EBook Gratis

APRENDIZAJE plsql

Free unaffiliated eBook created from
Stack Overflow contributors.

#plsql

Tabla de contenido

Acerca de ... 1

Capítulo 1: Empezando con plsql ... 2

Observaciones ... 2

Examples ... 2

Definición de PLSQL ... 2

Hola Mundo ... 3

Acerca de PLSQL ... 4

Diferencia entre% TYPE y% ROWTYPE ... 4

Crear o reemplazar una vista ... 4

Crear una tabla ... 5

Capítulo 2: Asignaciones modelo y lenguaje. ... 6

Examples ... 6

Modelo de asignaciones en PL / SQL ... 6

Capítulo 3: Colecciones y registros ... 8

Examples ... 8

Utilice una colección como un tipo de retorno para una función de división ... 8

Capítulo 4: Cursores ... 9

Sintaxis ... 9

Observaciones ... 9

Examples ... 9

Cursor parametrizado "FOR loop" ... 9

Cursor de bucle "FOR" implícito ... 9

Trabajando con SYS_REFCURSOR ... 10

función devolviendo un cursor ... 10

Y cómo usarlo: ... 10

Manejando un CURSOR ... 11

Capítulo 5: Declaración IF-THEN-ELSE ... 12

Sintaxis ... 12

Examples ... 12

IF-THEN ... 12

IF-THEN-ELSE 12
IF-THEN-ELSIF-ELSE 13
Capítulo 6: Funciones 14
Sintaxis 14
Examples 14
Generar GUID 14
Funciones de llamada 14
Capítulo 7: Gatillos 16
Introducción 16
Sintaxis 16
Examples 16
Antes de activar INSERT o ACTUALIZAR 16
Capítulo 8: Lazo 18
Sintaxis 18
Examples 18
Bucle simple 18
Mientras que bucle 18
En bucle 19
Capítulo 9: Manejo de excepciones 21
Introducción 21
Examples 21
Manejo de excepciones 21
Sintaxis 21
Excepciones definidas internamente 22
Excepciones predefinidas 23
Excepciones definidas por el usuario 24
Defina una excepción personalizada, levántela y vea de dónde viene 24
Manejo de excepciones de error de conexión 26
Capítulo 10: Manejo de excepciones 28
Introducción 28
Examples 28
Manejo de excepciones de error de conexión 28

Defina una excepción personalizada, levántela y vea de dónde viene. 29
Capítulo 11: Paquetes 31
Sintaxis 31
Examples 32
Uso del paquete 33
Sobrecarga 33
Restricciones en la sobrecarga 34
Definir un encabezado de paquete y un cuerpo con una función. 34
Capítulo 12: Procedimiento PLSQL 36
Introducción 36
Examples 36
Sintaxis 36
Hola Mundo 36
Parámetros de entrada / salida 37
Capítulo 13: Recoger a granel 38
Examples 38
Procesamiento de datos a granel 38
Capítulo 14: Tipos de objeto 39
Observaciones 39
Examples 39
BASE_TYPE 39
MID_TYPE 40
LEAF_TYPE 41
Accediendo a objetos almacenados 42
Creditos 44

Acerca de

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: plsql

It is an unofficial and free plsql ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official plsql.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Capítulo 1: Empezando con plsql

Observaciones

Esta sección proporciona una descripción general de qué es plsql y por qué un desarrollador puede querer usarlo.

También debe mencionar cualquier tema grande dentro de plsql, y vincular a los temas relacionados. Dado que la Documentación para plsql es nueva, es posible que deba crear versiones iniciales de esos temas relacionados.

Examples

Definición de PLSQL

PL / SQL (Lenguaje de procedimiento / lenguaje de consulta estructurado) es la extensión de procedimiento de Oracle Corporation para SQL y la base de datos relacional de Oracle. PL / SQL está disponible en Oracle Database (desde la versión 7), en la base de datos en memoria de TimesTen (desde la versión 11.2.1) e IBM DB2 (desde la versión 9.7).

La unidad básica en PL / SQL se llama bloque, que se compone de tres partes: una parte declarativa, una parte ejecutable y una parte de creación de excepciones.

```
DECLARE
   <declarations section>
BEGIN
   <executable command(s)>
EXCEPTION
   <exception handling>
END;
```

Declaraciones - Esta sección comienza con la palabra clave DECLARAR. Es una sección opcional y define todas las variables, cursores, subprogramas y otros elementos que se utilizarán en el programa.

Comandos ejecutables : esta sección se encuentra entre las palabras clave BEGIN y END y es una sección obligatoria. Consiste en las sentencias PL / SQL ejecutables del programa. Debe tener al menos una línea de código ejecutable, que puede ser solo un comando NULO para indicar que no se debe ejecutar nada.

Manejo de excepciones : esta sección comienza con la palabra clave EXCEPTION. Esta sección es nuevamente opcional y contiene excepciones que manejan errores en el programa.

Cada sentencia PL / SQL termina con un punto y coma (;). Los bloques PL / SQL se pueden anidar dentro de otros bloques PL / SQL usando BEGIN y END.

En el bloque anónimo, solo se requiere la parte ejecutable del bloque, otras partes no son

necesarias. A continuación se muestra un ejemplo de código anónimo simple, que no hace nada pero funciona sin informe de errores.

```
BEGIN
    NULL;
END;
/
```

La falta de una instrucción ejecutable genera un error, ya que PL / SQL no admite bloques vacíos. Por ejemplo, la ejecución del código a continuación conduce a un error:

```
BEGIN
END;
/
```

La aplicación generará un error:

```
END;
*
ERROR at line 2:
ORA-06550: line 2, column 1:
PLS-00103: Encountered the symbol "END" when expecting one of the following:
( begin case declare exit for goto if loop mod null pragma
raise return select update while with <an identifier>
<a double-quoted delimited-identifier> <a bind variable> <<
continue close current delete fetch lock insert open rollback
savepoint set sql execute commit forall merge pipe purge
```

Símbolo "*" en la línea debajo de la palabra clave "FIN"; significa, que el bloque que termina con este bloque está vacío o mal construido. Cada bloque de ejecución necesita instrucciones para hacerlo, incluso si no hace nada, como en nuestro ejemplo.

Hola Mundo

```
set serveroutput on

DECLARE
   message constant varchar2(32767):= 'Hello, World!';
BEGIN
   dbms_output.put_line(message);
END;
/
```

El `set serveroutput on` comandos `set serveroutput on` es necesario en los clientes SQL * Plus y SQL Developer para habilitar la salida de `dbms_output` . Sin el comando no se muestra nada.

El `end;` La línea señala el final del bloque PL / SQL anónimo. Para ejecutar el código desde la línea de comandos SQL, es posible que tenga que escribir `/` al principio de la primera línea en blanco después de la última línea del código. Cuando el código anterior se ejecuta en el indicador SQL, produce el siguiente resultado:

```
Hello, World!

PL/SQL procedure successfully completed.
```

Acerca de PLSQL

PL / SQL significa extensiones de lenguaje de procedimiento a SQL. PL / SQL está disponible solo como una "tecnología habilitadora" dentro de otros productos de software; no existe como un lenguaje independiente. Puede usar PL / SQL en la base de datos relacional de Oracle, en el servidor de Oracle y en las herramientas de desarrollo de aplicaciones del lado del cliente, como Oracle Forms. Estas son algunas de las formas en que podría usar PL / SQL:

1. Para construir procedimientos almacenados. .
2. Para crear disparadores de base de datos.
3. Para implementar la lógica del lado del cliente en su aplicación de Oracle Forms.
4. Para vincular una página de inicio de World Wide Web a una base de datos de Oracle.

Diferencia entre% TYPE y% ROWTYPE.

`%TYPE` : se utiliza para declarar un campo con el mismo tipo que el de una columna de tabla específica.

```
DECLARE
        vEmployeeName   Employee.Name%TYPE;
BEGIN
        SELECT Name
        INTO   vEmployeeName
        FROM   Employee
        WHERE  RowNum = 1;

        DBMS_OUTPUT.PUT_LINE(vEmployeeName);
END;
/
```

% ROWTYPE: se utiliza para declarar un registro con los mismos tipos que se encuentran en la tabla, vista o cursor especificados (= columnas múltiples).

```
DECLARE
        rEmployee     Employee%ROWTYPE;
BEGIN
        rEmployee.Name := 'Matt';
        rEmployee.Age := 31;

        DBMS_OUTPUT.PUT_LINE(rEmployee.Name);
        DBMS_OUTPUT.PUT_LINE(rEmployee.Age);
END;
/
```

Crear o reemplazar una vista

En este ejemplo vamos a crear una vista.

Una vista se utiliza principalmente como una forma sencilla de obtener datos de varias tablas.

Ejemplo 1:
Ver con una selección en una tabla.

```
CREATE OR REPLACE VIEW LessonView AS
       SELECT     L.*
       FROM       Lesson L;
```

Ejemplo 2:
Ver con una selección en varias tablas.

```
CREATE OR REPLACE VIEW ClassRoomLessonView AS
       SELECT     C.Id,
                  C.Name,
                  L.Subject,
                  L.Teacher
       FROM       ClassRoom C,
                  Lesson L
       WHERE      C.Id = L.ClassRoomId;
```

Para llamar a estas vistas en una consulta, puede utilizar una declaración de selección.

```
SELECT * FROM LessonView;
SELECT * FROM ClassRoomLessonView;
```

Crear una tabla

A continuación vamos a crear una tabla con 3 columnas.
El `Id` columna que se debe completar es, por lo que lo definimos `NOT NULL` .
En la columna `Contract` , también agregamos un cheque para que el único valor permitido sea 'Y' o 'N'. Si una inserción está en hecho y esta columna no se especifica durante la inserción, entonces se inserta una 'N' por defecto.

```
CREATE TABLE Employee (
        Id            NUMBER NOT NULL,
        Name          VARCHAR2(60),
        Contract      CHAR DEFAULT 'N' NOT NULL,
        ---
        CONSTRAINT p_Id PRIMARY KEY(Id),
        CONSTRAINT c_Contract CHECK (Contract IN('Y','N'))
);
```

Lea Empezando con plsql en línea: https://riptutorial.com/es/plsql/topic/1962/empezando-con-plsql

Capítulo 2: Asignaciones modelo y lenguaje.

Examples

Modelo de asignaciones en PL / SQL.

Todos los lenguajes de programación nos permiten asignar valores a las variables. Por lo general, un valor se asigna a la variable, de pie en el lado izquierdo. El prototipo de las operaciones de asignación generales en cualquier lenguaje de programación contemporáneo se ve así:

```
left_operand assignment_operand right_operand instructions_of_stop
```

Esto asignará el operando derecho al operando izquierdo. En PL / SQL esta operación se ve así:

```
left_operand := right_operand;
```

El operando izquierdo **debe ser siempre una variable** . El operando derecho puede ser valor, variable o función:

```
set serveroutput on
declare
  v_hello1 varchar2(32767);
  v_hello2 varchar2(32767);
  v_hello3 varchar2(32767);
  function hello return varchar2 is begin return 'Hello from a function!'; end;
begin
   -- from a value (string literal)
  v_hello1 := 'Hello from a value!';
   -- from variable
  v_hello2 := v_hello1;
  -- from function
  v_hello3 := hello;

  dbms_output.put_line(v_hello1);
  dbms_output.put_line(v_hello2);
  dbms_output.put_line(v_hello3);
end;
/
```

Cuando el bloque de código se ejecuta en SQL * Plus, se imprime la siguiente salida en la consola:

```
Hello from a value!
Hello from a value!
Hello from a function!
```

Hay una característica en PL / SQL que nos permite asignar "de derecha a izquierda". Es posible hacerlo en sentencia SELECT INTO. Prototipo de este instrumento que se encuentra a continuación:

```
SELECT [ literal | column_value ]

INTO local_variable

FROM [ table_name | aliastable_name ]

WHERE comparison_instructions;
```

Este código asignará el carácter literal a una variable local:

```
set serveroutput on
declare
  v_hello varchar2(32767);
begin
  select 'Hello world!'
  into v_hello
  from dual;

  dbms_output.put_line(v_hello);
end;
/
```

Cuando el bloque de código se ejecuta en SQL * Plus, se imprime la siguiente salida en la consola:

```
Hello world!
```

La asignación "de derecha a izquierda" **no es un estándar** , pero es una característica valiosa para los programadores y usuarios. Generalmente se usa cuando el programador está usando cursores en PL / SQL: esta técnica se usa cuando queremos devolver un único valor escalar o un conjunto de columnas en la línea de cursor del cursor SQL.

Otras lecturas:

- Asignación de valores a variables

Lea Asignaciones modelo y lenguaje. en línea:
https://riptutorial.com/es/plsql/topic/6959/asignaciones-modelo-y-lenguaje-

Capítulo 3: Colecciones y registros

Examples

Utilice una colección como un tipo de retorno para una función de división

Es necesario declarar el tipo; aquí `t_my_list` ; una colección es una `TABLE OF` *`something`*

```
CREATE OR REPLACE TYPE t_my_list AS TABLE OF VARCHAR2(100);
```

Aquí está la función. Observe el `()` utilizado como un tipo de constructor, y las palabras clave `COUNT` y `EXTEND` que le ayudan a crear y hacer crecer su colección;

```
CREATE OR REPLACE
FUNCTION cto_table(p_sep in Varchar2, p_list IN VARCHAR2)
  RETURN t_my_list
AS
--- this function takes a string list, element being separated by p_sep
--                                                    as separator
  l_string VARCHAR2(4000) := p_list || p_sep;
  l_sep_index PLS_INTEGER;
  l_index PLS_INTEGER := 1;
  l_tab t_my_list     := t_my_list();
BEGIN
  LOOP
    l_sep_index := INSTR(l_string, p_sep, l_index);
    EXIT
  WHEN l_sep_index = 0;
    l_tab.EXTEND;
    l_tab(l_tab.COUNT) := TRIM(SUBSTR(l_string,l_index,l_sep_index - l_index));
    l_index            := l_sep_index + 1;
  END LOOP;
  RETURN l_tab;
END cto_table;
/
```

Luego puede ver el contenido de la colección con la función `TABLE()` de SQL; se puede usar como una lista dentro de una sentencia SQL `IN ( ..)` :

```
select * from A_TABLE
 where A_COLUMN in ( TABLE(cto_table('|','a|b|c|d')) )
--- gives the records where A_COLUMN in ('a', 'b', 'c', 'd') --
```

Lea Colecciones y registros en línea: https://riptutorial.com/es/plsql/topic/9779/colecciones-y-registros

Capítulo 4: Cursores

Sintaxis

- Cursor *cursor_name* Is *your_select_statement*
- Cursor *cursor_name* (param TYPE) es *your_select_statement_using_param*
- FOR x in (*your_select_statement*) LOOP ...

Observaciones

Los cursores declarados son difíciles de usar, y usted prefiere los bucles `FOR` en la mayoría de los casos. Lo que es muy interesante en cursores en comparación con los bucles `FOR` simples, es que puede parametrizarlos.

Es mejor evitar hacer bucles con PL / SQL y cursores en lugar de usar Oracle SQL de todos modos. Sin embargo, para las personas acostumbradas al lenguaje de procedimientos, puede ser mucho más fácil de entender.

Si desea verificar si existe un registro, y luego hacer cosas diferentes dependiendo de si el registro existe o no, entonces tiene sentido usar las instrucciones `MERGE` en consultas SQL de ORACLE puras en lugar de usar bucles de cursor. (Tenga en cuenta que `MERGE` solo está disponible en las versiones de Oracle> = 9i).

Examples

Cursor parametrizado "FOR loop"

```
DECLARE
  CURSOR c_emp_to_be_raised(p_sal emp.sal%TYPE) IS
    SELECT * FROM emp WHERE  sal < p_sal;
BEGIN
  FOR cRowEmp IN c_emp_to_be_raised(1000) LOOP
    dbms_Output.Put_Line(cRowEmp .eName ||' ' ||cRowEmp.sal||'... should be raised ;)');
  END LOOP;
END;
/
```

Cursor de bucle "FOR" implícito

```
BEGIN
  FOR x IN (SELECT * FROM emp WHERE sal < 100) LOOP
    dbms_Output.Put_Line(x.eName ||' '||x.sal||'... should REALLY be raised :D');
  END LOOP;
END;
/
```

- La primera ventaja es que no hay ninguna declaración tediosa que hacer (piense en esta

horrible cosa "CURSOR" que tenía en versiones anteriores)

- La segunda ventaja es que primero construye su consulta de selección, luego, cuando tiene lo que desea, puede acceder de inmediato a los campos de su consulta (`x.<myfield>`) en su bucle PL / SQL
- El bucle abre el cursor y obtiene un registro a la vez para cada bucle. Al final del bucle se cierra el cursor.
- Los cursores implícitos son más rápidos porque el trabajo del intérprete crece a medida que el código se alarga. Cuanto menos código menos trabajo tiene que hacer el intérprete.

Trabajando con SYS_REFCURSOR

`SYS_REFCURSOR` se puede usar como un tipo de retorno cuando necesita manejar fácilmente una lista devuelta no desde una tabla, sino más específicamente desde una función:

función devolviendo un cursor

```
CREATE OR REPLACE FUNCTION list_of (required_type_in IN VARCHAR2)
   RETURN SYS_REFCURSOR
IS
   v_ SYS_REFCURSOR;
BEGIN
   CASE required_type_in
      WHEN 'CATS'
      THEN
         OPEN v_ FOR
           SELECT nickname FROM (
                select 'minou' nickname from dual
      union all select 'minâ'           from dual
      union all select 'minon'          from dual
           );
      WHEN 'DOGS'
      THEN
         OPEN v_ FOR
              SELECT dog_call FROM (
                select 'bill'   dog_call from dual
      union all select 'nestor'          from dual
      union all select 'raoul'           from dual
           );
   END CASE;
   -- Whit this use, you must not close the cursor.
   RETURN v_;
END list_of;
/
```

Y cómo usarlo:

```
DECLARE
   v_names   SYS_REFCURSOR;
   v_        VARCHAR2 (32767);
BEGIN
   v_names := list_of('CATS');
```

```
   LOOP
      FETCH v_names INTO v_;
      EXIT WHEN v_names%NOTFOUND;
      DBMS_OUTPUT.put_line(v_);
   END LOOP;
   -- here you close it
   CLOSE v_names;
END;
/
```

Manejando un CURSOR

- Declara el cursor para escanear una lista de registros
- Abrelo
- Obtener el registro actual en variables (esto incrementa la posición)
- Utilice `%notfound` para detectar el final de la lista
- No olvide cerrar el cursor para limitar el consumo de recursos en el contexto actual

-

```
DECLARE
  CURSOR curCols IS -- select column name and type from a given table
         SELECT column_name, data_type FROM all_tab_columns where table_name='MY_TABLE';
  v_tab_column all_tab_columns.column_name%TYPE;
  v_data_type all_tab_columns.data_type%TYPE;
  v_ INTEGER := 1;
BEGIN
  OPEN curCols;
  LOOP
     FETCH curCols INTO v_tab_column, v_data_type;
     IF curCols%notfound OR v_ > 2000 THEN
       EXIT;
     END IF;
     dbms_output.put_line(v_||':Column '||v_tab_column||' is of '|| v_data_type||' Type.');
     v_:= v_ + 1;
  END LOOP;

  -- Close in any case
  IF curCols%ISOPEN THEN
     CLOSE curCols;
  END IF;
END;
/
```

Lea Cursores en línea: https://riptutorial.com/es/plsql/topic/5303/cursores

Capítulo 5: Declaración IF-THEN-ELSE

Sintaxis

- SI [condición 1] ENTONCES
- [instrucciones para ejecutar cuando la condición 1 es VERDADERA];
- ELSIF [condición 2] LUEGO
- [instrucciones para ejecutar cuando la condición 2 es VERDADERA];
- MÁS
- [declaraciones a ejecutar cuando tanto la condición 1 como la condición 2 son FALSAS];
- TERMINARA SI;

Examples

IF-THEN

```
DECLARE
v_num1 NUMBER(10);
v_num2 NUMBER(10);

BEGIN
  v_num1 := 2;
  v_num2 := 1;

  IF v_num1 > v_num2 THEN
     dbms_output.put_line('v_num1 is bigger than v_num2');
  END IF;
END;
```

IF-THEN-ELSE

```
DECLARE
v_num1 NUMBER(10);
v_num2 NUMBER(10);

BEGIN
  v_num1 := 2;
  v_num2 := 10;

  IF v_num1 > v_num2 THEN
     dbms_output.put_line('v_num1 is bigger than v_num2');
  ELSE
    dbms_output.put_line('v_num1 is NOT bigger than v_num2');
  END IF;
END;
```

IF-THEN-ELSIF-ELSE

```
DECLARE
v_num1 NUMBER(10);
v_num2 NUMBER(10);

BEGIN
  v_num1 := 2;
  v_num2 := 2;

  IF v_num1 > v_num2 THEN
     dbms_output.put_line('v_num1 is bigger than v_num2');
  ELSIF v_num1 < v_num2 THEN
    dbms_output.put_line('v_num1 is NOT bigger than v_num2');
  ELSE
    dbms_output.put_line('v_num1 is EQUAL to v_num2');
  END IF;
END;
```

Lea Declaración IF-THEN-ELSE en línea: https://riptutorial.com/es/plsql/topic/5871/declaracion-if-then-else

Capítulo 6: Funciones

Sintaxis

- CREAR [O REEMPLAZAR] FUNCIÓN nombre_función [(parámetro [, parámetro])]

  VOLVER return_datatype

  Es | COMO

  [declaración_sección]

  COMENZAR_sección_ejecutable

  [EXCEPCIÓN excepción_sección]

  END [function_name];

Examples

Generar GUID

```
Create Or Replace Function Generateguid
Return Char Is
    V_Guid Char(40);
Begin
    Select Substr(Sys_Guid(),1,8)||'-'||Substr(Sys_Guid(),9,4)||'-'
                        ||Substr(Sys_Guid(),13,4)||'-'||Substr(Sys_Guid(),17,4)||'-'
                        ||Substr(Sys_Guid(),21) Into V_Guid
                        From Dual;
    Return V_Guid;
Exception
    When Others Then
    dbms_output.put_line('Error '|| SQLERRM);
End Generateguid;
```

Funciones de llamada

Hay algunas formas de usar las funciones.

Llamar a una función con una sentencia de asignación

```
DECLARE
    x NUMBER := functionName(); --functions can be called in declaration section
BEGIN
    x := functionName();
END;
```

Llamando a una función en la declaración IF

```
IF functionName() = 100 THEN
    Null;
END IF;
```

Llamando una función en una instrucción SELECT

```
SELECT functionName() FROM DUAL;
```

Lea Funciones en línea: https://riptutorial.com/es/plsql/topic/4005/funciones

Capítulo 7: Gatillos

Introducción

Introducción:

Los disparadores son un concepto útil en PL / SQL. Un disparador es un tipo especial de procedimiento almacenado que no requiere ser llamado explícitamente por el usuario. Es un grupo de instrucciones, que se activan automáticamente en respuesta a una acción de modificación de datos específica en una tabla o relación específica, o cuando se cumplen ciertas condiciones específicas. Los activadores ayudan a mantener la integridad y la seguridad de los datos. Hacen el trabajo conveniente tomando la acción requerida automáticamente.

Sintaxis

- CREAR [O REEMPLAZAR] TRIGGER trigger_name
- ANTES DE ACTUALIZAR [o INSERTAR] [o BORRAR]
- ON table_name
- [POR CADA FILA]
- DECLARAR
- - Declaraciones variables
- EMPEZAR
- - código de activación
- EXCEPCIÓN
- CUANDO ...
- -- manejo de excepciones
- FIN;

Examples

Antes de activar INSERT o ACTUALIZAR

```
CREATE OR REPLACE TRIGGER CORE_MANUAL_BIUR
  BEFORE INSERT OR UPDATE ON CORE_MANUAL
  FOR EACH ROW
BEGIN
  if inserting then
    -- only set the current date if it is not specified
    if :new.created is null then
      :new.created := sysdate;
    end if;
  end if;

  -- always set the modified date to now
  if inserting or updating then
    :new.modified := sysdate;
  end if;
```

```
end;
/
```

Lea Gatillos en línea: https://riptutorial.com/es/plsql/topic/7674/gatillos

Capítulo 8: Lazo

Sintaxis

1. LAZO
2. [declaraciones];
3. SALIR CUANDO [condición para el bucle de salida];
4. Bucle final;

Examples

Bucle simple

```
DECLARE
v_counter NUMBER(2);

BEGIN
  v_counter := 0;
  LOOP
    v_counter := v_counter + 1;
    dbms_output.put_line('Line number' || v_counter);

    EXIT WHEN v_counter = 10;
  END LOOP;
END;
```

Mientras que bucle

El bucle WHILE se ejecuta hasta que se cumpla la condición de fin. Ejemplo simple:

```
DECLARE
v_counter NUMBER(2); --declaration of counter variable

BEGIN
  v_counter := 0; --point of start, first value of our iteration

  WHILE v_counter < 10 LOOP --exit condition

    dbms_output.put_line('Current iteration of loop is ' || v_counter); --show current
iteration number in dbms script output
    v_counter := v_counter + 1; --incrementation of counter value, very important step

  END LOOP; --end of loop declaration
END;
```

Este bucle se ejecutará hasta que el valor actual de la variable v_counter sea menor que diez.

El resultado:

```
Current iteration of loop is 0
```

```
Current iteration of loop is 1
Current iteration of loop is 2
Current iteration of loop is 3
Current iteration of loop is 4
Current iteration of loop is 5
Current iteration of loop is 6
Current iteration of loop is 7
Current iteration of loop is 8
Current iteration of loop is 9
```

Lo más importante es que nuestro bucle comienza con el valor '0', por lo que la primera línea de resultados es 'La iteración actual del bucle es 0'.

En bucle

Loop FOR trabaja en reglas similares a las de otros bucles. El bucle FOR se ejecuta el número exacto de veces y este número se conoce al principio; los límites inferior y superior se establecen directamente en el código. En cada paso de este ejemplo, el bucle se incrementa en 1.

Ejemplo simple:

```
DECLARE
v_counter NUMBER(2); --declaration of counter variable

BEGIN
  v_counter := 0; --point of start, first value of our iteration, execute of variable

  FOR v_counter IN 1..10 LOOP --The point, where lower and upper point of loop statement is
declared - in this example, loop will be executed 10 times, start with value of 1

    dbms_output.put_line('Current iteration of loop is ' || v_counter); --show current
iteration number in dbms script output

  END LOOP; --end of loop declaration
END;
```

Y el resultado es:

```
Current iteration of loop is 1
Current iteration of loop is 2
Current iteration of loop is 3
Current iteration of loop is 4
Current iteration of loop is 5
Current iteration of loop is 6
Current iteration of loop is 7
Current iteration of loop is 8
Current iteration of loop is 9
Current iteration of loop is 10
```

Loop FOR tiene propiedad adicional, que está funcionando a la inversa. El uso de la palabra adicional 'REVERSE' en la declaración del límite inferior y superior del bucle permite hacer eso. Cada ejecución del valor de decremento de bucle de v_counter por 1.

Ejemplo:

```
DECLARE
v_counter NUMBER(2); --declaration of counter variable

BEGIN
  v_counter := 0; --point of start

  FOR v_counter IN REVERSE 1..10 LOOP

    dbms_output.put_line('Current iteration of loop is ' || v_counter); --show current
iteration number in dbms script output

  END LOOP; --end of loop declaration
END;
```

Y el resultado:

```
Current iteration of loop is 10
Current iteration of loop is 9
Current iteration of loop is 8
Current iteration of loop is 7
Current iteration of loop is 6
Current iteration of loop is 5
Current iteration of loop is 4
Current iteration of loop is 3
Current iteration of loop is 2
Current iteration of loop is 1
```

Lea Lazo en línea: https://riptutorial.com/es/plsql/topic/6157/lazo

Capítulo 9: Manejo de excepciones

Introducción

Oracle produce una variedad de excepciones. Es posible que se sorprenda de lo tedioso que puede ser que su código se detenga con un mensaje poco claro. Para mejorar la capacidad de su código PL / SQL para arreglarse fácilmente, es necesario manejar las excepciones en el nivel más bajo. Nunca oculte una excepción "debajo de la alfombra", a menos que esté aquí para guardar su parte del código solo para usted y para que nadie más lo pueda mantener.

Los errores predefinidos .

Examples

Manejo de excepciones

1. ¿Qué es una excepción?

   La excepción en PL / SQL es un error creado durante la ejecución de un programa.

   Tenemos tres tipos de excepciones:

   - Excepciones definidas internamente
   - Excepciones predefinidas
   - Excepciones definidas por el usuario

2. ¿Qué es una excepción de manejo?

   El manejo de excepciones es una posibilidad de mantener nuestro programa en ejecución incluso si aparece un error de tiempo de ejecución como resultado de, por ejemplo, errores de codificación, fallas de hardware. Evitamos que salga abruptamente.

Sintaxis

La sintaxis general para la sección de excepción:

```
declare
    declaration Section
begin
    some statements

exception
    when exception_one then
        do something
    when exception_two then
        do something
    when exception_three then
        do something
    when others then
```

```
        do something
end;
```

Una sección de excepción debe estar al final del bloque PL / SQL. PL / SQL nos da la oportunidad de anidar bloques, entonces cada bloque puede tener su propia sección de excepción, por ejemplo:

```
create or replace procedure nested_blocks
is
begin
    some statements
    begin
        some statements

    exception
        when exception_one then
            do something
    end;
exception
    when exception_two then
        do something
end;
```

Si la excepción se generará en el bloque anidado, debe manejarse en la sección de excepción interna, pero si la sección de excepción interna no maneja esta excepción, esta excepción irá a la sección de excepción del bloque externo.

Excepciones definidas internamente

Una excepción definida internamente no tiene un nombre, pero tiene su propio código.

¿Cuándo usarlo?

Si sabe que la operación de su base de datos puede generar excepciones específicas a aquellas que no tienen nombres, puede darles nombres para que pueda escribir controladores de excepciones específicamente para ellos. De lo contrario, puede usarlos solo con `others` controladores de excepciones.

Sintaxis

```
declare
    my_name_exc exception;
    pragma exception_init(my_name_exc,-37);
begin
    ...
exception
    when my_name_exc then
        do something
end;
```

`my_name_exc exception;` Esa es la declaración del nombre de excepción.

`pragma exception_init(my_name_exc,-37);` Asigne un nombre al código de error de la excepción

definida internamente.

Ejemplo

Tenemos un emp_id que es una clave primaria en la tabla emp y una clave externa en la tabla dept. Si intentamos eliminar emp_id cuando tiene registros secundarios, se lanzará una excepción con el código -2292.

```
create or replace procedure remove_employee
is
    emp_exception exception;
    pragma exception_init(emp_exception,-2292);
begin
    delete from emp where emp_id = 3;
exception
    when emp_exception then
        dbms_output.put_line('You can not do that!');
end;
/
```

> La documentación de Oracle dice: "Una excepción definida internamente con un nombre declarado por el usuario sigue siendo una excepción definida internamente, no una excepción definida por el usuario".

Excepciones predefinidas

Las excepciones predefinidas son excepciones definidas internamente pero tienen nombres. La base de datos Oracle eleva este tipo de excepciones automáticamente.

Ejemplo

```
create or replace procedure insert_emp
is
begin
    insert into emp (emp_id, ename) values ('1','Jon');

exception
    when dup_val_on_index then
        dbms_output.put_line('Duplicate value on index!');
end;
/
```

A continuación se muestran ejemplos de excepciones con sus códigos:

Nombre de excepción	Código de error
DATOS NO ENCONTRADOS	-1403
ACCESS_INTO_NULL	-6530
CASE_NOT_FOUND	-6592
ROWTYPE_MISMATCH	-6504

Nombre de excepción	Código de error
TOO_MANY_ROWS	-1422
CERO_DIVIDIR	-1476

Lista completa de nombres de excepciones y sus códigos en el sitio web de Oracle.

Excepciones definidas por el usuario

Como el nombre sugiere, los usuarios crean excepciones definidas por el usuario. Si desea crear su propia excepción, debe:

1. Declarar la excepción
2. Levántalo de tu programa
3. Crea un manejador de excepciones adecuado para atraparlo.

Ejemplo

Quiero actualizar todos los salarios de los trabajadores. Pero si no hay trabajadores, levante una excepción.

```
create or replace procedure update_salary
is
    no_workers exception;
    v_counter number := 0;
begin
    select count(*) into v_counter from emp;
    if v_counter = 0 then
        raise no_workers;
    else
        update emp set salary = 3000;
    end if;

    exception
        when no_workers then
            raise_application_error(-20991,'We don''t have workers!');
end;
/
```

¿Qué significa `raise` ?

Las excepciones son generadas por el servidor de la base de datos automáticamente cuando es necesario, pero si lo desea, puede generar explícitamente cualquier excepción mediante el uso de `raise` .

Procedimiento `raise_application_error(error_number,error_message);`

- error_number debe estar entre -20000 y -20999
- mensaje de error mensaje que se muestra cuando se produce un error.

Defina una excepción personalizada, levántela y vea de dónde viene.

Para ilustrar esto, aquí hay una función que tiene 3 comportamientos "incorrectos" diferentes

- el parámetro es completamente estúpido: usamos una expresión definida por el usuario
- el parámetro tiene un error tipográfico: utilizamos el error `NO_DATA_FOUND` estándar de Oracle
- otro caso, pero no manejado

Siéntete libre de adaptarlo a tus estándares:

```
DECLARE
  this_is_not_acceptable EXCEPTION;
  PRAGMA EXCEPTION_INIT(this_is_not_acceptable, -20077);
  g_err varchar2 (200) := 'to-be-defined';
  w_schema all_tables.OWNER%Type;

  PROCEDURE get_schema( p_table in Varchar2, p_schema out Varchar2)
  Is
    w_err varchar2 (200) := 'to-be-defined';
  BEGIN
    w_err := 'get_schema-step-1:';
    If (p_table = 'Delivery-Manager-Is-Silly') Then
      raise this_is_not_acceptable;
    end if;
    w_err := 'get_schema-step-2:';
    Select owner Into p_schema
      From all_tables
     where table_name like(p_table||'%');
  EXCEPTION
  WHEN NO_DATA_FOUND THEN
    -- handle Oracle-defined exception
     dbms_output.put_line('[WARN]'||w_err||'This can happen. Check the table name you
entered.');
  WHEN this_is_not_acceptable THEN
    -- handle your custom error
     dbms_output.put_line('[WARN]'||w_err||'Please don''t make fun of the delivery manager.');
  When others then
     dbms_output.put_line('[ERR]'||w_err||'unhandled exception:'||sqlerrm);
     raise;
  END Get_schema;

BEGIN
  g_err := 'Global; first call:';
  get_schema('Delivery-Manager-Is-Silly', w_schema);
  g_err := 'Global; second call:';
  get_schema('AAA', w_schema);
  g_err := 'Global; third call:';
  get_schema('', w_schema);
  g_err := 'Global; 4th call:';
  get_schema('Can''t reach this point due to previous error.', w_schema);

EXCEPTION
  When others then
     dbms_output.put_line('[ERR]'||g_err||'unhandled exception:'||sqlerrm);
  -- you may raise this again to the caller if error log isn't enough.
--  raise;
END;
/
```

Dar en una base de datos regular:

```
[WARN]get_schema-step-1:Please don't make fun of the delivery manager.
[WARN]get_schema-step-2:This can happen. Check the table name you entered.
[ERR]get_schema-step-2:unhandled exception:ORA-01422: exact fetch returns more than requested
number of rows
[ERR]Global; third call:unhandled exception:ORA-01422: exact fetch returns more than requested
number of rows
```

Recuerde que la excepción está aquí para manejar casos *raros* . Vi aplicaciones que generaron una excepción en cada acceso, solo para solicitar la contraseña del usuario, diciendo "no conectado" ... tanto desperdicio de cómputo.

Manejo de excepciones de error de conexión

Cada error estándar de Oracle está asociado con un número de error. Es importante anticipar lo que podría salir mal en su código. Aquí para una conexión a otra base de datos, puede ser:

- `-28000` cuenta está bloqueada
- `-28001` contraseña caducada
- `-28002` periodo de gracia
- `-1017` usuario incorrecto / contraseña

Aquí hay una manera de probar qué falla con el usuario que usa el enlace de la base de datos:

```
declare
  v_dummy number;
begin
  -- testing db link
  execute immediate 'select COUNT(1) from dba_users@pass.world' into v_dummy ;
  -- if we get here, exception wasn't raised: display COUNT's result
  dbms_output.put_line(v_dummy||' users on PASS db');

EXCEPTION
  -- exception can be referred by their name in the predefined Oracle's list
    When LOGIN_DENIED
    then
        dbms_output.put_line('ORA-1017 / USERNAME OR PASSWORD INVALID, TRY AGAIN');
    When Others
    then
  -- or referred by their number: stored automatically in reserved variable SQLCODE
        If  SQLCODE = '-2019'
        Then
          dbms_output.put_line('ORA-2019 / Invalid db_link name');
        Elsif SQLCODE = '-1035'
        Then
          dbms_output.put_line('ORA-1035 / DATABASE IS ON RESTRICTED SESSION, CONTACT YOUR
DBA');
        Elsif SQLCODE = '-28000'
        Then
          dbms_output.put_line('ORA-28000 / ACCOUNT IS LOCKED. CONTACT YOUR DBA');
        Elsif SQLCODE = '-28001'
        Then
          dbms_output.put_line('ORA-28001 / PASSWORD EXPIRED. CONTACT YOUR DBA FOR CHANGE');
        Elsif SQLCODE  = '-28002'
        Then
          dbms_output.put_line('ORA-28002 / PASSWORD IS EXPIRED, CHANGED IT');
        Else
```

```
   -- and if it's not one of the exception you expected
          dbms_output.put_line('Exception not specifically handled');
          dbms_output.put_line('Oracle Said'||SQLCODE||':'||SQLERRM);
       End if;
END;
/
```

Lea Manejo de excepciones en línea: https://riptutorial.com/es/plsql/topic/6050/manejo-de-excepciones

Capítulo 10: Manejo de excepciones

Introducción

Oracle produce una variedad de excepciones. Es posible que se sorprenda de lo tedioso que puede ser que su código se detenga con un mensaje poco claro. Para mejorar la capacidad de su código PL / SQL para arreglarse fácilmente, es necesario manejar las excepciones en el nivel más bajo. Nunca oculte una excepción "debajo de la alfombra", a menos que esté aquí para guardar su parte del código solo para usted y para que nadie más lo pueda mantener.

Los errores predefinidos .

Examples

Manejo de excepciones de error de conexión

Cada error estándar de Oracle está asociado con un número de error. Es importante anticipar lo que podría salir mal en su código. Aquí para una conexión a otra base de datos puede ser:

- `-28000` cuenta está bloqueada
- `-28001` contraseña caducada
- `-28002` periodo de gracia
- `-1017` usuario incorrecto / contraseña

Aquí hay una manera de probar qué falla con el usuario que usa el enlace de la base de datos:

```
declare
  v_dummy number;
begin
  -- testing db link
  execute immediate 'select COUNT(1) from dba_users@pass.world' into v_dummy ;
  -- if we get here, exception wasn't raised: display COUNT's result
  dbms_output.put_line(v_dummy||' users on PASS db');

EXCEPTION
  -- exception can be referred by their name in the predefined Oracle's list
    When LOGIN_DENIED
    then
        dbms_output.put_line('ORA-1017 / USERNAME OR PASSWORD INVALID, TRY AGAIN');
    When Others
    then
  -- or referred by their number: stored automatically in reserved variable SQLCODE
        If  SQLCODE = '-2019'
        Then
          dbms_output.put_line('ORA-2019 / Invalid db_link name');
        Elsif SQLCODE = '-1035'
        Then
          dbms_output.put_line('ORA-1035 / DATABASE IS ON RESTRICTED SESSION, CONTACT YOUR
DBA');
        Elsif SQLCODE = '-28000'
        Then
```

```
          dbms_output.put_line('ORA-28000 / ACCOUNT IS LOCKED. CONTACT YOUR DBA');
        Elsif SQLCODE = '-28001'
        Then
          dbms_output.put_line('ORA-28001 / PASSWORD EXPIRED. CONTACT YOUR DBA FOR CHANGE');
        Elsif SQLCODE  = '-28002'
        Then
          dbms_output.put_line('ORA-28002 / PASSWORD IS EXPIRED, CHANGED IT');
        Else
   -- and if it's not one of the exception you expected
          dbms_output.put_line('Exception not specifically handled');
          dbms_output.put_line('Oracle Said'||SQLCODE||':'||SQLERRM);
        End if;
END;
/
```

Defina una excepción personalizada, levántela y vea de dónde viene.

Para ilustrar esto, aquí hay una función que tiene 3 comportamientos "incorrectos" diferentes

- el parámetro es completamente estúpido: usamos una expresión definida por el usuario
- el parámetro tiene un error tipográfico: `NO_DATA_FOUND` error estándar `NO_DATA_FOUND` Oracle
- otro caso, pero no manejado

Siéntete libre de adaptarlo a tus estándares:

```
DECLARE
  this_is_not_acceptable EXCEPTION;
  PRAGMA EXCEPTION_INIT(this_is_not_acceptable, -20077);
  g_err varchar2 (200) := 'to-be-defined';
  w_schema all_tables.OWNER%Type;

  PROCEDURE get_schema( p_table in Varchar2, p_schema out Varchar2)
  Is
    w_err varchar2 (200) := 'to-be-defined';
  BEGIN
    w_err := 'get_schema-step-1:';
    If (p_table = 'Delivery-Manager-Is-Silly') Then
      raise this_is_not_acceptable;
    end if;
    w_err := 'get_schema-step-2:';
    Select owner Into p_schema
      From all_tables
     where table_name like(p_table||'%');
  EXCEPTION
  WHEN NO_DATA_FOUND THEN
    -- handle Oracle-defined exception
     dbms_output.put_line('[WARN]'||w_err||'This can happen. Check the table name you
entered.');
  WHEN this_is_not_acceptable THEN
    -- handle your custom error
     dbms_output.put_line('[WARN]'||w_err||'Please don''t make fun of the delivery manager.');
  When others then
     dbms_output.put_line('[ERR]'||w_err||'unhandled exception:'||sqlerrm);
     raise;
  END Get_schema;

BEGIN
  g_err := 'Global; first call:';
```

```
  get_schema('Delivery-Manager-Is-Silly', w_schema);
  g_err := 'Global; second call:';
  get_schema('AAA', w_schema);
  g_err := 'Global; third call:';
  get_schema('', w_schema);
  g_err := 'Global; 4th call:';
  get_schema('Can''t reach this point due to previous error.', w_schema);

EXCEPTION
  When others then
     dbms_output.put_line('[ERR]'||g_err||'unhandled exception:'||sqlerrm);
  -- you may raise this again to the caller if error log isn't enough.
--  raise;
END;
/
```

Dar en una base de datos regular:

```
[WARN]get_schema-step-1:Please don't make fun of the delivery manager.
[WARN]get_schema-step-2:This can happen. Check the table name you entered.
[ERR]get_schema-step-2:unhandled exception:ORA-01422: exact fetch returns more than requested
number of rows
[ERR]Global; third call:unhandled exception:ORA-01422: exact fetch returns more than requested
number of rows
```

Recuerde que la excepción está aquí para manejar casos *raros* . Vi aplicaciones que generaron una excepción en cada acceso, solo para solicitar una contraseña de usuario, diciendo "no conectado" ... tanto desperdicio de cómputo.

Lea Manejo de excepciones en línea: https://riptutorial.com/es/plsql/topic/9480/manejo-de-excepciones

Capítulo 11: Paquetes

Sintaxis

- CREAR [O REEMPLAZAR] PAQUETE nombre_paquete

  [AUTHID {CURRENT_USER | DEFINER}]

  {IS | COMO}

  [PRAGMA SERIALLY_REUSABLE;]

  [collection_type_definition ...]

  [record_type_definition ...]

  [subtipo_definición ...]

  [colección_declaración ...]

  [constante_declaración ...]

  [excepción_declaración ...]

  [object_declaration ...]

  [record_declaration ...]

  [variable_declaración ...]

  [cursor_spec ...]

  [function_spec ...]

  [procedure_spec ...]

  [call_spec ...]

  [PRAGMA RESTRICT_REFERENCES (aserciones) ...]

  END [nombre_paquete];

- CREAR O REEMPLAZAR EL PAQUETE PackageName IS

  FUNCIÓN FunctionName (parámetro 1 EN VARCHAR2, paramter2 EN NÚMERO) RETURN VARCHAR2;

  END PackageName;

- CREAR [O REEMPLAZAR] PACKAGE BODY package_name

{IS | COMO}

[PRAGMA SERIALLY_REUSABLE;]

[collection_type_definition ...]

[record_type_definition ...]

[subtipo_definición ...]

[colección_declaración ...]

[constante_declaración ...]

[excepción_declaración ...]

[object_declaration ...]

[record_declaration ...]

[variable_declaración ...]

[cursor_body ...]

[function_spec ...]

[procedure_spec ...]

[call_spec ...]

END [nombre_paquete];

- CREAR O REEMPLAZAR EL CUERPO DEL PAQUETE PackageName IS

  FUNCIÓN FunctionName (parámetro 1 EN VARCHAR2, paramter2 EN NÚMERO) RETURN VARCHAR2 IS

  declaraciones

  EMPEZAR

  declaraciones para ejecutar

  VOLVER *varchar2 variable*

  END FunctionName;

  END PackageName;

Examples

Uso del paquete

Los paquetes en PLSQL son una colección de procedimientos, funciones, variables, excepciones, constantes y estructuras de datos. En general, los recursos de un paquete están relacionados entre sí y realizan tareas similares.

Por qué usar paquetes

- Modularidad
- Mejor rendimiento / funcionalidad

Partes de un paquete

Especificación - A veces se llama un encabezado de paquete. Contiene declaraciones de variables y tipos y las firmas de las funciones y procedimientos que se encuentran en el paquete, que son **públicos** para ser llamado desde el exterior del paquete.

Cuerpo del paquete: contiene el código y **las** declaraciones **privadas** .

La especificación del paquete debe compilarse antes que el cuerpo del paquete, de lo contrario, la compilación del cuerpo del paquete informará un error.

Sobrecarga

Las funciones y procedimientos en los paquetes pueden estar sobrecargados. El siguiente paquete de **PRUEBA** tiene dos procedimientos llamados número de **impresión** , que se comportan de manera diferente según los parámetros con los que se llama.

```
create or replace package TEST is
  procedure print_number(p_number in integer);
  procedure print_number(p_number in varchar2);
end TEST;
/
create or replace package body TEST is

  procedure print_number(p_number in integer) is
  begin
    dbms_output.put_line('Digit: ' || p_number);
  end;

  procedure print_number(p_number in varchar2) is
  begin
    dbms_output.put_line('String: ' || p_number);
  end;

end TEST;
/
```

Llamamos a ambos procedimientos. El primero con parámetro entero, el segundo con varchar2.

```
set serveroutput on;
-- call the first procedure
exec test.print_number(3);
```

```
-- call the second procedure
exec test.print_number('three');
```

La salida del script anterior es:

```
SQL>
Digit: 3
PL/SQL procedure successfully completed
String: three
PL/SQL procedure successfully completed
```

Restricciones en la sobrecarga

Solo se pueden sobrecargar los subprogramas locales o empaquetados, o los métodos de tipo. Por lo tanto, no puede sobrecargar subprogramas independientes. Además, no puede sobrecargar dos subprogramas si sus parámetros formales difieren solo en el modo de nombre o parámetro

Definir un encabezado de paquete y un cuerpo con una función.

En este ejemplo, definimos un encabezado de paquete y un cuerpo de paquete con una función. Después de eso, estamos llamando a una función del paquete que devuelve un valor de retorno.

Encabezado del paquete :

```
CREATE OR REPLACE PACKAGE SkyPkg AS

       FUNCTION GetSkyColour(vPlanet IN VARCHAR2)
       RETURN VARCHAR2;

END;
/
```

Cuerpo del paquete :

```
CREATE OR REPLACE PACKAGE BODY SkyPkg AS

        FUNCTION GetSkyColour(vPlanet IN VARCHAR2)
        RETURN VARCHAR2
        AS
               vColour VARCHAR2(100) := NULL;
        BEGIN
               IF vPlanet = 'Earth' THEN
                       vColour := 'Blue';
               ELSIF vPlanet = 'Mars' THEN
                       vColour := 'Red';
               END IF;

               RETURN vColour;
        END;

END;
/
```

Llamando a la función desde el cuerpo del paquete :

```
DECLARE
        vColour VARCHAR2(100);
BEGIN
        vColour := SkyPkg.GetSkyColour(vPlanet => 'Earth');
        DBMS_OUTPUT.PUT_LINE(vColour);
END;
/
```

Lea Paquetes en línea: https://riptutorial.com/es/plsql/topic/4764/paquetes

Capítulo 12: Procedimiento PLSQL

Introducción

El procedimiento PLSQL es un grupo de sentencias SQL almacenadas en el servidor para su reutilización. Aumenta el rendimiento porque las instrucciones SQL no tienen que volver a compilarse cada vez que se ejecutan.

Los procedimientos almacenados son útiles cuando varias aplicaciones requieren el mismo código. Tener procedimientos almacenados elimina la redundancia e introduce la simplicidad en el código. Cuando se requiere la transferencia de datos entre el cliente y el servidor, los procedimientos pueden reducir los costos de comunicación en ciertas situaciones.

Examples

Sintaxis

```
CREATE [OR REPLACE] PROCEDURE procedure_name
[(parameter_name [IN | OUT | IN OUT] type [, ...])]
{IS | AS}
  < declarations >
BEGIN
  < procedure_body >
EXCEPTION                         -- Exception-handling part begins
  <exception handling goes here >
   WHEN exception1 THEN
       exception1-handling-statements
END procedure_name;
```

- nombre-procedimiento especifica el nombre del procedimiento.
- La opción [O REEMPLAZAR] permite modificar un procedimiento existente.
- La lista de parámetros opcionales contiene nombre, modo y tipos de parámetros. IN representa que el valor se pasará desde afuera y OUT representa que este parámetro se usará para devolver un valor fuera del procedimiento. Si no se especifica ningún modo, se supone que el parámetro está en modo IN.
- En la sección de declaración podemos declarar variables que serán utilizadas en la parte del cuerpo.
- procedure-body contiene la parte ejecutable.
- La palabra clave AS se utiliza en lugar de la palabra clave IS para crear un procedimiento independiente.
- La sección de excepciones manejará las excepciones del procedimiento. Esta sección es opcional.

Hola Mundo

El siguiente procedimiento simple muestra el texto "Hello World" en un cliente que admite `dbms_output` .

```
CREATE OR REPLACE PROCEDURE helloworld
AS
BEGIN
   dbms_output.put_line('Hello World!');
END;
/
```

Debe ejecutar esto en el indicador de SQL para crear el procedimiento en la base de datos, o puede ejecutar la consulta a continuación para obtener el mismo resultado:

```
SELECT 'Hello World!' from dual;
```

Parámetros de entrada / salida

PL / SQL usa las palabras clave IN, OUT, IN OUT para definir qué puede pasar con un parámetro pasado.

IN especifica que el parámetro es de solo lectura y el procedimiento no puede cambiar el valor.

OUT especifica que el parámetro es solo de escritura y un procedimiento puede asignarle un valor, pero no hacer referencia al valor.

IN OUT especifica que el parámetro está disponible para referencia y modificación.

```
PROCEDURE procedureName(x IN INT, strVar IN VARCHAR2, ans OUT VARCHAR2)
...
...
END procedureName;


procedureName(firstvar, secondvar, thirdvar);
```

Las variables que se pasaron en el ejemplo anterior deben escribirse tal como se definen en la sección de parámetros del procedimiento.

Lea Procedimiento PLSQL en línea: https://riptutorial.com/es/plsql/topic/2580/procedimiento-plsql

Capítulo 13: Recoger a granel

Examples

Procesamiento de datos a granel

colecciones locales no están permitidas en declaraciones selectas. Por lo tanto, el primer paso es crear una colección de nivel de esquema. Si la colección no es de nivel de esquema y se utiliza en las instrucciones SELECT, se generará "PLS-00642: los tipos de colección local no están permitidos en las instrucciones SQL"

```
CREATE OR REPLACE TYPE table1_t IS OBJECT (
a_1 INTEGER,
a_2 VARCHAR2(10)
);
```

- Permitir permisos en la colección para que pueda usarse públicamente en la base de datos

```
     GRANT EXECUTE ON table1_t TO PUBLIC;
     CREATE OR REPLACE TYPE table1_tbl_typ IS TABLE OF table1_t;
     GRANT EXECUTE ON table1_tbl_typ TO PUBLIC;
```

--seque los datos de la tabla a la colección y luego recorra la colección e imprima los datos.

```
    DECLARE
     table1_tbl table1_tbl_typ;
    BEGIN
     table1_tbl := table1_tbl_typ();
      SELECT table1_t(a_1,a_2)
      BULK COLLECT INTO table1_tbl
      FROM table1 WHERE ROWNUM<10;

     FOR rec IN (SELECT a_1 FROM TABLE(table1_tbl))--table(table1_tbl) won't give error)
     LOOP
       dbms_output.put_line('a_1'||rec.a_1);
       dbms_output.put_line('a_2'||rec.a_2);
     END LOOP;
    END;
/
```

Lea Recoger a granel en línea: https://riptutorial.com/es/plsql/topic/6855/recoger-a-granel

Capítulo 14: Tipos de objeto

Observaciones

Es importante tener en cuenta que un cuerpo de objeto puede no ser siempre necesario. Si el constructor predeterminado es suficiente y no es necesario implementar ninguna otra funcionalidad, no se debe crear.

Un constructor predeterminado es el constructor suministrado por Oracle, que consta de todos los atributos listados en orden de declaración. Por ejemplo, una instancia de BASE_TYPE podría construirse mediante la siguiente llamada, aunque no la declaremos explícitamente.

```
l_obj := BASE_TYPE(1, 'Default', 1);
```

Examples

BASE_TYPE

Declaración de tipo:

```
CREATE OR REPLACE TYPE base_type AS OBJECT
(
   base_id     INTEGER,
   base_attr   VARCHAR2(400),
   null_attr   INTEGER, -- Present only to demonstrate non-default constructors
   CONSTRUCTOR FUNCTION base_type
   (
      i_base_id INTEGER,
      i_base_attr VARCHAR2
   ) RETURN SELF AS RESULT,
   MEMBER FUNCTION get_base_id RETURN INTEGER,
   MEMBER FUNCTION get_base_attr RETURN VARCHAR2,
   MEMBER PROCEDURE set_base_id(i_base_id INTEGER),
   MEMBER PROCEDURE set_base_attr(i_base_attr VARCHAR2),
   MEMBER FUNCTION to_string RETURN VARCHAR2
) INSTANTIABLE NOT FINAL
```

Tipo de cuerpo:

```
CREATE OR REPLACE TYPE BODY base_type AS
   CONSTRUCTOR FUNCTION base_type
   (
      i_base_id INTEGER,
      i_base_attr VARCHAR2
   ) RETURN SELF AS RESULT
   IS
   BEGIN
      self.base_id := i_base_id;
      self.base_attr := i_base_attr;
      RETURN;
```

```
   END base_type;

   MEMBER FUNCTION get_base_id RETURN INTEGER IS
   BEGIN
      RETURN self.base_id;
   END get_base_id;

   MEMBER FUNCTION get_base_attr RETURN VARCHAR2 IS
   BEGIN
      RETURN self.base_attr;
   END get_base_attr;

   MEMBER PROCEDURE set_base_id(i_base_id INTEGER) IS
   BEGIN
      self.base_id := i_base_id;
   END set_base_id;

   MEMBER PROCEDURE set_base_attr(i_base_attr VARCHAR2) IS
   BEGIN
      self.base_attr := i_base_attr;
   END set_base_attr;

   MEMBER FUNCTION to_string RETURN VARCHAR2 IS
   BEGIN
      RETURN 'BASE_ID ['||self.base_id||']; BASE_ATTR ['||self.base_attr||']';
   END to_string;
END;
```

MID_TYPE

Declaración de tipo:

```
CREATE OR REPLACE TYPE mid_type UNDER base_type
(
   mid_attr DATE,
   CONSTRUCTOR FUNCTION mid_type
   (
      i_base_id   INTEGER,
      i_base_attr VARCHAR2,
      i_mid_attr  DATE
   ) RETURN SELF AS RESULT,
   MEMBER FUNCTION get_mid_attr RETURN DATE,
   MEMBER PROCEDURE set_mid_attr(i_mid_attr DATE),
   OVERRIDING MEMBER FUNCTION to_string RETURN VARCHAR2
) INSTANTIABLE NOT FINAL
```

Tipo de cuerpo:

```
CREATE OR REPLACE TYPE BODY mid_type AS
   CONSTRUCTOR FUNCTION mid_type
   (
      i_base_id   INTEGER,
      i_base_attr VARCHAR2,
      i_mid_attr  DATE
   ) RETURN SELF AS RESULT
   IS
   BEGIN
      self.base_id := i_base_id;
```

```
      self.base_attr := i_base_attr;
      self.mid_attr := i_mid_attr;
      RETURN;
   END mid_type;

   MEMBER FUNCTION get_mid_attr RETURN DATE IS
   BEGIN
      RETURN self.mid_attr;
   END get_mid_attr;

   MEMBER PROCEDURE set_mid_attr(i_mid_attr DATE) IS
   BEGIN
      self.mid_attr := i_mid_attr;
   END set_mid_attr;

   OVERRIDING MEMBER FUNCTION to_string RETURN VARCHAR2
   IS
   BEGIN
      RETURN (SELF AS base_type).to_string || '; MID_ATTR [' || self.mid_attr || ']';
   END to_string;
END;
```

LEAF_TYPE

Declaración de tipo:

```
CREATE OR REPLACE TYPE leaf_type UNDER mid_type
(
   leaf_attr VARCHAR2(1000),
   CONSTRUCTOR FUNCTION leaf_type
   (
      i_base_id   INTEGER,
      i_base_attr VARCHAR2,
      i_mid_attr  DATE,
      i_leaf_attr VARCHAR2
   ) RETURN SELF AS RESULT,
   MEMBER FUNCTION get_leaf_attr RETURN VARCHAR2,
   MEMBER PROCEDURE set_leaf_attr(i_leaf_attr VARCHAR2),
   OVERRIDING MEMBER FUNCTION to_string RETURN VARCHAR2
) INSTANTIABLE FINAL
```

Tipo de Cuerpo:

```
CREATE OR REPLACE TYPE BODY leaf_type AS
   CONSTRUCTOR FUNCTION leaf_type
   (
      i_base_id   INTEGER,
      i_base_attr VARCHAR2,
      i_mid_attr  DATE,
      i_leaf_attr VARCHAR2
   ) RETURN SELF AS RESULT
   IS
   BEGIN
      self.base_id := i_base_id;
      self.base_attr := i_base_attr;
      self.mid_attr := i_mid_attr;
      self.leaf_attr := i_leaf_attr;
      RETURN;
```

```
   END leaf_type;

   MEMBER FUNCTION get_leaf_attr RETURN VARCHAR2 IS
   BEGIN
      RETURN self.leaf_attr;
   END get_leaf_attr;

   MEMBER PROCEDURE set_leaf_attr(i_leaf_attr VARCHAR2) IS
   BEGIN
      self.leaf_attr := i_leaf_attr;
   END set_leaf_attr;

   OVERRIDING MEMBER FUNCTION to_string RETURN VARCHAR2 IS
   BEGIN
      RETURN (SELF AS mid_type).to_string || '; LEAF_ATTR [' || self.leaf_attr || ']';
   END to_string;
END;
```

Accediendo a objetos almacenados

```
CREATE SEQUENCE test_seq START WITH 1001;

CREATE TABLE test_tab
(
   test_id  INTEGER,
   test_obj base_type,
   PRIMARY KEY (test_id)
);

INSERT INTO test_tab (test_id, test_obj)
VALUES (test_seq.nextval, base_type(1,'BASE_TYPE'));
INSERT INTO test_tab (test_id, test_obj)
VALUES (test_seq.nextval, base_type(2,'BASE_TYPE'));
INSERT INTO test_tab (test_id, test_obj)
VALUES (test_seq.nextval, mid_type(3, 'MID_TYPE',SYSDATE - 1));
INSERT INTO test_tab (test_id, test_obj)
VALUES (test_seq.nextval, mid_type(4, 'MID_TYPE',SYSDATE + 1));
INSERT INTO test_tab (test_id, test_obj)
VALUES (test_seq.nextval, leaf_type(5, 'LEAF_TYPE',SYSDATE - 20,'Maple'));
INSERT INTO test_tab (test_id, test_obj)
VALUES (test_seq.nextval, leaf_type(6, 'LEAF_TYPE',SYSDATE + 20,'Oak'));
```

Devuelve referencia de objeto:

```
SELECT test_id
      ,test_obj
  FROM test_tab;
```

Devuelve la referencia del objeto, empujando todo a subtipo

```
SELECT test_id
      ,TREAT(test_obj AS mid_type) AS obj
  FROM test_tab;
```

Devuelve un descriptor de cadena de cada objeto, por tipo

```
SELECT test_id
      ,TREAT(test_obj AS base_type).to_string() AS to_string -- Parenthesis are needed after
the function name, or Oracle will look for an attribute of this name.
  FROM test_tab;
```

Lea Tipos de objeto en línea: https://riptutorial.com/es/plsql/topic/7699/tipos-de-objeto

Creditos

S. No	Capítulos	Contributors
1	Empezando con plsql	Community, Dinidu, JDro04, m.misiorny, Prashant Mishra, Tenzin, user272735
2	Asignaciones modelo y lenguaje.	m.misiorny, user272735
3	Colecciones y registros	J. Chomel
4	Cursores	dipdapdop, J. Chomel, Jucan
5	Declaración IF-THEN-ELSE	massko
6	Funciones	JDro04, Jon Clements, user3216906
7	Gatillos	Harjot, jiri.hofman
8	Lazo	m.misiorny, massko
9	Manejo de excepciones	Ice, J. Chomel, jiri.hofman, Tony Andrews, Zug Zwang
10	Paquetes	JDro04, jiri.hofman, StewS2, Tenzin
11	Procedimiento PLSQL	Dinidu, Doruk, Harjot, JDro04, Kekar, William Robertson
12	Recoger a granel	Prashant Mishra
13	Tipos de objeto	HepC