1. Introducció a la programació en PL/SQL

UF3 - Llenguatges SQL: extensió procedimental

1.1. Introducción

El **lenguaje PL/SQL**, incorpora todas las características propias de los lenguajes de tercera generación, es decir, manejo de variables, estructura modular (procedimientos y funciones), estructuras de control (condiciones, bucles, etc.), control de excepciones, etc.

El lenguaje PL/SQL es un lenguaje procedimental diseñado por Oracle para trabajar con bases de datos.

Algunas ventajas de PL/SQL son: .

- Mejor rendimiento.
- Desarrollo de programas modulares.
- Es portable: es un lenguaje nativo de Oracle Server y por tanto los programas se pueden ejecutar en cualquier sistema operativo y/o plataforma.
- Permite declarar variables.
- Se puede programar con estructuras de control de lenguaje procedural.
- Se puede manejar errores.
- Fácil mantenimiento.
- Mayor integridad y seguridad de los datos.
- Mayor claridad del código.

Delimitadores de código: son símbolos simples o compuestos.

Símbolos simples	
Símbolo	Descripción
+	Operador suma
-	Operador resta
*	Operador multiplicación
/	Operador división
=	Operador relacional
@	Indicador acceso remoto
;	Terminador de sentencias

Símbolos compuestos	
Símbolo	Descripción
<>	Operador relacional
!=	Operador relacional
||	Operador concatenación
--	Indicador de comentarios de una línea
/*	Delimitador del principio del comentario
*/	Delimitador del final del comentario
:=	Operador de asignación

Literales: es un valor numérico, de carácter, de cadena o booleano. Y se ha de tener en cuenta que:

- Los literales de fecha y carácter han de ir entre comillas simples.
- Los números pueden ser valores simples o notaciones complejas. Por ejemplo:

```
v_nombre := 'Maria';
```

Comentarios: para incluir comentarios en el código se usa dos guiones (--) para comentar una línea, o bien, /* y */ para comentar más de una línea.

```
VARIABLE v_salario NUMBER  BEGIN
                           Variable no PL/SQL
 SELECT salary
 INTO :v_salario /* hacer referencia a una variable no PL/SQL */
 FROM employees

 WHERE employee_id= 100; /* imprimir una variable no PL/SQL */
 END;
 /
```

Funciones SQL en PL/SQL

Las funciones que estudiamos en SQL, se usan también en PL/SQL, como son:

- Conversión de tipos de dato: `TO_CHAR`, `TO_DATE` y `TO_NUMBER`.
- Fecha
- Registro de hora
- etc

Las que no se pueden usar en PL/SQL, pues no son sentencias procedurales son el `DECODE` y las `funciones de grupo`.

Common Functions

- LENGTH(string): Returns the length of the provided string
- INSTR(string, substring, [start_position], [occurrence]): Returns the position of the substring within the specified string.
- TO_CHAR(input_value, [fmt_mask], [nls_param]): Converts a date or a number to a string
- TO_DATE(charvalue, [fmt_mask], [nls_date_lang]): Converts a string to a date value.
- TO_NUMBER(input_value, [fmt_mask], [nls_param]): Converts a string value to a number.
- ADD_MONTHS(input_date, num_months): Adds a number of months to a specified date.
- SYSDATE: Returns the current date, including time.
- CEIL(input_val): Returns the smallest integer greater than the provided number.
- FLOOR(input_val): Returns the largest integer less than the provided number.
- ROUND(input_val, round_to): Rounds a number to a specified number of decimal places.
- TRUNC(input_value, dec_or_fmt): Truncates a number or date to a number of decimals or format.
- REPLACE(whole_string, string_to_replace, [replacement_string]): Replaces one string inside the whole string with another string.
- SUBSTR(string, start_position, [length]): Returns part of a value, based on a position and length.

1.2. Bloque PL/SQL

```
[DECLARE
    <declaraciones de variables, cursores,
    excepciones definidas por el usuario>]
 BEGIN
       <cuerpo: sentencias SQL y PL/SQL>
[EXCEPTION
    <gestiónde las excepciones
    (control de errores)>]
 END;
```

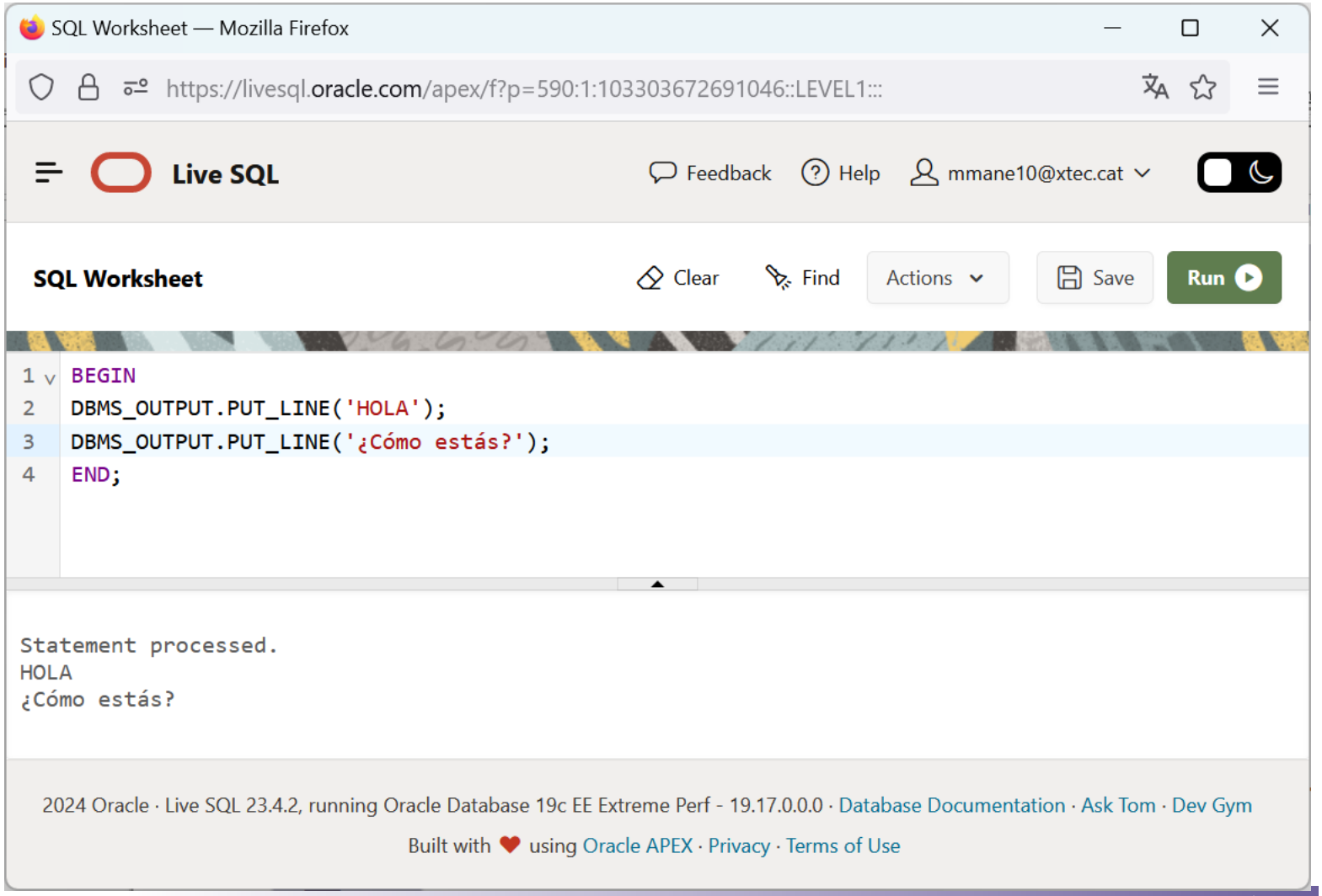
} Opcional

} Obligatorio

} Opcional

} Obligatorio

1.2.1. Ejecución de un programa PL/SQL



1.2.2. Tipos de bloques

Anónimo

Son sentencias SQL. Se ejecutan una sola vez, es decir, es semejante a un script. La sintaxis es la siguiente:

```
[DECLARE]
 BEGIN
 …
[EXCEPTION]
 END;
```

Procedimiento

En este bloque se almacena en el servidor y se ejecuta cada vez que se invoque. La sintaxis es la siguiente:

```
PROCEDURE nombre_procedimiento
IS/AS
  Variable
  BEGIN
  …
  [EXCEPTION]
  END nombre_procedimiento;
```

Función

Es semejante a un procedimiento, con la diferencia de que se le pasa un parámetro (como mínimo) y devuelve un valor. La sintaxis es la siguiente:

```
FUNCTION nombre_funcion
RETURN tipo_dato_a_devolver
IS/AS
 BEGIN
 …
RETURN valor_a_devolver;
 [EXCEPTION]
END nombre_funcion;
```

1.3 Identificadores, Variables y Constantes

1.3.1 Identificadores

Los **identificadores** se usan para nombrar los objetos que intervienen en un programa y son:

- variables, constantes, cursores, excepciones, procedimientos, funciones, etiquetas, etc.

Aspectos a tener en cuenta con los identificadores:

✓Pueden contener hasta 30 caracteres y siempre ha de comenzar por un carácter. Pueden contener números, signos de dólar, subrayados y almohadillas. No pueden contener caracteres como guiones, barras inclinadas y espacios.

✓No deberían tener el mismo nombre que una columna de la tabla de base de datos o tabla.

✓No debería ser palabras reservadas.

1.3.2. Variables

En PL/SQL se pueden declarar **variables** y se usan entre otras para almacenar cosas temporalmente un dato y manipular valores previamente almacenados.
La sintaxis para declarar una variable, es:

```
identificador [CONSTANT] tipo_dato [NOT NULL]
    [{:=|DEFAULT} valor];
```

donde:

`identificador` ⇨ es el nombre de la variable.

`CONSTANT` ⇨ restringe la variable para que no se pueda modificar el valor. Las constantes se han de inicializar.

`tipo_dato` ⇨ tipo de dato de la variable (escalar, compuesto, LOB, de referencia).

`NOT NULL` ⇨ restringe la variable para que obligatoriamente tenga un valor. Las variables NOT NULL se han de inicializar.

`valor` ⇨ es cualquier expresión PL/SQL que pueda ser una expresión literal, otra variable o una expresión que implique el uso de operadores y funciones.

Ejemplos variables

```
DECLARE
  var1    NUMBER(5);
  radio   NUMBER(3):=2;
  fecha   DATE;
  cont    NUMBER(2) NOT NULL:= 10;
  v_comm  CONSTANT NUMBER:=200;
  pobla   VARCHAR2(15):='Cornella';
```

Reglas de nomenclatura

- Dos variables pueden tener el mismo nombres si están en diferentes bloques.
- El nombre o identificador de las variables no deben coincidir con el nombre de las columnas de la tabla o el de las tablas.

Por ejemplo:

```
DECLARE  /*  Ejemplo  ERRONEO,  se usa
como  identificador  de  una  variable  el
mismo identificador que un campo de una
tabla */
  employee_id NUMBER(2);
 BEGIN
  SELECT employee_id
  INTO employee_id  FROM
  employees
   WHERE UPPER(last_name) = 'KING';
END;
/
```

```
DECLARE /* Ejemplo CORRECTO */
 v_employee_id NUMBER(2);  BEGIN
 SELECT employee_id
  INTO v_employee_id
 FROM employees
 WHERE UPPER(last_name) = 'KING';
 END;
  /
```

Inicializar variables

Para inicializar variables se usa la asignación :=. La sintaxis es:

```
identificador := expr;
```

Ejemplo:

```
v_nombre := 'Manel';
v_fecha := '10-ENE-2010';
v_sal := 1000;
```

Palabras reservadas

- la palabra reservada DEFAULT
- la restricción NOT NULL

1.3.2.1.Tipos de datos (para declarar variables):

ESCALAR ⇒ Contienen un valor único. Son los tipos más comunes que podemos encontrar

en PL/SQL, como son VARCHAR2, NUMBER, DATE, CHAR, BOOLEAN, etc.

COMPUESTA ⇒ Se usan para definir y manipular grupos de campos en bloques PL/SQL. Son las tablas PL/SQL y registros PL/SQL. Este tipo de variables la estudiaremos brevemente.

REFERENCIADA ⇒ Son las variables llamadas también punteros y se usan para designar otros artículos de programa. Este curso no vamos a estudiar dicha variable.

LOB ⇒ Contienen los valores llamados *localizadores*, que especifican la ubicación de imágenes gráficas. Este curso no vamos a estudiar dicha variable.

Tipos de datos escalares

Tipo de dato	Descripción
`CHAR [(long_max)]`	Carácter de longitud fija hasta 32.767 bytes y no se especifica la longitud, por defecto es 1.
`VARCHAR2[(long_max)]`	Carácter de longitud variable hasta 32.767 bytes y no existe un tamaño por defecto para este tipo de dato y tampoco para las constantes.
`LONG`	Caracteres de variable de longitud variable hasta 32.760 bytes.
`LONG RAW`	Datos binarios y cadenas de bytes de hasta 32.760 bytes. PL/SQL no interpreta los datos LONG RAW.
`NUMBER [(precisión, escala)]`	Número con una precisión (que va entre 1 y 38) y una escala que va entre -84 y 127.
`BINARY_INTEGER`	Enteros.
`BOOLEAN`	Tipo de dato lógico que almacena uno de los tres valores siguientes: TRUE, FALSE o NULL:
`DATE`	Fecha y hora.

Ejemplo de variables de datos escalares

```
DECLARE
  v_oficio        VARCHAR2(9);
  v_contador      BINARY_INTEGER    :=0;
  v_total         NUMBER(9,2) :=    0;
  v_fecha         DATE := SYSDATE + 9;
  v_valido        BOOLEAN NOT NULL := TRUE;
  v_comision      CONSTANT NUMBER(3) := 100;
```

Resumen de tipos de datos pl-sql-data-types

Tipo	Descripción
CHAR(n)	Almacena cadenas de caracteres de longitud fija. Se puede dar opcionalmente la longitud máxima que ha de tener la cadena. Este parámetro se ha da entre paréntesis (n).
VARCHAR2(n)	Almacena cadenas de caracteres de longitud variable, cuyo límite se debe especificar en número de bytes (n).
LONG(n)	Almacena cadenas de caracteres de longitud variable. Es parecido al VARCHAR2.
NUMBER(n,m)	Almacena datos numéricos donde *n* es el número total de dígitos y *m* es el número de decimales. Tanto *n* y *m* son opcionales. PL/SQL dispone de subtipos de NUMBER, que son: DECIMAL, NUMERIC, INTEGER, REAL, SMALLINT, etc.
BINARY_INTEGER	Es un tipo numérico entero que se almacena en memoria en formato binario, con el fin de facilitar los cálculos. Se usa para contadores, índices, etc.
PLS_INTEGER	Es un tipo nuevo y es similar a BINARY_INTEGER; aunque tiene dos ventajas: es más rápido y en caso de desbordamiento en el cálculo, se produce un error y se da la excepción correspondiente.
BOOLEAN	Almacena los valores Verdadero, Falso y Nulo. Las bases de datos no soporta este tipo.
DATE	Almacena fechas y el formato estándar es 'dd-mmm-aaaa'. También almacena la hora.
RAW(n)	Almacena datos binarios en longitud fija y se usa para almacenar cadenas de caracteres evitando las conversiones entre conjuntos de caracteres que realiza Oracle.
LONG RAW	Almacena datos binarios en longitud fija evitando conversiones entre conjuntos de caracteres.
ROWID	Almacena identificadores de fila.

1.3.2.2. Variables no PL/SQL o variables de HOST o variables de sustitución

Cabe destacar que las variables que se definen dentro de un bloque PL/SQL en la zona DECLARE, sea el tipo de bloque que sea, reciben el nombre de **Variables PL/SQL**.

Pero también, podemos tener variables que no se definen dentro de un bloque PL/SQL y reciben el nombre de **Variables no PL/SQL** o **variables de Host** o **variables de sustitución.**

Hay dos tipos de definir una variable no PL/SQL:

a) Definirla mientras se le pregunta al usuario un valor determinado a introducir:

```
ACCEPT nom_variable PROMPT 'literal'
```

b) Definirla para poder operar con ella (asignarle un valor)

```
VARIABLE nom_variable tipo_dato
```

➢ Referencia a variables no PL/SQL

Ahora nos podemos preguntar, **¿cómo referencia a variables No PL/SQL?**

Para hacer referencia a las variables que no son de tipo PL/SQL (variables HOST o variables de sustitución):

a) Si la variables se ha definido con la sintaxis (para preguntar al usuario un valor)

```
ACCEPT nom_variable PROMPT 'literal'
```

Se ha de coger el valor que contiene la variable ⇒ se ha de preceder del signo & a la variable.

b) Si la variable se ha definido con la sintaxis

```
VARIABLE <nom_variable> <tipo_variable>
```

Se le quiera asignar un valor ⇒ se ha de preceder de dos puntos (:) a la variable.

Ejemplo de variables no PL/SQL o variables ligadas

```
ACCEPT var_emp PROMPT 'Introduce el código del empleado:'
VARIABLE v_salario NUMBER
BEGIN
 SELECT salary
 INTO :v_salario /* escribe en una variable no PL/SQL */
 FROM employees
 WHERE employee_id= &var_emp; /* var_deptno es una variable de sustitución,
            que se declara en el ACCEPT cuando se le pregunta al usuario por el
           valor */
END;
/
PRINT v_salario
```

```
bloque anónimo terminado
V_SALARIO
---------
    24000
```

1.3.3. Constantes

Las **constantes** se han de declarar mediante la siguiente sintaxis:

```
<nombre_constante> CONSTANT <tipo> := <valor>
```

Por ejemplo:

```
DECLARE
    IVA CONSTANT      NUMBER := 16;
    pi CONSTANT       NUMBER(9,7):=  3.1415927;
    …
```

1.3.4. Los atributos %TYPE y %ROWTYPE

La idea de usar estos atributos es declarar variables que sean del mismo tipo que otros objetos que ya están definidos. Es decir, hereda de la variable el tipo y la longitud del objeto, pero no los atributos `NOT NULL`, ni valores por defecto.

Estos atributos son el `%TYPE` y el `%ROWTYPE`, vamos a estudiar cada uno de ellos:

%TYPE:

Este atributo declara una variable del mismo tipo que otra, o que una columna de una tabla. Por tanto, declara una variable basada en otras variables previamente declaradas, o en la definición de una columna de la base de datos.

Por ejemplo, si queremos declarar una variable llamada **var_nombre** que sea del mismo tipo que el **nombre de un empleado** (`first_name`) que tenemos en nuestra base de datos, concretamente en la tabla **empleado** (`employees`).

Tenemos la siguiente sintaxis dentro de un bloque PL/SQL:

```
DECLARE
  var_nombre              employees.first_name%TYPE;
  …
```

%ROWTYPE:

Este atributo crea una variable de registro cuyos campos se corresponden con las columnas de una tabla o vista de la base de datos.

En esta unidad didáctica estudiaremos con más detalle este atributo.

Por ejemplo, si declaras una variable que tenga la estructura de la tabla **empleado** (EMPLOYEES), seria:

```
DECLARE
  var_empleado   employees%ROWTYPE;
  …
```

1.4. Tipos de datos compuestos

Hasta ahora hemos estudiado las variables de tipo escalar, y ahora vamos a estudiar brevemente las variables compuestas, que son los **registros PL/SQL** y las **tablas PL/SQL**.

Registros PL/SQL

Son un grupo de elementos de datos relacionados en campos, cada uno con su propio nombre y tipo de datos; es decir, tratan una colección de campos como una unidad lógica.

La sintaxis para crear un registro PL/SQL, es la siguiente:

```
TYPE nombre_registro IS RECORD  (declaración_campo
    [,declaración_campo])
```

Es importante destacar que los registros no son los mismo que las filas de una tabla de una base de datos.

Ejemplo de registros PL/SQL:

•Creación de un registro usando tipos de variables escalares:

```
TYPE tipo_registro_empleado IS RECORD
  (employee_id       NUMBER(4),
   first_name        VARCHAR2(10),
   job_id            VARCHAR2(9),
   salary            NUMBER(7,2));
registro_empleado      tipo_registro_empleado;
   …
```

•Creación de un registro usando el atributo %TYPE:

```
TYPE tipo_registro_empleado IS RECORD
  (employee_id       employees.employee_id%TYPE,
   first_name        employees.first_name%TYPE,
   job_id            employees.job_id%TYPE,
   salary            employees.salary%TYPE);
registro_empleado      tipo_registro_empleado;
   …
```

Campo1 (tipo de datos)	Campo2 (tipo de datos)	Campo3 (tipo de datos)	Campo4 (tipo de datos)
department_id NUMBER(4)	first_name VARCHAR2(10)	job_id VARCHAR2(9)	salary NUMBER(7,2)

Esta estructura es similar a trabajar con el atributo `%ROWTYPE`:

```
DECLARE
var_emp employees%ROWTYPE;
```

```
LOS DATOS DEL EMPLEADO QUE HA CONSULTADO SON:
CODIGO EMPLEADO: 100
NOMBRE EMPLEADO: Steven
SALARIO EMPLEADO: 24000
```

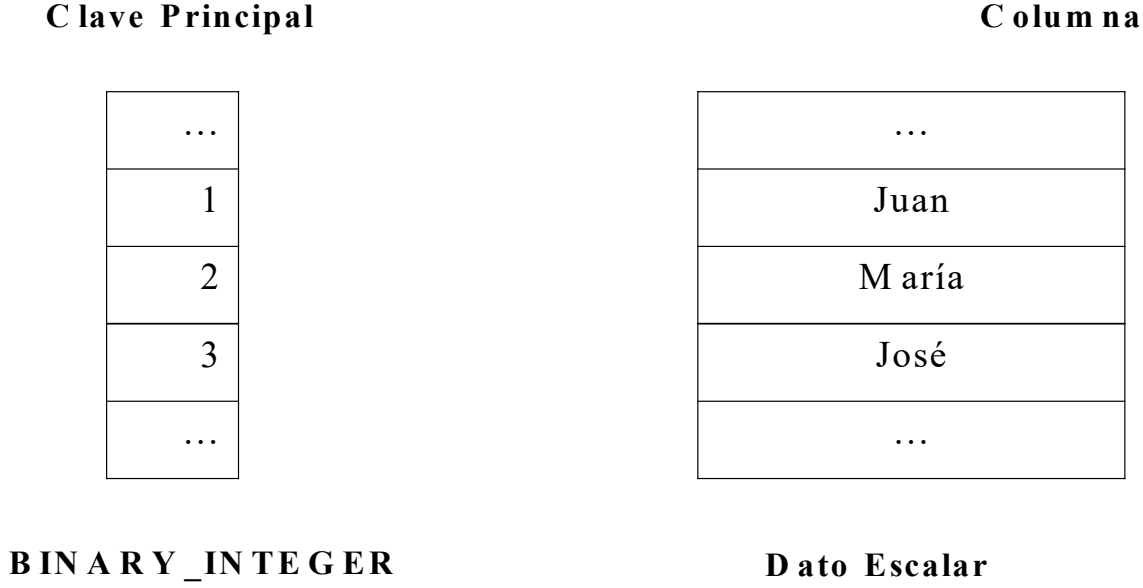


```
DECLARE
 variable employees%ROWTYPE;
BEGIN
 SELECT * INTO variable
 FROM employees
 WHERE employee_id = 100;

 DBMS_OUTPUT.PUT_LINE ('LOS DATOS DEL EMPLEADO QUE HA CONSULTADO  SON:');
 DBMS_OUTPUT.PUT_LINE ('CODIGO EMPLEADO: '||variable.employee_id);
 DBMS_OUTPUT.PUT_LINE ('NOMBRE EMPLEADO: '||variable.first_name);
 DBMS_OUTPUT.PUT_LINE ('SALARIO EMPLEADO: '||variable.salary);
 END;
 /
```

Tablas PL/SQL

Son tipos de datos compuestos y están estructurados como tablas de la base de datos, aunque no se ha de confundir con las propias tablas donde se almacenan información en la base de datos



La sintaxis para crear una tabla PL/SQL es la siguiente:

```
TYPE tipo_nombre_tabla IS TABLE OF
    {tipo_campo | variable%TYPE |
            nombretabla.campo%TYPE}[NOT NULL]
INDEX BY BINARY_INTEGER;

identificador  tipo_nombre_tabla;
```

Preguntes!!!!!