Escuela Técnica Superior
de Ingeniería Informática
Departamento de Lenguajes
y Sistemas Informáticos

Funciones, procedimientos, secuencias y cursores en Oracle

Grupo de Ingeniería del Software y Bases de Datos
Departamento de Lenguajes y Sistemas Informáticos
Universidad de Sevilla
diciembre 2011

Escuela Técnica Superior
de Ingeniería Informática
Departamento de Lenguajes
y Sistemas Informáticos

- Objetivos de este tema
  - Conocer la definición y utilización de funciones y procedimientos en Oracle.
  - Conocer la definición y utilización de las secuencias en Oracle.
  - Conocer la definición y utilización de cursores en Oracle.

© Diseño de Amador Durán Toro, 2011

Escuela Técnica Superior
de Ingeniería Informática
Departamento de Lenguajes
y Sistemas Informáticos

Procedimientos y funciones

- Oracle permite acceder y manipular información de la base de datos definiendo objetos procedurales (subprogramas) que se almacenan en la base de datos. Estos objetos procedurales son unidades de programa PL/SQL: **Funciones** y **Procedimientos almacenados.**

- Los procedimientos o funciones son **bloques PL/SQL** con nombre, que pueden recibir parámetros y pueden ser invocados desde distintos entornos: SQL*PLUS, Oracle*Forms, desde otros procedimientos y funciones y desde otras herramientas Oracle y aplicaciones.

- Los procedimientos y funciones llevan a cabo tareas específicas, y su mayor diferencia radica en que las funciones devuelven un valor.

Escuela Técnica Superior
de Ingeniería Informática
Departamento de Lenguajes
y Sistemas Informáticos

Procedimientos y funciones

- Sintaxis Procedimientos

CREATE [OR REPLACE} PROCEDURE [esquema].nombre-procedimiento
(nombre-parámetro {IN | OUT | IN OUT} tipo de dato, ..) {IS | AS}
Declaración de variables;
Declaración de constantes;
Declaración de cursores;
BEGIN
Cuerpo del subprograma PL/SQL;
EXCEPTION
Bloque de excepciones PL/SQL;
END;

Procedimientos y funciones

- Sintaxis Funciones

CREATE [OR REPLACE] FUNCTION [esquema].nombre-función
(nombre-parámetro {IN | OUT | IN OUT} tipo-de-dato, ...)
RETURN tipo-de-dato {IS | AS}
Declaración de variables;
Declaración de constantes;
Declaración de cursores;
BEGIN
Cuerpo del subprograma PL/SQL;
EXCEPTION
Bloque de excepciones PL/SQL;
END;

Procedimientos y Funciones

Descripción de la sintaxis:

- **Nombre-parámetro**: es el nombre que queramos dar al parámetro. Podemos utilizar múltiples parámetros. En caso de no necesitarlos, podemos omitir los paréntesis.
- **IN**: especifica que el parámetro es de entrada y que por tanto dicho parámetro tiene que tener un valor en el momento de llamar a la función o procedimiento. Si no se especifica nada, los parámetros son por defecto de tipo entrada.
- **OUT**: especifica que se trata de un parámetro de salida. Son parámetros cuyo valor es devuelto después de la ejecución el procedimiento al bloque PL/SQL que lo llamó. Las funciones PLSQL no admiten parámetros de salida.
- **IN OUT**: Son parámetros de entrada y salida a la vez.
- **Tipo-de-dato**: Indica el tipo de dato PLSQL que corresponde al parámetro (NUMBER, VARCHAR2, etc).

Procedimientos y funciones

- Ejemplo de creación de un procedimiento

```
CREATE OR REPLACE PROCEDURE contratar_empleado
        (w_codigo_emp    IN    emp.codigo_emp%TYPE,
         w_depart        IN    emp.cod_depart%TYPE,
         w_fecha_alta    IN    emp.fecha_alta%TYPE)
        IS
BEGIN
        INSERT INTO emp(código_emp, fecha_alta, cod_depart)
        VALUES (w_código_emp, w_fecha_alta, w_depart);
END contratar_empleado;
```

En este procedimiento se ha definido el tipo de dato de los parámetros de entrada como del mismo tipo que los campos de la tabla “emp” , es decir:

nombreParametro IN nombreTabla.nombreColumna**%TYPE**.

Sería equivalente a poner:

```
w_codigo_emp    number,
w_depart        varchar..
```


Escuela Técnica Superior de Ingeniería Informática
Departamento de Lenguajes y Sistemas Informáticos

Procedimientos y funciones

- Ejemplo de creación de una función

```
CREATE OR REPLACE FUNCTION obtener_salario
        (w_código_emp IN emp.código_emp%TYPE)
        RETURN NUMBER
        IS      w_salario emp.salario_emp%TYPE;
BEGIN
        SELECT  salario_emp     INTO    w_salario
        FROM    emp
        WHERE   código _emp     = w_código_emp;
        RETURN  w_salario;
END obtener_salario;
```

- Cada función debe devolver un valor del tipo especificado utilizando la sentencia RETURN.

Escuela Técnica Superior
de Ingeniería Informática
Departamento de Lenguajes
y Sistemas Informáticos

Procedimientos y funciones

- Cuando se crea un procedimiento o función, Oracle automáticamente compila el código fuente, guarda el código objeto en un área compartida de la SGA (System Global Area) y almacena tanto el código fuente como el código objeto en catálogos del diccionario de datos.

- El código objeto permanece en la SGA, por tanto, los procedimientos o funciones se ejecutan más rápidamente y lo pueden compartir muchos usuarios. Cuando es necesario liberar áreas de la SGA, Oracle aplica el algoritmo 'menos-usado-recientemente'. Si en un momento determinado se libera el área SQL de un procedimiento o función, la próxima vez que se ejecute se vuelve a cargar el código objeto, que está almacenado en catálogo, en la SGA

Escuela Técnica Superior
de Ingeniería Informática
Departamento de Lenguajes
y Sistemas Informáticos

Procedimientos y funciones

- Llamadas a procedimientos
- **Desde otro procedimiento, función y triggers**

```
CREATE PROCEDURE proceso ...                    IS          ...
BEGIN     ...
          /* llamada al procedimiento contratar_empleado  */
          contratar_empleado (2645, 'Contabilidad','19/12/1999');

END;
```

- **Herramientas de desarrollo de aplicaciones de Oracle:**

  SQL*Plus, SQL*Dba, SQL*Forms, SQL*Menu, SQL*ReportWriter, etc.

```
EXECUTE contratar_empleado (2645, 'Contabilidad','19/12/1999');
```

Procedimientos y funciones

- Llamadas a funciones
- **Desde otro procedimiento, función y triggers**

```
CREATE PROCEDURE proceso ...                    IS          ...
BEGIN      ...
           /* llamada a la función obtener_salario */
           w_sal :=obtener_salario (w_código);
END;
```

- **Desde un bloque anónimo**

```
BEGIN
      DBMS_OUTPUT.PUT_LINE('Salario cod_emp 1  '||obtener_salario(1));
END;
```

- **Desde una instrucción SQL**

```
SELECT cod_emp,  nom_emp,   obtener_salario(cod_emp)
FROM emp;
```


Procedimientos y funciones

- Documentación procedimientos y funciones
- Para obtener los nombres de todos los procedimientos y funciones se puede consultar la VISTA **USER_OBJECTS**

```
SELECT object_name, object_type  FROM USER_OBJECTS
WHERE object_type IN ('PROCEDURE' , 'FUNCTION');
```

- Para obtener el texto de un procedimiento o función almacenado se puede consultar la VISTA **USER_SOURCE**

```
SELECT text   FROM USER_SOURCE
WHERE type = 'PROCEDURE'
AND name = 'CONTRATAR_EMPLEADO'
```

Procedimientos y funciones

- Depuración de procedimientos y funciones
- Para visualizar los errores de compilación se puede consultar la VISTA USER_ERRORS o el comando **SHOW ERRORS.**
- Se pueden visualizar valores o mensajes desde un procedimiento o función, invocando al package standard **DBMS_OUPUT**.

Procedimiento	**Descripción**
DBMS_OUTPUT.PUT	Añade texto a la línea actual
DBMS_OUTPUT.NEW_LINE	Marca un final de línea
DBMS_OUTPUT.PUT_LINE	Combina PUT y NEW_LINE

Es necesario activar **SERVEROUTPUT** (SET SERVEROUTPUT ON) para ver las salidas desde procedimientos o funciones almacenados

Secuencias

- Las secuencias (sequences) son objetos que facilitan la generación automática de series numéricas.
- Los usos más frecuentes de las secuencias, son:
  - La generación automática de claves primarias
  - Coordinar las claves de múltiples filas o tablas.
- Las secuencias son independientes de las tablas; por tanto, una misma secuencia se puede usar para generar valores de columnas numéricas de una o más tablas.

Escuela Técnica Superior
de Ingeniería Informática
Departamento de Lenguajes
y Sistemas Informáticos

Secuencias

```
CREATE SEQUENCE   nombre_secuencia
        [INCREMENT  BY          n]
        [START      WITH        n]
        [MAXVALUE               n]
        ...;
```

- Para referenciar al número actual de una secuencia:
  nombre_secuencia**.Currval**
- Para generar el siguiente número de una secuencia:
  nombre_secuencia**.Nextval**
- Los usos posibles de Nextval y Currval son:
  Clausula “Values” del comando Insert
  Lista “Select” del comando Select
  Clausula “Set” del comando Update

Escuela Técnica Superior
de Ingeniería Informática
Departamento de Lenguajes
y Sistemas Informáticos

Secuencias

- Ejemplo de creación de secuencia

  CREATE SEQUENCE sec_emp;

- **Obtener los valores de la clave primaria** de la tabla Empleado y **coordinar los valores de las claves** primarias de las tabla Empleado y la tabla Salario.
  - Insertar fila en la tabla Empleado

    INSERT INTO Empleado (cod_emp,codemp...)
    VALUES (**sec_emp.NEXTVAL**, ‘emp01’...);

  - Insertar empleado en la tabla Salarios

    INSERT INTO Salario (cod_emp,mes...)
    VALUES (**sec_emp.CURRVAL**,..)

Escuela Técnica Superior
de Ingeniería Informática
Departamento de Lenguajes
y Sistemas Informáticos

Cursores

- Los cursores permiten realizar operaciones sobre los registros devueltos por una sentencia Select. La utilización de cursores es necesaria cuando:
  - Se necesita tratamiento fila a fila
  - En sentencias SELECT que devuelven más de una fila

- **Operaciones con cursores**
  - **Declare**
    - Se declara el cursor asignándole nombre y asociándole a una consulta.
  - **Open**
    - Abre el cursor y lo inicializa para que devuelva las filas.
    - Ejecuta la consulta asociada al cursor.
  - **Fetch**
    - Lee los datos del cursor con la sentencia FETCH.
    - Devuelve la siguiente fila en el conjunto activo.
    - Los datos devueltos se almacenan en variables de control o en un registro. FETCH ... INTO ...
  - **Close**
    - Desactiva el cursor y libera los recursos. CLOSE cursor_1;

Escuela Técnica Superior
de Ingeniería Informática
Departamento de Lenguajes
y Sistemas Informáticos

Cursores

- Ejemplo uso de cursor

```
DECLARE
        CURSOR cursor_1 IS
                SELECT nombre, número, salario
                FROM emp        ORDER BY salario;
        w_nombre                emp.nombre%TYPE;
        w_número                emp.número%TYPE;
        w_salar io              emp.salario%TYPE;
BEGIN
        OPEN cursor_1;
        FETCH cursor_1 INTO w_nombre, w_número, w_salario;
        ...
        CLOSE cursor_1;
END;
```

Cursores

- Ejemplo uso de cursor

```
DECLARE
        CURSOR cursor_1 IS
                SELECT nombre, número, salario
                FROM emp        ORDER BY salario;
        w_registro              cursor_1%ROWTYPE;

BEGIN
        OPEN cursor_1;
        FETCH cursor_1 INTO w_registro;
        ...
        CLOSE cursor_1;
END;
```

El registro ha sido declarado basado en el cursor:
w_registro <nombre_cursor>%ROWTYPE

Cursores

- Bucle de cursor FOR
  - De forma implícita declara una variable REGISTRO de tipo ROWTYPE, abre el cursor y de forma repetitiva realiza el FETCH de las filas sobre la variable registro. Por último cierra el cursor cuando todas las filas han sido procesadas

```
DECLARE
        CURSOR  c1  IS
        SELECT  empno, ename  FROM emp;
BEGIN
        FOR  c1rec  IN  c1  LOOP
        /* De forma implícita hace OPEN y FETCH del cursor*/
        ...
        END LOOP;
END;
```

Escuela Técnica Superior de Ingeniería Informática
Departamento de Lenguajes y Sistemas Informáticos

Cursores

- Atributos del cursor
  - Cada cursor definido tiene cuatro atributos a los que se puede acceder para conocer el estado del cursor.
    - **%FOUND** Devuelve true si el último FETCH evaluado devuelve la siguiente fila.
    - **%NOTFOUND** Devuelve true si el último FETCH evaluado no devuelve ninguna fila.
    - **%ROWCOUNT** Contador inicialmente a cero, que se incrementa en uno tras el FETCH de cada fila.
    - **%ISOPEN** Devuelve true si el cursor especificado está abierto.

Escuela Técnica Superior de Ingeniería Informática
Departamento de Lenguajes y Sistemas Informáticos

Cursores

- Ejemplo uso de atributos del cursor

```
DECLARE
        CURSOR cursor_1   IS
                          SELECT nombre, salario FROM empleados;
        registro          cursor_1%ROWTYPE

BEGIN
        IF NOT (cursor_1%ISOPEN) THEN OPEN cursor_1;
        ENDIF;

        LOOP
                FETCH cursor_1 INTO registro;
                EXIT WHEN cursor_1%NOTFOUND;
                ...
        END LOOP;
        CLOSE cursor_1;
        END;
```

Escuela Técnica Superior de Ingeniería Informática
Departamento de Lenguajes y Sistemas Informáticos

Ejercicio 1

- Añadir registros a la tabla de empleados, utilizando una **secuencia** que genere el código de empleado.

```
CREATE TABLE empleados
        (cod_emp        integer,
        nom_emp         char(10)        not null,
        salario         number(9,2)     DEFAULT 100000,
        fecha_nac       date            DEFAULT SYSDATE,
        comision        number(3,2)
                CHECK (comision>=0 AND comision <=1),
        cod_jefe                integer,
        PRIMARY KEY (cod_emp),
        FOREIGN KEY (cod_jefe) REFERENCES empleados
                                ON DELETE CASCADE);
```


Escuela Técnica Superior de Ingeniería Informática
Departamento de Lenguajes y Sistemas Informáticos

Ejercicio 2

- Crear un procedimiento para insertar un nuevo empleado en la tabla Empleados.
  - Los argumentos del procedimiento son los valores de los atributos del empleado.
  - Utilizar una secuencia para obtener el valor de la clave primaria del nuevo empleado.

```
CREATE OR REPLACE PROCEDURE proc_1
 (w_nombre VarChar, w_sal number, w_com number, w_jefe Varchar) IS
BEGIN
INSERT INTO empleados(cod_emp, nom_emp, salario, comision, cod_jefe)
VALUES (secl.nextval, w_nombre, w_sal, w_com, w_jefe);
END;
/

EXECUTE proc_1('Manuel', 2000, 0.25, null);
```

Escuela Técnica Superior
de Ingeniería Informática
Departamento de Lenguajes
y Sistemas Informáticos

Ejercicio 3

- Crear una función para calcular el sueldo total de un empleado pasado como parámetro. Tenga en cuenta que hay que añadir la comisión (que es un porcentaje adicional del salario) al salario.

- Es posible también llamar a la función desde un bloque PL/SQL: Genere un bloque PL/SQL anónimo (un procedimiento **BEGIN ….. END** que no es necesario nominar) y pruébelo con una instrucción DBMS_OUTPUT

Escuela Técnica Superior
de Ingeniería Informática
Departamento de Lenguajes
y Sistemas Informáticos

Ejercicio 4

- Obtener los tres empleados con más subordinados

Script ejercicios 1,2 y3

```
DROP TABLE empleados;
CREATE TABLE empleados
  ( cod_emp integer,
    nom_emp char(10) not null,
    salario number(9,2) DEFAULT 100000,
    fecha_nac date DEFAULT SYSDATE,
    comision number(5,2), CHECK (comision>=0 AND comision<=1),
    cod_jefe integer,
    PRIMARY KEY (cod_emp),
    FOREIGN KEY (cod_jefe) REFERENCES empleados ON DELETE CASCADE);

DROP SEQUENCE sec_emp;
CREATE SEQUENCE sec_emp INCREMENT BY 1 START WITH 1;

/* Procedimiento */
CREATE OR REPLACE PROCEDURE contratar_empleado
(w_nom_emp IN empleados.nom_emp%TYPE,
 w_salario IN empleados.salario%TYPE,
 w_comision IN empleados.comision%TYPE,
 w_cod_jefe IN empleados.cod_jefe%TYPE) IS
BEGIN
INSERT  INTO empleados  (cod_emp,nom_emp, salario,comision,cod_jefe)
       VALUES         (sec_emp.nextval, w_nom_emp, w_salario, w_comision,
w_cod_jefe);
COMMIT WORK;
END contratar_empleado;
/
```


Escuela Técnica Superior
de Ingeniería Informática
Departamento de Lenguajes
y Sistemas Informáticos

1. Procedimientos y funciones
   1.1 Definición
   1.2 Llamadas
   1.3 Documentación
   1.4 Depuración
2. Secuencias
   2.1 Definición
3. Cursores
   3.1 Bucle FOR
   3.2 Atributos
4. Ejercicios
5. Scripts

Script ejercicios 1,2 y3

```
EXECUTE contratar_empleado('Primero',1000.00,.07,null);
EXECUTE contratar_empleado('Segundo',2000,.10,1);
EXECUTE contratar_empleado('Tercero',2300.25,.15,2);

--SELECT * FROM empleados;

CREATE OR REPLACE FUNCTION obtener_salario(w_cod_emp IN
empleados.cod_emp%TYPE)
RETURN NUMBER IS w_salario_bruto empleados.salario%TYPE;
BEGIN
  SELECT salario*(1+comision) INTO w_salario_bruto FROM empleados
       WHERE cod_emp = w_cod_emp;
RETURN (w_salario_bruto);
END obtener_salario;
/
/*  Prueba de función desde una instrucción SQL */
SELECT cod_emp,nom_emp,salario,comision,obtener_salario(cod_emp) FROM
empleados;

/*  Prueba de función desde un bloque */
SET serveroutput ON;
BEGIN
DBMS_OUTPUT.PUT_LINE('Probando el salario de COD_EMP 1  '|| >>>>
'||obtener_salario(1));
END;
/
```

Script ejercicios 4

```
DROP TABLE empleados;
CREATE TABLE      empleados (dni char(4) PRIMARY KEY,
nomemp varchar2(15),
cojefe char(4),
                        FOREIGN KEY (cojefe) references empleados);

--
-- Inserta datos de ejemplo en la tabla
--
INSERT INTO empleados VALUES ('D1','Director',null);
INSERT INTO empleados VALUES ('D2','D.Comercial','D1');
INSERT INTO empleados VALUES ('D3','D.Producción','D1');
INSERT INTO empleados VALUES ('D4','Jefe Ventas','D2');
INSERT INTO empleados VALUES ('D5','Jefe Marketing','D2');
INSERT INTO empleados VALUES ('D6','Vendedor 1','D4');
INSERT INTO empleados VALUES ('D7','Vendedor 2','D4');
INSERT INTO empleados VALUES ('D8','Vendedor 3','D4');
INSERT INTO empleados VALUES ('D9','Vendedor 4','D4');
INSERT INTO empleados VALUES ('D10','Obrero 1','D3');
INSERT INTO empleados VALUES ('D11','Obrero 2','D3');
INSERT INTO empleados VALUES ('D12','Obrero 3','D3');
INSERT INTO empleados VALUES ('D13','Secretario','D5');
```


Script ejercicios 4

```
--
-- Procedimientos anónimos para obtener los tres empleados con más subordinados con bucle for
--

SET SERVEROUTPUT ON
DECLARE
        CURSOR c IS
                SELECT cojefe,count(*) AS cuenta FROM empleados
                GROUP BY cojefe ORDER BY 2 DESC;

BEGIN
        DBMS_OUTPUT.PUT_LINE('Prueba cursor (3 superjefes) bucle FOR');
        FOR fila IN c LOOP
                EXIT WHEN C%ROWCOUNT >3;
                DBMS_OUTPUT.PUT_LINE(fila.cojefe||' '||fila.cuenta);
        END LOOP;
END;
/
```

Script ejercicios 4

```
--
-- Procedimientos anónimos para obtener los tres empleados con más subordinados
con bucle normal

--
DECLARE
        wjefe CHAR(4);
        wcount INTEGER;
        CURSOR c IS SELECT cojefe,count(*) AS cuenta FROM empleados
                        GROUP BY cojefe ORDER BY 2 DESC;
        fila c%ROWTYPE;
BEGIN
        DBMS_OUTPUT.PUT_LINE('Prueba de cursor (3 superjefes) con
        Open/Fetch/Close ** BUCLE NORMAL');
        OPEN c;
        LOOP
                FETCH c INTO fila;
                EXIT WHEN C%NOTFOUND OR c%ROWCOUNT >3;
                DBMS_OUTPUT.PUT_LINE(fila.cojefe||' '||fila.cuenta);
        END LOOP;
        CLOSE c;
END;
/
```


Script ejercicios 4

```
--
-- Procedimientos anónimos para obtener los tres empleados con más subordinados
con bucle while

DECLARE
        wjefe CHAR(4);
        wcount INTEGER;
        CURSOR c IS
                SELECT cojefe,count(*) AS cuenta FROM empleados
                GROUP BY cojefe ORDER BY 2 DESC;
        fila c%ROWTYPE;
BEGIN
        DBMS_OUTPUT.PUT_LINE('Prueba de cursor (3 superjefes) con
        Open/Fetch/Close ** BUCLE WHILE');
        OPEN c;
        WHILE c%ROWCOUNT<3 LOOP
                FETCH c INTO fila;
                EXIT WHEN C%NOTFOUND;
                DBMS_OUTPUT.PUT_LINE(fila.cojefe||' '||fila.cuenta);
        END LOOP;
CLOSE c;
END;
/

 --  Para no crear ningún objeto en la BD
ROLLBACK WORK;
```