
ORACLE®

ORACLE

Oracle Database 12cでの PL/SQL新機能

Oracle HQ

Database Server Technologies Division、
Distinguished Product Manager
Bryn Llewellyn

下記事項は、弊社の一般的な製品の方向性に関する概要を説明するものです。
また、情報提供を唯一の目的とするものであり、いかなる契約にも組み込むことはできません。
マテリアルやコード、機能の提供をコミットメント（確約）するものではなく、購買を決定する際の判断材料になさらないで下さい。オラクルの製品に関して記載されている機能の開発、リリース、および時期については、弊社の裁量により決定されます。

アジェンダ

- クライアント、PL/SQL、SQL間の相互運用性の強化
- セキュリティの新機能
- プログラマのユーザビリティの改善
- その他

SQLから呼び出されるPL/SQLファンクションのパフォーマンス向上

- *例*：整数のプリティプリント

```
select      PK,
  Print(n1) "n1",
  Print(n2) "n2",
  Print(n3) "n3"
from t
```

```
 1       1 K          1 G        566 G
 2       1 K        157 M        416 G
 3       2 K          1 G        971 G
 4     578 byte       1 G          1 T
 5       2 K          1 G        220 G
 6       1 K          2 G          1 T
 7      48 byte       1 G          2 T
 8     992 byte      42 M          3 T
 9     794 byte       2 G          1 T
10       2 K        302 M        672 G
```

アルゴリズム

- 適切な1024の指数の倍数として整数をプリティプリント：K、M、G、T

```
function Print(n in integer) return varchar2 authid Definer is
  K  constant number not null := 1024;
  M  constant number not null := K*K;
  G  constant number not null := M*K;
  T  constant number not null := G*K;
begin
  return
    case
      when n   <= K-1 then To_Char(n,   '999999')||'byte'
      when n/K <= K-1 then To_Char(n/K, '999999')||'K'
      when n/M <= K-1 then To_Char(n/M, '999999')||'M'
      when n/G <= K-1 then To_Char(n/G, '999999')||'G'
      else                 To_Char(n/T, '999999')||'T'
    end;
end Print;
```

純粋なSQLで試した場合

```
select
  PK,
  case
    when n1            <= 1023 then To_Char(n1,               '999999')||' byte'
    when n1/1024       <= 1023 then To_Char(n1/1024,          '999999')||' K'
    when n1/1048576    <= 1023 then To_Char(n1/1048576,       '999999')||' M'
    when n1/1073741824 <= 1023 then To_Char(n1/1073741824,    '999999')||' G'
    else                            To_Char(n1/1099511627776, '999999')||' T'
  end
  "n1",
  case
    when n2            <= 1023 then To_Char(n2,               '999999')||' byte'
    when n2/1024       <= 1023 then To_Char(n2/1024,          '999999')||' K'
    when n2/1048576    <= 1023 then To_Char(n2/1048576,       '999999')||' M'
    when n2/1073741824 <= 1023 then To_Char(n2/1073741824,    '999999')||' G'
    else                            To_Char(n2/1099511627776, '999999')||' T'
  end
  "n2",
  case
    when n3            <= 1023 then To_Char(n3,               '999999')||' byte'
    when n3/1024       <= 1023 then To_Char(n3/1024,          '999999')||' K'
    when n3/1048576    <= 1023 then To_Char(n3/1048576,       '999999')||' M'
    when n3/1073741824 <= 1023 then To_Char(n3/1073741824,    '999999')||' G'
    else                            To_Char(n3/1099511627776, '999999')||' T'
  end
  "n3"
from t
```

PL/SQLの再利用性と分かりやすさで、SQLのパフォーマンスを実現

```
function Print(n in integer) return varchar2 authid Definer is
  pragma UDF;
  K  constant number not null := 1024;
  M  constant number not null := K*K;
  G  constant number not null := M*K;
  T  constant number not null := G*K;
begin
  return
    case
      when n   <= K-1 then To_Char(n,   '999999')||'byte'
      when n/K <= K-1 then To_Char(n/K, '999999')||'K'
      when n/M <= K-1 then To_Char(n/M, '999999')||'M'
      when n/G <= K-1 then To_Char(n/G, '999999')||'G'
      else                 To_Char(n/T, '999999')||'T'
    end;
end Print;
```

副問合せの*with*句内で PL/SQLファンクションを宣言

```
with
  function Print(n in integer) return varchar2 is
    K  constant number not null := 1024;
    M  constant number not null := K*K;
    G  constant number not null := M*K;
    T  constant number not null := G*K;
  begin
    return
      case
        when n   <= K-1 then To_Char(n,   '999999')||' byte'
        when n/K <= K-1 then To_Char(n/K, '999999')||' K'
        when n/M <= K-1 then To_Char(n/M, '999999')||' M'
        when n/G <= K-1 then To_Char(n/G, '999999')||' G'
        else                 To_Char(n/T, '999999')||' T'
      end;
  end Print;
select       PK,
  Print(n1)  "n1",
  Print(n2)  "n2",
  Print(n3)  "n3"
from t
```

パフォーマンスの比較

▪ 純粋なSQL	5.0倍 最速
▪ プラグマUDFを使用したスキーマレベル・ファンクション	3.9倍 かなり高速
▪ with句内のファンクション	3.8倍 ほぼ同等
▪ 12.1以前の通常のスキーマレベル・ファンクション	1.0– ベースライン もっとも低い パフォーマンス

 |

PL/SQL固有のデータ型からSQL文に対する値のバインド

- 12.1以前でバインドできたのはSQLデータ型のみ
- 12.1では、PL/SQLのindex by pls_integer表（レコード）とブールをバインド可能
  - クライアント側のプログラム（OCI、2種類のJDBC）とPL/SQL**から**
  - 無名ブロック、ファンクションを使用した文、*table* 演算子を使用した文**に対して**

PL/SQL索引付き表のバインド

- 12.1以前では、実際のコレクションを使用したファンクション呼出し、またはコレクションからのSELECTが可能だが、以下の制限がある
  - スキーマレベルでの型定義が必要
  - つまり、ネストした表またはVARRAYになる
  - 非スカラーのペイロードはADTのみ
- 12.1の新機能
  - パッケージ仕様での型定義が可能 - *index by pls_integer* 表を含む
  - ペイロードに*レコード*を使用可能 - ただし、フィールドはSQLデータ型のみ

コレクション

```
package Pkg authid Definer is
  type r is record(n integer, v varchar2(10));
  type t is table of r index by pls_integer;
  x t;
end Pkg;
```


例：
SQL内のPL/SQLファンクションへのIBPIのバインド

```
function f(x in Pkg.t) return varchar2 authid Definer is
  r varchar2(80);
begin
  for j in 1..x.Count() loop r
    := r||...;
  end loop;
  return r;
end f;
```

```
procedure Bind_IBPI_To_Fn_In_SQL authid Definer is
  v varchar2(80);
begin
  select f(Pkg.x) into v from Dual;
  ...
  execute immediate 'select f(:b) from Dual' into v
    using Pkg.x;
end Bind_IBPI_To_Fn_In_SQL;
```


例：
table 演算子のオペランドへのバインド

```
procedure Select_From_IBPI authid Definer is
  y Pkg.t;
begin
  for j in (select n, v from table(Pkg.x)) loop
    ...
  end loop;

  execute immediate 'select n, v from table(:b)'
  bulk collect into y
  using Pkg.x;
  for j in 1..y.Count() loop
    ...
  end loop;
end Select_From_IBPI;
```

例：
無名ブロックへのIBPIのバインド

```
procedure p1(x in Pkg.t) authid Definer is
begin
  for j in 1..x.Count() loop
    ...;
  end loop;
end p1;
```

```
procedure Bind_IBPI_To_Anon_Block authid Definer is
begin
  execute immediate 'begin p1(:b); end;' using Pkg.x;
end Bind_IBPI_To_Anon_Block;
```


例：
無名ブロックへのブールのバインド

```
procedure p2(b in boolean) authid Definer is
begin
  DBMS_Output.Put_Line(case b
                         when  true   then 'True'
                         when  false  then 'False'
                         else              'Null'
                       end);
end p2;
```

```
procedure Bind_Boolean_To_Anon_Block authid Definer is
  Nil constant boolean := null; -- workaround for existing bug
begin
  execute immediate 'begin p2(:b); end;' using true;
  execute immediate 'begin p2(:b); end;' using false;
  execute immediate 'begin p2(:b); end;' using Nil;
end Bind_Boolean_To_Anon_Block;
```

JDBCでのPL/SQL型のバインド

- 12.1以前
  - スキーマレベルのオブジェクト型を生成して、非SQLパッケージ型の構造を反映
  - オブジェクトを移入して、目的のPL/SQLサブプログラムに対するカスタムPL/SQLラッパー内にバインド
  - オブジェクトをラッパー内のパッケージ型に変換し、このパッケージ型を使用してPL/SQLサブプログラムを呼出し

JDBCでのPL/SQL型のバインド

- 12.1の新機能
  - PL/SQLパッケージ型をJDBC内のバインドとしてサポート
  - 非SQL型を使用したPL/SQLサブプログラムの実行が可能
  - *レコード、索引付き表、ネストした表、VARRAY*を含む型をサポート
  - *Table%rowtype*、*view%rowtype*、パッケージ定義の*cursor%rowtype*もサポート（これらは厳密にはレコード型）

例1：JavaからPL/SQLプロシージャへのシングル・レコードのバインド、変更後、Javaから出力として_再バインド

```
package Emp_Info is
  type employee is record(First_Name      Employees.First_Name%type,
                          Last_Name       Employees.Last_Name%type,
                          Employee_Id     Employees.Employee_Id%type,
                          Is_CEO          boolean);

  procedure Get_Emp_Name(Emp_p in out Employee);
end;
```


例1：

- JPublisherで生成した*EmpinfoEmployee*クラスを使用して、*Employee*仮パラメータを実装

```
{  …
      EmpinfoEmployee Employee = new EmpinfoEmployee();
      Employee.setEmployeeId(new java.math.BigDecimal(100));  // Use Employee ID 100

      // Call Get_Emp_Name() with the Employee object
      OracleCallableStatement cstmt =
         (OracleCallableStatement)conn.prepareCall("call EmpInfo.Get_Emp_Name(?)");
      cstmt.setObject(1, Employee, OracleTypes.STRUCT);

      // Use "PACKAGE.TYPE NAME" as the type name
      cstmt.registerOutParameter(1, OracleTypes.STRUCT, "EMPINFO.EMPLOYEE");
      cstmt.execute();

      // Get and print the contents of the Employee object
      EmpinfoEmployee oraData =
          (EmpinfoEmployee)cstmt.getORAData(1, EmpinfoEmployee.getORADataFactory());
      System.out.println("Employee: " + oraData.getFirstName() + " " + oraData.getLastName());
      System.out.println("Is the CEO? " + oraData.getIsceo());
}
```


例2：bulk collect文を使用した*table%rowtype*コレクションの移入と、コール元への出力パラメータとしてのコレクションの引渡し

```
package EmpRow is
  type Table_of_Emp is table of Employees%Rowtype;
  procedure GetEmps(Out_Rows out Table_of_Emp);
end;
```

```
package Body EmpRow is
  procedure GetEmps(Out_Rows out Table_of_Emp) is
  begin
    select *
    bulk collect into Out_Rows
    from Employees;
  end;
end;
```


例2：

```
{ …
   // Call GetEmps() to get the ARRAY of table row data objects
   CallableStatement cstmt = conn.prepareCall("call EmpRow.GetEmps(?)");

    // Use "PACKAGE.COLLECTION NAME" as the type name
   cstmt.registerOutParameter(1, OracleTypes.ARRAY, "EMPROW.TABLE_OF_EMP");
   cstmt.execute();

   // Print the Employee Table rows
   Array a = cstmt.getArray(1);
   String s = Debug.printArray ((ARRAY)a, "",
                                ((ARRAY)a).getSQLTypeName () +"( ", conn);
   System.out.println(s);
}
```


PL/SQL固有のデータ型からSQL文に対するバインド：制限事項

- PL/SQL固有のデータ型はパッケージ仕様内で宣言が必要
- IBPIのレコード・フィールドはSQLデータ型でなければならない
- IBPIのみ、index-by-varchar2は不可
- *insert*、*update*、*delete*、*merge*へのバインドは不可
- DBMS_Sqlを使用したバインドは不可

アジェンダ

- クライアント、PL/SQL、SQL間の相互運用性の強化
- **セキュリティの新機能**
- プログラマのユーザビリティの改善
- その他

PL/SQLユニットへのロールの付与

- 検討すべきベスト・プラクティス
  - PL/SQLサブプログラムを介した場合のみ、アプリケーション・データへのアクセスを許可
  - さらに、エンドユーザー・セッションは、アプリケーション成果物の所有者ではなく、別のデータベース所有者として認可
  - 具体的には、単一のスキーマまたは2、3のスキーマ内で定義者権限のユニットを使用。次に、これらの上での実行権限をエンドユーザーに付与するが、表に対する権限はエンドユーザーに付与しない
- ユニットの所有者は非常に多くの表にアクセスできるため、各ユニットも同様に多くの表にアクセス可能

PL/SQLユニットへのロールの付与

- 12.1では細部化されたスキームを使用できるため、同じ所有者を持つユニットごとに所有者の表に対して異なる権限を付与できる
  - 旧スキームと同様、エンドユーザーの権限は低い
  - ユニットは起動者権限であるため、"そのまま"ではエンドユーザーがデータにアクセスできない
  - ユニットに対して適切な権限を持つロールを付与することで、厳密にユニットの目的に合う権限のみが付与される。このようなロールの無効化は不可
  - ユニットの所有者にはあらかじめ同じロールの付与が必要（有効化は不要）

PL/SQLユニットへのロールの付与

- 概念を示すシナリオ
  - 2種類のユーザー：*App*と*Client*
  - 2種類の表：*App.t1*と*App.t2*
  - 2種類のIRプロシージャ：*App.Show_t1*と*App.Show_t2*が表に対して*SELECT*文を実行
  - クライアントは*App.Show_t1*と*App.Show_t2*に対する実行権限を持つ
  - *App*が2種類のロール：*r_Show_t1*と*r_Show_t2*を作成
  - *App*が*App.t1*の*Select*権限を*r_Show_t1*に付与、～2も同様
  - *App*が*r_Show_t1*を*App.Show_t1*に付与、～2も同様

PL/SQLユニットへのロールの付与

```
create procedure Show_t1 authid Current_User is
begin
  for j in (select Fact from App.t1 order by 1) loop -- Notice the schema-qualification
    ...
  end loop;
end Show_t1;
/
grant Execute on App.Show_t1 to Client
/
-- this has the side-effect of granting the role to App with Admin option
-- other non-schema object types like directories and editions behave the same
create role r_Show_t1
/
grant select on t1 to r_Show_t1
/
grant r_Show_t1 to procedure Show_t1
/
```

```
select Object_Name, Object_Type, Role
from User_Code_Role_Privs
/
```

```
.......     .........   .........
SHOW_T1     PROCEDURE   R_SHOW_T1
```


PL/SQLユニットへのロールの付与

- クライアントが*App.Show_t1*を呼び出す場合、このプロシージャのプログラマが後からどんな不注意な間違いをしても、ロールに与えられた権限に限定される

PL/SQLユニットへのロールの付与

- この新機能による、PL/SQLコンパイル時の静的参照への影響はない

"inherit privileges"権限

- 機能要件
  - 万一、特権ユーザー（特にSysなど）が所有するOracleの提供するコードにSQLインジェクションの脆弱性があった場合、これによるリスクを軽減する
  - IRユニットは起動者のセキュリティ権限を使用して実行される。このため、Sysが所有するDRユニットにインジェクションの脆弱性がある場合、Scottなどのユーザーとしてセッションを認可できる悪意のある個人が、IRユニットを作成してインジェクションの脆弱性を悪用し、Sysのセキュリティ権限でこのIRユニットを実行できる。
  - 出荷時に、Sysは少数のOracle管理ユーザーに対してのみ"inherit privileges"を付与しているため、新機能ではこの抜け道が解消される。約30のその他のOracle管理ユーザーも同様。

"inherit privileges"権限

- その他の要件
  - （少なくともオラクルのガイドラインに従っている限りは）顧客作成コードの動作には影響を与えないこと
  - Sysが所有する既存の顧客の作成したDRユニットから、顧客の作成したユーザーが所有するIRユニットを呼び出している場合は、注意が必要。12.1へのアップグレード時に機能しなくなるが、これはルールに反しているため、問題にはならない。

"bequeath Current_User"ビュー

- ビューが定義する副問合せ内で起動されるIRファンクションで、ビューに対してSQLを実行する現在のユーザーの権限が使用される
- "従来の"DRビューでは、ビューが定義する副問合せ内で起動されるIRファンクションで、ビューの所有者権限が使用される

アジェンダ

- クライアント、PL/SQL、SQL間の相互運用性の強化
- セキュリティの新機能
- **プログラマのユーザビリティの改善**
- その他

ホワイトリスト

- 特定のユニットを、リストに記載したユニットからのみ参照できるように宣言できる
- 無名ブロックをリストに含むことはできないため、ホワイトリストに記載したユニットを動的に呼び出したり、データベースの外部から起動したりすることはできない

accessible by句

```
package Helper authid Definer accessible by (Good_Unit, Bad_Unit)
is
  procedure p;
end Helper;
```

```
package body Good_Unit is
  procedure p is
  begin
    Helper.p();
    ...
  end p;
end Good_Guy;
```

```
package body Bad_Unit is
  procedure p is
  begin
    Helper.p();
     ...
   end p;
 end Bad_Guy;
```

PLS-00904: insufficient privilege to access object HELPER

コール・スタックのイントロスペクションの改善

- 12.1以前では、*DBMS_Utility*パッケージ内で3つのファクションを使用
  - *Format_Call_Stack()*
  - *Format_Error_Stack()*
  - *Format_Error_Backtrace()*
- 12.1の新機能
  - *UTL_Call_Stack*パッケージが同じ問題を正しく解決

イントロスペクションの対象となるコード

```
package body Pkg is
  procedure p is
    procedure q is
      procedure r is
        procedure p is
        begin
          Print_Call_Stack();
        end p;
      begin
        p();
      end r;
    begin
      r();
    end q;
  begin
    q();
  end p;
end Pkg;
```


12.1以前のPrint_Call_Stack()

```
procedure Print_Call_Stack authid Definer is

begin
  DBMS_Output.Put_Line(DBMS_Utility.Format_Call_Stack());
end;
```

```
----- PL/SQL Call Stack -----
  object      line  object
  handle    number  name
0x631f6e88        12  procedure USR.PRINT_CALL_STACK
0x68587700         7  package body USR.PKG
0x68587700        10  package body USR.PKG
0x68587700        13  package body USR.PKG
0x68587700        16  package body USR.PKG
0x69253ca8         1  anonymous block
```

- バグ2769809（2003年1月Bryn提出）を参照

12.1のPrint_Call_Stack()

```
procedure Print_Call_Stack authid Definer is
  Depth pls_integer := TL_Call_Stack.Dynamic_Depth();
begin
  for j in reverse 2..Depth loop
    DBMS_Output.Put_Line(
      (j - 1)||
      To_Char(UTL_Call_Stack.Unit_Line(j), '99')||
      UTL_Call_Stack.Concatenate_Subprogram(UTL_Call_Stack.Subprogram(j)));
  end loop;
end;
```

```
5   1 __anonymous_block
4  16 PKG.P
3  13 PKG.P.Q
2  10 PKG.P.Q.R
1   7 PKG.P.Q.R.P
```


コール・スタックのイントロスペクションの改善

- エラー・スタックとバックトレースに対する対照的なサブプログラム
- 上記に加えて
  - Owner(Depth)
  - Current_Edition(Depth)
  - Lexical_Depth(Depth)

Agenda

- クライアント、PL/SQL、SQL間の相互運用性の強化
- セキュリティの新機能
- プログラマのユーザビリティの改善
- **その他**

12.1で導入されたその他の拡張機能

- 起動者権限のファンクションの結果をキャッシュ可能
  （現在のユーザーがキャッシュの検索キーの一部になる）
- 安全なコールアウト（*extproc*を介した実装）の高速化
  （20倍の高速化を実現するOracle R Enterpriseによる）
- スキーマ間でのオブジェクトの配置方法を変えることなく、
  エディションベースの再定義（EBR）を導入可能 - そのため、
  PL/SQLビューやシノニムのみを変更するすべてのパッチで
  EBRの使用を推奨

12.1で導入されたその他の拡張機能

- *pga_aggregate_limit* - 例：大きすぎるコレクションなどによるこの制限の超過は、致命的エラーを発生させる
- *DBMS_Scheduler*に追加された新しい*Job_Type*
  - *Sql_Script*
  - *Backup_Script*
- 新しい資格証明の使用による制御
  - データベースのユーザー名、パスワード、ロールのカプセル化（AS SYSDBA、AS SYSBACKUPなど）

Hardware and Software

ORACLE

Engineered to Work Together


ORACLE®