UA4B. PL/SQL

4.16.1. Subprogramas

Los subprogramas son bloques nombrados a los cuales les podemos pasar parámetros y los podemos invocar. Además, los subprogramas pueden estar almacenados en la base de datos o estar encerrados en otros bloques. Si el programa está almacenado en la base de datos, podremos invocarlo si tenemos permisos suficientes y si está encerrado en otro bloque lo podremos invocar si tenemos visibilidad sobre el mismo.

Hay dos clases de subprogramas: las funciones y los procedimientos. Las funciones devuelven un valor y los procedimientos no.

Para declarar un subprograma utilizamos la siguiente sintaxis:

Sintaxis para la declaración de sub

Funciones.	Procedimien

`FUNCTION nombre [(parametro [, parametro] ...)]`
`RETURN tipo_dato IS`
`[declaraciones_locales]`
`BEGIN`
`sentencias_ejecutables`
`[EXCEPTION`
`manejadores_de_excepciones]`
`END [nombre];`
	`PROCEDURE nom`
	`[declara`
	`BEGIN`
	`sentencias_ej`
	`[EXCEPTION`
	`manejadores_d`
	`END [nombre];`

Donde:

```
parametro := nombre_parametro [IN|OUT|IN OUT] tipo_dato [{:=|DEFAULT} e
```

Algunas consideraciones que debes tener en cuenta son las siguientes:

- No podemos imponer una restricción `NOT NULL` a un parámetro.
- No podemos especificar una restricción del tipo:

```
PROCEDURE KK(a NUMBER(10)) IS ...          --ilegal
```

- Una función siempre debe acabar con la sentencia `RETURN`.

Podemos definir subprogramas al final de la parte declarativa de cualquier bloque. En Oracle, cualquier identificador debe estar declarado antes de usarse, y eso mismo pasa con los subprogramas, por lo que deberemos declararlos antes de usarlos.

```
DECLARE
hijos NUMBER;
FUNCTION hijos_familia( id_familia NUMBER )
     RETURN NUMBER IS
     hijos NUMBER;
BEGIN
     SELECT COUNT(*) INTO hijos FROM agentes
          WHERE familia = id_familia;
     RETURN hijos;
END hijos_familia;
BEGIN
...
END;
```

Si quisiéramos definir subprogramas en orden alfabético o lógico, o necesitamos definir subprogramas mutuamente recursivos (uno llama a otro, y éste a su vez llama al

anterior), deberemos usar la definición hacia delante, para evitar errores de compilación.

```
DECLARE
     PROCEDURE calculo(...);          --declaración hacia delante
     --Definimos subprogramas agrupados lógicamente
     PROCEDURE inicio(...) IS
     BEGIN
          ...
          calculo(...);
          ...
     END;
     ...
BEGIN
     ...
END;
```