列式を印刷されるようにします。PUT_LINE を使用すると、改行文字が出力の最後に追加されます。

留意事項

DBMS_OUTPUT 手順から出力を取得するには、出力を OSS ファイルに出力する必要があります。出力をリダイレクトするには、環境変数 SQLMX_UDR_STDOUT に値を割り当てます。OSS シェルでは、`export` コマンドを使用します。`export` コマンドの詳細については、「export マンページ」を参照してください。

例

文字列と変数を次の分離された手順により出力します。

```
DBMS_OUTPUT.PUT( 'The result = ' );
DBMS_OUTPUT.PUT_LINE( grand_total );
```

複数の文字列と変数を 1 つの PUT_LINE 呼び出しで出力します。

```
DBMS_OUTPUT.PUT_LINE( 'Error ' || err_num || ' in step ' || step_no );
```

NULL の可能性のある値を出力します。

```
IF x IS NULL THEN
    DBMS_OUTPUT.PUT_LINE ('x = <NULL>');
ELSE
    DBMS_OUTPUT.PUT_LINE ('x = ' || x );
END IF;
```

DECLARE/BEGIN 文 (ブロック)

DECLARE/BEGIN ブロックは、文入れ子に新しいレベルを定義します。さらに、それはこのレベル内部にのみ表示される変数を宣言ためにも使用できます。

Syntax

```
[ DECLARE
-- 各変数と各定数の宣言はここに配置します  ]
  BEGIN
--- 1 つ以上の PL/MX 文がここに入ります。
END  ;
```

DECLARE/BEGIN ブロックは、実行可能な文が許可されている場所であればどこにでも配置することができます。さらに、DECLARE/BEGIN 文も EXCEPTION 句を持つことができます。詳細については、「**例外処理**」を参照してください。

EXIT

EXIT 文は、囲みループの実行を終了し、ループの後の次の文で実行を再開します。ループとは FOR 文、LOOP 文、または WHILE 文を表します。

Syntax

```
EXIT [label] [when-clause];
when-clause := WHEN boolean-expression
```

ここで、

label

これは複数の囲みループ上にあるラベルです。

boolean-expression

これは任意のブール式です。

ラベルが指定されている場合、ラベルは囲みループ内のラベルである必要があり、これはこの EXIT 文が適用されるループです。ラベルが指定されていない場合、その最も内側の囲みループはこの EXIT 文が適用されるループです。*when-clause* が指定されるようになる場合で、*boolean-expression* が真になるようになる場合にのみ、EXIT 文が実行されます。

例

FOR ループ内で終了：

```
<< outer >> FOR loop_counter in 1..10 LOOP
    << inner >> WHILE x LOOP
        EXIT outer WHEN some_value = 5;
    END LOOP;
END LOOP;
```

通常、外側のループは 10 回実行され、内側のループの先頭に到達するたびに x が真である限り、内側のループが実行されます。しかしながら、この場合の EXIT は、内側ループを完全に終了し、外側ループの次のステートメントにジャンプします。

GOTO

GOTO 文は、制御をラベルに渡します。

Syntax

```
GOTO  label;
```

ラベルは可視化されている必要があります。それは、現在の GOTO 文と同じ構文レベルであるか、または囲み文内にあるか、である必要があります。変数名の可視性に関するものと同じルールが、GOTO 文に関して適用されます。このラベルがまだ定義されていない可能性があります。このラベルが以降に出現する可能性があるため、この GOTO 文が前方にジャンプすることになります。しかしながら、このラベルは内部文リストの内部にあることはできないため、GOTO は文リストにジャンプすることができなくなります。GOTO は、FOR 文または DECLARE/BLOCK の内部のラベルにはジャンプできません。FOR および DECLARE/BEGIN の構成でラベル可視性を定義することにより、そのラベルがその GOTO 文のポイントで表示されないようにできます。GOTO は、これらの構文内のラベルがその GOTO 文のポイントで表示可能であったとしても、以下に示すような文リストにはジャンプすることはできません。

- IF 文の THEN または ELSE 句
- CASE 文の WHEN 句
- LOOP および WHILE 文
- EXCEPTION 句に関連付けられた BEGIN または WHEN

例

```
FUNCTION f RETURN INTEGER IS
    BEGIN
        GOTO i; [1]
        FOR j IN 1..2 LOOP
        GOTO i;  [2]
            IF TRUE THEN
                << i >> NULL;  [3]
            GOTO i; [4]
            END IF;
        END LOOP;
        << i >> NULL;  [5]
END;
```

これらのラベルが異なる可視性レベルにあるため、行 3 および 行 5 に同じ名前「i」を持つこれらのラベルはリーガルとなります。行 1 にある GOTO 文はリーガルであり、前方ジャンプ先は行 5 にある外側ラベルです。行 4 にある GOTO 文はリーガルであり、後方ジャンプ先は行 3 にある内部ラベルです。

行 2 にある GOTO は、それを行 3 にある内部レベルに対して解決するため、リーガルとなります。けれども、GOTO は、そのジャンプ先が対象 IF 文の THEN 句内部にあるため行 3 に対してジャンプすることは許可されません。行 3 にある内側ラベルが存在しなかった場合は、行 5 にある外側のラベルを非表示とするために、全ての 3 つの GOTO 文は 行 5 にある外側ラベルにリーガルにジャンプするとになります。

FOR

FOR 文では、文のスコープ内でローカルな INTEGER インデックス変数が宣言され、開始値で始まり、終了値を超えるまでインクリメントまたはデクリメントされます。インデックス変数には、ループ本体内で値を代入することはできません。ループは、終了値に達するか、ループ内で EXIT 文または RETURN 文が実行されるまで続きます。

構文

```
FOR index-identifier IN [REVERSE] beginning-value .. ending-value
  LOOP statements
END LOOP;

beginning-value := arithmetic-expression
```

```
ending-value := arithmetic-expression

statements :=
  statement; [statements]
```

上記において、

index-identifier

FOR 文のインデックス変数に関連付ける名前を指定する有効な PL/MX 識別子。FOR 文の本体内のどこからでも参照できますが、FOR 文の外部からは参照できません。

REVERSE

指定された場合、*index-identifier* は *ending-value* に初期化され、*beginning-value* より小さくなるまで FOR 文の各ループでデクリメントされます。REVERSE を省略すると、*index-identifier* は *beginning-value* に初期化され、*ending-value* より大きくなるまで各ループでインクリメントされます。バインドされた値はループの開始前に 1 回評価され、その後は一定のままです。

statements

ループの各繰り返しで実行される一連の PL/MX 文です。

例

文を 10 回ループし、インデックス識別子を 1 から 10 までインクリメントする FOR 文。

```
FOR loop_counter in 1..10 LOOP
        ----いくつかのタスクを実行します...
END LOOP;
```

文を n-2+1 回ループし、インデックス識別子を n から 2 までデクリメントする FOR 文。

```
FOR inner_n IN REVERSE 2..n LOOP
  fact := fact * inner_n;
END LOOP;
```

IF

ブール式が TRUE の場合、IF 文は一連の文を実行し、そうでない場合は一連の文をスキップします。ELSE ... IF の簡単な書き方である ELSIF 句のオプションもあります。ネストされた IF ... ELSIF 文の場合、ブール式が TRUE に評価される最初の一連の文のみが実行され、ほかの文はすべてスキップされます。つまり、最後の "END IF"の後から実行が再開されます。ブール式がどれも TRUE に評価されない場合に実行される文を含む ELSE 句を、オプションで最後に置くことができます。

構文

```
IF boolean-expression THEN
  statements
[ELSIF boolean-expression THEN statements [elsif-clauses]]
[ELSE statements]
END IF;

elsif-clauses :=
  ELSIF boolean-expression THEN statements [elsif-clauses]
```

上記において、

boolean-expression

ブール式。詳しくは、**ブール式**を参照してください。

statements

一連の 1 つ以上の PL/MX 文。

例

他の句のない IF 文：

```
IF finished THEN
  DBMS_OUTPUT.PUT_LINE( 'Done!' );
END IF;
```

ELSIF および ELSE 句を含む IF 文：

```
IF finished THEN
  DBMS_OUTPUT.PUT_LINE( 'Done!' );
ELSIF more_work_to_do THEN
  DBMS_OUTPUT.PUT_LINE( 'More work still to do!' );
ELSE
  DBMS_OUTPUT.PUT_LINE( 'Totally Finshed!' );
END IF;
```

LOOP

LOOP 文は無限ループを作成します。ループ内の EXIT 文または RETURN 文が実行されるまで、その中に含まれる文が繰り返し実行されます。

構文

```
LOOP
  statements
END LOOP;
```

上記において、

statements

一連の 1 つ以上の PL/MX 文。

例

条件が満たされたときに終了する LOOP 文：

```
LOOP
  ---ここで何らかの作業を実行します。
  IF finished THEN  -- ループを終了します。
   EXIT;
  END IF;
END LOOP;
```

NULL

NULL 文は、名前からわかるように何もしません。これは、コード内の場所をマークするため、または IF 文の分岐内で配置された場所では何も起こらないことを示すためのプレースホルダとして役立ちます（ドキュメントに「このページは意図的に空白にされています」という記述を追加することに似ています）。

構文

```
NULL;
```

例

IF 文の一部として NULL 文を使用する場合：

```
IF NOT output_was_written THEN
  DBMS_OUTPUT.PUT_LINE( 'Some output' );
```

```
  output_was_written := TRUE;
ELSE
  ---
  ---出力が既に書き込まれている場合は何もしません。
  ---
  NULL;
END IF;
```

RAISE

RAISE 文の説明については、「**例外処理**(49 ページ)」を参照してください。

Syntax

```
RAISE [ expression-name ];
```

RETURN

RETURN 文は、その手順または関数の実行を終了させ、その呼び出し元に制御を返します。

Syntax

```
RETURN [expression];

expression :=
    arithmetic-expression
  | boolean-expression
```

手順に関して、RETURN 文は式を持つことはできません。RETURN 文に遭遇することなく、制御がその手順の終了に到達するようになる場合は、暗黙的に RETURN が発生します。

関数に関して、RETURN 文は式を持つ必要があります。式の値がその呼び出し元に返されます。式はその関数の戻り値タイプに変換されます。式の値が戻り値タイプの制約を満たさない場合は、VALUE_ERROR 例外が発生します。制御が RETURN 文に遭遇することなく関数の最後に到達するようになる場合は、END_OF_FUNCTION 例外が発生します。

例

計算済み算術値を持つ関数からの戻り値は以下のとおりです。

```
RETURN (X*X + 2*X + 5);
```

変数値を持つ関数からのリターン :

```
RETURN (counter);
```

WHILE

WHILE 文は、ブール式が FALSE と評価されるか、ループ内の EXIT 文または RETURN 文が実行されるまで、その中に含まれる文を実行するループを作成します。

構文

```
WHILE boolean-expression LOOP
  statements
END LOOP;
```

上記において、

boolean-expression

ブール式。

statements

一連の 1 つ以上の PL/MX 文。

例

ブール変数に基づいてループする WHILE 文。

```
WHILE doing_work LOOP
  ---ここで何らかの作業を実行します。
END LOOP;
```

変数の値に基づいてループする WHILE 文。

```
WHILE counter < limit LOOP
  ---何らかの作業を実行します...
  counter := counter + 1;
END LOOP;
```

PL/MX 内部の SQL 文

PL/MX 手順または関数の内部で見つかる SQL 文は､SQL/MX データベースに対して照会を行います。手順や関数により使用されるテーブルは、手順や関数がコンパイルされる前に、作成される必要があります。PL/MX 内部の SQL 文はテーブルを作成することができません。

手順や関数のコンパイルは、モジュール定義ファイル (MDF ファイル) を作成します。それから、MDF ファイルは、SQL/MX コンパイラを使用して、コンパイルされます。PL/MX コンパイルからのものだけではない、エラーメッセ―が表示される場合もありますが、さらに、その SQL/MX コンパイラからもエラーメッセージが表示される場合もあります。

デフォルトでは、PL/MX 内部の SQL 文に手順や関数を入れることは許可されません。SQL 文を許可するには､スタンドアロン PL/MX コンパイラを実行している場合の *sql_usage* オプションを指定します。それぞれの SQL 文は、*sql_usage* オプション値に基づき許可されます。

-sql_usage パラメータ値	意味
none	組み込み SQL 文は許可されません。デフォルト動作です。
contains	CLOSE、および CURSOR が許可されます。
reads	上記に加えて、FETCH、OPEN、および SELECT が許可されます。
modifies	上記に加えて、DELETE、INSERT、および UPDATE が許可されます。

サブプログラムが SQL 文を使用するようになる場合､PL/MX コンパイラは、そのサブプログラムがそのデータベースにその後格納されることになるため、そのサブプログラムの完全修飾名を認識する必要があります。サブプログラム名には、1 つ、2 つ、または 3 つのコンポーネントを持つ場合があります。左から右への順で、これらのコンポーネントは、カタログ、スキーマ、および、そのサブプログラムに関するオブジェクト名を参照します。3 つのコンポーネントが指定されるような場合は、この名前は「完全修飾」です。それが 3 つに満たない場合、不明なコンポーネントはそのカタログとスキーマに関するデフォルト名で埋められます。スタンドアロンのコンパイラ―の場合は、これらは、- CATALOG および - SCHEMA のオプションにより指定されます。指定しない場合、そのデフォルト値は CAT および SCH となります。

例

ブロックの宣言内部で･･･

```
CURSOR C IS SELECT J FROM T WHERE I >= 2;
```

それから、そのブロックの文内部で･･･

```
OPEN C;
FETCH C INTO HostVariable;
CLOSE C;

    INSERT INTO CAT.SCH.T VALUES (HostVariable, 100);
    DELETE FROM CAT.SCH.T WHERE I = 17;
    UPDATE CAT.SCH.T SET J = 102 WHERE I = 18;
    SELECT J INTO HostVariable FROM CAT.SCH.T WHERE I = 18;
```

HostVariable は、PL/MX サブプログラム内部の変数を参照します。SQL 文に依存し、そのホスト変数は、その SQL 文の入力、または 出力である可能性があります。PL/MX による SQL 文がテーブルを変更する場合は、その PL/MX サブプログラムは トランザクション内部からのみ起動させる必要があります。組込み SQL 文が エラーを発生させる場合、「**例外処理**」で説明されるように例外が発生します。

組み込み関数

この章には、PL/MX 手順と関数の内部で使用できる組み込み関数が一覧表示されます。関数は、SQL/MX により対話式に提供される組込み関数のサブセットに対応します。一般に、セマンティックは類似しています。入力パラメータは、必要に応じてタイプ変換が行われるため、リストされているものと正確に一致しない場合があります。組み込み関数の戻り値のタイプは、結果が PL/MX 手順または関数内でどのように処理される方法を示します。詳細については、「SQL/MX リファレンスマニュアル」を参照してください。

パラメータが存在しない場合は、空のカッコまたは空のパラメータリストを使用します。

function	説明
ASCII (s)	*s* は文字列です。戻り値は、*s* 内の最初の文字に関する ASCII コードを表示する符号なし 32 ビット整数です。
CHAR (i)	*i* は整数です。戻り値は、ASCII コード値を持つ文字長 1 の CHAR です。
CHAR_LENGTH (s)	*s* は文字列です。戻り値は、その文字列の長さを表示する符号なし 32 ビット整数です。 CHARACTER_LENGTH は CHAR_LENGTH に関する同義語です。
COALESCE (A, B, C, ⋯)	NULL でないものを見つけるまでパラメータを 1 つずつ評価し、それを返します。すべてが NULL の場合、最終的な値は NULL です。戻り値の型は、すべての A、B、C、⋯のタイプに変換できるようなタイプです。
CONCAT (s1、s2)	s1 および s2 を連結します。
CONVERTTIMESTAMP (i)	*i* は、特定の開始時間から秒の特定の秒数を代表する 64 ビット整数型ユリウス日形式タイムスタンプです。それはタイムスタンプに変換されます。
CONVERTTOHEX (x)	*x* は任意のタイプにすることができます。それは *x* の内部形式の 16 進表現に変換されます。
CURRENT (i)	現在の (ローカル) 時間の TIMESTAMP を返します。*i* はオプションの整数型パラメータでその精度 (マイクロ秒での秒数) を表示します。指定しない場合、それはデフォルト値の 6 となります。 CURRENT_TIMESTAMP は、CURRENT の同義語です。
CURRENT_DATE	現地時間で現在の DATE を返します。
CURRENT_TIME (i)	ローカル時間の現在の TIME を返します。*i* はオプションの整数型パラメータでその精度 (マイクロ秒での秒数) を表示します。指定しない場合、それはデフォルト値の 0 となります。

表は続く

DATE_ADD (d, i)	*d* は datetime であり、*i* は interval です。interval は datetime を加算し、同じたタイプの datetime を返します
DATEADD (unit, n, d)	*n* は numeric であり、*d* は datetime です。この *unit* (単位) は、以下の表示形式の内の 1 つとなります。 · YEAR YY YYYY · MONTH M MM · DAY D DD · HOUR H HH · MINUTE MI · SECOND S SS *n* は、別の同じタイプの datetime を取得するために、定義された *unit* に加算される *d* の数を表示します。
DATE_SUB (d, i)	*d* は datetime であり、*i* は interval です。interval は datetime から減算され、同じたタイプの datetime を返します
DATEFORMAT (d, x)	*d* は、それは文字列に変換される datetime です。この *x* パラメータは要求されますが、無視されます。
DAY (d)	*d* は、DATE または TIMESTAMP です。32 ビットの符号なし整数が、*d* により指定された月の日に対応する 1 ～ 31 の値を付与されて返されます。 DAYOFMONTH は DAY の同義語です。
DAYOFWEEK (d)	*d* は、DATE または TIMESTAMP です。32 ビットの符号なし整数が、d により指定された週の日に対応する 1 ～ 7 の値を付与されて返されます。例えば、1 は日曜日、2 は月曜日です。
DECODE (E { , S , R } ··· [, D])	CASE 式 `CASE E { WHEN S THEN R } … [ ELSE D ] END` と同一です。この関数は、評価 E (式) を評価します。それから、E と等しいものが見つかるまで検索を続け、それぞれの S (検索) を評価してから、対応する R (結果) を返します。E と等しい S が見つからず、D (デフォルト) が存在するなら、D を返します。D が存在しないなら、結果は NULL となります。戻り値のタイプ、すべての A、B、C、··· のタイプに変換できるようなタイプです。
EXTRACT (unit FROM x)	この *unit* は DATEADD に関するものと同じです。*x* は datetime または interval です。*x* 内部で指定された値を返します。戻り値は NUMERIC であり、その *unit* が SECOND である場合、小数部分を持つことができます。

表は続く

HOUR (t)	*t* は TIME または TIMESTAMP です。32 ビットの符号なし整数が、*t* の時間コンポーネントに対応する 0 ～ 23 の値を付与されて返されます。
IFNULL (a, b)	*a* と *b* は任意のタイプの式を取ることができます。*a* が NULL になるならば、この式は *b* に対して評価します。*a* が NULL でならないなら、この式は *a* に対して評価し、*b* に対して評価されません。戻り値のタイプは *a* と *b* のタイプにそれぞれが変換できるようなタイプです。 ISNULL は IFNULL の同義語です。NVL は IFNULL の同義語です。
JULIANTIMESTAMP (t)	*t* は TIMESTAMP です。それは、特定の開始時間からの特定の秒の秒数を代表する 64 ビット整数型ユリウス日形式の timestamp です。
LAST_DAY (d)	*d* は、DATE または TIMESTAMP です。*d* と同じ月の最終日に関する DATE を返します。
LCASE (s)	*s* は文字列です。戻り値は文字列と同じですが、小文字に変換します。 LOWER は LCASE の同義語です。
LEFT (s, i)	*s* は文字列であり、*i* は整数です。戻り値は、*s* の最初の *i* 個の文字から構成される文字列です。
LTRIM (s1 [, s2])	*s1* と *s2* は文字列です。戻り値は、*s1* の先頭から文字が削除されることを除き、*s1* と同じ文字列です。これらの文字は *s2* 内部で検出される必要があります。*s2* が指定されることがない場合は、*s2* はスペースのみで構成される文字列のデフォルト値になります。先行のスペースは *s1* から削除 されます。
MINUTE (t)	*t* は TIME または TIMESTAMP です。32 ビットの符号なし整数が、*t* の時間コンポーネントに対応する 0 ～ 59 の値を付与されて返されます。
MOD (i1, i2)	*i1* および *i2* が符号なし 32 ビット整数である場合、mod 関数は、それを符号なし量に対して適用されるものとして計算されます。それ以外の場合、*i1* および *i2* を同じタイプの整数にする変換が発生することになります。mod 関数は、それを符号付き量に対して適用されるものとして計算されることになります。結果は 64 ビット符号付き量として処理されます。
MONTH (d)	*d* は、DATE または TIMESTAMP です。32 ビットの符号なし整数が、*d* により代表される月に対応する 1 ～ 12 までの値を付与されて返されます。

表は続く

MONTHS_BETWEEN (d1, d2)	月の数を戻すための数値を戻します。*d1* から *d2* に対して、前の月数を表示するために NUMERIC を返します。*d1* が *d2* より以前である場合、その結果は負となります。
NULLIF (a, b)	*a* と *b* は、これらに相当する、任意のタイプの式をすることができます。*a* = *b* になるならば、NULL が返され、それ以外の場合は、*a* が返されます。この戻り値のタイプは *a* と同じタイプです。
NULLIFZERO (a)	*a* は、0 に対して比較することができる任意のタイプであることができます。*a* = 0 になるならば、NULL が返され、それ以外の場合は、*a* が返されます。この戻り値のタイプは *a* と同じタイプです。
NVL2 (a, b, c)	*a*、*b*、および *c* は任意のタイプであることができます。*a* が NULL になるならば、*c* が返され、それ以外の場合は、*b* が返されます。それぞれの場合は、*b* および *c* の内のその他の 1 つは評価されません。戻り値のタイプは *b* と *c* のタイプにそれぞれが変換できるようなタイプです。
POSITION (s1 IN s2)	*s1* と *s2* は文字列です。この関数は、*s2* のサブストリングとして、*s1* を検索し、符号なし 32 ビット整数を返します。*s1* が *s2* のサブストリングとなるならば、その戻り値は、*s2* の起首の文字が位置 1 と見なされ、*s1* が *s2* 内で開始される状況で、その位置を返します。*s1* が *s2* のサブストリングではない場合、この関数は 0 を返します。
REPEAT (s, i)	*s* は文字列であり、*i* は整数です。結果は、*s* が *i* 回連結された 文字列です。
REPLACE (s1, s2, s3)	*s1*, *s2*、および *s3* は文字列です。戻り値は、*s1* 内部での *s2* の出現のそれぞれが *s3* により置き換えられることを除外し、*s1* と同じ文字列です。
RIGHT (s, i)	*s* は文字列であり、*i* は整数です。戻り値は、*s* の最後の *i* 個の文字から構成される文字列です。
ROUND (n, i)	*n* は NUMERIC であり、*i* はオプション整数型パラメータです。戻り値は、*i* により指定される精度に *n* が丸められている状況の NUMERIC です。例えば、*i* が -3 となるならば、*n* は、小数点以下 3 桁に丸められることになります。*i* は負である可能性があります。例えば、*i* が -3 となるならば、*n* は 1000 の倍数に丸められることになります。指定されない場合、*i* はデフォルト値の 0 となります。

表は続く

RTRIM (s1 [, s2])	*s1* と *s2* は文字列です。戻り値は *s2* 内部でそれらの文字が検出される限り、文字が *s1* の末尾から削除されることを除外し、*s1* と同じ文字列です。*s2* が指定されることがない場合は、*s2* はスペースのみで構成される文字列のデフォルト値になります。後続のスペースは *s1* から削除されます。
SECOND (t)	*t* は TIME または TIMESTAMP です。32 ビットの符号なし整数が、*t* の秒コンポーネントに対応する 0 ～ 59 の値を付与されて返されます。
SPACE (i)	*i* は整数です。戻り値は、*i* 個のスペースを含んでいる文字列です。
SQLCODE	「**例外処理**」を参照してください。
SUBSTRING (s, i1, i2)	*s* は文字列であり、*i1* と *i2* は整数です。戻り値は、s 内部の *i1* の位置から開始される *s* のサブストリングである文字列です。*s* の最初の文字は位置 1 とみなされ、その文字長は *i2* です。
TO_CHAR (n, s)	*n* は NUMERIC であり、s は 文字列です。戻り値は、パターン *s* に従い、文字列値に変換されている、*n* の中の文字列です。
TO_DATE (s1, s2, s3)	*s1* は、DATE 変換される文字列です。*s2* は *s1* の形式で記述される パターンであるオプション文字列パラメータです。*s2* が指定されることがない場合、*s1* がこの形式で与えられる必要があることを意味しており、それはデフォルト値の「YYYY-MM-DD」となります。*s2* が与えられるようになる場合、*s3* はオプションで与えられますが、それは無視されます。
TO_TIMESTAMP (s1, s2)	*s1* は TIMESTAMP に変換される文字列であり、*s2* はパターンです。
TRIM ([[LEADING | TRAILING | BOTH] [s2] FROM] s3)	*s2* と *s3* は文字列です。戻り値は、s2 から文字が削除されることを除外し、*s3* と同じ文字列です。LEADING が指定されている場合、文字は起首から削除されます。TRAILING が指定されている場合、文字は末尾から削除されます。両方が指定されている場合、文字は起首および末尾の両方から削除されます。これらの文字は *s2* 内部で検出される必要があります。*s2* が指定されることがない場合は、*s2* はスペースのみで構成される文字列のデフォルト値になり、即ち、後続のスペースは、起首、末尾またはその両方から削除されることになります。LEADING、TRAILING、または BOTH が指定されないような場合は、そのデフォルト値は BOTH となります。
UCASE (s)	*s* は文字列です。戻り値は文字列と同じですが、大文字に変換します。 UPPER および UPSHIFT は UCASE の同義語です。

表は続く

YEAR (d)	*d* は、DATE または TIMESTAMP です。32 ビットの符号なし整数が、*d* により代表される年に対応する 0 ～ 1999 までの値を付与されて返されます。
ZEROIFNULL (a)	*a* は、0 の値が存在することが意味を持つような任意のタイプであることができます。*a* が NULL になるならば、0 が返され、それ以外の場合は、*a* が返されます。この戻り値のタイプは *a* と同じタイプです。

例外処理

実行時に発生して実行を停止させるエラーは「例外」と呼ばれ、「発生」していると言われます。いくつかの例は、ゼロによる除算か、PL/MX 内部の SQL コールを実行中にエラーかです。例外の各タイプには名前とその「SQLCODE」と呼ばれる、内部番号があります。明示的に以下の文により、例外を発生させることができます。

```
RAISE  exception-name ;
```

デフォルトでは、例外があれば実行を終了させます。しかしながら、以下の DECLARE/BEGIN 拡張構文を使用して、サブプログラム内部で例外を処理することができます。

```
[ DECLARE
    { variable_declaration } … ]
BEGIN
    { statement } …
EXCEPTION
    { WHEN  exception-name  [ OR  exception-name ] … THEN
        { statement }  … } …
    [ WHEN OTHERS
        { statement } … ]
END ;
```

最も外側のサブプログラムの開始を含む関連の宣言があるかどうかにかかわらず、例外句は任意の BEGIN で使用できる。

例外が発生し、それを処理するどの BEGIN 内部でもない場合は、サブプログラムは終了します。関数文が最後まで実行してしまう場合は、END_OF_FUNCTION 例外が発生します。これは自体をその関数内部で処理されることはできず、代わりに関数の呼び出し元により確認されます。

DECLARE の元で定義されている変数は以下に示す内部で可視化されます。

- BEGIN の文
- その BEGIN 内部で発生した例外を処理する WHEN 句の文

以下のような場合は、実行はその END 以降も続行されます。

- 実行が BEGIN 文の文リストの最後に到達している。そして例外が発生していない。
- 例外が発生し、そして WHEN 句がそれを処理した。さらにその WHEN 句の文リスト内では何の追加の例外が起きなかった。

注記: 以下のような場合には、DECLARE 文または EXCEPTION 句と共に使用される BEGIN 文内部で例外が発生しているとはみなされません。

- 対応する BEGIN に先立ち、DECLARE 文の宣言を実行中に、例外が発生しています。
- EXCEPTION 句の内部で例外が発生しています。

このような例外を処理するための試行は、以降の囲み BEGIN 文から開始されます。

例外を処理する任意の BEGIN 内部に例外がない場合は、そのサブプログラムが終了します。関数のメイン BEGIN 文リストにある文が暴走してしまう場合は、END_OF_FUNCTION 例外が発生します。その最も外部の BEGIN の文リスト後方で、サブプログラムの最も外部レベルの、EXCEPTION 句のサブプログラム下の WHEN 句で、その例外を処理することができます。しかしながら、このような最も外側にある句を最後に実行すると、処理できない END_OF_FUNCTION 例外が発生します。

SQLCODE と命名された組み込み関数が、その例外に関連付けられた 64 ビット量の「例外番号」を返します。SQLCODE が任意の WHEN 句内部の入れ子になっていない場所で呼び出されることになる場合は、それは 0 を返すようになります。WHEN 句内部では、例外名を指定せずに RAISE を使用することができます。例外が 1 か所の外部スコープに伝達されます。効果は、例外の数に対応する例外名 RAISE を呼び出すことと同等です。例外番号は SQLCODE がその時点で返すことになるものです。

入れ子になった BEGIN/EXCEPTION/END ブロックの例

```
BEGIN
BEGIN
         1
EXCEPTION
    WHEN
     2
END
EXCEPTION
    WHEN
 3
         BEGIN
  4
  EXCEPTION
         WHEN
  5
  END
  6
END
```

例は、以下に示す構文です。

- 外部 BEGIN/EXCEPTION/END
- 外部 BEGIN 下に入れ子になったの BEGIN/EXCEPTION/END
- 外部 BEGIN/EXCEPTION/END の 1 つの WHEN 句内の下で入れ子になった BEGIN/EXCEPTION/END

例外が 1 で発生するようになる場合、2 の WHEN によるなどのように、例外がそのレベルで処理される可能性があります。または例外は、3 での WHEN によるなどのように、さらに外部で処理される可能性があります。2 での WHEN 内部で文に関して発生する例外は、次の外部レイヤー内で処理されることができます。例外が 4 で発生するようになる場合、5 の WHEN によるなどのように、例外がそのレベルで処理できる場合があります 位置 4 は外部 BEGIN に関する文リスト内部ではなく、それは外側 EXCEPTION 句により処理されることはできません。

SQLCODE 関数が 6 で呼び出されるようになる場合、関数は発生した例外を返すようになります。例えば、3 の WHEN 制御を渡すための制御は、1 または 2 にてです。例外は、生成されている直近の例外ではない場合があります。例えば、3 にある WHEN が制御を取得するようになる場合、この WHEN に関する分が実行されるようになります。例外が、現在の WHEN 内部で入れ子になった 4 の BEGIN により開始されます。例外が 4 で発生する場合、例外が 5 の WHEN により処理できる場合があります。何もその他の例外が現在の WHEN 句内で発生していない場合、実行は、6 で、END の後で、続行することになります。制御は、3 で開始されている WHEN 句に復帰します。そして、SQLCODE が 6 で呼び出される場合、SQLCODE は現在の WHEN 句まで到達する実行を発生させる例外を返します。けれども、4 で発生した例外は、より直近で発生したものなのです。

例外

例外番号は 16 ビットに収まり、負の数です。

SQLCODE	名前	説明
-17400	ACCESS_INTO_NULL	実行するプログラムは、初期化されていないオブジェクトの属性に対して値を割り当てることを試行します。
-17401	CASE_NOT_FOUND	CASE 文の WHEN 句内のいずれの選択肢も選択されない場合、何の ELSE 句も存在しません。
-17403	CURSOR_ALREADY_OPEN	プログラムは既に開いているカーソルを開くことを試行します。
-17404	DUP_VAL_ON_INDEX	実行するプログラムは、一意のインデックスにより制約されるデータベース内に重複する値を格納することを試行します。
-17405	END_OF_FUNCTION	関数は最も外部レベルの文リストの最後まで実行された。
-17406	INVALID_CURSOR	実行するプログラムはまだ開いていないカーソルを閉じるなどの無効なカーソル操作を試行します。
-17407	INVALID_NUMBER	算術計算またはデータ変換の際にオーバーフローが発生しました。または、1 科の変換が失敗しました。または、文字式の評価中に文字列のオーバーフローが発生しました。または、SQL 文内では、対象文字列が有効な数値を代表しないため、文字列から数値への変換が失敗することになります。または、一括 FETCH での LIMIT 句式が正の数値に対して評価されない。
-17409	NO_DATA_FOUND	A SELECT_INTO 文は何も列を返しません。
-17412	PROGRAM_ERROR	内部エラーが発生しました。
-17415	STORAGE_ERROR	プログラムが、メモリが不足しているか、またはメモリが壊れているかです。
-17419	TIMEOUT_ON_RESOURCE	リソースを待機中にタイムアウトが発生しました。
-17420	TOO_MANY_ROWS	SELECT INTO 文は 1 つ以上の列を返します。
-17421	VALUE_ERROR	変換または制約エラー (ただし SQLCODE-17407 に関する説明の場合とは違います)。
-17422	ZERO_DIVIDE	実行するプログラムは、ゼロにより数値の除算を試行します。

さらに、以下に示すステップを使用して、内部例外番号の内の 1 つの新しい名前を定義することもできます。

```
identifier EXCEPTION ;
pragma EXCEPTION_INIT (identifier , value ) ;
```

値はリテラルで、マイナス符号が先行し、そのテーブル内の数字の内の 1 つに一致しています。それから、RAISE 文または WHEN 句内の、例外番号を代表するために、この識別子が使用されることができます。後続のプラグマ EXCEPTION_INIT なしで、EXCEPTION が宣言される場合、それはエラーとなります。

UDF の管理

UDF は、データベースの一部となるカスタムビジネスロジックを実装する方法を提供します。他のプログラミング言語の関数と同じように、UDF はパラメーターを受け取ってスカラー値を返します。PL/MX UDF も、ソースコードとオブジェクトコードの両方がデータベース内で完全に保存および管理されるという点で、真のストアドファンクションです。そのため、UDF はストアドファンクションと呼ばれることもあります。
L17.02 以降のバージョンの SQL/MX リリース 3.5 では、UDF の作成、使用、および管理がサポートされています。

UDF は SQL/MX 専用の機能です。SQL/MP テーブル、エイリアス、または NAMETYPE NSK での UDF の使用はサポートされず、動作は未定義です。

UDF とそのプロパティの管理

以下の管理操作は、UDF 上でサポートされています。これらの操作を実行するには、SQL NonStop/MX にログインします。

- UDF の作成 - CREATE FUNCTION 文を実行します。
- UDF のドロップ - DROP FUNCTION 文を実行します。
- UDF を暗号化 - GRANT EXECUTE, GRANT... を実行 (UDF 実行可能形式またはソースの場合)、REVOKE EXECUTE、または REVOKE....(UDF 実行可能形式またはソースの場合) の文
- UDF の所有権譲渡 - GIVE Object コマンドを実行します。
- UDF の閲覧 - SHOWDDL コマンドを実行します。

文およびコマンドの詳細については、「SQL/MX リファレンスマニュアル」を参照してください。

前提条件

- UDF を作成するには、ターゲットスキーマに対する CREATE 特権を持っているか、または SUPER.SUPER ユーザーとしてログインしている必要があります。
- UDF をドロップするには、その関数を含むスキーマに対する DROP 特権を持っているか、または SUPER.SUPER ユーザーとしてログインしている必要があります。
- UDF に特権を付与するには、ユーザーは WITH GRANT OPTION を付与された特権を保持している必要があります。セキュリティ管理者が有効で、ユーザーがセキュリティ管理者としてログインしている場合、またはセキュリティ管理者が有効になっておらず、ユーザーが SUPER.SUPER ユーザーとしてログインしている場合も、特権を付与することができます。
- UDF から特権を取り消すには、特権の付与者でなければなりません。セキュリティ管理者が有効で、ユーザーがセキュリティ管理者としてログインしている場合、またはセキュリティ管理者が有効になっておらず、ユーザーが SUPER.SUPER ユーザーとしてログインしている場合も、特権を取り消すことができます。
- UDF のソースを表示するには、その関数の SOURCE に対する SELECT 特権を保持する必要があります。

クエリでの UDF の使用

メインクエリの選択リストまたはメインクエリの WHERE 句で UDF を使用できます。

前提条件

- UDF を SQL クエリにコンパイルするには、UDF を CREATE FUNCTION 文で事前に作成しておく必要があります。
- UDF を含む SQL クエリを実行するには、クエリを実行するユーザーは、クエリで参照されるすべての UDF に対して EXECUTE 特権を持っていなければなりません。また、ユーザーは、クエリで参照されるすべての UDF に対して SELECT EXECUTABLE 特権を持っていなければなりません。これらの特権は、クエリの実行時に通常必要となる特権（クエリで参照されているテーブルに対する適切な特権など）に加えて適用されます。通常、UDF に対する EXECUTE 特権を持つユーザーは、その UDF に対して SELECT EXECUTABLE 特権も持ちます。これは、EXECUTE 特権が付与または取り消しされたときに NonStop SQL/MX によって自動的に管理されます。
- UDF を含む SQL クエリは、トランザクションのコンテキスト内で実行する必要があります。UDF を含むクエリがトランザクションを使用せずに実行された場合、NonStop SQL/MX はクエリに代わって暗黙のトランザクションを開始します。
- UDF の各呼び出しは、10 億の命令の実行に限定されます。1 回の呼び出しで 10 億を超える命令を実行しようとすると、クエリは SVM 命令制限例外で終了します。

SQL クエリでの NonStop SQL/MX UDF の使用

NonStop SQL/MX では UDF を次の方法で使用できます。

- メインクエリの選択リストでの UDF の使用
- メインクエリの WHERE 句での UDF の使用

メインクエリの選択リストでの UDF の使用

1. NonStop SQL/MX にログオンします。
2. クエリの選択リスト部分に 1 つ以上の UDF を含む SELECT 文をコンパイルします。
3. コンパイルされた文を実行します。

SELECT

次のテキストは、SELECT 参照ドキュメントの要約形式です。SELECT 文の完全なドキュメントについては、SQL/MX Reference Manual を参照してください。

構文

```
SELECT [[ANY N ] | [FIRST N ]] [ALL | DISTINCT] select-list

select-list is:
  * | select-sublist [,select-sublist]...
```

```
select-sublist is:
  corr.* | [corr.] single-col [[AS]name] | [col-expr [[AS]name]
```

選択リストに関する考慮事項

- *col-expr* は単一の列名または派生列です。派生列は SQL 値式です。そのオペランドは、数値、文字列、日時またはインターバルリテラル、列、列に定義された組み込み関数（集約関数を含む）、列に定義されたユーザー定義関数、CASE 式、または CAST 式です。col-expr で指定された単一の列は、FROM 句で指定されたテーブルまたはビューからのものでなければなりません。サブクエリ内で UDF を使用することはサポートされていません。
- 集約された組み込み関数の戻り値は、型に互換性がある場合は、UDF への引数として渡すこともできます。
- UDF の戻り値を別の UDF または組み込み関数に渡したり、算術式で UDF の戻り値を使用したりすることは、まだサポートされていません。
- クエリがコンパイルされた後で、クエリで参照されている UDF が変更された（たとえば、ドロップされて再作成された）場合、クエリを再コンパイルする必要があります。AUTOMATIC_RECOMPILATION CQD が 'ON'の場合、再コンパイルが自動的に実行されます。

例文

factorial という名前の UDF がスキーマ `cat.sch` に作成されたとします。

```
SET SCHEMA CAT.SCH;

CREATE FUNCTION factorial(n in INTEGER) RETURN INTEGER
LANGUAGE PLMX
IS
  rslt      PLS_INTEGER;
  loopCount PLS_INTEGER;

 begin
   rslt    := 1;
   loopCount := n;

   while loopCount > 1 loop
     rslt := rslt * loopCount;
     loopCount := loopCount - 1;
   end loop;
   return rslt;

  end factorial;
/
```

また、データベースに次の値を持つテーブルが存在するとします。

```
I             I2
-----------------  ---------------------

          1          100
          2          200
          3          120
```

次のクエリがコンパイルされ、サンプルテーブルに対して実行されます。

```
SELECT factorial(i) from datatable;
```

出力例

```
(EXPR)
-----------------

          1
          2
          6

--- 3 行が選択されました。
```

例文

前の例の factorial UDF およびデータテーブルがあると仮定します。次のクエリがコンパイルされて実行されます。

```
SELECT i as "i", factorial(i) as "factorial(i)" from datatable;
```

出力例

```
i             factorial(i)
---------------  ---------------------

              1             1
              2             2
              3             6

--- 3 行が選択されました。
```

例文

前の例の factorial UDF およびデータテーブルがあると仮定します。次のクエリがコンパイルされて実行されます。

```
SELECT factorial(max(i)) as "factorial(max(i))" from datatable;
```

出力例

```
factorial(max(i))
---------------------------------

                          6

--- 1 行が選択されました。
```

メインクエリの WHERE 句での UDF の使用

1. NonStop SQL/MX にログオンします。
2. クエリの where 句に 1 つ以上の UDF を含む SELECT 文または DELETE 文をコンパイルします。
3. コンパイルされた文を実行します。

   SELECT 文または DELETE 文について詳しくは、SQL/MX Reference Manual を参照してください。

検索条件

検索条件は、行に条件を適用した結果に応じて、表またはビューから行を選択するために使用されます。条件は、OR、AND、および NOT 演算子で結合された述語で構成されるブール式です。

次の場所で検索条件を使用できます。

- SELECT、DELETE、または UPDATE 文の WHERE 句
- SELECT 文の HAVING 句
- CASE 式の検索形式
- 結合を含む SELECT 文の ON 句
- CHECK 制約
- ROWS SINCE シーケンス関数

UDF は、SELECT 文または DELETE 文の WHERE 句の特定の形式で使用できます。UPDATE 文でのユーザー定義関数の使用はサポートされていません。

述語

述語は、BETWEEN、比較、EXISTS、IN、LIKE、NULL、または定量化された比較述語です。述語は、行を選択するために満たす必要がある条件を指定します。UDF は、SELECT 文および DELETE 文の BETWEEN および比較述語で使用できます。

検索条件の例

AND 演算子で別の比較述語に結合された比較述語内に UDF を含む検索条件を使用して、行を選択します。

```
select * from datatable where i = factorial(i2) and i >= 3;

I                    I2
-------------------  ------------------

                  24                   4
```

--- 1 行が選択されました。

比較述語に UDF を含む検索条件を使用して行を削除します。

```
delete from datatable where factorial(i2) >= 120;;
```

--- 2 行が選択されました。

PL/MX エラー

PL/MX ではコンパイル時エラーおよび実行時エラーの 2 種類のエラーが発生します。

コンパイル時エラーは、PL/MX 関数のコンパイル中に構文解析および意味解析が実行されたときに検出されます。実行時エラーが発生すると、トラップ条件が出されます。トラップ条件により、通常は実行時例外が発生します。詳細については、「SQL/MX メッセージマニュアル」を参照してください。

スタンドアロンコンパイラ

PL/MX コンパイラ、PLMXCMP は、NonStop SQL/MX サブシステムが CREATE PROCEDURE、または CREATE FUNCTION 文を実行する場合に NonStop SQL/MX サブシステムにより自動的に開始され、管理されるようになります。しかしながら、コンパイラは、OSS 上のプログラムとして、分離して実行することができます。

Syntax

```
 PLMXCMP  options

options  ::=  option  options

option  ::=  -CATALOG  string
    -DC_DATE { 0 | 1 }


 -plmx_ccflags  ccflags_string
 -SCHEMA  string
 -SET DATASPACE_SIZE value
 -SET MAX_COMPLEX_OBJECTS value
 -SET MAX_INSTRUCTIONS value
 -source_dump  { on | off }
 -sql_usage   option

 input_filename
```

Parameters

input_filename

これは、PL/MX 入力ソースファイルの名前です。*input_filename* は必須です。

DC_DATE

この DATE タイプの 2 つのバージョンがサポートされます。デフォルトである、オプション -DC_DATE 0 とは、SQL/MX DATE が現在のサブプログラムにより使用されることを意味しています。オプション -DC_DATE one とは、DATE type2 が現在のサブプログラムにより使用されることを意味しています。DATE の両方のタイプを同じサブプログラム内で使用することはできません。このマニュアルでは、明記しない限り、DATE 情報は DATE の両方のタイプに適用されます。

source_dump

ソースファイルのコピーを標準出力に書き込む [on] かどうかを示します。このデフォルト設定は、[on] です。

-SET

-SET オプションにより設定される数量項目は、ランタイム構成パラメータと呼ばれます。以下に示す数量は、お互いに独立しています。

- DATASPACE_SIZE は、サブプログラムの実行中、そのスタックに関して割り当てられることになるストレージスペースの容量です。スタック上にある、それぞれの LARGEINT または DOUBLE PRECISION 変数ごとに関して対応している 8 バイトに加え、その他のタイプの変数に関して、4 バイトが追加されています。指定されているこの値は、1024 により乗算されます。指定できる最小値は、

またデフォルト値でもある、32 であり、その最大値は 1024 です。最小サイズ (またはデフォルト値) は、32 KB であり、その最大値は 1 MB です。

- MAX_COMPLEX_OBJECTS は、サブプログラムの実行中、同時に存在することができる複合データ オブジェクトの最大数です。複雑なデータ オブジェクトには、CHAR、DATE、INTERVAL、NUMERIC、TIME、TIMESTAMP、VARCHAR、および VARCHAR2 のタイプ、の変数が含まれます。MAX_COMPLEX_OBJECTS の設定は、同時にスタック上に存在することができる複合データオブジェクト変数の合計数に対応しています。指定できる最小値は、64 であり、そのデフォルト値は 128 であり、またその最大値は 4096 です。
- MAX_INSTRUCTIONS は、サブプログラムが終了される前に、内部「SQL 仮想マシン」内部で実行されることができる命令の最大数です。指定できる最小値は、100 であり、そのデフォルト値は 10 億 (10^9) であり、またその最大値は 2^{64}-1 です。PL/MX 文は、そのような命令のほんの数個または数十個にだけ対応している可能性があります。MAX_INSTRUCTIONS を使用し、無限ループを持つ暴走サブプログラムを制限することができます。

さらに、以下に示す PL/MX ソースコード内のプラグマを使用することにより、ランタイム構成パラメータを設定することができます。

```
pragma SET (DATASPACE_SIZE, value);
   pragma SET (MAX_COMPLEX_OBJECTS, value);
   pragma SET (MAX_INSTRUCTIONS, value);
```

ソースコード内のプラグマにより設定される値は、より高い優先順位を持ちます。

-plmx_ccflags

sql-usage

「**PL/MX 内部の SQL 文**」を参照してください。

例

/home/me/myudfsource.plmx ファイルの構文をコンパイルし、以下に示すように、compileThat オプションを TRUE に設定します。

```
PLMXCMP –plmx_ccflags “compileThat=TRUE” /home/me/myudfsource.plmx
```

Web サイト

全般的な Web サイト

Hewlett Packard Enterprise Information Library

http://www.hpe.com/info/EIL

Hewlett Packard Enterprise サポートセンター

http://www.hpe.com/support/hpesc

Contact Hewlett Packard Enterprise Worldwide

http://www.hpe.com/assistance

サブスクリプションサービス/サポートのアラート

http://www.hpe.com/support/e-updates-ja

Software Depot

http://www.hpe.com/support/softwaredepot

カスタマーセルフリペア

http://www.hpe.com/support/selfrepair

L シリーズのマニュアル

http://www.hpe.com/info/nonstop-ldocs

J シリーズのマニュアル

http://www.hpe.com/info/nonstop-jdocs

上記以外の Web サイトについては、**サポートと他のリソース**を参照してください。

サポートと他のリソース

Hewlett Packard Enterprise サポートへのアクセス

- ライブアシスタンスについては、Contact Hewlett Packard Enterprise Worldwide の Web サイトにアクセスします。

  http://www.hpe.com/assistance

- ドキュメントとサポートサービスにアクセスするには、Hewlett Packard Enterprise サポートセンターの Web サイトにアクセスします。

  http://www.hpe.com/support/hpesc

ご用意いただく情報

- テクニカルサポートの登録番号（該当する場合）
- 製品名、モデルまたはバージョン、シリアル番号
- オペレーティングシステム名およびバージョン
- ファームウェアバージョン
- エラーメッセージ
- 製品固有のレポートおよびログ
- アドオン製品またはコンポーネント
- 他社製品またはコンポーネント

アップデートへのアクセス

- 一部のソフトウェア製品では、その製品のインターフェイスを介してソフトウェアアップデートにアクセスするためのメカニズムが提供されます。ご使用の製品のドキュメントで、ソフトウェアの推奨されるソフトウェアアップデート方法を確認してください。
- 製品のアップデートをダウンロードするには、以下のいずれかにアクセスします。

  Hewlett Packard Enterprise サポートセンター

  http://www.hpe.com/support/hpesc

  Hewlett Packard Enterprise サポートセンター：ソフトウェアのダウンロード

  http://www.hpe.com/support/downloads

  Software Depot

  http://www.hpe.com/support/softwaredepot
- eNewsletters およびアラートをサブスクライブするには、以下にアクセスします。

http://www.hpe.com/support/e-updates-ja

- お客様の資格を表示したりアップデートしたり、契約や保証をお客様のプロファイルにリンクしたりするには、Hewlett Packard Enterprise サポートセンターの **More Information on Access to Support Materials** ページにアクセスします。

  http://www.hpe.com/support/AccessToSupportMaterials

重要: 一部のアップデートにアクセスするには、Hewlett Packard Enterprise サポートセンターからアクセスするときに製品資格が必要になる場合があります。関連する資格を使って HPE パスポートをセットアップしておく必要があります。

カスタマーセルフリペア（CSR）

Hewlett Packard Enterprise カスタマーセルフリペア（CSR）プログラムでは、ご使用の製品をお客様ご自身で修理することができます。CSR 部品を交換する必要がある場合、お客様のご都合のよいときに交換できるよう直接配送されます。一部の部品は CSR の対象になりません。Hewlett Packard Enterprise もしくはその正規保守代理店が、CSR によって修理可能かどうかを判断します。

リモートサポート（HPE 通報サービス）

リモートサポートは、保証またはサポート契約の一部としてサポートデバイスでご利用いただけます。リモートサポートは、インテリジェントなイベント診断を提供し、ハードウェアイベントを Hewlett Packard Enterprise に安全な方法で自動通知します。これにより、ご使用の製品のサービスレベルに基づいて、迅速かつ正確な解決が行われます。ご使用のデバイスをリモートサポートに登録することを強くおすすめします。

ご使用の製品にリモートサポートの追加詳細情報が含まれる場合は、検索を使用してその情報を見つけてください。

リモートサポートおよびプロアクティブケア情報

HPE 通報サービス

http://www.hpe.com/jp/hpalert

HPE プロアクティブケアサービス

http://www.hpe.com/services/proactivecare-ja

HPE プロアクティブケアサービス：サポートされている製品のリスト

http://www.hpe.com/services/proactivecaresupportedproducts

HPE プロアクティブケアアドバンストサービス：サポートされている製品のリスト

http://www.hpe.com/services/proactivecareadvancedsupportedproducts

保証情報

ご使用の製品の保証またはサーバー、ストレージ、電源、ネットワーク、およびラック製品の安全と準拠に関する情報に関するドキュメントを確認するには、下記の Web サイトを参照してください。

http://www.hpe.com/support/Safety-Compliance-EnterpriseProducts

追加保証情報

HPE ProLiant と x86 サーバーおよびオプション

http://www.hpe.com/support/ProLiantServers-Warranties

HPE エンタープライズサーバー

http://www.hpe.com/support/EnterpriseServers-Warranties

HPE ストレージ製品

http://www.hpe.com/support/Storage-Warranties

HPE ネットワーク製品

http://www.hpe.com/support/Networking-Warranties

規定に関する情報

安全、環境、および規定に関する情報については、Hewlett Packard Enterprise サポートセンターからサーバー、ストレージ、電源、ネットワーク、およびラック製品の安全と準拠に関する情報を参照してください。

http://www.hpe.com/support/Safety-Compliance-EnterpriseProducts

規定に関する追加情報

Hewlett Packard Enterprise は、REACH（欧州議会と欧州理事会の規則 EC No 1907/2006）のような法的な要求事項に準拠する必要に応じて、弊社製品の含有化学物質に関する情報をお客様に提供することに全力で取り組んでいます。この製品の含有化学物質情報レポートは、次を参照してください。

http://www.hpe.com/info/reach

RoHS、REACH を含む Hewlett Packard Enterprise 製品の環境と安全に関する情報と準拠のデータについては、次を参照してください。

http://www.hpe.com/info/ecodata

社内プログラム、製品のリサイクル、エネルギー効率などの Hewlett Packard Enterprise の環境に関する情報については、次を参照してください。

http://www.hpe.com/info/environment

ドキュメントに関するご意見、ご指摘

Hewlett Packard Enterprise では、お客様により良いドキュメントを提供するように努めています。ドキュメントを改善するために役立てさせていただきますので、何らかの誤り、提案、コメントなどがございましたら、ドキュメントフィードバック担当（**docsfeedback@hpe.com**）へお寄せください。この電子メールには、ドキュメントのタイトル、部品番号、版数、およびドキュメントの表紙に記載されている刊行日をご記載ください。オンラインヘルプの内容に関するフィードバックの場合は、製品名、製品のバージョン、ヘルプの版数、およびご利用規約ページに記載されている刊行日もお知らせください。

PL/MX と Oracle PL/SQL の違い

以下は PL/SQL でまだサポートされていない、言語エリアごとに整理された PL/MX での PL/SQL の機能の一部のサマリーです。

エリア	PL/MX 内に存在しない機能
組み込み関数	PL/MX をサポートする組み込み関数のリストを参照してください。
コンパイルの構造	グローバル変数パッケージ -- コンパイルは単なるサブプログラムであり、入れ子済みサブプログラムはありません。
例外処理	SQLERRM ユーザー定義例外。
入力/出力	PUT および DBMS_OUTPUT 内の PUT_LINE を除外する、入力/出力文。
オペレーター	IN、LIKE SET オペレーター。
プラグマ	EXCEPTION_INIT のみがサポートされます。
サブプログラム	パラメータ NOCOPY に関する IN パラメータ命名表記法に関するデフォルト値 (ただし、NOCOPY は IN OUT および OUT パラメータに関して暗黙指定されています)。関数が呼び出し元に戻らずに終了する場合、コンパイル時に再帰警告が出る可能性があります。
文	カーソル変数またはプレースホルダーへの割り当ては、ユーザー定義の手順と関数を呼び出します。PL/MX をサポートする埋め込み型 SQL 文の例を参照してください。
タイプ	BLOB、CLOB、NLOB、BFILE 複合型 (コレクションまたはレコードなど、浮動小数点型、LONG、RAW のマルチバイト文字およびその他のキャラクターセット (ASCII がサポートされています)。NCHAR、NVARCHAR2、NLS_COMP、NLS_SORT、および Oracle が提供する型は、%TYPE 属性 TIMEZONE ユーザー定義のサブタイプです。

さらに、以下は、PL/M 内にも存在しているが言語エリアごとに違いを整理した Oracle PL/SQL の一部の機能のサマリーです。

エリア	PL/SQL と Oracle PL/SQL での動作が異なる機能
例外処理	SQLCODE により返される異なる数。

表は続く

パラメータ	NOCOPY は常に IN OUT および OUT パラメータに関して暗黙指定されます。これらのパラメータは、値によるのではなく、参照により渡されます。
タイプ	DATE はデフォルトで SQL/MX DATE ですが、コンパイラオプションにより変更できます。DECIMAL は、38 ではなく、18 と同じ大きさの精度のみを持つことができます。 NUMERIC と DECIMAL は負のスケールをサポートしていません。 NUMERIC (または DECIMAL) (9, 0) は、最大精度およびスケールです。INTERVAL では、精度は秒単位で最大 6 秒間までのみ許可されますが、9 は許可されません。TIMESTAMP では、精度は最大で 6 秒間までのみ許可されますが、9 は許可されません。