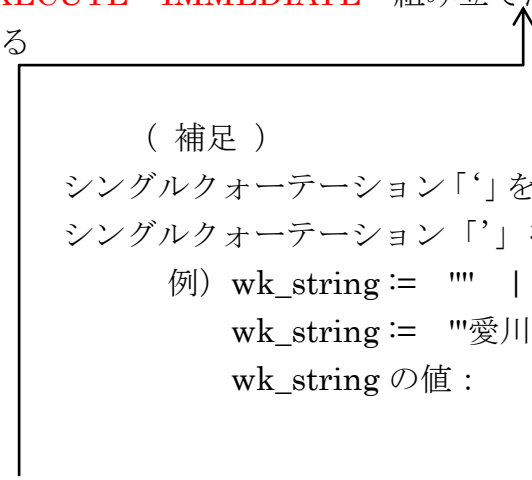
【動的 SQL 文の実行】　　(PL/SQL)

プロシージャ内のロジック判断により、SELECT、INSERT、UPDATE（複数レコード)、CREATE を実行する SQL 文を動的に作成して実行する方法

WHERE 文の条件列の変更や条件の個数変更がある動的な Select 文を作成する記述方法
EXECUTE IMMEDIATE 文を使用して、
注意、1 レコード対象の SELECT や INSERT、UPDATE（複数レコード)、CREATE にしか使用できない。

EXECUTE　IMMEDIATE　組み立てた動的 SQL 文　；　で、動的 SQL の実行が行われる



（ 補足 ）
シングルクォーテーション「‘」を文字列データとして扱ってほしい場合には、
シングルクォーテーション「’」を２個並べて、連結させる。
例）wk_string := ''' || '愛川 こずえ' || ''' ;
wk_string := '''愛川 こずえ''' ;
wk_string の値 : '愛川 こずえ'

目的文字列：(実行するまで、Where 条件が分からなく、動的に SQL 文を作成する方法)

```
SELECT user_addr FROM user_table WHERE user_ name = 'str_para' AND user_no = num_para
```

もしくは、

```
SELECT user_addr FROM user_table WHERE user_ name = 'str_para'
```

もしくは、

```
SELECT user_addr FROM user_table WHERE user_no = num_para
```

（使用例１）動作テスト済　　PL/SQL 無名ブロックでの動的 SQL 作成対応

［SQL 文を組み立てて動的 SQL 文を作成する方法］

注）　INTO 句は、結果セットが１件の時のみ使用可能

```
DECLARE
  wk_sql   VARCHAR2(1024);
  o_ename   VARCHAR2(1024);
BEGIN
  wk_sql  := 'SELECT  member  FROM  emp  WHERE  empno = 1' ;
  EXECUTE  IMMEDIATE   wk_sql  INTO   o_ename;
  DBMS_OUTPUT.PUT_LINE ( '名前は、'  || o_ename  || '  です' ) ;
END;
/
```

単独コマンドで、動的 SQL 文内のバインド変数を使う方法

SQL インジェクション防止対応のため

SQL インジェクション防止対応の方法

プロシージャの中で、SQL インジェクション防止対応を行うようにして、このプロシージャを呼び出せば、防止対応を実施したことになる

プロシージャを使った SQL インジェクション防止対応については、このドキュメントの後半部分を参照のこと

（使用例２）動作テスト済　　１件分のレコード処理

［SQL 文を組み立てて動的 SQL 文を作成する方法］

注）　INTO 句は、結果セットが１件の時のみ使用可能

```
CREATE　OR　REPLACE　PROCEDURE　dynamic_test2　(
    str_where　IN　VARCHAR2 ;
)　AS
    emp_record　emp_table%ROWTYPE　;　　--　レコード型変数の定義
    wk_sql　VARCHAR2(1024)　;
```

参考)

複数個の列への対応

%ROWTYPE を使用しなかったときのレコード変数の定義方法

```
TYPE　emp_record_type　IS　RECORD　(
                                    empno　NUMBER　,
                                    ename　VARCHAR2(14) ,
                                    deptno　NUMBER
                                )　;

emp_record　　emp_record_type　　;
```

```
BEGIN
    /*　動的 SQL 文の組み立て　　*/
    wk_sql　:=　'SELECT　*　FROM　emp_table　'　;
    wk_sql　:=　wk_sql　||　str_where ;　　--　WHERE 条件の追加
    DBMS_OUTPUT.PUT_LINE (　wk_sql　) ;

    EXECUTE　IMMEDIATE　wk_sql　INTO　emp_record　;　--SQL 文の実行
   DBMS_OUTPUT.PUT_LINE ( '番号：　' || emp_record.empno
                          || '　名前：　' || emp_record.ename　) ;

END　dynamic_test2　;
/
```

/* 呼出し */

EXECUTE　dynamic_test2　('WHERE　empno = 2　')　;

実行結果

SELECT　*　FROM　emp_table　　WHERE　empno = 2

番号：　2　　名前：　IKURA

PL/SQL プロシージャが正常に完了しました。

EXECUTE　dynamic_test2　('WHERE　empno > 2　')　;

実行結果

SELECT　*　FROM　emp_table　　WHERE　empno > 2

*

行 1 でエラーが発生しました。:

ORA-01422: 完全フェッチがリクエストよりも多くの行を戻しました

ORA-06512: "KOZUE.DYNAMIC_TEST2", 行 13

※ 複数レコードに対応するためには、プロシージャ側を複数件数対応するように修正する必要がある

次ページに修正コーディングを記述

参考)

複数個の列へテーブルのレコード型である ROWTYPE を使用しないでの対応

%ROWTYPE を使用しなかったときのレコード変数の定義方法

```
TYPE   emp_record_type   IS   RECORD   (
                                          empno   NUMBER   ,
                                          ename   VARCHAR2(14) ,
                                          deptno   NUMBER
                                        )   ;

emp_record          emp_record_type     ;
```

もしくは、

テーブルのレコード型である IS　RECORD 句を使用しない場合

列の値を保存するための**配列変数**を、列の個数分用意する

```
TYPE   wk_event_type      IS TABLE OF
                          V$ACTIVE_SESSION_HISTORY.event%TYPE ;
wk_event                  wk_event_type ;

TYPE   wk_event_count_type        IS TABLE OF   NUMBER   ;
wk_event_count            wk_event_count_type ;

TYPE   wk_sample_time_type        IS TABLE OF   VARCHAR2( 32 ) ;
wk_sample_time            wk_sample_time_type ;
```

用意した複数個の配列変数を、BULK　COLLECT　INTO 句の後に、並べる

```
EXECUTE   IMMEDIATE   wk_sql   BULK   COLLECT   INTO
                          wk_sample_time ,   wk_event ,   wk_event_count    ;
```

（使用例３）動作テスト済

EXECUTE　IMMEDIATE での**複数件数対応**　　バインド変数未使用

［SQL 文を組み立てて動的 SQL 文を作成する方法］

```
CREATE　OR　REPLACE　PROCEDURE　dynamic_test3　(
    str_where　IN　VARCHAR2　≒
)　AS
    wk_sql　VARCHAR2(1024)　;

    /*　　テーブルのデータ型定義　　*/
    TYPE　typ_emp_tab　IS　TABLE　OF　emp%ROWTYPE
                INDEX　BY　PLS_INTEGER　;
    /*　　配列の変数定義　　*/
    dim_emp_record　typ_emp_tab　;


BEGIN
    /*　動的 SQL 文の組み立て　　*/
    wk_sql　:=　'SELECT　*　FROM　emp　'　;
    wk_sql　:=　wk_sql　||　str_where ;　　--　WHERE 条件の追加
    DBMS_OUTPUT.PUT_LINE (　wk_sql　) ;

    --　SQL 文の実行
    EXECUTE　IMMEDIATE　wk_sql　BULK　COLLECT　INTO
        dim_emp_record　;

     IF　dim_emp_record.count　=　0
     THEN
       DBMS_OUTPUT.PUT_LINE (　' データ無し') ;
       RETURN;
     END　IF;

     FOR　i　IN　dim_emp_record.FIRST　..　dim_emp_record.LAST　LOOP
       DBMS_OUTPUT.PUT_LINE (　' 番号：　' || dim_emp_record(i).empno
                        || ' 名前：　' || dim_emp_record(i).member　) ;
     END LOOP;

END　dynamic_test3　;

/

show errors
```

```
/* 呼出し */
EXECUTE  dynamic_test3 （' WHERE  empno >= 1  ORDER BY  empno '）；
```

実行結果

```
SELECT  *  FROM  emp_table  WHERE  empno >= 2  ORDER BY empno
番号： 2 名前： IKURA
番号： 3 名前： MINKA
番号： 4 名前： HATSUNE
PL/SQL プロシージャが正常に完了しました。
```

```
EXECUTE  dynamic_test3 （' WHERE  empno >= 99  ORDER BY  empno '）；
```

実行結果

```
SELECT  *  FROM  emp  WHERE  empno >= 99  ORDER BY  empno
データ無し
PL/SQL プロシージャが正常に完了しました。
```

PL/SQL プロシージャにおける
バインド変数を使った動的 SQL 文の作成と呼び出し方法

注意事項）

PL/SQL プロシージャの中では、バインド変数は使用出来ない

PL/SQL プロシージャの動的 SQL 文中に使われるバインド変数は、SQL*Plus で定義したホスト変数とは異なるものである

PL/SQL ブロックの動的 SQL 文中の中だけで有効となる特殊な変数となる

⇩

このバインド変数の使用は、外部から不正な処理を目的とした SQL インジェクションを防止するための目的のものである

よって、EXECUTE IMMEDIATE の中でセットされる動的 SQL 文に使われるバインド変数には、列値と比較するための値（パラメータ）の数値もしくは文字列しかセット出来ない

⇩

表の列名そのものを示す文字列や'<'などの比較演算子および OR などの条件連結子などは、バインド変数にはセット出来ない

SQL 文の中の WHERE 条件式そのものを動的に変更指定したい場合には、SQL 文の内容を文字列として作成して、EXECUTE IMMEDIATE 文の第一パラメータとして受け渡す

EXECUTE IMMEDIATE　'＜SQL 文の文字列＞';

PL/SQL プロシージャの中でのバインド変数への値のセット方法のは、

動的 SQL 文の実行時の EXECUTE IMMEDIATE の USING 句でプロシージャの仮引数の値を、動的 SQL 文中に使われているバインド変数に受け渡すようにして実行を行う

※ バインド変数の記述の仕方は、'（シングルクォーテェーション）の区切り中にそのまま記述するだけでよい
途中で区切りを中断して、バインド変数を外に出す必要なない

また、バインド変数のデータ型の宣言ステートメントは不要である

EXECUTING ・・・・・ USING 句で指定された値（もしくは、変数）と同一データ型が、自動で割り当てられる

~~EXECUTE IMMEDIATE を使った SELECT 文の中で USING 句を使用した場合には、BULK COLLECT INT 句を使用してテーブル型配列に結果セットを処理することは出来ない~~

【使い方】

EXECUTE IMMEDIATE

'SQL 文 ・・・・・ :バインド変数１ ,:バインド変数２ , ・・'

BULK COLLECT INTO 実行結果を入れる配列

USING 値１（もしくは、変数） , 値２（もしくは、変数） ・・・ ;

バインド変数への値は、SQL 文で使用された順番でセットされていく
USING 句で使用する変数名とバインド変数の名前の指定には関係は無い

~~USING 句で指定できる値（もしくは、変数）は、文字型に限られる~~

~~USING 句を使ってしまうと「 BULK COLLECT INTO 実行結果を入れる配列 」は、使えなくなる~~

（使用例４）動作テスト済

EXECUTE　IMMEDIATE とバインド変数を使用した動的 SQL 文

カーソル OPEN を使用

記述例）

プロシージャ定義

```
CREATE　OR　REPLACE　PROCEDURE　Dynamic_SQL_Used1
  (　 Retsu　　VARCHAR2 ,
      Val　　VARCHAR2
  )
IS

  sql_stmt VARCHAR2( 300 ) ;
/*　 テーブル用の変数定義　 */
emp_record　emp%ROWTYPE　;

/*　 カーソル型の変数定義　 */
TYPE　cursor_type　IS　REF　CURSOR ;
cur_emp　cursor_type ;
BEGIN
  sql_stmt　:='Select　*　FROM　emp　WHERE '　|| Retsu
                          ||　' =　:Bind_Y' ;

  OPEN　cur_emp　FOR　sql_stmt　USING　Val ;

    LOOP
      FETCH　cur_emp　INTO　emp_record ;
      EXIT　WHEN　cur_emp%NOTFOUND;
      DBMS_OUTPUT.PUT_LINE('No : ' || emp_record.empno || ' NAME : '
                                  || emp_record.member );
    END LOOP;

  CLOSE　cur_emp;

END　Dynamic_SQL_Used1　;

 /

  Show　errors
```

プロシージャの呼び出し

```
set   serveroutput   on ;

variable   retsu_name     VARCHAR2 ( 300 ) ;
variable   hikaku_parameter     NUMBER ;

execute   :retsu_name   :=   'EMPNO' ;
execute   :hikaku_parameter   :=   1 ;
execute   Dynamic_SQL_Used1 (:retsu_name ,   :hikaku_parameter )
          No : 1 NAME : 愛川こずえ

execute   :retsu_name   :=   'DEPTNO' ;
execute   :hikaku_parameter   :=   1 ;
execute   Dynamic_SQL_Used1 ( :retsu_name ,   :hikaku_parameter ) ;
          No : 1 NAME : 愛川こずえ
          No : 2 NAME : いとくとら
          No : 3 NAME : ミンカ・リー

execute   :retsu_name   :=   'EMPNO' ;
execute   :hikaku_parameter   :=   9 ;
execute   Dynamic_SQL_Used1 ( :retsu_name ,   :hikaku_parameter ) ;
```

（使用例５）動作テスト済

EXECUTE　IMMEDIATE とバインド変数を使用した動的 SQL 文

BULK　COLLECT　INTO を使用

記述例）

プロシージャ定義

```
CREATE　OR　REPLACE　PROCEDURE　Dynamic_SQL_Used2
  (　 Retsu1　　VARCHAR2 ,　Val1　　VARCHAR2 ,
　　Retsu2　　VARCHAR2 ,　Val2　　VARCHAR2　 )
IS

  sql_stmt VARCHAR2( 300 ) ;

/*　 テーブルのデータ型定義　 */
TYPE　typ_emp_tab　IS　TABLE　OF　emp%ROWTYPE
                INDEX　BY　PLS_INTEGER　;
/*　 配列の変数定義　 */
dim_emp_record　typ_emp_tab　;

BEGIN
  sql_stmt　 := 'Select　 *　 FROM　 emp　 WHERE '
                  || Retsu1　 ||　 ' =　 :Bind1　 AND '
                  || Retsu2　 ||　 ' =　 :Bind2'　 ;

--　SQL 文の実行
EXECUTE　IMMEDIATE　sql_stmt　BULK　COLLECT　INTO
        dim_emp_record　USING　Val1 ,　Val2 ;

 IF　dim_emp_record.count　=　0
 THEN
   DBMS_OUTPUT.PUT_LINE (　' データ無し') ;
   RETURN;
 END　IF;

 FOR　i　IN　dim_emp_record.FIRST　..　dim_emp_record.LAST　LOOP
   DBMS_OUTPUT.PUT_LINE (　' 番号：　' || dim_emp_record(i).empno
                              || ' 名前：　' || dim_emp_record(i).member　 ) ;
 END LOOP;

END　Dynamic_SQL_Used2　;

 /
 Show　 errors
```

プロシージャの呼び出し

```
set  serveroutput  on ;

variable  retsu1_name    VARCHAR2 ( 300 ) ;
variable  hikaku1_parameter    NUMBER ;
variable  retsu2_name    VARCHAR2 ( 300 ) ;
variable  hikaku2_parameter    NUMBER ;

execute  :retsu1_name  :=  'EMPNO' ;
execute  :hikaku1_parameter  :=  1 ;
execute  :retsu2_name  :=  'DEPTNO' ;
execute  :hikaku2_parameter  :=  1 ;
execute  Dynamic_SQL_Used2(  :retsu1_name ,  :hikaku1_parameter ,  -
                    :retsu2_name ,  :hikaku2_parameter )
```

No : 1 NAME : 愛川こずえ

```
execute  :retsu1_name  :=  'DEPTNO' ;
execute  :hikaku1_parameter  :=  1 ;
execute  :retsu2_name  :=  'DEPTNO' ;
execute  :hikaku2_parameter  :=  1 ;
execute  Dynamic_SQL_Used2(  :retsu1_name ,  :hikaku1_parameter ,  -
                    :retsu2_name ,  :hikaku2_parameter )
```

No : 1 NAME : 愛川こずえ
No : 2 NAME : いとくとら
No : 3 NAME : ミンカ・リー

```
execute  :retsu1_name  :=  'EMPNO' ;
execute  :hikaku1_parameter  :=  9 ;
execute  :retsu2_name  :=  'DEPTNO' ;
execute  :hikaku2_parameter  :=  1 ;
execute  Dynamic_SQL_Used2(  :retsu1_name ,  :hikaku1_parameter ,  -
                    :retsu2_name ,  :hikaku2_parameter )
```

データ無し

SQL 文の組み立て方法
（文字列の中のシングルクォーテーション「’」）
WHERE の条件式中の文字指定値の記述

文字列の値として、WHERE　user_name　＝　‘愛川こずえ’；　を設定する場合の記述方法

シングルクォーテーション「‘」を文字列データとして扱ってほしい場合には、シングルクォーテーション「’」を２個並べて、連結させる。

例）user_name := ''　||　'愛川 こずえ'　||　''　；
user_name := '''愛川 こずえ'''　；
user_name の値：　　'愛川 こずえ'

（使用例）
目的文字列：

```
SELECT　user_no　FROM　user_table　WHERE　user_ name　=　'愛川こずえ' ≒
```

対応ソース・コードの記述

```
DECLARE

BEGIN
   wk_sql　:= ' SELECT　user_no　FROM　user_table　WHERE　user_name　= '
                 ||　''                    --　シングルクォーテーション「’」を４個
                 ||　'愛川こずえ'
                 ||　''　；                --　シングルクォーテーション「’」を４個
   EXECUTE IMMEDIATE　wk_sql ;
END　；

/
```

目的文字列：

```
SELECT  user_no  FROM  user_table  WHERE  user_ name
        LIKE  '*こずえ*' ≒
```

対応ソース・コードの記述

```
DECLARE
   wk_ename  SCOTT.user_table.user_name%TYPE  ;
   wk_sql  VARCHAR2( 150 )  ;
BEGIN
   wk_ename  :=  '愛川こずえ'  ;
   wk_sql   := ' SELECT   user_no   FROM   user_table
        WHERE  user_ name  LIKE  '
                ||  '"*'                  --  シングルクォーテーション「’」を 3+1 個
                ||  wk_ename
                ||  '*"'  ;               --  シングルクォーテーション「’」を 1+3 個
   EXECUTE IMMEDIATE    wk_sql ;
END  ;

/
```