



EDB Postgres Advanced Server

リリース **13**

Database Compatibility for Oracle Developers Reference Guide

2020 年 12 月 30 日

目次

第 1 章	Introduction	1
第 2 章	The SQL Language	3
2.1	SQL Syntax	3
2.1.1	Lexical Structure	3
2.1.2	Identifiers and Key Words	4
2.1.3	Constants	5
2.1.3.1	文字列定数	5
2.1.3.2	数値定数	5
2.1.3.3	他のタイプの定数	6
2.1.4	Comments	7
2.2	Data Types	9
2.2.1	Numeric Types	9
2.2.1.1	整数型	10
2.2.1.2	任意の精度番号	10
2.2.1.3	浮動小数点型	11
2.2.2	Character Types	12
2.2.3	Binary Data	13
2.2.4	Date/Time Types	14
2.2.4.1	区間タイプ	15
2.2.4.2	日付/時刻入力	16
2.2.4.2.1	日程	16
2.2.4.2.2	回	17
2.2.4.2.3	タイムスタンプ	17
2.2.4.3	日付/時刻出力	17
2.2.4.4	内部	18
2.2.5	Boolean Types	18
2.2.6	XML Type	18
2.3	Functions and Operators	20
2.3.1	Logical Operators	20
2.3.2	Comparison Operators	20
2.3.3	Mathematical Functions and Operators	22

2.3.4	String Functions and Operators	26
2.3.4.1	NULL との連結から生じる文字列テキストの切り捨て	33
2.3.4.2	SYS_GUID	36
2.3.5	Pattern Matching String Functions	37
2.3.5.1	REGEXP_COUNT	37
2.3.5.2	REGEXP_INSTR	38
2.3.5.3	REGEXP_SUBSTR	41
2.3.6	Pattern Matching Using the LIKE Operator	43
2.3.7	Data Type Formatting Functions	44
2.3.7.1	TO_CHAR、TO_DATE、TO_TIMESTAMP、TO_TIMESTAMP_TZ	46
2.3.7.1.1	日付と時刻の修飾子	49
2.3.7.2	IMMUTABLETO_CHAR (TIMESTAMP、format) 関数	51
2.3.7.3	TO_NUMBER	52
2.3.7.3.1	数値修飾子	52
2.3.8	Date/Time Functions and Operators	53
2.3.8.1	ADD_MONTHS	56
2.3.8.2	現在の日付/時刻	56
2.3.8.3	抽出物	58
2.3.8.4	MONTHS_BETWEEN	59
2.3.8.5	NEXT_DAY	60
2.3.8.6	NEW_TIME	60
2.3.8.7	NUMTODSINTERVAL	61
2.3.8.8	NUMTOYMINTERVAL	62
2.3.8.9	ラウンド	63
2.3.8.10	SYSDATE	68
2.3.8.11	TRUNC	68
2.3.9	Sequence Manipulation Functions	72
2.3.10	Conditional Expressions	73
2.3.10.1	事例	73
2.3.10.2	合体	74
2.3.10.3	NULLIF	75
2.3.10.4	NVL	75
2.3.10.5	NVL2	75
2.3.10.6	最大かつ最低	76
2.3.11	Aggregate Functions	77
2.3.11.1	LISTAGG	80
2.3.11.2	中央値	82
2.3.11.3	STATS_MODE	84
2.3.12	Subquery Expressions	86
2.3.12.1	EXISTS	86
2.3.12.2	IN	87

2.3.12.3	NOT IN	87
2.3.12.4	ANY/SOME	87
2.3.12.5	ALL	88
第 3 章	System Catalog Tables	89
3.1	dual	89
3.2	edb_dir	89
3.3	edb_password_history	90
3.4	edb_policy	90
3.5	edb_profile	91
3.6	edb_variable	93
3.7	pg_synonym	94
3.8	product_component_version	94
第 4 章	Conclusion	95
索引		97

第 1 章

Introduction

Oracle のデータベース互換性とは、アプリケーションコードをほとんどまたはまったく変更せずに、アプリケーションが Oracle 環境および EDBPostgresAdvancedServer (AdvancedServer) 環境で実行されることを意味します。

このガイドは、AdvancedServer でサポートされている互換性のあるデータ型に関する参照資料を提供します。以下に関する参照情報:

- 互換性のある SQL 言語構文は、*OracleDevelopersSQL* ガイドのデータベース互換性 で提供されています。
- 互換性のあるカタログビューは、*OracleDevelopers* カタログビューガイドのデータベース互換性 で提供されています。

AdvancedServer で Oracle データベースと互換性のあるアプリケーションを開発するには、アプリケーションの構築で使用される機能に特に注意する必要があります。たとえば、互換性のあるアプリケーションを開発するとは、次のものを選択することを意味します。

- Oracle データベースと互換性のあるアプリケーションのデータベーステーブルを定義するデータ型
- OracleSQL と互換性のある SQL ステートメント
- Oracle データベースと互換性のある SQL ステートメントおよび手続き型ロジックで使用するシステムおよび組み込み関数
- ストアドプロシージャ言語 (SPL) は、ストアドプロシージャ、関数、トリガー、およびパッケージのデータベースサーバー側アプリケーションロジックを作成します
- Oracle のデータディクショナリと互換性のあるシステムカタログビュー

AdvancedServer の互換性機能と拡張機能の詳細については、次の Web サイトで入手可能な AdvancedServer ドキュメントの完全なライブラリを参照してください。

<https://www.enterprisedb.com/edb-docs>

第 2 章

The SQL Language

以下のセクションでは、Oracle データベースと互換性のある AdvancedServerSQL 言語のサブセットについて説明します。次の SQL 構文、データ型、および関数は、EDBPostgresAdvancedServer と Oracle の両方で機能します。

AdvancedServer のドキュメントセットには、このガイドに含まれていない拡張機能（Oracle のデータベース互換性を提供せず、Oracle スタイルのアプリケーションをサポートしない機能）の構文とコマンドが含まれています。

このセクションは、次のセクションで構成されています。

- AdvancedServerSQL 構文と言語要素の一般的な議論
- データ型
- 組み込み関数

2.1 SQL Syntax

このセクションでは、SQL の一般的な構文について説明します。これは、データを定義および変更するために SQL コマンドがどのように適用されるかについての詳細を含む次の章を理解するための基盤を形成します。

2.1.1 Lexical Structure

SQL 入力は一連のコマンドで構成されます。command は、セミコロン（`;`）で終わる tokens のシーケンスで構成されます。入力ストリームの終わりもコマンドを終了します。有効なトークンは、特定のコマンドの構文によって異なります。

トークンは、key word、identifier、quoted identifier、literal（または constant）、または特殊文字記号です。トークンは通常 whitespace（スペース、タブ、改行）で区切られていますが、あいまいさがなければ必要ありません（通常、特殊文字が他のトークンタイプに隣接している場合のみです）。

また、SQL 入力では comments が発生する可能性があります。それらはトークンではありません-それらは事実上空白と同等です。

たとえば、次は（構文的に）有効な SQL 入力です。

```
SELECT * FROM MY_TABLE;  
UPDATE MY_TABLE SET A = 5;  
INSERT INTO MY_TABLE VALUES (3, 'hi there');
```

これは、1 行に 1 つずつの 3 つのコマンドのシーケンスです（ただし、これは必須ではありません。1 行に複数のコマンドを含めることができ、通常はコマンドを行に分割できます）。

SQL 構文は、どのトークンがコマンドを識別し、どれがオペランドまたはパラメーターであるかに関してあまり一貫性がありません。通常、最初のいくつかのトークンはコマンド名であるため、上記の例では、通常 SELECT、UPDATE、および INSERT コマンドについて説明します。しかし、たとえば、UPDATE コマンドは、特定の位置に表示される SET トークンを常に必要とし、INSERT のこの特定のバリエーションは、完了するために VALUES トークンも必要とします。各コマンドの正確な構文規則は、*Oracle Developers SQL ガイド*のデータベース互換性で説明されています。

2.1.2 Identifiers and Key Words

上記の例の SELECT、UPDATE、VALUES などのトークンは、key words、つまり SQL 言語で固定された意味を持つ単語の例です。トークン MY_TABLE と A は identifiers の例です。使用されるコマンドに応じて、テーブル、列、またはその他のデータベースオブジェクトの名前を識別します。したがって、単に names と呼ばれることもあります。キーワードと識別子の lexical structure は同じです。つまり、言語がわからないとトークンが識別子であるかキーワードであるかがわかりません。

SQL 識別子とキーワードは、文字（a-z または A-Z）で始まる必要があります。識別子またはキーワードの後続の文字には、文字、アンダースコア、数字（0-9）、ドル記号（\$）、または番号記号（#）を使用できます。

識別子とキーワード名は大文字と小文字を区別しません。だから

```
UPDATE MY_TABLE SET A = 5;
```

次のように書くこともできます。

```
uPDaTE my_TabLE SeT a = 5;
```

よく使用される規則は、大文字でキーワードを、小文字で名前を書くことです。

```
UPDATE my_table SET a = 5;
```

2 番目の種類の識別子があります: delimited identifier または quoted identifier。任意の文字シーケンスを二重引用符 (") で囲むことで形成されます。区切り識別子は常に識別子であり、キーワードではありません。したがって、/"select/" を使用して /"select/" という名前の列またはテーブルを参照できます。一

方、引用符で囲まれていない選択はキーワードとして使用されるため、テーブル名または列名が必要な場所で使用すると解析エラーが発生します。

```
UPDATE /"my_table/" SET /"a/" = 5;
```

引用符付き識別子には、数値コードがゼロの文字を除く、任意の文字を含めることができます。

二重引用符を含めるには、2つの二重引用符を使用します。これにより、他の方法では不可能なテーブル名または列名（スペースやアンパサンドを含む名前など）を作成できます。長さの制限は引き続き適用されます。

識別子を引用符で囲むと大文字と小文字が区別されますが、引用符で囲まれていない名前は常に小文字に変換されます。たとえば、識別子 `FOO`、`foo`、`/"foo/"` は `AdvancedServer` によって同じと見なされますが、`/"Foo/"` と `/"FOO/"` はこれらの3つとは異なります。引用符で囲まれていない名前の小文字への折りたたみは、`Oracle` データベースと互換性がありません。`Oracle` 構文では、引用符で囲まれていない名前は大文字に変換されます。たとえば、`foo` は `/"foo/"` ではなく `/"FOO/"` と同等です。ポータブルアプリケーションを作成する場合は、常に特定の名前を引用するか、決して引用しないことをお勧めします。

2.1.3 Constants

`AdvancedServer` の暗黙的に型指定された定数の種類は、`strings` と `numbers` です。定数を明示的な型で指定することもできます。これにより、システムによるより正確な表現とより効率的な処理が可能になります。これらの選択肢については、次のサブセクションで説明します。

2.1.3.1 文字列定数

SQL の `string constant` は、`'This is a string'` などの単一引用符 (') で区切られた任意の文字列です。文字列定数内に単一引用符を含めるには、2つの隣接する単一引用符を記述します。`'Dianne''s horse'`。これは、二重引用符 (/") と同じではないことに注意してください。

2.1.3.2 数値定数

数値定数は、次の一般的な形式で受け入れられます。

```
digits
digits.[digits][e[+-]digits]
[digits].digits[e[+-]digits]
digitse[+-]digits
```

ここで、`digits` は1つ以上の10進数 (0~9) です。小数点を使用する場合、少なくとも1桁は小数点の前または後になければなりません。指数マーカー (e) が存在する場合は、少なくとも1桁が続く必要があります。定数

にスペースやその他の文字を埋め込むことはできません。先頭のプラス記号またはマイナス記号は、実際には定数の一部とは見なされないことに注意してください。これは定数に適用される演算子です。

有効な数値定数の例を次に示します。

```
42
3.5
4.
.001
5e2
1.925e-3
```

小数点も指数も含まない数値定数は、その値が INTEGER 型（32 ビット）に収まる場合、最初は INTEGER 型と見なされます。それ以外の場合、値がタイプ BIGINT（64 ビット）に収まる場合、タイプ BIGINT であると推定されます。それ以外の場合は、タイプ NUMBER と見なされます。小数点や指数を含む定数は、最初は常に NUMBER タイプであると推定されます。

数値定数の最初に割り当てられたデータ型は、型解決アルゴリズムの出発点にすぎません。ほとんどの場合、定数はコンテキストに応じて自動的に最も適切な型に強制されます。必要に応じて、次のセクションで説明するようにキャストすることにより、数値を特定のデータ型として解釈させることができます。

2.1.3.3 他のタイプの定数

CAST

次の表記法を使用して、任意のタイプの定数を入力できます。

```
CAST('string' AS type)
```

文字列定数のテキストは、type というタイプの入力変換ルーチンに渡されます。結果は、示されたタイプの定数です。明示的な型キャストは、定数が必要な型にあいまいさがない場合（たとえば、テーブル列に直接割り当てられる場合）、省略される場合があります。この場合、自動的に強制されます。

CAST を使用して、任意の式の実行時型変換を指定することもできます。

CAST (MULTISET)

MULTISET は CAST の拡張で、サブクエリの結果をネストされたテーブルタイプに変換します。概要は次のとおりです。

```
CAST ( MULTISET ( < subquery > ) AS < datatype > )
```

subquery は 1 つ以上の行を返すクエリであり、datatype はネストされたテーブルタイプです。

CAST (MULTISET) は、データのコレクションをテーブルに格納するために使用されます。

Example

次の例は、MULTISET の使用方法を示しています。

```
edb=# CREATE OR REPLACE TYPE project_table_t AS TABLE OF VARCHAR2(25);
CREATE TYPE
edb=# CREATE TABLE projects (person_id NUMBER(10), project_name VARCHAR2(20));
CREATE TABLE
edb=# CREATE TABLE pers_short (person_id NUMBER(10), last_name VARCHAR2(25));
CREATE TABLE
```

```
edb=# INSERT INTO projects VALUES (1, 'Teach');
INSERT 0 1
edb=# INSERT INTO projects VALUES (1, 'Code');
INSERT 0 1
edb=# INSERT INTO projects VALUES (2, 'Code');
INSERT 0 1
edb=# INSERT INTO pers_short VALUES (1, 'Morgan');
INSERT 0 1
edb=# INSERT INTO pers_short VALUES (2, 'Kolk');
INSERT 0 1
edb=# INSERT INTO pers_short VALUES (3, 'Scott');
INSERT 0 1
edb=# COMMIT;
COMMIT
```

```
edb=# SELECT e.last_name, CAST(MULTISET(
edb(#   SELECT p.project_name
edb(#   FROM projects p
edb(#   WHERE p.person_id = e.person_id
edb(#   ORDER BY p.project_name) AS project_table_t)
edb-# FROM pers_short e;
  last_name | project_table_t
-----+-----
Morgan     | {Code,Teach}
Kolk       | {Code}
Scott      | {}
(3 rows)
```

2.1.4 Comments

コメントとは、二重ダッシュで始まり、行末まで続く任意の文字列です。例:

```
-- This is a standard SQL comment
```

または、C スタイルのブロックコメントを使用できます。

```
/* multiline comment  
* block  
*/
```

ここで、コメントは/で始まり、一致する//まで続きます。

2.2 Data Types

次の表に、組み込みの汎用データ型を示します。

お名前	エイリアス	説明
BLOB	LONGRAW、RAW (<i>n</i>) 、 BYTEA	バイナリデータ
ブーリアン		論理ブール (true/false)
CHAR[(<i>n</i>)]	キャラクター [(<i>n</i>)]	<i>n</i> 文字の固定長文字列
CLOB	LONG、LONGVARCHAR	長い文字列
DATE	タイムスタンプ	秒の日付と時刻
倍精度	フロート、 フロート (25) –フロート (53)	倍精度浮動小数点数
整数	INT、BINARY_INTEGER、 PLS_INTEGER	符号付き 4 バイト整数
NUMBER	12 月、10 進数、数値	オプションの小数点以下の桁数を持つ正確な数値
NUMBER (<i>p</i> [, <i>s</i>])	DEC (<i>p</i> [, <i>s</i>])、 DECIMAL (<i>p</i> [, <i>s</i>])、 NUMERIC (<i>p</i> [, <i>s</i>])	最大精度の正確な数値、 <i>p</i> 、およびオプションのスケール、 <i>s</i>
REAL	FLOAT (1) –FLOAT (24)	単精度浮動小数点数
タイムスタンプ [(<i>p</i>)]		オプションの小数秒精度 <i>p</i> を使用した日付と時刻
タイムゾーン付きタイムスタンプ [(<i>p</i>)]		オプションの小数秒精度、 <i>p</i> 、およびタイムゾーン付きの日付と時刻
VARCHAR2 (<i>n</i>)	CHARVARYING (<i>n</i>)、CHARACTERVARYING (<i>n</i>)、VARCHAR (<i>n</i>)	最大長が <i>n</i> 文字の可変長文字列
XMLTYPE		XML データ

2.2.1 Numeric Types

数値型は、4 バイトの整数、4 バイトと 8 バイトの浮動小数点数、および固定精度の小数で構成されます。次の表に、使用可能なタイプを示します。

お名前	スト レー ジ サイズ	説明	範囲
BINARY_INTEGER	4 バイト	符号付き整数、INTEGER のエイリアス	-2,147,483,648 から +2,147,483,647
倍精度	8 バイト	可変精度、不正確	15 桁の精度
整数	4 バイト	整数の通常の選択	-2,147,483,648 から +2,147,483,647
NUMBER	可変	ユーザー指定の精度、正確	最大 1000 桁の精度
NUMBER (p[, s])	可変	最大精度の正確な数値、p、およびオプションのスケール、s	最大 1000 桁の精度
PLS_INTEGER	4 バイト	符号付き整数、INTEGER のエイリアス	-2,147,483,648 から +2,147,483,647
REAL	4 バイト	可変精度、不正確	6 桁の精度
ROWID	8 バイト	符号付き 8 ビット整数。	-9223372036854775808 から 9223372036854775807 まで

次のセクションでは、タイプについて詳しく説明します。

2.2.1.1 整数型

BINARY_INTEGER、INTEGER、PLS_INTEGER、および ROWID タイプは、表 Numeric Types で指定されているように整数（小数部を含まない）を格納します。許容範囲外の値を保存しようとすると、エラーが発生します。

2.2.1.2 任意の精度番号

タイプ NUMBER は、事実上無制限の桁数の精度を格納し、正確に計算を実行できます。特に正確な金額が必要な金額やその他の数量を保存する場合に推奨されます。ただし、NUMBER 型は、次のセクションで説明する浮動小数点型に比べて非常に遅いです。

以下では、これらの用語を使用します。NUMBER の scale は、小数点の右側の小数部の小数点以下の桁数です。NUMBER の precision は、整数の有効桁数、つまり小数点の両側の桁数です。したがって、数値 23.5141 の精度は 6、スケールは 4 です。整数のスケールはゼロと見なすことができます。

NUMBER タイプの精度とスケールの両方を設定できます。タイプ NUMBER の列を宣言するには、構文を使用します

```
NUMBER(precision, scale)
```

精度は正、スケールはゼロまたは正でなければなりません。あるいは、

NUMBER (precision)

スケール 0 を選択します。精度またはスケールなしで NUMBER を指定すると、精度の実装制限まで、任意の精度およびスケールの数値を格納できる列が作成されます。この種類の列は入力値を特定のスケールに強制しませんが、宣言されたスケールを持つ NUMBER 列は入力値をそのスケールに強制します。(SQL 標準では、デフォルトのスケール 0、つまり整数精度への強制が必要です。移植性を最大にするには、精度とスケールを明示的に指定するのが最適です。)

値の精度またはスケールが宣言された列の精度またはスケールよりも大きい場合、システムは値を丸めようとします。宣言された制限を満たすために値を丸めることができない場合、エラーが発生します。

2.2.1.3 浮動小数点型

データ型 REAL および DOUBLE PRECISION は *inexact*、可変精度数値型です。実際には、これらの型は通常、基になるプロセッサ、オペレーティングシステム、およびコンパイラがサポートする範囲で、バイナリ浮動小数点演算（それぞれ単精度と倍精度）の IEEE 標準 754 の実装です。

不正確とは、一部の値を正確に内部形式に変換できず、近似値として保存されることを意味します。そのため、値の保存と印刷はわずかな矛盾を示す場合があります。これらのエラーを管理し、計算を通じてどのように伝播するかは、数学およびコンピューターサイエンスの分野全体の主題であり、以下の点を除き、ここではこれ以上説明しません。

正確な保存と計算が必要な場合（金額など）、代わりに NUMBER タイプを使用します。

特に境界の場合（無限大、アンダーフロー）の特定の動作に依存している場合、重要なものについてこれらの型を使用して複雑な計算を行いたい場合は、実装を慎重に評価する必要があります。

2 つの浮動小数点値が等しいかどうかを比較すると、期待どおりに機能する場合と機能しない場合があります。

ほとんどのプラットフォームでは、REAL タイプの範囲は少なくとも $1\text{E}-37$ から $1\text{E}+37$ までで、精度は少なくとも 6 桁の 10 進数です。DOUBLE PRECISION タイプは通常、少なくとも 15 桁の精度で $1\text{E}-307$ から $1\text{E}+308$ 程度の範囲にあります。値が大きすぎるか小さすぎると、エラーが発生します。入力数値の精度が高すぎる場合、丸めが行われる場合があります。ゼロとは異なると表現できないゼロに近い数値は、アンダーフローエラーの原因になります。

AdvancedServer は、不正確な数値タイプを指定するための SQL 標準表記「10」および FLOAT(p) もサポートしています。ここで、p は最小許容精度を 2 進数で指定します。AdvancedServer は、REAL タイプの選択として FLOAT(1) から FLOAT(24) を受け入れ、「11」の選択として FLOAT(25) から FLOAT(53) を受け入れます。許容範囲外の p の値はエラーになります。精度が指定されていない「10」は「11」を意味します。

2.2.2 Character Types

次の表に、AdvancedServer で使用可能な汎用文字タイプを示します。

お名前	説明
CHAR[(n)]	<i>n</i> で指定されたサイズまで空白が埋め込まれた固定長文字列
CLOB	最大 1GB の大きな可変長
長い	可変の無制限の長さ。
NVARCHAR (n)	制限付きの可変長の国別文字ストリング。
NVARCHAR2 (n)	制限付きの可変長の国別文字ストリング。
STRING	VARCHAR2 のエイリアス。
VARCHAR (n)	制限付きの可変長文字列（非推奨と見なされますが、互換性のためにサポートされます）
VARCHAR2 (n)	制限付きの可変長文字列

n は正の整数です。これらのタイプは、*n* 文字までの文字列を保存できます。*n* の長さを超える値を割り当てようとすると、余分な文字がすべてスペースでない場合、エラーが発生します。その場合、文字列は最大長に切り捨てられます。

これらのタイプのデータのストレージ要件は、実際の文字列に文字列が 127 バイト未満の場合は 1 バイト、文字列が 127 バイト以上の場合は 4 バイトです。CHAR の場合、パディングにもストレージが必要です。長い文字列はシステムによって自動的に圧縮されるため、ディスク上の物理的要件は少なくなります。長い値はバックグラウンドテーブルに格納されるため、短い列値への迅速なアクセスを妨げません。

データベースの文字セットは、テキスト値の格納に使用される文字セットを決定します。

CHAR

n の値を指定しない場合、*n* はデフォルトで 1 になります。割り当てられる文字列が *n* より短い場合、CHAR 型の値は指定された幅（*n*）までスペースが埋め込まれ、そのように格納および表示されます。

パディングスペースは意味的に重要でないものとして扱われます。つまり、CHAR 型の 2 つの値を比較する場合、末尾のスペースは無視され、CHAR 値を他の文字列型の 1 つに変換するときに削除されます。

長すぎる値を明示的に CHAR(*n*) 型にキャストすると、エラーは発生せずに値が *n* 文字に切り捨てられます（SQL 標準で指定されています）。

VARCHAR, VARCHAR2, NVARCHAR and NVARCHAR2

割り当てる文字列が *n* より短い場合、タイプ VARCHAR、VARCHAR2、NVARCHAR、NVARCHAR2 の値は、パディングなしで短い文字列を保存します。

VARCHAR の値で意味的に重要な末尾のスペース are に注意してください。

値を明示的に VARCHAR 型にキャストすると、長すぎる値はエラーを発生させることなく *n* 文字に切り捨てられます（SQL 標準で指定されています）。

CLOB

大きな文字列を CLOB タイプで保存できます。CLOB は、長さの制限が指定されていないことを除き、VARCHAR2 と意味的に同等です。通常、最大文字列長が不明な場合は CLOB タイプを使用する必要があります。

CLOB タイプに保存できる最長の文字列は約 1GB です。

注釈: CLOB データ型は、実際には PostgreSQL の TEXT データ型に基づいた DOMAIN です。DOMAIN の詳細については、<https://www.postgresql.org/docs/current/static/sql-createdomain.html> の PostgreSQL コードドキュメントを参照してください。

したがって、CLOB タイプの使用は、約 1GB の最大サイズなど、TEXT に対してできることによって制限されます。

大量のデータを使用するには、CLOB データタイプを使用する代わりに、pg_largeobject システムカタログに依存する PostgreSQL の Large Objects 機能を使用します。ラージオブジェクトの詳細については、次の PostgreSQL コードドキュメントを参照してください。

<https://www.postgresql.org/docs/current/static/largeobjects.html>

2.2.3 Binary Data

次の表に、バイナリ文字列を保存できるデータ型を示します。

お名前	ストレージサイズ	説明
バイナリ	バイナリ文字列の長さ。	長さが 1～8300 の固定長バイナリ文字列。
BLOB	実際のバイナリ文字列と、バイナリ文字列が 127 バイト未満の場合は 1 バイト、バイナリ文字列が 127 バイト以上の場合には 4 バイト。	最大サイズが 1GB の可変長バイナリ文字列。
VARBINARY	バイナリ文字列の長さ	長さが 1～8300 の可変長バイナリ文字列。

バイナリ文字列は、オクテット（またはバイト）のシーケンスです。バイナリ文字列は、2つの特性によって文字列と区別されます。1つ目は、バイナリ文字列は、値ゼロのオクテットとその他の「非印刷可能」オクテット（32～126 の範囲外のオクテットとして定義）を保存できることです。第二に、バイナリ文字列の操作は実際のバイトを処理しますが、文字列のエンコードと処理はロケール設定に依存します。

2.2.4 Date/Time Types

以下の日付/時刻タイプの説明では、テーブルが作成または変更されるたびに、構成パラメーター `edb_redwood_date` が TRUE に設定されていることを前提としています。

AdvancedServer は、次の表に示す日付/時刻タイプをサポートしています。

お名前	ストレージ サイズ	説明	低価値	高価値	解像度
DATE	8 バイト	日時	紀 元 前 4713 年	5874897AD	1 秒
間隔日から 2 日目 [(p)]	12 バイト	期間	- 178000000 年	178000000 年	1 マイクロ秒 /14 桁
間隔月	12 バイト	期間	- 178000000 年	178000000 年	1 マイクロ秒 /14 桁
タイムスタンプ [(p)]	8 バイト	日時	紀 元 前 4713 年	5874897AD	1 マイクロ秒
タイムゾーン付きタイムスタンプ [(p)]	8 バイト	タイムゾーン付きの 日付と時刻	紀 元 前 4713 年	5874897AD	1 マイクロ秒

データ定義言語 (DDL) コマンドの列のデータ型として DATE が CREATE TABLE または ALTER TABLE として表示される場合、テーブル定義がデータベースに保存されるときに TIMESTAMP に変換されます。したがって、時間コンポーネントも日付とともに列に格納されます。

DATE が SPL 宣言セクションの変数のデータ型、SPL プロシージャまたは SPL 関数の仮パラメーターのデータ型、または SPL 関数の戻り型として表示される場合、常に TIMESTAMP。したがって、存在する場合は時間コンポーネントを処理できます。

TIMESTAMP は、秒フィールドに保持される小数桁数を指定するオプションの精度値 `p` を受け入れます。`p` の許容範囲は 0 から 6 までで、デフォルトは 6 です。

TIMESTAMP 値が倍精度浮動小数点数として格納されている場合 (現在はデフォルト)、有効な精度の制限は 6 未満です。TIMESTAMP 値は 2000-01-01 の午前 0 時前後の秒数として格納されます。2000 年 1 月 1 日から数年以内の日付ではマイクロ秒の精度が達成されますが、それより遠い日付では精度が低下します。TIMESTAMP 値が 8 バイトの整数として保存されている場合 (コンパイル時オプション)、値の全範囲でマイクロ秒の精度が利用可能です。ただし、8 バイトの整数のタイムスタンプには、上記の日付よりも限られた日付範囲 (紀元前 4713 年から西暦 294276 年まで) があります。

TIMESTAMP (p) WITH TIME ZONE は TIMESTAMP (p) に似ていますが、タイムゾーンも含まれます。

2.2.4.1 区間タイプ

INTERVAL の値は期間を指定します。INTERVAL タイプの値は、データの値を記述するフィールドで構成されます。次の表に、INTERVAL タイプで許可されるフィールドを示します。

フィールド名	許可される間隔値
年	整数値（正または負）
月	0 から 11
DAY	整数値（正または負）
営業時間	0 から 23
MINUTE	0 から 59
第二	0 から 59.9 (p) (9 (p) は秒の小数部の精度)

フィールドは YEARS から MONTHS、DAYS から HOURS、MINUTES、SECONDS の降順で表示する必要があります。

AdvancedServer は、Oracle データベースと互換性のある 2 つの INTERVAL タイプをサポートしています。

AdvancedServer がサポートする最初のバリエーションは INTERVAL DAY TO SECOND [(p)] です。INTERVAL DAY TO SECOND [(p)] は、日、時間、分、秒の時間間隔を保存します。

p は second フィールドの精度を指定します。

AdvancedServer は値を解釈します:

```
INTERVAL '1 2:34:5.678' DAY TO SECOND(3)
```

1 日、2 時間、34 分、5 秒、678 分の 1 秒。

AdvancedServer は値を解釈します:

```
INTERVAL '1 23' DAY TO HOUR
```

1 日と 23 時間。

AdvancedServer は値を解釈します:

```
INTERVAL '2:34' HOUR TO MINUTE
```

2 時間 34 分。

AdvancedServer は値を解釈します:

```
INTERVAL '2:34:56.129' HOUR TO SECOND(2)
```

2 時間 34 分 56 秒 13 分の 1 秒。指定された精度のため、小数秒は 13 に切り上げられることに注意してください。

Oracle データベースと互換性のある AdvancedServer がサポートする 2 番目のバリエーションは INTERVAL YEAR TO MONTH です。このバリエーションは、年と月の時間間隔を保存します。

AdvancedServer は値を解釈します:

```
INTERVAL '12-3' YEAR TO MONTH
```

12 年 3 ヶ月。

AdvancedServer は値を解釈します:

```
INTERVAL '456' YEAR(2)
```

12 年 3 ヶ月。

AdvancedServer は値を解釈します:

```
INTERVAL '300' MONTH
```

25 年として。

2.2.4.2 日付/時刻入力

日付と時刻の入力は、ISO8601SQL 互換形式、Oracle のデフォルトの dd-MON-yy 形式、およびどのコンポーネントが年、月、日であるかについてあいまいさが無い限り、他の多くの形式で受け入れられます。ただし、あいまいさを避けるため、TO_DATE 関数の使用を強くお勧めします。

日付または時刻のリテラル入力は、テキスト文字列のように単一引用符で囲む必要があります。次の SQL 標準構文も使用できます。

```
type 'value'
```

type は DATE または TIMESTAMP です。

value は日付/時刻のテキスト文字列です。

2.2.4.2.1 日程

次のブロックは、1999 年 1 月 8 日に相当する日付のいくつかの可能な入力形式を示しています。

```
Example
January 8, 1999
1999-01-08
1999-Jan-08
Jan-08-1999
08-Jan-1999
08-Jan-99
Jan-08-99
19990108
990108
```

日付値は、DATE または TIMESTAMP 列または変数に割り当てることができます。日付の値に時刻の値が付加されていない場合、時間、分、および秒のフィールドはゼロに設定されます。

2.2.4.2.2 回

次の表に、日付またはタイムスタンプの時間コンポーネントの例を示します。

例	説明
04:05:06.789	ISO8601
04:05:06	ISO8601
04:05	ISO8601
040506	ISO8601
04:05AM	04:05 と同じ;AM は値に影響しません
04:05PM	16:05 と同じ; 入力時間は 12 以下でなければなりません

2.2.4.2.3 タイムスタンプ

タイムスタンプの有効な入力、日付と時刻の連結で構成されます。タイムスタンプの日付部分は、セクション Dates の下のブロックに示されている例のいずれかに従ってフォーマットできます。タイムスタンプの時間部分は、Times セクションの表に示されている例のいずれかに従ってフォーマットできます。

以下は、Oracle のデフォルト形式に従ったタイムスタンプの例です。

```
08-JAN-99 04:05:06
```

以下は、ISO8601 標準に従うタイムスタンプの例です。

```
1999-01-08 04:05:06
```

2.2.4.3 日付/時刻出力

日付/時刻型のデフォルトの出力形式は、アプリケーションインターフェースに応じて、(dd-MON-yy) Redwood date style と呼ばれ、Oracle データベースと互換性があり、(yyyy-mm-dd) と呼ばれ、ISO8601 形式と呼ばれます。データベースに、SQLInteractive などの JDBC を使用するアプリケーションは、常に日付を ISO8601 形式で提示します。PSQL などの他のアプリケーションは、Redwood 形式で日付を提示します。

次の表に、Redwood と ISO8601 の 2 つのスタイルの出力形式の例を示します。

説明	例
レッドウッドスタイル	05/12/3107:37:16
ISO8601/SQL 標準	1997-12-1707:37:16

2.2.4.4 内部

AdvancedServer は、すべての日付/時刻の計算にユリウス日付を使用します。ユリウス日は、年の長さが 365.2425 日であるという仮定に基づいて、紀元前 4713 年以降の日付を正しく予測または計算します。

2.2.5 Boolean Types

AdvancedServer は、標準の SQL タイプ BOOLEAN を提供します。BOOLEAN には、TRUE または FALSE の 2 つの状態のいずれかのみを指定できます。3 番目の状態 UNKNOWN は、SQL の NULL 値で表されます。

お名前	ストレージサイズ	説明
ブーリアン	1 バイト	論理ブール (true/false)

真の状態を表すための有効なリテラル値は TRUE です。偽の状態を表す有効なリテラルは FALSE です。

2.2.6 XML Type

XMLTYPE データ型は、XML データを格納するために使用されます。XML データを文字フィールドに格納することの利点は、入力値の整形形式をチェックすること、およびタイプセーフな操作を実行するためのサポート関数があることです。

XML タイプは、XML 標準で定義されている整形形式の「ドキュメント」と、XML 標準のプロダクション XMLDecl? content で定義されている「コンテンツ」フラグメントを格納できます。大まかに言うと、これはコンテンツフラグメントが複数のトップレベル要素またはキャラクターノードを持つことができることを意味します。

注釈: Oracle は、XMLTYPE 列のコンテンツフラグメントの保存をサポートしていません。

次の例は、XMLTYPE 列のあるテーブルへの行の作成と挿入を示しています。

```
CREATE TABLE books (
    content          XMLTYPE
);

INSERT INTO books VALUES (XMLPARSE (DOCUMENT '<?xml
version="1.0"><book><title>Manual</title><chapter>...</chapter></book>'));

SELECT * FROM books;
```

content

(次のページに続く)

(前のページからの続き)

```
<book><title>Manual</title><chapter>...</chapter></book>  
(1 row)
```

2.3 Functions and Operators

AdvancedServer は、組み込みデータ型用の多数の関数と演算子を提供します。

2.3.1 Logical Operators

通常の論理演算子が利用可能です:AND、OR、NOT

SQL は、null 値が「不明」を表す 3 値のブールロジックを使用します。次の真理値表を確認してください。

AND/OR Truth Table

a	b	aANDb	a または b
本当だ	本当だ	本当だ	本当だ
本当だ	偽	偽	本当だ
本当だ	ヌル	ヌル	本当だ
偽	偽	偽	偽
偽	ヌル	偽	ヌル
ヌル	ヌル	ヌル	ヌル

NOT Truth Table

a	ない
本当だ	偽
偽	本当だ
ヌル	ヌル

演算子 AND と OR は可換です。つまり、結果に影響を与えずに左右のオペランドを切り替えることができます。

2.3.2 Comparison Operators

通常の比較演算子を次の表に示します。

オペレーター	説明
<	より小さい
>	より大きい
<=	より小さいか等しい
>=	より大きい等しい
=	等しい
<>	等しくない
!=	等しくない

比較演算子は、これが理にかなっているすべてのデータ型で使用できます。すべての比較演算子は、`1 < 2 < 3` などの `BOOLEAN` 型の値を返すバイナリ演算子です（ブール値と 3 を比較する `<` 演算子がないため）。

比較演算子に加えて、特別な `BETWEEN` 構造が利用可能です。

```
a BETWEEN x AND y
```

に等しい

```
a >= x AND a <= y
```

同様に、

```
a NOT BETWEEN x AND y
```

に等しい

```
a < x OR a > y
```

内部で最初の形式を 2 番目の形式に書き換えるのに必要な CPU サイクルを除いて、2 つのそれぞれの形式に違いはありません。

値が `null` かどうかを確認するには、構造を使用します

```
expression IS NULL
```

```
expression IS NOT NULL
```

`NULL` は `NULL` に等しくないため、`expression = NULL` を書き込まないでください。（`null` 値は不明な値を表し、2 つの不明な値が等しいかどうかは不明です。）この動作は `SQL` 標準に準拠しています。

一部のアプリケーションでは、`expression` がヌル値と評価された場合に `expression = NULL` が `true` を返すことを期待する場合があります。これらのアプリケーションは、`SQL` 標準に準拠するように変更することを強くお勧めします。

2.3.3 Mathematical Functions and Operators

多くの AdvancedServer タイプに数学演算子が用意されています。考えられるすべての順列に共通の数学的規則がない型（日付/時刻型など）の実際の動作については、以降のセクションで説明します。

次の表に、使用可能な数学演算子を示します。

オペレーター	説明	例	結果
+	加算	2+3	5
-	減算	2-3	-1
*	乗算	2*3	6
/	分割（次の注を参照してください。）	4/2	2
**	べき乗演算子	2** 3	8

注釈: postgresql.conf ファイルの db_dialect 構成パラメーターが redwood に設定されている場合、INTEGER データ型のペアを分割しても、値が切り捨てられません。次の例に示すように、小数の結果は保持されます。

```
edb=# SET db_dialect TO redwood;
SET
edb=# SHOW db_dialect;
 db_dialect
-----
 redwood
(1 row)

edb=# SELECT CAST('10' AS INTEGER) / CAST('3' AS INTEGER) FROM dual;
      ?column?
-----
 3.3333333333333333
(1 row)
```

この動作は、ネイティブの INTEGER データ型がなく、INTEGER データ型の指定が内部的に NUMBER(38) に変換される Oracle データベースと互換性があり、結果の端数が保持されます。

db_dialect 構成パラメーターが postgres に設定されている場合、INTEGER データ型のペアを分割すると、次の例に示すように値が切り捨てられます。

```
edb=# SET db_dialect TO postgres;
SET
edb=# SHOW db_dialect;
```

(次のページに続く)

(前のページからの続き)

```

db_dialect
-----
postgres
(1 row)

edb=# SELECT CAST('10' AS INTEGER) / CAST('3' AS INTEGER) FROM dual;
?column?
-----
      3
(1 row)

```

この動作は PostgreSQL データベースと互換性があり、INTEGER、SMALLINT、または BIGINT のデータ型のいずれかのペアを含む除算の結果が切り捨てられます。前の例に示すように、db_dialect が postgres に設定されている場合、同じ切り捨てられた結果が AdvancedServer によって返されます。

ただし、db_dialect が redwood に設定されている場合でも、INTEGER のデータ型のペアによる除算のみが結果の切り捨てにならないことに注意してください。INTEGER データ型の有無にかかわらず、SMALLINT または BIGINT データ型のみを含む除算は、INTEGER および SMALLINT 部門に関与しています:

```

edb=# SHOW db_dialect;
db_dialect
-----
redwood
(1 row)

edb=# SELECT CAST('10' AS INTEGER) / CAST('3' AS SMALLINT) FROM dual;
?column?
-----
      3
(1 row)

```

次の表に、使用可能な数学関数を示します。これらの関数の多くは、異なる引数タイプを持つ複数の形式で提供されます。特に明記されていない限り、関数の特定の形式は、引数と同じデータ型を返します。DOUBLE PRECISION データを処理する関数は、主にホストシステムの C ライブラリの上に実装されます。したがって、境界の場合の精度と動作は、ホストシステムによって異なる場合があります。

機能	戻りタイプ	説明	例	結果
ABS (x)	x と同じ	絶対値	ABS (-17.4)	17.4
CEIL (DOUBLEPRECISION または NUMBER)	入力と同じ	引数以上の最小整数	セイル (-42.8)	-42

次のページに続く

表 1 – 前のページからの続き

EXP (DOUBLE- PRECISION または NUMBER)	入力と同じ	指数関数	EXP (1.0)	2.7182818284590452
FLOOR (DOUBLE- PRECISION または NUMBER)	入力と同じ	引数より大きくない最大の整数	フロア (-42.8)	43
LN (DOUBLEPRECI- SION または NUM- BER)	入力と同じ	自然対数	LN (2.0)	0.6931471805599453
LOG (bNUMBER、 xNUMBER)	NUMBER	底 b の対数	LOG (2.0、64.0)	6.0000000000000000
MOD (y、x)	引数の型 と同じ	y/x の残り	MOD (9、4)	1
NVL (x、y)	引数の型 と同じで す。両方 の引数が 同じデー タ型であ る場合	x がヌルの場合、NVL は y を返します	NVL (9、0)	9
パワー (aDOUBLE- PRECISION、BDOU- BLEPRECISION)	倍精度	a の b 乗	パワー (9.0、3.0)	729.0000000000000000
POWER (aNUMBER、 bNUMBER)	NUMBER	a の b 乗	パワー (9.0、3.0)	729.0000000000000000
ROUND (DOUBLE- PRECISION または NUMBER)	入力と同じ	最も近い整数に丸める	ラウンド (42.4)	42
ROUND(vNUMBER、 sINTEGER)	NUMBER	小数点以下の桁数に丸 めます	ラウンド (42.4382、2)	42.44
SIGN (DOUBLEPRE- CISION または NUM- BER)	入力と同じ	引数の符号 (-1、0、+1)	SIGN (-8.4)	-1
SQRT (DOUBLEPRE- CISION または NUM- BER)	入力と同じ	平方根	SQRT (2.0)	1.414213562373095

次のページに続く

表 1 – 前のページからの続き

TRUNC (DOUBLE- PRECISION または NUMBER)	入力と同じ	ゼロに向かって切り捨 てる	TRUNC (42.8)	42
TRUNC (vNUMBER、 sINTEGER)	NUMBER	小数点以下の桁数に切 り捨て	TRUNC (42.4382、2)	42.43
WIDTH_BUCKET (opNUMBER、 b1NUMBER、 b2NUMBER、 countINTEGER)	整数	op が割り当てられる バケットを、b1 から b2 の範囲の count バ ケットの等深度ヒスト グラムで返します	WIDTH_BUCKET (5.35、0.024、10.06、 5)	3

次の表に、使用可能な三角関数を示します。すべての三角関数は引数を取り、DOUBLE PRECISION 型の値を返します。

機能	説明
ACOS (x)	逆余弦
ASIN (x)	逆正弦
ATAN (x)	逆正接
ATAN2 (x, y)	x/y の逆正接
COS (x)	コサイン
SIN (x)	サイン
TAN (x)	接線

2.3.4 String Functions and Operators

このセクションでは、文字列値を調べて操作するための関数と演算子について説明します。このコンテキストの文字列には、タイプ CHAR、VARCHAR2、および CLOB の値が含まれます。特に明記されていない限り、以下にリストされているすべての機能はこれらすべてのタイプで機能しますが、CHAR タイプを使用する場合の自動パディングの潜在的な影響には注意してください。一般に、ここで説明する関数は、最初にそのデータを文字列表現に変換することにより、非文字列型のデータでも機能します。

機能	戻りタイプ	説明	例	結果
文字列// ひも	CLOB	ストリング 連結	「エンタープライズ」// 「DB」	EnterpriseDB
CONCAT (string, string)	CLOB	ストリング 連結	'a'// 'b'	ab
HEXTORAW (varchar2)	RAW	VARCHAR2 値を RAW 値に変換し ます	HEXTORAW ('303132')	'012'
RAWTOHEX (生)	VARCHAR2	RAW 値 を HEX- ADECI- MAL 値に 変換します	RAWTOHEX ('012')	'303132'

次のページに続く

表 2 – 前のページからの続き

INSTR(string, set, [start[, オカレンス]])	整数	文字列内の 文字セット の位置を検 索し、文字 列 string の 位 置 start か ら 開 始 し、セット の最初、2 番目、3 番 目などを検 索します。 セットが見 つからない 場合は 0 を 返します。	INSTR ('PETERPIPER- PICKEDaPECKOFPICK- LEDPEPPERS', 'PI', 1,3)	30
INSTRB (string, set)	整数	文字列内の セットの位 置を返しま す。セット が見つから ない場合は 0 を返しま す。	INSTRB (「PEPER- PIPERPICKEDaPECK- OFPICKLEDPEPPERS」、 「PICK」)	13
INSTRB(string, set, start)	整数	文 字 列 内 の開始位置 から始まる セットの位 置を返しま す。セット が見つから ない場合は 0 を返しま す。	INSTRB (「PEPER- PIPERPICKEDaPECK- OFPICKLEDPEPPERS」、 「PICK」、14)	30

次のページに続く

表 2 – 前のページからの続き

INSTRB (string, set, start, occurrence)	整数	文字列内の指定された集合の開始位置を開始位置から返します。セットが見つからない場合は 0 を返します。	INSTRB (「PEPER-PIPERPICKEDaPECK-OFPICKLEDPEPPERS」、 「PICK」、1、2)	30
LOWER (文字列)	CLOB	文字列を小文字に変換	LOWER ('TOM')	トム
SUBSTR (string, start[, count])	CLOB	start から始まり、count 文字に向かって部分文字列を抽出します。count が指定されていない場合、文字列は最初から最後まで切り取られます。	SUBSTR ('これはテストです', 6, 2)	は
SUBSTRB (string, start[, count])	CLOB	start と count がバイト数であることを除いて、SUBSTR と同じです。	SUBSTRB ('abc', 3) (2 バイト文字セットを想定)	c
SUBSTR2 (string, start[, count])	CLOB	SUBSTR のエイリアス。	SUBSTR2 ('これはテストです', 6, 2)	は

次のページに続く

表 2 – 前のページからの続き

SUBSTR2 (string、start[、count])	CLOB	SUBSTRB のエイリアス。	SUBSTR2 ('abc', 3) (2 バイト文字セットを想定)	c
SUBSTR4 (string、start[、count])	CLOB	SUBSTR のエイリアス。	SUBSTR4 ('これはテストです', 6,2)	は
SUBSTR4 (string、start[、count])	CLOB	SUBSTRB のエイリアス。	SUBSTR4 ('abc', 3) (2 バイト文字セットを想定)	c
SUBSTRC (string、start[、count])	CLOB	SUBSTR のエイリアス。	SUBSTRC ('これはテストです', 6,2)	は
SUBSTRC (string、start[、count])	CLOB	SUBSTRB のエイリアス。	SUBSTRC ('abc', 3) (2 バイト文字セットを想定)	c
TRIM ([LEADING/TRAILING/BOTH][文字]FROM 文字列)	CLOB	文字列の開始/終了/両端から文字のみ（デフォルトではスペース）を含む最も長い文字列を削除します。	TRIM (両 方 の 'x'FROM'xTomxx')	トム

次のページに続く

表 2 – 前のページからの続き

LTRIM (string[, set])	CLOB	指定された文字列の左から、set で指定されたすべての文字を削除します。set が指定されていない場合、デフォルトとして空白スペースが使用されます。	LTRIM('abcdefghi','abc')	デフィ
RTRIM (string[, set])	CLOB	指定された文字列の右側から、set で指定されたすべての文字を削除します。set が指定されていない場合、デフォルトとして空白スペースが使用されます。	RTRIM('abcdefghi','ghi')	abcdef
UPPER (文字列)	CLOB	文字列を大文字に変換	UPPER ('tom')	トム

追加の文字列操作関数が利用可能であり、次の表にリストされています。それらのいくつかは、上記の表にリストされている SQL 標準の文字列関数を実装するために内部的に使用されます。

機能	戻りタイプ	説明	例	結果
----	-------	----	---	----

次のページに続く

表 3 – 前のページからの続き

ASCII (文字列)	整数	引数の最初のバイトの ASCII コード	ASCII ('x')	120
CHR (整数)	CLOB	指定された ASCII コードの文字	CHR (65)	A
DECODE (expr, expr1a, expr1b[, expr2a, expr2b]...[, デフォルト])	expr1b, expr2b,... の引数タイプと同じ、デフォルト	expr1a, expr2a などと expr の最初の一致を検索します。一致が見つかった場合、対応するパラメーターペア expr1b, expr2bなどを返します。一致が見つからなかった場合、デフォルトを返します。一致が見つからず、デフォルトが指定されていない場合、null を返します。	DECODE (3, 1, 'One', 2, 'Two', 3, 'Three', '見つかりません')	三
INITCAP (文字列)	CLOB	各単語の最初の文字を大文字に変換し、残りを小文字に変換します。単語は、英数字以外の文字で区切られた英数字のシーケンスです。	INITCAP ('hiTHOMAS')	こんにちはトーマス
長さ	整数	文字列値の文字数を返します。	LENGTH ('Côté d'Azur')	11
長さ	整数	この関数の機能は、LENGTHと同じです。関数名は互換性のためにサポートされています。	LENGTHC ('コート d'アズール')	11
長さ 2	整数	この関数の機能は、LENGTHと同じです。関数名は互換性のためにサポートされています。	LENGTH2 ('コート d'アズール')	11

次のページに続く

表 3 – 前のページからの続き

長さ 4	整数	この関数の機能は、 LENGTH と同じです。関数名は互換性のためにサポートされています。	LENGTH4 ('コート d" アズール')	11
長さ	整数	指定された値を保持するために必要なバイト数を返します。	LENGTHB ('コート d" アズール')	12
LPAD (文字列、長さ INTEGER [, 塗りつぶし])	CLOB	文字列をサイズ、長さ、文字を先頭に追加して埋めます (デフォルトではスペース)。文字列がすでに長さよりも長い場合、(右側で) 切り捨てられます。	LPAD ('hi', 5, 'xy')	xyxhi
REPLACE (string, search_string[, replace_string])	CLOB	文字列の 1 つの値を別の値に置き換えます。replace_string の値を指定しない場合、search_string 値が見つかった場合は削除されます。	REPLACE ('GEORGE', 'GE', 'EG')	EGOREG
RPAD (文字列、長さ INTEGER [, 塗りつぶし])	CLOB	文字列をサイズ、長さ、文字を追加して塗りつぶします (デフォルトではスペース)。文字列がすでに長さよりも長い場合、切り捨てられます。	RPAD ('hi', 5, 'xy')	ヒックス
TRANSLATE (string, from, to)	CLOB	from セット内の文字に一致する string 内の文字は、to セット内の対応する文字に置き換えられます。	TRANSLATE ('12345', '14', 'ax')	a23x5

2.3.4.1 NULL との連結から生じる文字列テキストの切り捨て

注釈: このセクションでは、Oracle データベースと互換性のない機能について説明します。これにより、Oracle から AdvancedServer にデータを変換するときに、矛盾が生じる可能性があります。

AdvancedServer では、列の値が NULL の場合、列とテキスト文字列の連結により次のいずれかが発生する可能性があります。

- テキスト文字列の戻り
- テキスト文字列の消失（つまり、null 結果）

結果は、NULL 列のデータ型と連結の方法に依存します。

文字列連結演算子 '||' を使用する場合、テキストへの暗黙的な強制を持つ表データ型にリストされているテキストへの暗黙的な強制を持つ型は、入力パラメーターの 1 つが NULL である場合、文字列を切り捨てません明示的な型キャスト（つまり、::text）が使用されない限り、文字列は切り捨てられます。また、ヌルが存在する場合の一貫した動作を確認するには、CONCAT 関数を使用できます。

次のクエリは、テキストへの暗黙の強制があるデータ型をリストします。

```
SELECT castsource::regtype, casttarget::regtype, castfunc::regproc,
       CASE castcontext
         WHEN 'e' THEN 'explicit'
         WHEN 'a' THEN 'implicit in assignment'
         WHEN 'i' THEN 'implicit in expressions'
       END as castcontext,
       CASE castmethod
         WHEN 'f' THEN 'function'
         WHEN 'i' THEN 'input/output function'
         WHEN 'b' THEN 'binary-coercible'
       END as castmethod
FROM pg_cast
WHERE casttarget::regtype::text = 'text'
AND castcontext='i';
```

次の表に、クエリの結果を示します。

castsource	casttarget	castfunc	キャストコンテキスト	castmethod
性格	テキスト	pg_catalog.text	暗黙の表現	機能
さまざまなキャラクター	テキスト	–	暗黙の表現	バイナリ強制
/"char/"	テキスト	pg_catalog.text	暗黙の表現	機能
お名前	テキスト	pg_catalog.text	暗黙の表現	機能
pg_node_tree	テキスト	–	暗黙の表現	バイナリ強制

次のページに続く

表 4 – 前のページからの続き

pg_ndistinct	テキスト	–	暗黙の表現	入出力機能
pg_dependencies	テキスト	–	暗黙の表現	入出力機能
整数	テキスト	–	暗黙の表現	入出力機能
小さい	テキスト	–	暗黙の表現	入出力機能
oid	テキスト	–	暗黙の表現	入出力機能
日付	テキスト	–	暗黙の表現	入出力機能
倍精度	テキスト	–	暗黙の表現	入出力機能
本当の	テキスト	–	暗黙の表現	入出力機能
タイムゾーン付きの時間	テキスト	–	暗黙の表現	入出力機能
時間帯なしの時間	テキスト	–	暗黙の表現	入出力機能
タイムゾーン付きのタイムスタンプ	テキスト	–	暗黙の表現	入出力機能
間隔	テキスト	–	暗黙の表現	入出力機能
ビッグ	テキスト	–	暗黙の表現	入出力機能
数値	テキスト	–	暗黙の表現	入出力機能
タイムゾーンなしのタイムスタンプ	テキスト	–	暗黙の表現	入出力機能
記録する	テキスト	–	暗黙の表現	入出力機能
ブール値	テキスト	pg_catalog.text	暗黙の表現	機能
バイトア	テキスト	–	暗黙の表現	入出力機能

列の出力の詳細については、次の場所にある PostgreSQL コアドキュメントの pg_cast システムカタログを参照してください。

<https://www.postgresql.org/docs/current/static/catalog-pg-cast.html>

したがって、たとえば、データ型 UUID はこのリストに含まれていないため、テキストへの暗黙の強制はありません。その結果、NULL UUID 列を使用した特定の連結試行では、テキストが切り捨てられます。

この例では、すべての NULL 列の値を持つ単一の行で次の表が作成されます。

```
CREATE TABLE null_concat_types (
    boolean_type    BOOLEAN,
    uuid_type       UUID,
    char_type       CHARACTER
);

INSERT INTO null_concat_types VALUES (NULL, NULL, NULL);
```

boolean_type 列と char_type 列には、テキストに対する暗黙の強制がありますが、uuid_type 列にはありません。

したがって、列 boolean_type または char_type に対して連結演算子 ‘||’ を使用して文字列を連結すると、次のようになります。

```
SELECT 'x=' || boolean_type || 'y' FROM null_concat_types;
```

```
?column?
-----
x=y
(1 row)
```

```
SELECT 'x=' || char_type || 'y' FROM null_concat_types;
```

```
?column?
-----
x=y
(1 row)
```

ただし、列 `uuid_type` と連結すると、`x=` 文字列が失われます。

```
SELECT 'x=' || uuid_type || 'y' FROM null_concat_types;
```

```
?column?
-----
y
(1 row)
```

ただし、`::text` で明示的なキャストを使用すると、`x=` 文字列が失われなくなります。

```
SELECT 'x=' || uuid_type::text || 'y' FROM null_concat_types;
```

```
?column?
-----
x=y
(1 row)
```

`CONCAT` 関数を使用すると、`x=` 文字列も保持されます。

```
SELECT CONCAT('x=',uuid_type) || 'y' FROM null_concat_types;
```

```
?column?
-----
x=y
(1 row)
```

したがって、`NULL` 列のデータ型によっては、明示的なキャストまたは `CONCAT` 関数を使用して、テキスト文字列が失われないようにする必要があります。

2.3.4.2 SYS_GUID

SYS_GUID 関数は、グローバルに一意の識別子を生成して返します。識別子は、16 バイトの RAW データの形式を取ります。SYS_GUID 機能は uuid-osspl モジュールに基づいており、普遍的に一意の識別子を生成します。概要は次のとおりです。

`SYS_GUID()`

Example

次の例では、テーブル EMP に列を追加し、一意の識別子を挿入して、16-byte RAW 値を返します。

```
edb=# CREATE TABLE EMP (C1 RAW (16) DEFAULT SYS_GUID() PRIMARY KEY, C2 INT);
CREATE TABLE
edb=# INSERT INTO EMP (C2) VALUES (1);
INSERT 0 1
edb=# SELECT * FROM EMP;
           c1                               | c2
-----+-----
 \xb944970d3a1b42a7a2119265c49cbb7f | 1
(1 row)
```

2.3.5 Pattern Matching String Functions

AdvancedServer は、`REGEXP_COUNT`、`REGEXP_INSTR`、`REGEXP_SUBSTR` 機能のサポートを提供します。これらの関数は、正規表現で指定されたパターンの文字列を検索し、文字列内のパターンの出現に関する情報を返します。パターンは POSIX スタイルの正規表現でなければなりません。POSIX スタイルの正規表現の作成の詳細については、次のコアドキュメントを参照してください。

<https://www.postgresql.org/docs/current/static/functions-matching.html>

2.3.5.1 REGEXP_COUNT

`REGEXP_COUNT` は、文字列で正規表現を検索し、正規表現が発生した回数を返します。署名は次のとおりです。

```
INTEGER REGEXP_COUNT
(
    srcstr      TEXT,
    pattern     TEXT,
    position    DEFAULT 1
    modifier    DEFAULT NULL
)
```

Parameters

`srcstr`

`srcstr` は検索する文字列を指定します。

`pattern`

`pattern` は、`REGEXP_COUNT` が検索する正規表現を指定します。

`position`

`position` は、`REGEXP_COUNT` が検索を開始するソース文字列内の位置を示す整数値です。デフォルト値は 1 です。

`modifier`

`modifier` は、パターンマッチングの動作を制御する値を指定します。デフォルト値は `NULL` です。AdvancedServer でサポートされている修飾子の完全なリストについては、次から入手可能な PostgreSQL コアドキュメントを参照してください。

<https://www.postgresql.org/docs/current/static/functions-matching.html>

Example

次の簡単な例では、`REGEXP_COUNT` は、文字列 `'reinitializing'` で文字 `i` が使用された回数のカウントを返します。

```
edb=# SELECT REGEXP_COUNT('reinitializing', 'i', 1) FROM DUAL;
 regexp_count
-----
           5
(1 row)
```

最初の例では、コマンドは REGEXP_COUNT が最初の位置でカウントを開始するよう指示します。コマンドを変更して、6 番目の位置からカウントを開始する場合:

```
edb=# SELECT REGEXP_COUNT('reinitializing', 'i', 6) FROM DUAL;
 regexp_count
-----
           3
(1 row)
```

REGEXP_COUNT は 3 を返します。カウントは、6 番目の位置の前にある文字 i の出現を除外するようになりました。

2.3.5.2 REGEXP_INSTR

REGEXP_INSTR は、POSIX スタイルの正規表現の文字列を検索します。この関数は、一致が検出された文字列内の位置を返します。署名は次のとおりです。

```
INTEGER REGEXP_INSTR
(
  srcstr          TEXT,
  pattern          TEXT,
  position         INT  DEFAULT 1,
  occurrence       INT  DEFAULT 1,
  returnparam     INT  DEFAULT 0,
  modifier        TEXT DEFAULT NULL,
  subexpression    INT  DEFAULT 0,
)
```

Parameters:

srcstr

srcstr は検索する文字列を指定します。

pattern

pattern は、REGEXP_INSTR が検索する正規表現を指定します。

position

position は、ソース文字列の開始位置を示す整数値を指定します。デフォルト値は 1 です。

occurrence

occurrence は、検索される文字列で複数のパターンが発生した場合にどの一致が返されるかを指定します。デフォルト値は 1 です。

returnparam

returnparam は、REGEXP_INSTR が返す文字列内の位置を指定する整数値です。デフォルト値は 0 です。指定してください:

0 は、pattern に一致する最初の文字の文字列内の位置を返します。

pattern の末尾に続く最初の文字の位置を返す 0 より大きい値。

modifier

modifier は、パターンマッチングの動作を制御する値を指定します。デフォルト値は NULL です。AdvancedServer でサポートされている修飾子の完全なリストについては、次から入手可能な PostgreSQL コアドキュメントを参照してください。

<https://www.postgresql.org/docs/current/static/functions-matching.html>

subexpression

subexpression は、REGEXP_INSTR によって返される pattern の部分を識別する整数値です。subexpression のデフォルト値は 0 です。

subexpression の値を指定する場合、検索対象の値の一部を分離する pattern に 1 つ（または複数）の括弧を含める必要があります。subexpression で指定された値は、どの括弧のセットが返されるべきかを示します。たとえば、subexpression が 2 の場合、REGEXP_INSTR は括弧の 2 番目のセットの位置を返します。

Example

次の簡単な例では、REGEXP_INSTR は、3 つの連続した数字を含むパターンの最初の出現について、電話番号を含む文字列を検索します。

```
edb=# SELECT REGEXP_INSTR('800-555-1212', '[0-9][0-9][0-9]', 1, 1) FROM DUAL;
 regexp_instr
-----
          1
(1 row)
```

このコマンドは、REGEXP_INSTR に指示して最初に出現した位置を返します。コマンドを変更して、3 桁の連続した数字の 2 番目の出現の開始を返す場合:

```
edb=# SELECT REGEXP_INSTR('800-555-1212', '[0-9][0-9][0-9]', 1, 2) FROM DUAL;
 regexp_instr
```

(次のページに続く)

(前のページからの続き)

5
(1 row)

REGEXP_INSTR は 5 を返します。3 番目の連続した数字の 2 番目の出現は、5/:sup:th` の位置から始まります。

2.3.5.3 REGEXP_SUBSTR

REGEXP_SUBSTR 関数は、POSIX 準拠の正規表現で指定されたパターンの文字列を検索します。REGEXP_SUBSTR は、関数の呼び出しで指定されたパターンに一致する文字列を返します。関数のシグネチャは次のとおりです。

```
TEXT REGEXP_SUBSTR
(
  srcstr          TEXT,
  pattern          TEXT,
  position         INT DEFAULT 1,
  occurrence       INT DEFAULT 1,
  modifier         TEXT DEFAULT NULL,
  subexpression    INT DEFAULT 0
)
```

Parameters:

srcstr

srcstr は検索する文字列を指定します。

pattern

pattern は、REGEXP_SUBSTR が検索する正規表現を指定します。

position

position は、ソース文字列の開始位置を示す整数値を指定します。デフォルト値は 1 です。

occurrence

occurrence は、検索される文字列で複数のパターンが発生した場合にどの一致が返されるかを指定します。デフォルト値は 1 です。

modifier

modifier は、パターンマッチングの動作を制御する値を指定します。デフォルト値は NULL です。AdvancedServer でサポートされている修飾子の完全なリストについては、次から入手可能な PostgreSQL コアドキュメントを参照してください。

<https://www.postgresql.org/docs/current/static/functions-matching.html>

subexpression

subexpression は、REGEXP_SUBSTR によって返される pattern の部分を識別する整数値です。subexpression のデフォルト値は 0 です。

subexpression の値を指定する場合、検索対象の値の一部を分離する pattern に 1 つ（または複数）の括弧を含める必要があります。subexpression で指定された値は、どの括弧のセットが返されるべき

かを示します。たとえば、`subexpression` が 2 の場合、`REGEXP_SUBSTR` は括弧の 2 番目のセットに含まれる値を返します。

Example

次の簡単な例では、`REGEXP_SUBSTR` は 3 つの連続した数字の最初のセットの電話番号を含む文字列を検索します。

```
edb=# SELECT REGEXP_SUBSTR('800-555-1212', '[0-9][0-9][0-9]', 1, 1) FROM
DUAL;
 regexp_substr
-----
      800
(1 row)
```

3 桁の最初の出現箇所を見つけ、文字列 (800) を返します。コマンドを変更して、連続する 3 桁の数字の 2 番目の出現を確認する場合:

```
edb=# SELECT REGEXP_SUBSTR('800-555-1212', '[0-9][0-9][0-9]', 1, 2) FROM
DUAL;
 regexp_substr
-----
      555
(1 row)
```

`REGEXP_SUBSTR` は、2 番目のサブストリングの内容である 555 を返します。

2.3.6 Pattern Matching Using the LIKE Operator

AdvancedServer は、従来の SQL LIKE 演算子を使用したパターンマッチングを提供します。LIKE 演算子の構文は次のとおりです。

```
string LIKE pattern [ ESCAPE escape-character ]
string NOT LIKE pattern [ ESCAPE escape-character ]
```

すべての pattern は文字列のセットを定義します。pattern で表される文字列のセットに string が含まれている場合、LIKE 式は TRUE を返します。予想どおり、LIKE が TRUE を返す場合、NOT LIKE 式は FALSE を返します。同等の式は NOT (string LIKE pattern) です。

pattern にパーセント記号またはアンダースコアが含まれていない場合、パターンは文字列のみを表します。その場合、LIKE は等号演算子のように機能します。pattern のアンダースコア () は、任意の 1 文字を表します (一致します)。パーセント記号 () は、0 個以上の文字の文字列に一致します。

いくつかの例:

```
'abc' LIKE 'abc'      true
'abc' LIKE 'a%'       true
'abc' LIKE '_b_'      true
'abc' LIKE 'c'        false
```

LIKE パターンマッチは常に文字列全体をカバーします。したがって、文字列内の任意の場所でパターンに一致するには、パターンの先頭と末尾をパーセント記号にする必要があります。

他の文字と一致せずにリテラルのアンダースコアまたはパーセント記号と一致するには、pattern の各文字の前にエスケープ文字が必要です。デフォルトのエスケープ文字はバックスラッシュですが、ESCAPE 句を使用して別のエスケープ文字を選択できます。エスケープ文字自体に一致させるには、2 つのエスケープ文字を記述します。

バックスラッシュは文字列リテラルですすでに特別な意味を持つため、バックスラッシュを含むパターン定数を記述するには、SQL ステートメントに 2 つのバックスラッシュを記述する必要があります。したがって、実際にリテラルのバックスラッシュに一致するパターンを記述することは、ステートメントに 4 つのバックスラッシュを記述することを意味します。これを回避するには、ESCAPE で別のエスケープ文字を選択します。その場合、バックスラッシュは LIKE にとって特別ではなくなりました。(ただし、文字列リテラルパーサーにとっては特別なものなので、まだ 2 つ必要です。)

ESCAPE ' ' を書き込むことにより、エスケープ文字を選択しないこともできます。これにより、エスケープメカニズムが事実上無効になり、パターン内のアンダースコアとパーセント記号の特別な意味をオフにすることができなくなります。

2.3.7 Data Type Formatting Functions

次の表で説明する AdvancedServer フォーマット関数は、さまざまなデータ型（日付/時刻、整数、浮動小数点、数値）をフォーマットされた文字列に変換し、フォーマットされた文字列から特定のデータ型に変換するための強力なツールセットを提供します。これらの関数はすべて共通の呼び出し規則に従います。最初の引数はフォーマットされる値であり、2 番目の引数は出力または入力フォーマットを定義する文字列テンプレートです。

機能	戻りタイプ	説明	例	結果
TO_CHAR (DATE[, format])	VARCHAR2	日付/時刻を出力形式の文字列に変換します。省略した場合、デフォルトの形式は DD-MON-YY です。	TO_CHAR (SYSDATE, 'MM/DD/YYYYHH12:MI:SSAM')	07/25/200709:43:02AM
TO_CHAR (TIMESTAMP[, format])	VARCHAR2	タイムスタンプを出力形式の文字列に変換します。省略した場合、デフォルトの形式は DD-MON-YY です。	TO_CHAR (CURRENT_TIMESTAMP, 'MM/DD/YYYYHH12:MI:SSAM')	08/13/201508:55:22PM

次のページに続く

表 5 – 前のページからの続き

TO_CHAR (INTEGER[, format])	VARCHAR2	整数を出力形式で文字列に変換します	TO_CHAR (2412, '999,999S')	2,412+
TO_CHAR (NUMBER[, format])	VARCHAR2	10 進数を出力形式の文字列に変換します	TO_CHAR (10125.35, '999,999.99')	10,125.35
TO_CHAR (DOUBLEPRECISION, 形式)	VARCHAR2	浮動小数点数を出力形式の文字列に変換します	TO_CHAR (CAST (123.5282ASREAL), '999.99')	123.53
TO_DATE (string[, format])	DATE	日付形式の文字列を DATE データ型に変換する	TO_DATE ('2007-07-0413:39:10', 'YYYY-MM-DDHH24:MI:SS')	04-JUL-07
TO_NUMBER (string[, format])	NUMBER	数値形式の文字列を NUMBER データ型に変換します	TO_NUMBER ('2,412-', '999,999S')	-2412
TO_TIMESTAMP (string, format)	タイムスタンプ	タイムスタンプ形式の文字列を TIMESTAMPTZ データ型に変換します	TO_TIMESTAMP ('05Dec200008:30:25pm', 'DDMonYYYYhh12:mi:sspm')	05-DEC-0020:30:25+05:30

次のページに続く

表 5 – 前のページからの続き

TO_TIMESTAMP_TZ (string, format)	タイムスタンプ	タイムスタンプ形式の文字列をTIMESTAMP_TZデータ型に変換します	TO_TIMESTAMP_TZ ('2003/12/13 10:13:18-8:00', 'YYYY/MM/DDHH:MI:SSSTZH:TZM')	12 月 13 日 23:43:18+05:30
----------------------------------	---------	--------------------------------------	--	--------------------------------

2.3.7.1 TO_CHAR、TO_DATE、TO_TIMESTAMP、TO_TIMESTAMP_TZ

出力テンプレート文字列（TO_CHAR の場合）には、認識される特定のパターンがあり、フォーマットされる値から適切にフォーマットされたデータに置き換えられます。テンプレートパターンではないテキストは、そのままコピーされます。同様に、入力テンプレート文字列（TO_CHAR 以外の場合）で、テンプレートパターンは入力データ文字列の一部とそこにある値を特定します。

TO_TIMESTAMP、TO_TIMESTAMP_TZ、または TO_DATE を呼び出すときに日付、月、または年を指定しない場合、デフォルトでは、出力形式はそれぞれ現在の月または現在の年の最初の日付を考慮します。次の例では、日付、月、年は入力文字列で指定されていません。TO_TIMESTAMP、TO_TIMESTAMP_TZ、TO_DATE は、現在の月と現在の年の最初の日付のデフォルト値を返します。

```
edb=# select to_timestamp('12', 'HH');
         to_timestamp
-----
01-MAY-20 12:00:00 +05:30
(1 row)

edb=# select to_timestamp_tz('12', 'HH');
         to_timestamp_tz
-----
01-MAY-20 12:00:00 +05:30
(1 row)

edb=# select to_date('12', 'HH');
         to_date
-----
01-MAY-20 12:00:00
(1 row)
```

次の表は、TO_CHAR、TO_DATE、TO_TIMESTAMP、および TO_TIMESTAMP_TZ 機能を使用して日付値をフォーマットするために使用可能なテンプレートパターンを示しています。

模様	説明
HH	時刻 (01-12)
HH12	時刻 (01-12)
HH24	時刻 (00-23)
MI	分 (00-59)
SS	2 番目 (00-59)
SSSSS	午前 0 時からの秒数 (0-86399)
FFn	秒の小数部。n は、返される桁数の 1~9 のオプションの整数です。省略した場合、デフォルトは 6 です。
午前または午前または PM または P.M.	子午線インジケーター (大文字)
午前または午前または午後または午後	子午線インジケーター (小文字)
Y、YYY	カンマ付きの年 (4 桁以上)
年	年 (つづり)
シアー	年 (スペルアウト) (BC の日付の先頭にマイナス記号が付きます)
YYYY	年 (4 桁以上)
SYYYYY	年 (4 桁以上) (BC 日付の前にマイナス記号が付きます)
YYY	年の下 3 桁
YY	年の下 2 桁
Y	年の下 1 桁
IYYY	ISO 年 (4 桁以上)
IYY	ISO 年の下 3 桁
IY	ISO 年の下 2 桁
私は	ISO 年の下 1 桁
BC または B.C. または AD または A.D.	時代インジケーター (大文字)
bc または b.c. または ad または a.d.	時代インジケーター (小文字)
月	完全な大文字の月名
月	完全に大文字と小文字が混在する月名
月	完全な小文字の月名
月	省略形の大文字の月名 (英語では 3 文字、ローカライズされた長さは異なります)
月	大/小文字混合の月名 (英語では 3 文字、ローカライズされた長さは異なります)
月	短縮された小文字の月名 (英語では 3 文字、ローカライズされた長さは異なります)

次のページに続く

表 6 – 前のページからの続き

MM	月番号 (01-12)
DAY	完全な大文字の曜日名
日	完全な大文字と小文字の曜日名
日	完全な小文字の曜日名
DY	短縮された大文字の曜日名 (英語では 3 文字、ローカライズされた長さは異なります)
ダイ	大/小文字混合の曜日名 (英語では 3 文字、ローカライズされた長さは異なります)
dy	短縮された小文字の曜日名 (英語では 3 文字、ローカライズされた長さは異なります)
DDD	年の日 (001-366)
DD	月の日 (01-31)
D	曜日 (1~7。日曜日は 1)
W	月の週 (1~5) (最初の週は月の最初の日から始まります)
WW	年の週番号 (1~53) (最初の週は年の最初の日から始まります)
IW	年の ISO 週番号。新年の最初の木曜日は週 1 です
CC	世紀 (2 桁) ;21 世紀は 2001-01-01 に始まります
SCC	BC 日付の前にマイナス記号が付いていることを除き、CC と同じ
J	ユリウス日 (紀元前 4712 年 1 月 1 日からの日)
Q	四半期
RM	ローマ数字の月 (I-XII;I=January) (大文字)
rm	ローマ数字の月 (i-xii;i=January) (小文字)
RR	年の下 2 桁のみを指定した場合、年の上 2 桁。結果は、現在の年と指定された 2 桁の年を使用するアルゴリズムに基づいています。指定された 2 桁の年の最初の 2 桁は、次の例外を除き、現在の年の最初の 2 桁と同じです。 指定された 2 桁の年が 50 未満で、現在の年の下 2 桁が 50 以上の場合、指定された年の最初の 2 桁は現在の年の最初の 2 桁より 1 大きくなります。 指定された 2 桁の年が 50 以上で、現在の年の下 2 桁が 50 未満の場合、指定された年の最初の 2 桁は現在の年の最初の 2 桁よりも 1 少なくなります。
RRRR	TO_DATE 関数にのみ影響します。2 桁または 4 桁の年を指定できます。2 桁の年を指定すると、RR 形式のように年の最初の 2 桁を返します。4 桁の年が指定されている場合、指定された 4 桁の年を返します。
TZH	タイムゾーン時間
TZM	タイムゾーン分

2.3.7.1.1 日付と時刻の修飾子

テンプレートパターンに特定の修飾子を適用して、その動作を変更できます。たとえば、FMMonth は、FM 修飾子を持つ Month パターンです。次の表に、日付/時刻形式の修飾子パターンを示します。

修飾子	説明	例
FM プレフィックス	塗りつぶしモード（空白とゼロの埋め込みを抑制）	FMMonth
TH サフィックス	大文字の序数の接尾辞	DDTH
番目の接尾辞	小文字の序数の接尾辞	DDth
FX プレフィックス	固定形式のグローバルオプション（使用上の注意を参照）	FX 月 DD 日
SP サフィックス	スペルモード	DDSP

日付/時刻形式の使用上の注意:

- FM は、パターンの出力を固定幅にするために追加される先行ゼロと後続ブランクを抑制します。
- FX オプションを使用しない場合、TO_TIMESTAMP、TO_TIMESTAMP_TZ、TO_DATE は入力文字列の複数の空白スペースをスキップします。テンプレートの最初の項目として FX を指定する必要があります。例:

```
TO_TIMESTAMP('2000 - JUN', 'YYYY-MON') is correct, but
TO_TIMESTAMP('2000    JUN', 'FXYYYY MON') and
TO_TIMESTAMP_TZ('2000    JUN', 'FXYYYY MON') returns an error
because TO_TIMESTAMP and TO_TIMESTAMP_TZ expects one space
only.
```

- 通常のテキストは TO_CHAR テンプレートで許可され、そのまま出力されます。
- 文字列から timestamp、timestampz、または date への変換では、YYY、YYYY または Y,YYY フィールドがある場合、CC フィールドは無視されます。CC が YY または Y とともに使用される場合、年は「10」として計算されます。

次の表に、TO_CHAR および TO_DATE 機能の使用例をいくつか示します。

表現	結果
TO_CHAR (CURRENT_TIMESTAMP, 'Day, DDHH12:MI:SS')	「火曜日、0605:39:18」
TO_CHAR (CURRENT_TIMESTAMP, 'FMDay, FMDDHH12:MI:SS')	「火曜日、605:39:18」
TO_CHAR (-0.1, '99.99')	「-.10」
TO_CHAR (-0.1, 'FM9.99')	「-.1」
TO_CHAR (0.1, '0.9')	「0.1」
TO_CHAR (12, '9990999.9')	「0012.0」
TO_CHAR (12, 'FM9990999.9')	「0012」

次のページに続く

表 7 – 前のページからの続き

TO_CHAR (485, '999')	「485」
TO_CHAR (-485, '999')	「-485」
TO_CHAR (1485, '9,999')	「1,485」
TO_CHAR (1485, '9G999')	「1,485」
TO_CHAR (148.5, '999.999')	「148.500」
TO_CHAR (148.5, 'FM999.999')	「148.5」
TO_CHAR (148.5, 'FM999.990')	「148.500」
TO_CHAR (148.5, '999D999')	「148.500」
TO_CHAR (3148.5, '9G999D999')	「3,148.500」
TO_CHAR (-485, '999S')	「485-」
TO_CHAR (-485, '999MI')	「485-」
TO_CHAR (485, '999MI')	「485」
TO_CHAR (485, 'FM999MI')	「485」
TO_CHAR (-485, '999PR')	'<485>'
TO_CHAR (485, 'L999')	「\$485」
TO_CHAR (485, 'RN')	「CDLXXXV」
TO_CHAR (485, 'FMRN')	「CDLXXXV」
TO_CHAR (5.2, 'FMRN')	「V」
TO_CHAR (12, '99V999')	「12000」
TO_CHAR (12.4, '99V999')	「12400」
TO_CHAR (12.45, '99V9')	「125」

次の表に、TO_TIMESTAMP_TZ 関数の使用例をいくつか示します。

表現	結果
TO_TIMESTAMP_TZ ('12-JAN-2010', 'DD-MONTH-YYYY')	'12-JAN-1000:00:00+05:30'
TO_TIMESTAMP_TZ ('03-APR-0709:12:21P.M', 'DD-MON-YYHH12:MI:SSA.M')	'03-APR-0709:12:21+05:30'
TO_TIMESTAMP_TZ ('2003/12/1310:13:18-8:00', 'YYYY/MM/DDHH:MI:SSTZH:TZM')	'13-DEC-0323:43:18+05:30'
TO_TIMESTAMP_TZ ('20-MAR-2004:30:00+08:00', 'DD-MON-YYHH:MI:SSTZH:TZM') ;	'20-MAR-2002:00:00+05:30'
TO_TIMESTAMP_TZ ('10-Sep-0214:10:10.123000', 'DD-MON-RRHH24:MI:SS.FF') ;	'10-SEP-0214:10:10.123+05:30'

2.3.7.2 IMMUTABLE TO_CHAR (TIMESTAMP, format) 関数

IMMUTABLE 形式の関数を使用する可能性がある TO_CHAR 関数の特定のケースがあります。基本的に、関数がデータベースを変更しない場合、関数は IMMUTABLE であり、関数は入力パラメーターのみに依存して同じ一貫した値を返します。つまり、構成パラメーター、ロケール、データベースの内容などの設定は、関数によって返される結果に影響しません。

関数のボラティリティカテゴリ VOLATILE、STABLE、および IMMUTABLE の詳細については、次の PostgreSQL Core ドキュメントを参照してください。

<https://www.postgresql.org/docs/current/static/xfunc-volatility.html>

IMMUTABLE 関数の特別な利点は、CREATE INDEX コマンドで使用して、その関数に基づいてインデックスを作成できることです。

TO_CHAR 機能が IMMUTABLE フォームを使用するには、次の条件が満たされている必要があります。

- TO_CHAR 関数の最初のパラメーターは、データ型 TIMESTAMP でなければなりません。
- TO_CHAR 関数の 2 番目のパラメーターで指定された形式は、言語、ロケールなどの要因に基づいて関数の戻り値に影響を与えてはなりません。たとえば、IMMUTABLE フォームには 'YYYY-MM-DD HH24:MI:SS' の形式を使用できます。ロケールの設定に関係なく、関数の結果は数値形式でのみ表現された日付と時刻であるため、関数のただし、'DD-MON-YYYY' の形式は IMMUTABLE 形式の関数には使用できません。これは、3 文字の月の省略形がロケール設定に応じて異なる結果を返す可能性があるためです。

不変ではない関数を生成する形式パターンには、スペルアウトまたは短縮された月 (MONTH, MON)、日 (DAY, DY)、中央値インジケータ (AM, PM)、または時代インジケータ (BC, AD) のバリエーションが含まれます。

次の例では、TIMESTAMP 列を持つテーブルが作成されます。

```
CREATE TABLE ts_tbl (ts_col TIMESTAMP);
```

以下は、IMMUTABLE 形式の TO_CHAR 関数を使用したインデックスの正常な作成を示しています。

```
edb=# CREATE INDEX ts_idx ON ts_tbl (TO_CHAR(ts_col, 'YYYY-MM-DD HH24:MI:SS'));
CREATE INDEX
edb=# \ds ts_idx
```

Column	Type	Index "public.ts_idx"	Definition
to_char	character varying	to_char(ts_col, 'YYYY-MM-DD HH24:MI:SS'::character varying)	

```
btree, for table "public.ts_tbl"
```

3 文字の月の省略形 MON はロケール設定に基づいて異なる戻り値になる可能性があるため、TO_CHAR 関数で指定された形式により IMMUTABLE フォームの使用が妨げられるため、次の結果はエラーになります。

```
edb=# CREATE INDEX ts_idx_2 ON ts_tbl1 (TO_CHAR(ts_col, 'DD-MON-YYYY'));
ERROR:  functions in index expression must be marked IMMUTABLE
```

2.3.7.3 TO_NUMBER

次の表に、数値の書式設定に使用できるテンプレートパターンを示します。

模様	説明
9	指定された桁数の値
0	先行ゼロ付きの値
。(期間)	小数点
、(コンマ)	グループ(千)区切り記号
\$	ドル記号
S	数字に固定された記号(ロケールを使用)。
L	通貨記号(ロケールを使用)

9 は、9s と同じ桁数の値になります。数字が使用できない場合、サーバーは対応する 9s を無視します。S パターンは + をサポートせず、\$ パターンは式の小数点をサポートしません。

次の表に、TO_NUMBER 関数の使用例をいくつか示します。

表現	結果
TO_NUMBER ('-65', 'S99')	'-65'
TO_NUMBER ('\$65', 'L99')	「65」
TO_NUMBER ('9678584', '9999999')	'9678584'
TO_NUMBER ('123,456,789', '999,999,999')	'123456789'
TO_NUMBER ('1210.73', '9999.99')	'1210.73'
TO_NUMBER ('1210.73')	'1210.73'
TO_NUMBER ('0101.010', 'FM99999999.99999')	'101.010'

2.3.7.3.1 数値修飾子

次の表は、数値フォーマットの修飾子パターンを示しています。

模様	説明	例
FM プレフィックス	塗りつぶしモード(末尾のゼロと空白の埋め込みを抑制します)	FM99.99

2.3.8 Date/Time Functions and Operators

日付/時刻関数の表は、日付/時刻値の処理に使用可能な関数を示し、詳細は次のサブセクションに表示されます。次の表は、基本的な算術演算子 (+, -) の動作を示しています。書式設定関数については、IMMUTABLETO_CHAR (TIMESTAMP、format) 関数を参照してください。

オペレーター	例	結果
プラス (+)	日付'2001-09-28'+7	05-OCT-0100:00:00
プラス (+)	タイムスタンプ'2001-09-2813:30:00'+3	01-OCT-0113:30:00
マイナス (-)	日付'2001-10-01'-7	24-SEP-0100:00:00
マイナス (-)	タイムスタンプ'2001-09-2813:30:00'-3	25-SEP-0113:30:00
マイナス (-)	タイムスタンプ'2001-09-2903:00:00'-タイムスタンプ'2001-09-2712:00:00'	@1 日 15 時間

次の表の日付/時刻関数では、DATE データ型と TIMESTAMP データ型の使用は交換可能です。

機能	戻りタイプ	説明	例	結果
ADD_MONTHS (DATE、NUMBER)	DATE	日付に月を追加する	ADD_MONTHS ('28-FEB-97'、3.8	97 年 5 月 31 日 00:00:00
CURRENT_DATE	DATE	現在の日付	CURRENT_DATE	04-JUL-07
CURRENT_TIMESTAMP	タイムスタンプ	現在の日付と時刻を返します	CURRENT_TIMESTAMP	04-JUL-0715:33:23.484
抽出物 (タイムスタンプからのフィールド)	倍精度	サブフィールドを取得	抽出 (TIMESTAMP'2001-02-1620:38:40' からの時間)	20

次のページに続く

表 8 – 前のページからの続き

LAST_DAY (DATE)	DATE	指定された日付で表される月の最終日を返します。指定された日付に時刻部分が含まれる場合、変更されずに結果に繰り越されます。	LAST_DAY ('14-APR-98')	98 年 4 月 30 日 00:00:00
LOCALTIMESTAMP[(精度)]		現在の日付と時刻（現在のトランザクションの開始	LOCALTIMESTAMP	04-JUL-07 15:33:23.484
MONTHS_BETWEEN (DATE, DATE)	NUMBER	2 つの日付間の月数	MONTHS_BETWEEN ('28-FEB-07', '30-NOV-06')	3
NEXT_DAY (DATE, dayofweek)	DATE	指定された日付の次の曜日に該当する日付	NEXT_DAY ('16-APR-07', 'FRI')	20-APR-07 00:00:00
NEW_TIME (DATE, VARCHAR, VARCHAR)	DATE	日付と時刻を別のタイムゾーンに変換します	NEW_TIME (TO_DATE '2005/05/29 12:45:00', 'AST', 'PST')	2005/05/29 14:45:00

次のページに続く

表 8 – 前のページからの続き

NUMTODSINTERVAL (数値、間隔)	間隔	数値を指定された日または秒の間隔に変換します	SELECT numtodsinterval (100, 'hour') ;	4 日間 04:00:00
NUMTOYMINTERVAL (数値、間隔)	間隔	数値を指定された年または月の間隔に変換します。	SELECT numtoyminterval (100, 'month') ;	8 年 4 ヶ月
ROUND (DATE[, format])	DATE	形式に従って丸められた日付	ROUND (TO_DATE ('29-MAY-05') , 'MON')	01-JUN-0500:00:00
SYS_EXTRACT_UTC (TIMESTAMPWITH- TIMEZONE)	タイムスタンプ	協定世界時を返します	SYS_EXTRACT_UTC (CAST ('24-MAR- 1112:30:00PM- 04:00' AS TIMESTAMPWITH- TIMEZONE)))	24-MAR-1116:30:00
SYSDATE	DATE	現在の日付と時刻を返します	SYSDATE	01-AUG-1211:12:34
SYSTIMESTAMP ()	タイムスタンプ	現在の日付と時刻を返します	SYSTIMESTAMP	01-AUG-1211:11:23。 665229-07:00
TRUNC (DATE[フ ォーマット])	DATE	形式に従って切り捨てる	TRUNC (TO_DATE ('29-MAY-05') , 'MON')	05 年 5 月 1 日 00:00:00

2.3.8.1 ADD_MONTHS

ADD_MONTHS 関数は、指定された日付に指定された月数を加算（または 2 番目のパラメーターが負の場合は減算）します。月の最終日は、特定の日付の月の日と同じです。ただし、日が月の最終日である場合は、結果の日付は常に月の最終日になります。

月数パラメータの小数部分は、計算を実行する前に切り捨てられます。

指定された日付に時刻部分が含まれる場合、変更されずに結果に繰り越されます。

以下は ADD_MONTHS 機能の例です。

```
SELECT ADD_MONTHS('13-JUN-07',4) FROM DUAL;

      add_months
-----
13-OCT-07 00:00:00
(1 row)

SELECT ADD_MONTHS('31-DEC-06',2) FROM DUAL;

      add_months
-----
28-FEB-07 00:00:00
(1 row)

SELECT ADD_MONTHS('31-MAY-04',-3) FROM DUAL;

      add_months
-----
29-FEB-04 00:00:00
(1 row)
```

2.3.8.2 現在の日付/時刻

AdvancedServer は、現在の日付と時刻に関連する値を返す多くの機能を提供します。これらの関数はすべて、現在のトランザクションの開始時間に基づいて値を返します。

- CURRENT_DATE
- CURRENT_TIMESTAMP
- LOCALTIMESTAMP
- LOCALTIMESTAMP (precision)

CURRENT_DATE は、現在のトランザクションの開始時間に基づいて現在の日付と時刻を返します。トランザクション内で複数呼び出された場合、CURRENT_DATE の値は変わりません。

```
SELECT CURRENT_DATE FROM DUAL;
```

```
date
```

```
-----
06-AUG-07
```

CURRENT_TIMESTAMP は現在の日付と時刻を返します。単一の SQL ステートメントから呼び出されると、ステートメント内の各出現に対して同じ値を返します。トランザクション内の複数のステートメントから呼び出された場合、発生ごとに異なる値を返すことがあります。関数から呼び出された場合、呼び出し元の `current_timestamp` によって返される値とは異なる値を返す場合があります。

```
SELECT CURRENT_TIMESTAMP, CURRENT_TIMESTAMP FROM DUAL;
```

```
current_timestamp | current_timestamp
```

```
-----+-----
02-SEP-13 17:52:29.261473 +05:00 | 02-SEP-13 17:52:29.261474 +05:00
```

LOCALTIMESTAMP には、オプションで精度パラメーターを指定できます。これにより、秒フィールドの小数桁に丸められます。精度パラメーターがない場合、結果は使用可能な完全な精度で提供されます。

```
SELECT LOCALTIMESTAMP FROM DUAL;
```

```
timestamp
```

```
-----
06-AUG-07 16:11:35.973
(1 row)
```

```
SELECT LOCALTIMESTAMP(2) FROM DUAL;
```

```
timestamp
```

```
-----
06-AUG-07 16:11:44.58
(1 row)
```

これらの関数は現在のトランザクションの開始時間を返すため、トランザクション中に値は変更されません。これは機能と見なされます。その目的は、単一のトランザクションに「現在の」時間の一貫した概念を持たせ、同じトランザクション内の複数の変更と同じタイムスタンプを付与することです。他のデータベースシステムでは、これらの値をより頻繁に進める場合があります。

2.3.8.3 抽出物

EXTRACT 関数は、日付/時刻値から年や時間などのサブフィールドを取得します。EXTRACT 関数は、DOUBLE PRECISION タイプの値を返します。有効なフィールド名は次のとおりです。

YEAR

年フィールド

```
SELECT EXTRACT(YEAR FROM TIMESTAMP '2001-02-16 20:38:40') FROM DUAL;

date_part
-----
      2001
(1 row)
```

MONTH

年内の月の番号 (1~12)

```
SELECT EXTRACT(MONTH FROM TIMESTAMP '2001-02-16 20:38:40') FROM DUAL;

date_part
-----
        2
(1 row)
```

DAY

日 (月) フィールド (1~31)

```
SELECT EXTRACT(DAY FROM TIMESTAMP '2001-02-16 20:38:40') FROM DUAL;

date_part
-----
       16
(1 row)
```

HOURL

時間フィールド (0-23)

```
SELECT EXTRACT(HOUR FROM TIMESTAMP '2001-02-16 20:38:40') FROM DUAL;

date_part
-----
       20
(1 row)
```

MINUTE

分フィールド (0-59)

```
SELECT EXTRACT(MINUTE FROM TIMESTAMP '2001-02-16 20:38:40') FROM DUAL;
```

```
date_part
-----
          38
(1 row)
```

SECOND

秒のフィールド (小数部分を含む) (0-59)

```
SELECT EXTRACT(SECOND FROM TIMESTAMP '2001-02-16 20:38:40') FROM DUAL;
```

```
date_part
-----
          40
(1 row)
```

2.3.8.4 MONTHS_BETWEEN

MONTHS_BETWEEN 関数は、2 つの日付間の月数を返します。結果は、最初の日付が 2 番目の日付より大きい場合は正、最初の日付が 2 番目の日付より小さい場合は負の数値です。

両方の日付パラメーターの月の日が同じである場合、または両方の日付パラメーターがそれぞれの月の最終日に該当する場合、結果は常に整数の月になります。

以下は MONTHS_BETWEEN 機能の例です。

```
SELECT MONTHS_BETWEEN('15-DEC-06','15-OCT-06') FROM DUAL;
```

```
months_between
-----
              2
(1 row)
```

```
SELECT MONTHS_BETWEEN('15-OCT-06','15-DEC-06') FROM DUAL;
```

```
months_between
-----
             -2
(1 row)
```

```
SELECT MONTHS_BETWEEN('31-JUL-00','01-JUL-00') FROM DUAL;
```

(次のページに続く)

(前のページからの続き)

```

months_between
-----
          0.967741935
(1 row)

SELECT MONTHS_BETWEEN('01-JAN-07','01-JAN-06') FROM DUAL;

months_between
-----
              12
(1 row)

```

2.3.8.5 NEXT_DAY

NEXT_DAY 関数は、指定された日付より厳密に大きい指定された曜日の最初の出現を返します。少なくとも曜日の最初の 3 文字を指定する必要があります (例:SAT)。指定された日付に時刻部分が含まれる場合、変更されずに結果に繰り越されます。

以下は NEXT_DAY 機能の例です。

```

SELECT NEXT_DAY(TO_DATE('13-AUG-07','DD-MON-YY'),'SUNDAY') FROM DUAL;

      next_day
-----
19-AUG-07 00:00:00
(1 row)

SELECT NEXT_DAY(TO_DATE('13-AUG-07','DD-MON-YY'),'MON') FROM DUAL;

      next_day
-----
20-AUG-07 00:00:00
(1 row)

```

2.3.8.6 NEW_TIME

NEW_TIME 関数は、日付と時刻をあるタイムゾーンから別のタイムゾーンに変換します。NEW_TIME は DATE タイプの値を返します。構文は次のとおりです。

```
NEW_TIME (DATE, time_zone1, time_zone2)
```

time_zone1 と time_zone2 は、次の表の「タイムゾーン」列の文字列値である必要があります。

タイムゾーン	UTC からのオフセット	説明
AST	UTC+4	大西洋標準時
ADT	UTC+3	大西洋夏時間
BST	UTC+11	ベーリング標準時間
BDT	UTC+10	ベーリング夏時間
CST	UTC+6	中部標準時
CDT	UTC+5	中央夏時間
EST	UTC+5	東部標準時
EDT	UTC+4	東部夏時間
GMT	UTC	グリニッジ標準時
HST	UTC+10	アラスカ-ハワイ標準時
HDT	UTC+9	アラスカ-ハワイ夏時間
MST	UTC+7	山地標準時
MDT	UTC+6	山の夏時間
NST	UTC+3:30	ニューファンドランド標準時
PST	UTC+8	太平洋標準時
PDT	UTC+7	太平洋夏時間
YST	UTC+9	ユーコン標準時
YDT	UTC+8	ユーコン夏時間

NEW_TIME 関数の例を次に示します。

```
SELECT NEW_TIME(TO_DATE('08-13-07 10:35:15','MM-DD-YY HH24:MI:SS'),'AST',
'PST') "Pacific Standard Time" FROM DUAL;
```

Pacific Standard Time

13-AUG-07 06:35:15
(1 row)

2.3.8.7 NUMTODSINTERVAL

NUMTODSINTERVAL 関数は、数値を 1 日から 2 番目の間隔単位を含む時間間隔に変換します。関数を呼び出すとき、結果セットに含まれる最小の小数間隔タイプを指定します。有効な間隔タイプは DAY、HOUR、MINUTE、および SECOND です。

次の例では、数値を日と時間を含む時間間隔に変換します。

```
SELECT numtodsinterval(100, 'hour');
```

numtodsinterval

(次のページに続く)

(前のページからの続き)

```
4 days 04:00:00
(1 row)
```

次の例では、数値を分と秒を含む時間間隔に変換します。

```
SELECT numtodsinterval(100, 'second' );
numtodsinterval
-----
1 min 40 secs
(1 row)
```

2.3.8.8 NUMTOYMINTERVAL

NUMTOYMINTERVAL 関数は、数値を年から月の間隔単位を含む時間間隔に変換します。関数を呼び出すとき、結果セットに含まれる最小の小数間隔タイプを指定します。有効な間隔タイプは YEAR および MONTH です。

次の例では、数値を年と月を含む時間間隔に変換します。

```
SELECT numtoyminterval(100, 'month' );
numtoyminterval
-----
8 years 4 mons
(1 row)
```

次の例では、数値を年のみを含む時間間隔に変換します。

```
SELECT numtoyminterval(100, 'year' );
numtoyminterval
-----
100 years
(1 row)
```

2.3.8.9 ラウンド

ROUND 関数は、指定されたテンプレートパターンに従って丸められた日付を返します。テンプレートパターンを省略すると、日付は最も近い日に丸められます。次の表に、ROUND 機能のテンプレートパターンを示します。

模様	説明
CC, SCC	1 月 1 日、cc01 を返します。ここで、cc は、最後の 2 桁が 50 以下の場合、指定された年の最初の 2 桁、または最後の 2 桁が 50 より大きい場合、指定された年の最初の 2 桁より大きい 1 です; (AD 年の場合)
YYYY, YYYY, YEAR, SYEAR, YYY, YY, Y	1 月 1 日 yyyy を返します。yyyy は最も近い年に丸められます。6 月 30 日に切り捨て、7 月 1 日に切り上げ
IYYY, IYY, IY, I	ISO 年の初めに切り上げます。月と日が 6 月 30 日以前の場合は切り捨て、月と日が 7 月 1 日以降の場合は切り上げによって決定されます。
Q	月と日が四半期の 2 番目の月の 15 日以前である場合は切り捨て、または月と日が 2 番目の月の 16 日以降である場合は切り上げによって決定される、四半期の最初の日を返します。四半期
月、月、月、月	月の日が 15 日以前の場合、指定された月の最初の日を返します。月の日付が 16 日以降の場合、翌月の最初の日を返します
WW	年の最初の日と同じ曜日に対応する最も近い日付に丸めます
IW	ISO 年の最初の日と同じ曜日に対応する最も近い日付に丸めます
W	月の最初の日と同じ曜日に対応する最も近い日付に丸めます
DDD, DD, J	最も近い日の始まりに丸めます。11:59:59AM またはそれ以前のラウンドは、同じ日の開始まで。午後 12 時 0 分 00 秒以降、翌日の開始にラウンド
DAY, DY, D	最も近い日曜日に丸めます
HH, HH12, HH24	最も近い時間に丸める
MI	最も近い分に丸める

ROUND 機能の使用例を次に示します。

次の例では、最も近い 100 年に丸めています。

```
SELECT TO_CHAR(ROUND(TO_DATE('1950','YYYY'),'CC'),'DD-MON-YYYY') "Century"
FROM DUAL;

Century
-----
01-JAN-1901
(1 row)

SELECT TO_CHAR(ROUND(TO_DATE('1951','YYYY'),'CC'),'DD-MON-YYYY') "Century"
FROM DUAL;

Century
```

(次のページに続く)

(前のページからの続き)

```
-----
01-JAN-2001
(1 row)
```

次の例では、最も近い年に丸めます。

```
SELECT TO_CHAR(ROUND(TO_DATE('30-JUN-1999','DD-MON-YYYY'),'Y'),'DD-MON-
YYYY') "Year" FROM DUAL;
```

```
      Year
-----
01-JAN-1999
(1 row)
```

```
SELECT TO_CHAR(ROUND(TO_DATE('01-JUL-1999','DD-MON-YYYY'),'Y'),'DD-MON-
YYYY') "Year" FROM DUAL;
```

```
      Year
-----
01-JAN-2000
(1 row)
```

次の例では、最も近い ISO 年に丸めています。最初の例は 2004 年に丸められ、2004 年の ISO 年は 2003 年の 12 月 29 日 th に始まります。2 番目の例は 2005 年に丸められ、2005 年の ISO 年は 1 月 3 日 rd 同年の。

(ISO 年は、月曜日から日曜日までの 7 日間に新しい年の少なくとも 4 日が含まれる最初の月曜日に始まります。したがって、ISO 年の始まりは前年の 12 月に始まる可能性があります)

```
SELECT TO_CHAR(ROUND(TO_DATE('30-JUN-2004','DD-MON-YYYY'),'IYYY'),'DD-MON-
YYYY') "ISO Year" FROM DUAL;
```

```
      ISO Year
-----
29-DEC-2003
(1 row)
```

```
SELECT TO_CHAR(ROUND(TO_DATE('01-JUL-2004','DD-MON-YYYY'),'IYYY'),'DD-MON-
YYYY') "ISO Year" FROM DUAL;
```

```
      ISO Year
-----
03-JAN-2005
(1 row)
```

次の例では、最も近い四半期に丸めます。

```
SELECT ROUND(TO_DATE('15-FEB-07','DD-MON-YY'),'Q') "Quarter" FROM DUAL;
```

```

      Quarter
-----
01-JAN-07 00:00:00
(1 row)
```

```
SELECT ROUND(TO_DATE('16-FEB-07','DD-MON-YY'),'Q') "Quarter" FROM DUAL;
```

```

      Quarter
-----
01-APR-07 00:00:00
(1 row)
```

次の例では、最も近い月に丸めます。

```
SELECT ROUND(TO_DATE('15-DEC-07','DD-MON-YY'),'MONTH') "Month" FROM DUAL;
```

```

      Month
-----
01-DEC-07 00:00:00
(1 row)
```

```
SELECT ROUND(TO_DATE('16-DEC-07','DD-MON-YY'),'MONTH') "Month" FROM DUAL;
```

```

      Month
-----
01-JAN-08 00:00:00
(1 row)
```

次の例では、最も近い週に丸めます。2007 年の最初の日 は月曜日なので、最初の例では、1 月 18 日/:sup:th` は 1 月 15 日/:sup:th` に到着する月曜日に最も近いです。2 番目の例では、1 月 19 日/:sup:th` は、1 月 22 日/:sup:nd` に当たる月曜日に近くなります。

```
SELECT ROUND(TO_DATE('18-JAN-07','DD-MON-YY'),'WW') "Week" FROM DUAL;
```

```

      Week
-----
15-JAN-07 00:00:00
(1 row)
```

```
SELECT ROUND(TO_DATE('19-JAN-07','DD-MON-YY'),'WW') "Week" FROM DUAL;
```

```

      Week
-----
22-JAN-07 00:00:00
(1 row)
```

次の例では、最も近い ISO 週に丸めます。ISO 週は月曜日に始まります。最初の例では、2004 年 1 月 1 日は 2003 年 12 月 29 日の月曜日に最も近い。2 番目の例では、2004 年 1 月 2 日は 2004 年 1 月 5 日の月曜日に近い。

```
SELECT ROUND(TO_DATE('01-JAN-04','DD-MON-YY'),'IW') "ISO Week" FROM DUAL;
```

```
      ISO Week
-----
29-DEC-03 00:00:00
(1 row)
```

```
SELECT ROUND(TO_DATE('02-JAN-04','DD-MON-YY'),'IW') "ISO Week" FROM DUAL;
```

```
      ISO Week
-----
05-JAN-04 00:00:00
(1 row)
```

次の例では、最も近い週に丸めます。週は、月の最初の日と同じ日に始まると見なされます。

```
SELECT ROUND(TO_DATE('05-MAR-07','DD-MON-YY'),'W') "Week" FROM DUAL;
```

```
      Week
-----
08-MAR-07 00:00:00
(1 row)
```

```
SELECT ROUND(TO_DATE('04-MAR-07','DD-MON-YY'),'W') "Week" FROM DUAL;
```

```
      Week
-----
01-MAR-07 00:00:00
(1 row)
```

次の例では、最も近い日に丸めます。

```
SELECT ROUND(TO_DATE('04-AUG-07 11:59:59 AM','DD-MON-YY HH:MI:SS AM'),'J')
"Day" FROM DUAL;
```

```
      Day
-----
04-AUG-07 00:00:00
(1 row)
```

```
SELECT ROUND(TO_DATE('04-AUG-07 12:00:00 PM','DD-MON-YY HH:MI:SS AM'),'J')
"Day" FROM DUAL;
```

```
      Day
-----
```

(次のページに続く)

(前のページからの続き)

```
05-AUG-07 00:00:00
(1 row)
```

次の例では、最も近い曜日（日曜日）の開始日に丸めます。

```
SELECT ROUND(TO_DATE('08-AUG-07','DD-MON-YY'),'DAY') "Day of Week" FROM
DUAL;
```

```
Day of Week
-----
05-AUG-07 00:00:00
(1 row)
```

```
SELECT ROUND(TO_DATE('09-AUG-07','DD-MON-YY'),'DAY') "Day of Week" FROM
DUAL;
```

```
Day of Week
-----
12-AUG-07 00:00:00
(1 row)
```

次の例では、最も近い時間に丸めます。

```
SELECT TO_CHAR(ROUND(TO_DATE('09-AUG-07 08:29','DD-MON-YY HH:MI'),'HH'),'DD-
MON-YY HH24:MI:SS') "Hour" FROM DUAL;
```

```
Hour
-----
09-AUG-07 08:00:00
(1 row)
```

```
SELECT TO_CHAR(ROUND(TO_DATE('09-AUG-07 08:30','DD-MON-YY HH:MI'),'HH'),'DD-
MON-YY HH24:MI:SS') "Hour" FROM DUAL;
```

```
Hour
-----
09-AUG-07 09:00:00
(1 row)
```

次の例では、最も近い分に丸めます。

```
SELECT TO_CHAR(ROUND(TO_DATE('09-AUG-07 08:30:29','DD-MON-YY
HH:MI:SS'),'MI'),'DD-MON-YY HH24:MI:SS') "Minute" FROM DUAL;
```

```
Minute
-----
09-AUG-07 08:30:00
```

(次のページに続く)

(前のページからの続き)

```
(1 row)

SELECT TO_CHAR(ROUND(TO_DATE('09-AUG-07 08:30:30','DD-MON-YY
HH:MI:SS'),'MI'),'DD-MON-YY HH24:MI:SS') "Minute" FROM DUAL;

      Minute
-----
09-AUG-07 08:31:00
(1 row)
```

2.3.8.10 SYSDATE

SYSDATE 関数は、データベースサーバーが存在するオペレーティングシステムの現在の日付と時刻（タイムゾーンなしのタイムスタンプ）を返します。関数は STABLE で、引数は不要です。

単一の SQL ステートメントから呼び出されると、ステートメント内の各出現に対して同じ値を返します。トランザクション内の複数のステートメントから呼び出された場合、発生ごとに異なる値を返すことがあります。関数から呼び出された場合、呼び出し元の SYSDATE によって返される値とは異なる値を返す場合があります。

次の例は、SYSDATE の呼び出しを示しています。

```
SELECT SYSDATE, SYSDATE FROM DUAL;
      sysdate          |          sysdate
-----+-----
28-APR-20 16:45:28 | 28-APR-20 16:45:28
(1 row)
```

2.3.8.11 TRUNC

TRUNC 関数は、指定されたテンプレートパターンに従って切り捨てられた日付を返します。テンプレートパターンを省略すると、日付は最も近い日に切り捨てられます。次の表に、TRUNC 機能のテンプレートパターンを示します。

模様	説明
CC、SCC	1 月 1 日、cc01 を返します。cc は、指定された年の最初の 2 桁です。
YYYY、YYYY、YEAR、SYEAR、YYY、YY、Y	yyyy が指定された年である yyyy1 月 1 日を返します
IYYY、IYY、IY、I	指定された日付を含む ISO 年の開始日を返します
Q	指定された日付を含む四半期の最初の日を返します
月、月、月、月	指定された月の最初の日を返します
WW	年の最初の日と同じ曜日に対応する指定された日付の直前または同じ日付を返します
IW	指定された日付を含む ISO 週の始まりを返します
W	月の最初の日と同じ曜日に対応する指定された日付の直前または同じ最大の日付を返します
DDD、DD、J	指定された日付の一日の始まりを返します
DAY、DY、D	指定された日付を含む週の始まり（日曜日）を返します
HH、HH12、HH24	時間の開始を返します
MI	分の始まりを返します

TRUNC 機能の使用例を次に示します。

次の例では、100 年単位に切り捨てられます。

```
SELECT TO_CHAR(TRUNC(TO_DATE('1951','YYYY'),'CC'),'DD-MON-YYYY') "Century"
FROM DUAL;

      Century
-----
01-JAN-1901
(1 row)
```

次の例では、年まで切り捨てられます。

```
SELECT TO_CHAR(TRUNC(TO_DATE('01-JUL-1999','DD-MON-YYYY'),'Y'),'DD-MON-
YYYY') "Year" FROM DUAL;

      Year
-----
01-JAN-1999
(1 row)
```

次の例は、ISO 年の初めまで切り捨てます。

```
SELECT TO_CHAR(TRUNC(TO_DATE('01-JUL-2004','DD-MON-YYYY'),'IYYY'),'DD-MON-
YYYY') "ISO Year" FROM DUAL;
```

(次のページに続く)

(前のページからの続き)

```

ISO Year
-----
29-DEC-2003
(1 row)

```

次の例では、四半期の開始日まで切り捨てられます。

```

SELECT TRUNC(TO_DATE('16-FEB-07','DD-MON-YY'),'Q') "Quarter" FROM DUAL;

Quarter
-----
01-JAN-07 00:00:00
(1 row)

```

次の例は、月の初めに切り捨てます。

```

SELECT TRUNC(TO_DATE('16-DEC-07','DD-MON-YY'),'MONTH') "Month" FROM DUAL;

Month
-----
01-DEC-07 00:00:00
(1 row)

```

次の例では、年の最初の日によって決定される週の開始日まで切り捨てられます。2007 年の最初の日は月曜日であるため、1 月 19 日の直前の月曜日:sup:th`は 1 月 15 日の/:sup:th`です。

```

SELECT TRUNC(TO_DATE('19-JAN-07','DD-MON-YY'),'WW') "Week" FROM DUAL;

Week
-----
15-JAN-07 00:00:00
(1 row)

```

次の例では、ISO 週の始まりに切り捨てます。ISO 週は月曜日に始まります。2004 年 1 月 2 日は、2003 年 12 月 29 日月曜日に始まる ISO 週に該当します。

```

SELECT TRUNC(TO_DATE('02-JAN-04','DD-MON-YY'),'IW') "ISO Week" FROM DUAL;

ISO Week
-----
29-DEC-03 00:00:00
(1 row)

```

次の例では、週の始まりに切り捨てられます。この場合、週は月の最初の日と同じ日に始まると見なされます。

```
SELECT TRUNC(TO_DATE('21-MAR-07','DD-MON-YY'),'W') "Week" FROM DUAL;
```

```

      Week
-----
15-MAR-07 00:00:00
(1 row)
```

次の例は、一日の始めに切り捨てます。

```
SELECT TRUNC(TO_DATE('04-AUG-07 12:00:00 PM','DD-MON-YY HH:MI:SS AM'),'J')
"Day" FROM DUAL;
```

```

      Day
-----
04-AUG-07 00:00:00
(1 row)
```

次の例は、週の始まり（日曜日）に切り捨てます。

```
SELECT TRUNC(TO_DATE('09-AUG-07','DD-MON-YY'),'DAY') "Day of Week" FROM
DUAL;
```

```

Day of Week
-----
05-AUG-07 00:00:00
(1 row)
```

次の例では、時間の開始まで切り捨てます。

```
SELECT TO_CHAR(TRUNC(TO_DATE('09-AUG-07 08:30','DD-MON-YY HH:MI'),'HH'),'DD-
MON-YY HH24:MI:SS') "Hour" FROM DUAL;
```

```

      Hour
-----
09-AUG-07 08:00:00
(1 row)
```

次の例では、分に切り捨てます。

```
SELECT TO_CHAR(TRUNC(TO_DATE('09-AUG-07 08:30:30','DD-MON-YY
HH:MI:SS'),'MI'),'DD-MON-YY HH24:MI:SS') "Minute" FROM DUAL;
```

```

      Minute
-----
09-AUG-07 08:30:00
(1 row)
```

2.3.9 Sequence Manipulation Functions

このセクションでは、シーケンスオブジェクトを操作するための AdvancedServer の機能について説明します。シーケンスオブジェクト（シーケンスジェネレーターまたは単にシーケンスとも呼ばれる）は、CREATE SEQUENCE コマンドで作成された特別な単一行テーブルです。通常、シーケンスオブジェクトは、テーブルの行の一意的識別子を生成するために使用されます。以下にリストされているシーケンス関数は、シーケンスオブジェクトから連続したシーケンス値を取得するためのシンプルでマルチユーザーセーフなメソッドを提供します。

<code>sequence.NEXTVAL</code> <code>sequence.CURRVAL</code>
--

sequence は、CREATE SEQUENCE コマンドでシーケンスに割り当てられた識別子です。以下に、これらの機能の使用法を説明します。

NEXTVAL

シーケンスオブジェクトを次の値に進め、その値を返します。これはアトミックに行われます。複数のセッションが同時に NEXTVAL を実行する場合でも、各セッションは個別のシーケンス値を安全に受け取ります。

CURRVAL

現在のセッションでこのシーケンスの NEXTVAL によって最後に取得された値を返します。（このセッションでこのシーケンスに対して NEXTVAL が呼び出されていない場合はエラーが報告されます。）これはセッションローカル値を返すため、他のセッションが NEXTVAL を実行したかどうかにかかわらず、現在のセッションでした。

シーケンスオブジェクトが既定のパラメーターで作成されている場合、そのオブジェクトに対して NEXTVAL を呼び出すと、1 で始まる連続した値が返されます。CREATE SEQUENCE コマンドで特別なパラメーターを使用すると、他の動作を取得できます。

Important : 同じシーケンスから番号を取得する同時トランザクションのブロックを回避するために、NEXTVAL 操作はロールバックされません。つまり、値がフェッチされると、NEXTVAL を実行したトランザクションが後でアボートしても、使用されたと見なされます。これは、トランザクションが中止されると、割り当てられた値のシーケンスに未使用の「穴」が残る可能性があることを意味します。

2.3.10 Conditional Expressions

次のセクションでは、AdvancedServer で使用可能な SQL 準拠の条件式について説明します。

2.3.10.1 事例

SQL の CASE 式は、他の言語の if/else ステートメントに似た一般的な条件式です。

```
CASE WHEN condition THEN result
      [ WHEN ... ]
      [ ELSE result ]
END
```

「10」句は、式が有効であればどこでも使用できます。condition は BOOLEAN の結果を返す式です。結果が TRUE の場合、「10」式の値は条件に続く「11」です。結果が FALSE の場合、後続の WHEN 節が同じ方法で検索されます。no WHEN condition が TRUE の場合、「10」式の値は「13」節の「11」です。「13」節が省略され、一致する条件がない場合、結果は「14」になります。

例:

```
SELECT * FROM test;

a
---
1
2
3
(3 rows)

SELECT a,
       CASE WHEN a=1 THEN 'one'
            WHEN a=2 THEN 'two'
            ELSE 'other'
       END
FROM test;

a | case
---+-----
1 | one
2 | two
3 | other
(3 rows)
```

すべての result 式のデータ型は、単一の出力型に変換可能でなければなりません。

次の「単純な」CASE 式は、上記の一般的な形式の特殊なバリエーションです。

```

CASE expression
    WHEN value THEN result
    [ WHEN ... ]
    [ ELSE result ]
END

```

expression が計算され、WHEN 節のすべての value 仕様と比較され、等しいものが見つかるまで比較されます。一致するものが見つからない場合、ELSE 節の result（またはヌル値）が返されます。

上記の例は、単純な CASE 構文を使用して記述できます。

```

SELECT a,
       CASE a WHEN 1 THEN 'one'
              WHEN 2 THEN 'two'
              ELSE 'other'
       END
FROM test;

 a | case
---+-----
 1 | one
 2 | two
 3 | other
(3 rows)

```

CASE 式は、結果を決定するために必要のない部分式を評価しません。たとえば、これはゼロ除算失敗を回避する可能な方法です。

```
SELECT ... WHERE CASE WHEN x <> 0 THEN y/x > 1.5 ELSE false END;
```

2.3.10.2 合体

COALESCE 関数は、引数の最初のヌルではないものを返します。NULL は、すべての引数が NULL の場合にのみ返されます。

```
COALESCE(value [, value2 ] ... )
```

多くの場合、表示または追加の計算のためにデータを取得するときに、デフォルト値をヌル値に置き換えるために使用されます。例:

```
SELECT COALESCE(description, short_description, '(none)') ...
```

CASE 式のように、COALESCE は結果を決定するために必要ではない引数を評価しません。つまり、最初の非ヌル引数の右側の引数は評価されません。この SQL 標準関数は、他のデータベースシステムで 사용되는 NVL および IFNULL に類似した機能を提供します。

2.3.10.3 NULLIF

value1 と value2 が等しい場合、NULLIF 関数はヌル値を返します。それ以外の場合は、value1 を返します。

```
NULLIF (value1, value2)
```

これを使用して、上記の COALESCE の例の逆演算を実行できます。

```
SELECT NULLIF (value1, ' (none) ') ...
```

value1 が (なし) の場合は null を返し、それ以外の場合は value1 を返します。

2.3.10.4 NVL

NVL 関数は、ヌルではない最初の引数を返します。NVL は最初の式を評価します。その式が NULL と評価される場合、NVL は 2 番目の式を返します。

```
NVL (expr1, expr2)
```

戻り値の型は引数の型と同じです。すべての引数は同じデータ型でなければなりません（または共通の型に強制変換可能でなければなりません）。すべての引数が NULL の場合、NVL は NULL を返します。

次の例は、非委託従業員のボーナスを計算します。従業員が委託従業員である場合、この式は従業員の手数料を返します。従業員が委託された従業員でない場合（つまり、彼の手数料が NULL である場合）、この式は給与の 10 % のボーナスを返します。

```
bonus = NVL (emp.commission, emp.salary * .10)
```

2.3.10.5 NVL2

NVL2 は式を評価し、最初の式の値に応じて、2 番目または 3 番目の式を返します。最初の式が NULL でない場合、NVL2 は expr2 の値を返します。最初の式が NULL の場合、NVL2 は expr3 の値を返します。

```
NVL2 (expr1, expr2, expr3)
```

戻り値の型は引数の型と同じです。すべての引数は同じデータ型でなければなりません（または共通の型に強制変換可能でなければなりません）。

次の例では、委託された従業員のボーナスを計算します。特定の従業員が委託された従業員である場合、この式はその手数料の 110 % に等しい金額を返します。従業員がコミッションされた従業員でない場合（つまり、コミッションが NULL）、この式は 0 を返します。

```
bonus = NVL2(emp.commission, emp.commission * 1.1, 0)
```

2.3.10.6 最大かつ最低

GREATEST および LEAST 関数は、任意の数の式のリストから最大値または最小値を選択します。

```
GREATEST(value [, value2 ] ... )  
LEAST(value [, value2 ] ... )
```

式はすべて、結果の型となる共通のデータ型に変換可能である必要があります。リスト内のヌル値は無視されます。すべての式がヌルと評価された場合にのみ、結果はヌルになります。

注釈: GREATEST と LEAST は標準 SQL にはありませんが、一般的な拡張子です。

2.3.11 Aggregate Functions

Aggregate 関数は、入力値のセットから単一の結果値を計算します。組み込みの集計関数を次の表にリストします。

機能	引数のタイプ	戻りタイプ	説明
AVG (式)	INTEGER、REAL、DOUBLEPRECISION、NUMBER	任意の整数型の場合は NUMBER、浮動小数点引数の場合は DOUBLEPRECISION、それ以外は引数のデータ型と同じ	すべての入力値の平均（算術平均）
COUNT (*)		ビッグ	入力行の数
COUNT (式)	どれでも	ビッグ	expression の値が null でない入力行の数
MAX (式)	数値、文字列、日付/時刻、または bytea タイプ	引数の型と同じ	すべての入力値にわたる式の最大値
MEDIAN (式)	BIGINT、DOUBLEPRECISION、INTEGER、INTERVAL、NUMERIC、REAL、SMALLINT、TIMESTAMP、TIMESTAMPZ	任意の整数型の場合は NUMBER、浮動小数点引数の場合は DOUBLEPRECISION、それ以外は引数のデータ型と同じ	式の間接値を識別します
MIN (式)	数値、文字列、日付/時刻、または bytea タイプ	引数の型と同じ	すべての入力値にわたる式の最小値
SUM (式)	INTEGER、REAL、DOUBLEPRECISION、NUMBER	SMALLINT または INTEGER 引数の場合は BIGINT、BIGINT 引数の場合は NUMBER、浮動小数点引数の場合は DOUBLEPRECISION、その他の場合は引数のデータ型と同じ	すべての入力値にわたる式の合計

COUNT を除いて、行が選択されていない場合、これらの関数は null 値を返すことに注意してください。特に、行がない SUM は null を返しますが、期待どおりのゼロではありません。COALESCE 関数を使用して、必要に応じてヌルをゼロに置き換えることができます。

次の表は、統計分析で通常使用される集計関数を示しています。（これらは、より一般的に使用される集約のリストが乱雑になるのを避けるために分離されています。）説明が N に言及している場合、それはすべての入力式が非ヌルである入力行の数を意味します。すべての場合において、計算が無意味な場合、たとえば N がゼロの場合、null が返されます。

機能	引数のタイプ	戻りタイプ	説明
CORR (Y, X)	倍精度	倍精度	相関係数
COVAR_POP (Y, X)	倍精度	倍精度	母集団共分散
COVAR_SAMP (Y, X)	倍精度	倍精度	サンプル共分散
REGR_AVGX (Y, X)	倍精度	倍精度	独立変数の平均 ($\text{sum}(X)/N$)
REGR_AVGY (Y, X)	倍精度	倍精度	従属変数の平均 ($\text{sum}(Y)/N$)
REGR_COUNT (Y, X)	倍精度	倍精度	両方の式が null でない入力行の数
REGR_INTERCEPT (Y, X)	倍精度	倍精度	(X, Y) ペアによって決定される最小二乗適合線形方程式の y 切片
REGR_R2 (Y, X)	倍精度	倍精度	相関係数の二乗
REGR_SLOPE (Y, X)	倍精度	倍精度	(X, Y) ペアによって決定される最小二乗適合線形方程式の勾配
REGR_SXX (Y, X)	倍精度	倍精度	合計 (X^2) - 合計 (X) ² / 2N (独立変数の「平方和」)
REGR_SXY (Y, X)	倍精度	倍精度	合計 ($X*Y$) - 合計 (X) / * 合計 (Y) / N (独立した時間従属変数の「積の合計」)
REGR_SYY (Y, X)	倍精度	倍精度	合計 (Y^2) - 合計 (Y) ² / 2N (従属変数の「平方和」)
STDDEV (式)	INTEGER、REAL、DOUBLEPRECISION、NUMBER	浮動小数点引数の場合は DOUBLEPRECISION、それ以外の場合は NUMBER	STDDEV_SAMP の歴史的なエイリアス
STDDEV_POP (式)	INTEGER、REAL、DOUBLEPRECISION、NUMBER	浮動小数点引数の場合は DOUBLEPRECISION、それ以外の場合は NUMBER	入力値の母標準偏差
STDDEV_SAMP (式)	INTEGER、REAL、DOUBLEPRECISION、NUMBER	浮動小数点引数の場合は DOUBLEPRECISION、それ以外の場合は NUMBER	入力値の標準偏差のサンプル

次のページに続く

表 10 – 前のページからの続き

VARIANCE (式)	INTEGER、REAL、 DOUBLEPRECISION、NUMBER	浮動小数点引数の場合は DOUBLEPRECISION、それ以外の場合は NUMBER	VAR_SAMP の履歴 エイリアス
VAR_POP (式)	INTEGER、REAL、 DOUBLEPRECISION、NUMBER	浮動小数点引数の場合は DOUBLEPRECISION、それ以外の場合は NUMBER	入力値の母集団分散 (母標準偏差の二乗)
VAR_SAMP (式)	INTEGER、REAL、 DOUBLEPRECISION、NUMBER	浮動小数点引数の場合は DOUBLEPRECISION、それ以外の場合は NUMBER	入力値のサンプル分散 (サンプル標準偏差の二乗)

2.3.11.1 LISTAGG

AdvancedServer は、文字列の集約をサポートする LISTAGG 機能を追加しました。LISTAGG は、複数の行からのデータを順序付けられた方法で単一の行に連結する集約関数です。オプションで、データのカスタム区切り文字を含めることができます。

LISTAGG 関数は、WITHIN GROUP 句の下での ORDER BY 句を使用してメジャー列の値を連結し、順序付けられた集計データを生成することを義務付けています。

Objective

- LISTAGG はグループ化せずに使用できます。この場合、LISTAGG 関数はテーブル内のすべての行で動作し、単一の行を返します。
- LISTAGG は GROUP BY 節で使用できます。この場合、LISTAGG 関数は各グループで動作し、各グループの集約された出力を返します。
- LISTAGG は OVER 節で使用できます。この場合、LISTAGG 関数は `query_partition_by_clause` の式に基づいてクエリ結果セットをグループに分割し、各グループのデータを集計します。

Synopsis

```
LISTAGG( <measure_expr> [, <delimiter> ]) WITHIN GROUP( <order_by_clause> )  
[ OVER <query_partition_by_clause> ]
```

Parameters

`measure_expr`

`measure_expr` (必須) は、集計する値を割り当てる列または式を指定します。NULL 値は無視されます。

`delimiter`

`delimiter` (オプション) は、結果行の連結された値を区切る文字列を指定します。`delimiter` には、NULL の値、文字列、文字リテラル、列名、または定数式を指定できます。無視された場合、LISTAGG 関数はデフォルトで NULL 値を使用します。

`order_by_clause`

`order_by_clause` (必須) は、連結された値が返されるソート順を決定します。

`query_partition_by_clause`

`query_partition_by_clause` (オプション) を使用すると、LISTAGG 関数を分析関数として使用でき、OVER 句で各グループのレコードの範囲を設定できます。

Return Type

LISTAGG 関数は文字列値を返します。

Examples

次の例では、EMP テーブルの値を連結し、delimiter コンマで区切られたすべての従業員をリストします。

まず、EMP という名前のテーブルを作成してから、EMP テーブルにレコードを挿入します。

```
edb=# CREATE TABLE EMP
edb=#         (EMPNO NUMBER(4) NOT NULL,
edb(#         ENAME VARCHAR2(10),
edb(#         JOB VARCHAR2(9),
edb(#         MGR NUMBER(4),
edb(#         HIREDATE DATE,
edb(#         SAL NUMBER(7, 2),
edb(#         COMM NUMBER(7, 2),
edb(#         DEPTNO NUMBER(2));
CREATE TABLE

edb=# INSERT INTO EMP VALUES
edb=#         (7499, 'ALLEN', 'SALESMAN', 7698,
edb(#         TO_DATE('20-FEB-1981', 'DD-MON-YYYY'), 1600, 300, 30);
INSERT 0 1
edb=# INSERT INTO EMP VALUES
edb=#         (7521, 'WARD', 'SALESMAN', 7698,
edb(#         TO_DATE('22-FEB-1981', 'DD-MON-YYYY'), 1250, 500, 30);
INSERT 0 1
edb=# INSERT INTO EMP VALUES
edb=#         (7566, 'JONES', 'MANAGER', 7839,
edb(#         TO_DATE('2-APR-1981', 'DD-MON-YYYY'), 2975, NULL, 20);
INSERT 0 1
edb=# INSERT INTO EMP VALUES
edb=#         (7654, 'MARTIN', 'SALESMAN', 7698,
edb(#         TO_DATE('28-SEP-1981', 'DD-MON-YYYY'), 1250, 1400, 30);
INSERT 0 1
edb=# INSERT INTO EMP VALUES
edb=#         (7698, 'BLAKE', 'MANAGER', 7839,
edb(#         TO_DATE('1-MAY-1981', 'DD-MON-YYYY'), 2850, NULL, 30);
INSERT 0 1

edb=# SELECT LISTAGG(ENAME, ',') WITHIN GROUP (ORDER BY ENAME) FROM EMP;
          listagg
-----
ALLEN,BLAKE,JONES,MARTIN,WARD
(1 row)
```

次の例では、EMP テーブルで PARTITION BY 句と LISTAGG を使用し、DEPTNO によるパーティションに基づいて、テーブル全体ではなく各パーティションに適用される出力を生成します。

```
edb=# SELECT DISTINCT DEPTNO, LISTAGG(ENAME, ',') WITHIN GROUP (ORDER BY
ENAME) OVER(PARTITION BY DEPTNO) FROM EMP;
 deptno |          listagg
-----+-----
      30 | ALLEN, BLAKE, MARTIN, WARD
      20 | JONES
(2 rows)
```

次の例は、GROUP BY 句が含まれていることを除いて、前の例と同じです。

```
edb=# SELECT DEPTNO, LISTAGG(ENAME, ',') WITHIN GROUP (ORDER BY ENAME) FROM
EMP GROUP BY DEPTNO;
 deptno |          listagg
-----+-----
      20 | JONES
      30 | ALLEN, BLAKE, MARTIN, WARD
(2 rows)
```

2.3.11.2 中央値

MEDIAN 関数は、指定された値の範囲から式の間値を計算します。NULL 値は無視されます。クエリがユーザー定義テーブルを参照しない場合、MEDIAN 関数はエラーを返します。

Objective

- MEDIAN はグループ化せずに使用できます。この場合、MEDIAN 関数はテーブル内のすべての行で動作し、単一の行を返します。
- MEDIAN は OVER 節で使用できます。この場合、MEDIAN 関数は PARTITION BY 句で指定された expression に基づいてクエリ結果セットをグループに分割し、各グループのデータを集計します。

Synopsis

```
MEDIAN( <median_expression> ) [ OVER ( [ PARTITION BY... ] ) ]
```

Parameters

median_expression

median_expression (必須) は、MEDIAN 関数が動作し、中央値を返すターゲット列または式です。数値、日時、または間隔のデータ型を指定できます。

PARTITION BY

PARTITION BY 句 (オプション) は、MEDIAN 関数を分析関数として使用できるようにし、OVER 句で各グループのレコードの範囲を設定します。

Return Types

戻りタイプは、expression の入力データタイプによって決まります。次の表は、各入力タイプの戻り値のタイプを示しています。

入力タイプ	戻りタイプ
ビッグ	数値
フロート、ダブル精度	倍精度
整数	数値
間隔	間隔
数値	数値
REAL	REAL
小型	数値
タイムスタンプ	タイムスタンプ
タイムスタンプ	タイムスタンプ

Examples

次の例では、クエリは EMP テーブルの各部門の給与の中央値を返します。

```
edb=# SELECT * FROM EMP;
 empno|  ename  |   job   | mgr |      hiredate      |  sal  | comm  | deptno
-----+-----+-----+-----+-----+-----+-----+-----
 7369 | SMITH   | CLERK   | 7902 | 17-DEC-80 00:00:00 | 800.00 |      |    20
 7499 | ALLEN   | SALESMAN| 7698 | 20-FEB-81 00:00:00 | 1600.00 | 300.0 |    30
 7521 | WARD    | SALESMAN| 7698 | 22-FEB-81 00:00:00 | 1250.00 | 500.00 |    30
 7566 | JONES   | MANAGER | 7839 | 02-APR-81 00:00:00 | 2975.00 |      |    20
 7654 | MARTIN  | SALESMAN| 7698 | 28-SEP-81 00:00:00 | 1250.00 | 1400.00 |    30
(5 rows)

edb=# SELECT MEDIAN (SAL) FROM EMP;
 median
-----
    1250
(1 row)
```

次の例では、EMP テーブルの PARTITION BY 句と MEDIAN を使用し、DEPTNO によるパーティションに基づいて給与の中央値を返します。

```
edb=# SELECT EMPNO, ENAME, DEPTNO, MEDIAN (SAL) OVER (PARTITION BY DEPTNO)
FROM EMP;
 empno |  ename  | deptno | median
-----+-----+-----+-----
  7369 | SMITH   |    20 | 1887.5
  7566 | JONES   |    20 | 1887.5
```

(次のページに続く)

(前のページからの続き)

```

7499 | ALLEN   |      30 |    1250
7521 | WARD    |      30 |    1250
7654 | MARTIN  |      30 |    1250
(5 rows)

```

MEDIAN 機能は PERCENTILE_CONT と比較できます。次の例では、MEDIAN は PERCENTILE_CONT と同じ結果を生成します。

```

edb=# SELECT MEDIAN (SAL), PERCENTILE_CONT(0.5) WITHIN GROUP (ORDER BY SAL)
FROM EMP;
 median | percentile_cont
-----+-----
  1250 |          1250
(1 row)

```

2.3.11.3 STATS_MODE

STATS_MODE 関数は一連の値を引数として取り、最も高い頻度で発生する値を返します。同じ頻度で複数の値が表示される場合、STATS_MODE 関数は最初の値を任意に選択し、その 1 つの値のみを返します。

Objective

- STATS_MODE 機能はグループ化せずに使用できます。この場合、STATS_MODE 関数はテーブル内のすべての行で動作し、単一の値を返します。
- STATS_MODE は、WITHIN GROUP 句を使用して、順序集合集計関数として使用できます。この場合、STATS_MODE 関数は順序付けられたデータセットで動作します。
- STATS_MODE は GROUP BY 節で使用できます。この場合、STATS_MODE 関数は各グループで動作し、各グループの最も頻繁で集約された出力を返します。

Synopsis

```
STATS_MODE( <expr> )
```

または

```
STATS_MODE() WITHIN GROUP ( ORDER BY sort_expression )
```

Parameters

expr

列に割り当てる式または値。

Return Type

STATS_MODE 関数は、頻繁に表示される値を返します。ただし、列のすべての値が NULL の場合、STATS_MODE は NULL を返します。

Examples

次の例は、EMP テーブルの給与モードを返します。

```
edb=# SELECT * FROM EMP;
 empno|  ename  |  job   | mgr  |      hiredate      |  sal  | comm  | deptno
-----+-----+-----+-----+-----+-----+-----+-----
 7369 | SMITH   | CLERK  | 7902 | 17-DEC-80 00:00:00 | 800.00|       |    20
 7499 | ALLEN   | SALESMAN| 7698 | 20-FEB-81 00:00:00 | 1600.00| 300.0 |    30
 7521 | WARD    | SALESMAN| 7698 | 22-FEB-81 00:00:00 | 1250.00| 500.00 |    30
 7566 | JONES   | MANAGER | 7839 | 02-APR-81 00:00:00 | 2975.00|       |    20
 7654 | MARTIN  | SALESMAN| 7698 | 28-SEP-81 00:00:00 | 1250.00| 1400.00 |    30
(5 rows)

edb=# SELECT STATS_MODE(SAL) FROM EMP;
 stats_mode
-----
 1250.00
(1 row)
```

次の例では、EMP テーブルで GROUP BY および ORDER BY 句と STATS_MODE を使用し、DEPTNO によるパーティションに基づいて給与を返します。

```
edb=# SELECT STATS_MODE(SAL) FROM EMP GROUP BY DEPTNO ORDER BY DEPTNO;
 stats_mode
-----
 800.00
 1250.00
(2 rows)
```

次の例では、WITHIN GROUP 句と STATS_MODE 関数を使用して、順序付けられたデータセットで集計を実行します。

```
SELECT STATS_MODE() WITHIN GROUP (ORDER BY SAL) FROM EMP;
 stats_mode
-----
 1250.00
(1 row)
```

2.3.12 Subquery Expressions

このセクションでは、AdvancedServer で使用可能な SQL 準拠のサブクエリ式について説明します。このセクションに記載されているすべての式フォームは、ブール値 (TRUE/FALSE) を返します。

2.3.12.1 EXISTS

EXISTS の引数は、任意の SELECT ステートメント、またはサブクエリです。サブクエリは、行を返すかどうかを判断するために評価されます。少なくとも 1 行が返される場合、EXISTS の結果は TRUE です。サブクエリが行を返さない場合、EXISTS の結果は FALSE になります。

```
EXISTS (subquery)
```

サブクエリは、周囲のクエリから変数を参照できます。変数は、サブクエリの評価中に定数として機能します。

通常、サブクエリは、完了までではなく、少なくとも 1 つの行が返されるかどうかを判断するのに十分なだけ実行されます。副作用（シーケンス関数の呼び出しなど）を持つサブクエリを記述することは賢明ではありません。副作用が発生するかどうかを予測するのは難しいかもしれません。

結果は行が返されるかどうかのみ依存し、それらの行の内容には依存しないため、通常、サブクエリの出力リストは重要ではありません。一般的なコーディング規則は、すべての EXISTS テストを EXISTS (SELECT 1 WHERE ...) の形式で記述することです。ただし、INTERSECT を使用するサブクエリなど、このルールには例外があります。

この単純な例は deptno での内部結合に似ていますが、複数の一致する emp 行がある場合でも、dept 行ごとに最大 1 つの出力行を生成します。

```
SELECT dname FROM dept WHERE EXISTS (SELECT 1 FROM emp WHERE emp.deptno =
dept.deptno);

  dname
-----
ACCOUNTING
RESEARCH
SALES
(3 rows)
```

2.3.12.2 IN

右側は括弧で囲まれたサブクエリで、正確に 1 つの列を返す必要があります。左側の式が評価され、サブクエリの結果の各行と比較されます。等しいサブクエリ行が見つかった場合、IN の結果は TRUE になります。等しい行が見つからない場合（サブクエリが行を返さない特別な場合を含む）、結果は FALSE です。

```
expression IN (subquery)
```

左側の式が NULL をもたらす場合、または等しい右側の値がなく、少なくとも 1 つの右側の行が NULL をもたらす場合、IN 構造の結果は NULL になることに注意してください。FALSE ではありません。これは、NULL 値のブールの組み合わせに関する SQL の通常の規則に準拠しています。

EXISTS と同様に、サブクエリが完全に評価されると仮定するのは賢明ではありません。

2.3.12.3 NOT IN

右側は括弧で囲まれたサブクエリで、正確に 1 つの列を返す必要があります。左側の式が評価され、サブクエリの結果の各行と比較されます。（サブクエリが行を返さない特別な場合を含む）等しくないサブクエリ行のみが見つかった場合、NOT IN の結果は TRUE になります。等しい行が見つかった場合、結果は FALSE です。

```
expression NOT IN (subquery)
```

左側の式が NULL をもたらす場合、または等しい右側の値がなく、少なくとも 1 つの右側の行が NULL をもたらす場合、NOT IN 構造の結果は NULL になることに注意してください。TRUE ではありません。これは、NULL 値のブールの組み合わせに関する SQL の通常の規則に準拠しています。

EXISTS と同様に、サブクエリが完全に評価されると仮定するのは賢明ではありません。

2.3.12.4 ANY/SOME

右側は括弧で囲まれたサブクエリで、正確に 1 つの列を返す必要があります。左側の式が評価され、指定された演算子を使用してサブクエリ結果の各行と比較されます。演算子はブール値の結果を生成する必要があります。真の結果が得られた場合、ANY の結果は TRUE になります。真の結果が見つからない場合（サブクエリが行を返さない特別な場合を含む）、結果は FALSE です。

```
expression operator ANY (subquery)
expression operator SOME (subquery)
```

SOME は ANY の同義語です。IN は = ANY と同等です。

成功せず、少なくとも 1 つの右側の行が演算子の結果に対して NULL を生成する場合、ANY 構成の結果は FALSE ではなく NULL になることに注意してください。これは、NULL 値のブールの組み合わせに関する SQL の通常の規則に準拠しています。

EXISTS と同様に、サブクエリが完全に評価されると仮定するのは賢明ではありません。

2.3.12.5 ALL

右側は括弧で囲まれたサブクエリで、正確に 1 つの列を返す必要があります。左側の式が評価され、指定された演算子を使用してサブクエリの結果の各行と比較されます。演算子はブール値の結果を生成する必要があります。すべての行が **true** を返す場合、ALL の結果は **TRUE** です（サブクエリが行を返さない特別な場合を含む）。偽の結果が見つかった場合、結果は **FALSE** です。比較がどの行に対しても **FALSE** を返さない場合、結果は **NULL** であり、少なくとも 1 行に対して **NULL** を返します。

`expression operator ALL (subquery)`

NOT IN は <> ALL と同等です。EXISTS と同様に、サブクエリが完全に評価されると仮定するのは賢明ではありません。

第 3 章

System Catalog Tables

次のシステムカタログ表には、データベースオブジェクトの定義が含まれています。システムテーブルのレイアウトは変更される可能性があります。システムテーブルに格納されている情報に依存するアプリケーションを作成する場合は、既存のカタログビューを使用するか、カタログビューを作成してシステムテーブルの変更からアプリケーションを分離するのが賢明です。

3.1 dual

`dual` は、Oracle データベースとの互換性のためにのみ提供される単一行、単一系列のテーブルです。

コラム	種類	修飾子	説明
<code>dummy</code>	<code>VARCHAR2 (1)</code>		互換性のためにのみ提供されています。

3.2 edb_dir

`edb_dir` テーブルには、`CREATE DIRECTORY` コマンドで作成されたディレクトリを指すエイリアスごとに 1 行が含まれます。ディレクトリは、ホストファイルシステムへのユーザーの制限付きアクセスを許可するパス名のエイリアスです。

ディレクトリを使用して、ファイルシステム内の特定のディレクトリツリーにユーザーをフェンスできます。たとえば、`UTL_FILE` パッケージは、ユーザーがホストファイルシステムのファイルとディレクトリの読み取りと書き込みを許可する機能を提供しますが、データベース管理者が `CREATE DIRECTORY` コマンドを介してアクセスを許可したパスへのアクセスのみを許可します。

コラム	種類	修飾子	説明
dirname	「名前」	ヌルではない	エイリアスの名前。
diowner	oid	ヌルではない	エイリアスを所有するユーザーの OID。
dirpath	テキスト		アクセスが許可されるディレクトリ名。
diracl	aclitem[]		エイリアスにアクセスできるユーザーを決定するアクセス制御リスト。

3.3 edb_password_history

edb_password_history テーブルには、パスワードの変更ごとに 1 行が含まれます。テーブルは、クラスター内のすべてのデータベースで共有されます。

コラム	種類	参照資料	説明
passhistroleid	oid	pg_authid.oid	ロールの ID。
passhistpassword	テキスト		md5 暗号化形式のロールパスワード。
passhistpasswordsetat	タイムスタンプ		パスワードが設定された時刻。

3.4 edb_policy

edb_policy テーブルには、ポリシーごとに 1 行が含まれます。

コラム	種類	修飾子	説明
polycyname	name	ヌルではない	The policy name.
policygroup	oid	ヌルではない	現在使用されていません。
policyobject	oid	ヌルではない	このポリシーによって保護されたテーブルの OID (object_schema と object_name)。
policykind	char	ヌルではない	このポリシーで保護されているオブジェクトの種類: テーブルの 'r' ビューの場合は 'v' =同義語の場合 現在は常に 'r' です。
policyproc	oid	ヌルではない	ポリシー関数の OID (function_schema と policy_function)。

次のページに続く

表 1 – 前のページからの続き

コラム	種類	修飾子	説明
policyinsert	ブール値	ヌルではない	ポリシーが INSERT ステートメントによって実施されている場合は true。
policyselect	ブール値	ヌルではない	ポリシーが SELECT ステートメントによって実施されている場合は true。
policydelete	ブール値	ヌルではない	ポリシーが DELETE ステートメントによって実施されている場合は true。
policyupdate	ブール値	ヌルではない	ポリシーが UPDATE ステートメントによって実施されている場合は true。
policyindex	ブール値	ヌルではない	現在使用されていません。
policyenabled	ブール値	ヌルではない	ポリシーが有効な場合は true。
policyupdatecheck	ブール値	ヌルではない	UPDATE ステートメントによって更新された行がポリシーを満たす必要がある場合は true。
polycystatic	ブール値	ヌルではない	現在使用されていません。
policytype	整数	ヌルではない	現在使用されていません。
policyopts	整数	ヌルではない	現在使用されていません。
policyseccols	int2vector	ヌルではない	sec_relevant_cols にリストされている列の列番号。

3.5 edb_profile

edb_profile テーブルには、使用可能なプロファイルに関する情報が格納されます。edb_profiles はクラスター内のすべてのデータベースで共有されます。

コラム	種類	参照資料	説明
oid	oid		行識別子（非表示属性。明示的に選択する必要があります）。
prfname	name		プロファイルの名前。
prffailedloginattempts	整数		プロファイルによって許可されたログイン試行の失敗回数。-1 は、デフォルトプロファイルの値を使用することを示します。-2 は、失敗したログイン試行に制限がないことを示します。

次のページに続く

表 2 – 前のページからの続き

コラム	種類	参照資料	説明
prfpasswordlocktime	整数		プロファイルに関連付けられたパスワードロック時間（秒単位）。-1 は、デフォルトプロファイルの値を使用することを示します。-2 は、アカウントを永続的にロックする必要があることを示します。
prfpasswordlifetime	整数		プロファイルに関連付けられたパスワードの有効期間（秒単位）。-1 は、デフォルトプロファイルの値を使用することを示します。-2 は、パスワードの有効期限が切れないことを示します。
prfpasswordgracetime	整数		プロファイルに関連付けられたパスワード猶予時間（秒単位）。-1 は、デフォルトプロファイルの値を使用することを示します。-2 は、パスワードの有効期限が切れないことを示します。
prfpasswordreusetime	整数		ユーザーがパスワードを再利用するまで待機する必要がある秒数。-1 は、デフォルトプロファイルの値を使用することを示します。-2 は、古いパスワードを再利用できないことを示します。
prfpasswordreusemax	整数		パスワードを再利用する前に発生する必要があるパスワード変更の数。-1 は、デフォルトプロファイルの値を使用することを示します。-2 は、古いパスワードを再利用できないことを示します。

次のページに続く

表 2 – 前のページからの続き

コラム	種類	参照資料	説明
prfpasswordallowhashed	整数		passwordallowhashed パラメーターは、暗号化されたパスワードの使用を許可するかどうかを指定します。可能な値は、true/on/yes/1、false/off/no/0、および DEFAULT です。
prfpasswordverifyfuncdb	oid	pg_database.oid	パスワード検証機能が存在するデータベースの OID。
prfpasswordverifyfunc	oid	pg_proc.oid	プロファイルに関連付けられたパスワード検証機能の OID。

3.6 edb_variable

edb_variable テーブルには、パッケージレベル変数（パッケージ内で宣言された各変数）ごとに 1 行が含まれます。

コラム	種類	修飾子	説明
varname	「名前」	ヌルではない	変数の名前。
varpackage	oid	ヌルではない	パッケージを格納する pg_namespace 行の OID。
vartype	oid	ヌルではない	変数の型を定義する pg_type 行の OID。
varaccess	/"char/"	ヌルではない	変数がパッケージの外部から見える場合は +。 変数がパッケージ内でのみ表示される場合は -。 注:パブリック変数はパッケージヘッダー内で宣言されます。プライベート変数は、パッケージ本体内で宣言されます。
varsrc	テキスト		変数のデフォルト値式を含む、変数宣言のソースが含まれます。
varseq	smallint	ヌルではない	パッケージで変数が宣言された順序。

3.7 pg_synonym

pg_synonym テーブルには、CREATE SYNONYM コマンドまたは CREATE PUBLIC SYNONYM コマンドで作成された同義語ごとに 1 行が含まれます。

コラム	種類	修飾子	説明
synname	「名前」	ヌルではない	シノニムの名前。
synnamespace	oid	ヌルではない	synowner を置き換えます。シノニムが格納されている pg_namespace 行の OID が含まれています
synowner	oid	ヌルではない	シノニムを所有するユーザーの OID。
synobjschema	「名前」	ヌルではない	参照されるオブジェクトが定義されているスキーマ。
synobjname	「名前」	ヌルではない	参照されるオブジェクトの名前。
synlink	テキスト		参照されるオブジェクトが定義されているデータベースリンクの（オプションの）名前。

3.8 product_component_version

product_component_version テーブルには、機能の互換性に関する情報が含まれています。アプリケーションは、インストール時または実行時にこのテーブルを照会して、アプリケーションで使用される機能がこの展開で使用可能であることを確認できます。

コラム	種類	説明
製品	さまざまなキャラクター (74)	製品の名前。
バージョン	さまざまなキャラクター (74)	製品のバージョン番号。
状態	さまざまなキャラクター (74)	リリースのステータス。

第 4 章

Conclusion

OracleDevelopers 向け EDBPostgresAdvancedServer データベース互換性リファレンスガイド

著作権 ©2007-2020EnterpriseDBCorporation。

無断複写・転載を禁じます。

EnterpriseDB®Corporation

34CrosbyDrive、Suite201、Bedford、MA01730、アメリカ

T+17813573390F+19784671307E

info@enterprisedb.com

www.enterprisedb.com

- EnterpriseDB および PostgresEnterpriseManager は、EnterpriseDBCorporation の登録商標です。EDB および EDBPostgres は、EnterpriseDBCorporation の商標です。Oracle は、Oracle、Inc の登録商標です。その他の商標は、それぞれの所有者の商標です。
- EDB は、存在する場合、EDB 製品のログオン UI の設計、コーディングのベストプラクティスの確立、レビュー、および入力検証の検証を行います。EDB は追加の入力コンポーネントに対して同じアプローチに従いますが、製品の性質上、限定された検証が可能な信頼できるユーザーによって入力および送信されるフリーフォーム SQL、WMI またはその他の文字列を受け入れる必要がある場合があります。このような場合、ユーザーが誤った入力や危険な入力を禁止することはできません。
- EDB は、将来、認証された信頼できるユーザーからのフリーフォーム SQL、WMI またはその他の潜在的に危険な入力を受け入れる製品に機能を追加する権利を留保しますが、そのような機能はすべて、可能な限り最小のリスクを提供するように設計およびテストされます可能であれば、スーパーユーザーまたは同等の特権が必要です。
- EDB は、すべての潜在的な脅威を予測できることを保証しないため、すべての潜在的な脆弱性が対処または検討されたことを完全に保証することはできません。

索引

Aggregate Functions, 77

Binary Data, 13

Boolean Types, 18

Character Types, 12

Comments, 7

Comparison Operators, 20

Conclusion, 95

Conditional Expressions, 73

Constants, 5

Data Type Formatting Functions, 44

Data Types, 9

Date/Time Functions and Operators, 53

Date/Time Types, 14

Functions and Operators, 20

Identifiers and Key Words, 4

Introduction, 1

Lexical Structure, 3

Logical Operators, 20

Mathematical Functions and Operators, 22

Numeric Types, 9

Pattern Matching String Functions, 37

Pattern Matching Using the LIKE Operator, 43

Sequence Manipulation Functions, 72

SQL Syntax, 3

String Functions and Operators, 26

Subquery Expressions, 86

System Catalog Tables, 89

The SQL Language, 3

XML Type, 18