

# 1

## Introducción



ORACLE®

### Objetivos del Curso

Después de completar este curso, debería ser capaz de hacer lo siguiente:

- Identificar las extensiones de programación que PL/SQL proporciona a SQL
- Escribir código PL/SQL para interactuar con la base de datos
- Diseñar bloques anónimos de PL/SQL que ejecuten eficientemente
- Utilizar construcciones de programación PL/SQL y declaraciones de control condicional
- Manejar errores de tiempo de ejecución
- Describir los procedimientos y funciones almacenados



## Entornos de desarrollo PL/SQL

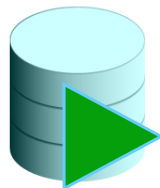
Esta configuración de curso proporciona las siguientes herramientas para desarrollar código PL/SQL:

- Oracle **SQL Developer** (utilizado en este curso)
  - Una herramienta gráfica
- Oracle **SQL\*Plus**
  - Una ventana o aplicación de línea de comandos

1 - 4

## ¿Qué es Oracle SQL Developer?


- Oracle SQL Developer es una herramienta gráfica gratuita que mejora la productividad y simplifica las tareas de desarrollo de bases de datos.
- Puede conectarse a cualquier esquema de base de datos Oracle de destino mediante la autenticación de base de datos Oracle estándar.



SQL Developer

1 - 5

## Codificación PL/SQL en SQL\*Plus



```
oracle@EDRSR9P1:~/Desktop
File Edit View Search Terminal Help
Copyright (c) 1982, 2012, Oracle. All rights reserved.

Enter user-name: ora41
Enter password:
Last Successful login time: Mon Sep 2012 21:55:44 +00:00

Connected to:
Oracle Database 12c Enterprise Edition Release 12.1.0.0.2 - 64bit Beta
With the Partitioning, OLAP, Data Mining and Real Application Testing options

SQL> set serveroutput on
SQL> create or replace procedure hello is
  2 begin
  3   dbms_output.put_line('Hello Class!');
  4 end;
  5 /

Procedure created.

SQL> execute hello
Hello Class!

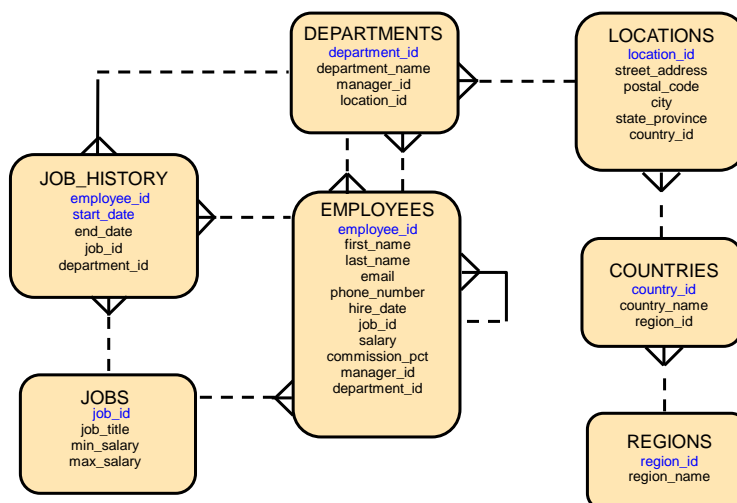
PL/SQL procedure successfully completed.

SQL>
```

- Oracle SQL\*Plus es una interfaz de línea de comandos que le permite enviar sentencias SQL y bloques PL/SQL.
- Recibe los resultados en una aplicación o una ventana de comandos.

1 - 6

## Esquema de recursos humanos (HR) para este curso



1 - 7

## Practice 1

Esta práctica abarca los siguientes temas:

- Inicio de `SQL Developer`
- Creación de una nueva conexión de base de datos
- Visualización de las tablas de esquema `HR`
- Configuración de las preferencias de `SQL Developer`

1 - 8

# 2

## Introducción a PLSQL



ORACLE®

## Objetivos

Después de completar esta lección, usted debería ser capaz de:

- Explicar la necesidad de PL/SQL
- Explicar los beneficios de PL/SQL
- Identificar los diferentes tipos de bloques PL/SQL
- Mensajes de salida en PL/SQL

1 - 10

## Agenda

- Comprender los beneficios y la estructura de PL/SQL
- Examinar bloques PL/SQL
- Generación de mensajes de salida en PL/SQL

1 - 11

## Acerca de PL/SQL

### SQL:

- Es el lenguaje principal utilizado para acceder y modificar datos en bases de datos relacionales

### PL/SQL:

- Soporte para "Procedural Language Extension to SQL "
- Es el lenguaje de acceso a datos estándar de Oracle para bases de datos relacionales
- Integra perfectamente las construcciones procedurales con SQL



1 - 12

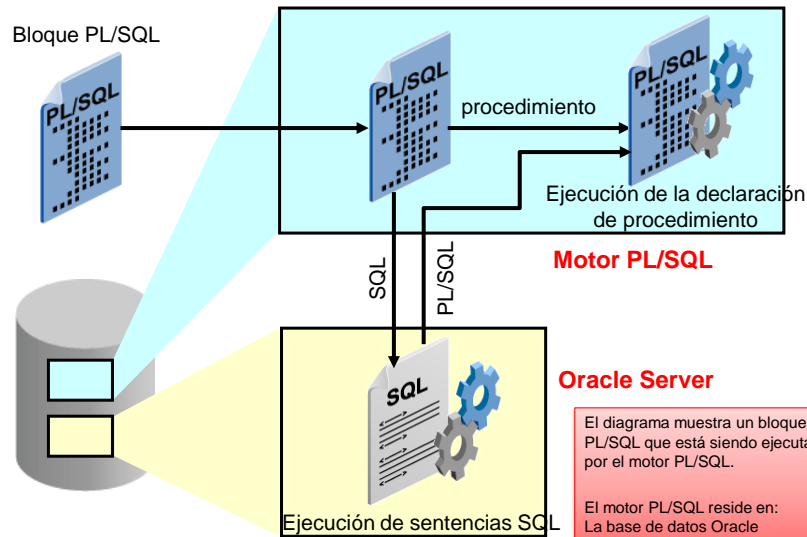
## Acerca de PL/SQL

### PL/SQL:

- Proporciona una **estructura de bloque** para unidades de código ejecutables.
- El mantenimiento del código se hace más fácil con una estructura tan bien definida.
- Proporciona construcciones procedimentales tales como:
  - Variables, constantes y tipos de datos
  - Estructuras de control como sentencias condicionales y bucles
  - Unidades de programa reutilizables que se escriben una vez y se ejecutan muchas veces

1 - 13

## Arquitectura de ejecución de PL/SQL



1 - 14

## Arquitectura de ejecución de PL/SQL

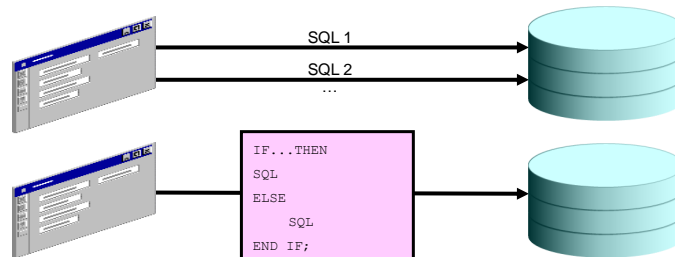
### NOTAS:

- Todas las sentencias de PL/SQL se procesan en el **Ejecutor de instrucciones de procedimiento**
- Todas las sentencias de SQL deben enviarse al **Ejecutor de sentencias de SQL** para su procesamiento por los procesos de Oracle Server.
- El entorno SQL también puede invocar el entorno PL/SQL.
  - Por ejemplo, el entorno PL/SQL se invoca cuando se utiliza una función PL/SQL en una instrucción SELECT.
- El motor PL/SQL es una máquina virtual que reside en la memoria y procesa las instrucciones de **m-code** de PL/SQL.
  - Cuando el motor PL/SQL encuentra una instrucción SQL, se hace un cambio de contexto para pasar la instrucción SQL a los procesos de Oracle Server.

1 - 15

## Ventajas de PL/SQL

- Integración de construcciones procedurales con SQL
  - Los comandos SQL le dice al servidor de la base de datos qué hacer. Sin embargo, no puede especificar cómo hacerlo
  - PL/SQL integra instrucciones de control y sentencias condicionales con SQL, lo que le da un mejor control de sus sentencias SQL y su ejecución
- Rendimiento mejorado
  - Las sentencias SQL se envían a la base de datos una a la vez. Esto resulta en muchos viajes en red y una llamada a la base de datos para cada sentencia SQL



1 - 16

## Ventajas de PL/SQL

- Desarrollo de programas modularizados
  - La unidad básica en todos los programas PL / SQL es el **bloque**.
  - Los bloques pueden ser secuenciales o pueden anidarse en otros bloques.
  - Permiten la agrupación de declaraciones relacionadas lógicamente dentro de ellos.
- Integración con herramientas de Oracle
  - El motor PL/SQL está integrado en herramientas de Oracle como **Oracle Forms y Oracle Reports**
  - Sólo las sentencias SQL se pasan a la base de datos
- Portabilidad
  - Los programas PL/SQL pueden ejecutarse en cualquier lugar donde se ejecute un servidor Oracle, independientemente del sistema operativo y la plataforma
- Manejo de excepciones
  - PL/SQL le permite manejar las excepciones de manera eficiente.
  - Puede definir bloques separados para tratar con excepciones

1 - 17



## Estructura de bloques PL/SQL

Un bloque PL / SQL consta de cuatro secciones

- **DECLARE** (opcional)
    - Variables, cursores, excepciones definidas por el usuario
  - **BEGIN** (obligatorio)
    - Sentencias SQL
    - Sentencias PL/SQL
  - **EXCEPTION** (opcional)
    - Acciones a realizar cuando se producen excepciones
  - **END;** (obligatorio)
    - Fin del bloque
    - Debe de finalizar con ;
- Las palabras clave **DECLARE**, **BEGIN** y **EXCEPTION** no terminan con un punto y coma. Sin embargo, la palabra clave **END**, todas las sentencias **SQL** y las instrucciones **PL / SQL** deben finalizarse con un punto y coma.



1 - 18

## Agenda

- Comprender los beneficios y la estructura de PL/SQL
- Examinar bloques PL/SQL
- Generación de mensajes de salida en PL/SQL

1 - 19

## Tipos de bloques

- Hay tres tipos de bloques que forman un programa PL/SQL:
  - Bloques anónimos
  - Procedimientos
  - Funciones

### Anónimos

```
[DECLARE]

BEGIN
  --statements

[EXCEPTION]

END;
```

### Procedure

```
PROCEDURE name
IS

BEGIN
  --statements

[EXCEPTION]

END;
```

### Function

```
FUNCTION name
RETURN datatype
IS
BEGIN
  --statements
  RETURN value;
[EXCEPTION]

END;
```

1 - 20

## Tipos de bloques. Bloques Anónimos

- Los bloques anónimos son bloques **sin nombre**.
- Se declaran en línea (online) en el punto en una aplicación donde se deben ejecutar y compilar cada vez que se ejecuta la aplicación.
  - Estos bloques **no se almacenan en la base de datos**.
- Si desea ejecutar el mismo bloque de nuevo, tiene que volver a escribir el bloque.
- No puede invocar o llamar al bloque que escribió anteriormente porque los bloques son anónimos y no existen después de ejecutados.

### Anónimos

```
[DECLARE]

BEGIN
  --statements

[EXCEPTION]

END;
```

1 - 21

## Tipos de bloques. Subprogramas

- Los **subprogramas** son complementarios a los bloques anónimos.
- Son bloques PL/SQL que se almacenan en la base de datos.
  - Disponen de un nombre y son almacenados, para poder ser invocarlos cuando quiera (dependiendo de su aplicación).
- Puede declararlos como **procedimientos** o como **funciones**.
  - Por lo general, se utiliza un procedimiento para realizar una acción y una función para calcular y devolver un valor.

### Procedure

```
PROCEDURE name
IS
BEGIN
    --statements
[EXCEPTION]
END;
```

### Function

```
FUNCTION name
RETURN datatype
IS
BEGIN
    --statements
    RETURN value;
[EXCEPTION]
END;
```

1 - 22

## Tipos de bloques. Procedures y Funciones

- Los **procedimientos** son objetos con nombre que contienen instrucciones SQL y / o PL/SQL.
- No devuelven ningún tipo de valor directamente al programa principal
- Las **funciones** son objetos con nombre que contienen instrucciones SQL y / o PL/SQL.
- A diferencia de un procedimiento, una función devuelve un valor de un tipo de datos especificado.

### Procedure

```
PROCEDURE name
IS
BEGIN
    --statements
[EXCEPTION]
END;
```

### Function

```
FUNCTION name
RETURN datatype
IS
BEGIN
    --statements
    RETURN value;
[EXCEPTION]
END;
```

1 - 23

## Construcciones del programa

La tabla siguiente describe una variedad de construcciones de programa PL/SQL que utilizan el bloque PL/SQL básico



Construcciones de herramientas
Bloques anónimos
Procedimientos o funciones de APP
Paquetes de APP
Triggers de APP
Tipos de objetos

Construcciones de servidor de base de datos
Bloques anónimos
Procedimientos almacenados o funciones
Paquetes Almacenados
Triggers de BBDD
Tipos de objetos

1 - 24

## Examinando un bloque anónimo

- Para crear un bloque anónimo utilizando SQL Developer, introduzca el bloque en el área de trabajo (como se muestra en la diapositiva).

```

DECLARE
  v_fname VARCHAR2(20);
BEGIN
  SELECT first_name INTO v_fname FROM employees
  WHERE employee_id=100;
END;
  
```

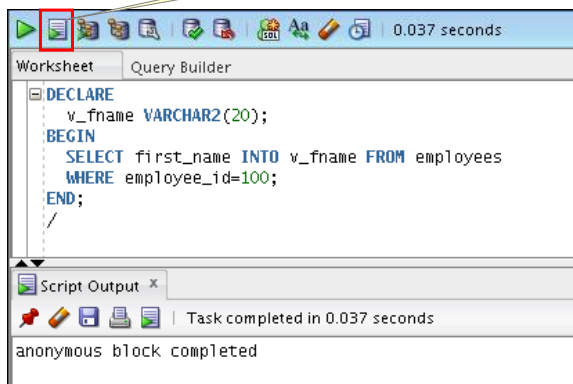
- El bloque de ejemplo tiene la sección **declarativa** y la sección **ejecutable**
- El bloque anónimo obtiene el primer nombre del empleado cuyo **employee\_id** es 100 y lo almacena en una variable denominada **v\_fname**

1 - 25

## Ejecutando un bloque anónimo

Haga clic en el botón Ejecutar script para ejecutar el bloque anónimo:

Run Script (or F5)



1 - 26

## Agenda

- Comprender los beneficios y la estructura de PL/SQL
- Examinar bloques PL/SQL
- Generación de mensajes de salida en PL/SQL

1 - 27

## Activación de la salida de un bloque PL / SQL

- El ejemplo anterior realiza unas operaciones pero no devuelve ningún valor.
- PL/SQL no tiene funcionalidad incorporada de entrada o salida. Por lo tanto, es necesario **utilizar paquetes de Oracle** predefinidos para la entrada y la salida

1. Ejecute el siguiente comando, para habilitar la salida de información

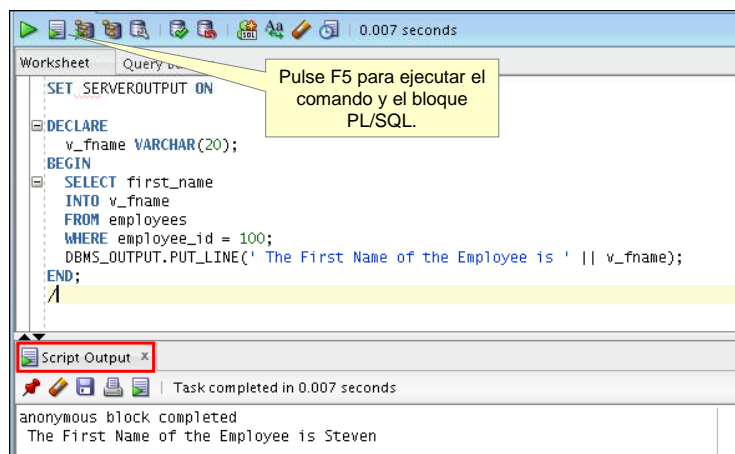
```
SET SERVEROUTPUT ON
```

2. Utilice el procedimiento **PUT\_LINE** del paquete **DBMS\_OUTPUT** para mostrar la salida deseada ( similar a PRINT)

```
DBMS_OUTPUT.PUT_LINE(' The First Name of the Employee  
is ' || v_fname);  
. . .
```

1 - 28

## Visualización de salida de un bloque PL/SQL



1 - 29

## Quiz

Un bloque PL/SQL debe constar de las tres secciones siguientes:

- Una sección declarativa, que comienza con la palabra clave `DECLARE` y termina cuando se inicia la sección ejecutable.
  - Una sección ejecutable, que comienza con la palabra clave `BEGIN` y termina con `END`
  - Una sección de manejo de excepciones, que comienza con la palabra clave `EXCEPTION` y está anidada dentro de la sección ejecutable.
- a. True
- b. False

1 - 30

## Resumen

En esta lección, debes haber aprendido a:

- Explicar la necesidad de `PL/SQL`
- Explicar los beneficios de `PL/SQL`
- Identificar los diferentes tipos de bloques `PL/SQL`
- Mensajes de salida en `PL/SQL`

1 - 31

## Prácticas 2

En esta lección, realiza las siguientes prácticas:

- Identificación de los bloques PL / SQL que se ejecutan correctamente
- Creación y ejecución de un simple bloque PL / SQL

1 - 32

# 3

## Declaración de variables PL / SQL



ORACLE®



## Objetivos

Después de completar esta lección, usted debería ser capaz de:

- Reconocer identificadores válidos e inválidos
- Enumerar los usos de las variables
- Declarar e inicializar variables
- Enumerar y describir varios tipos de datos
- Identificar los beneficios de usar el atributo `%TYPE`
- Declarar, utilizar e imprimir variables `bind`

1 - 34

## Agenda

- Introducción a las variables
- Examinar los tipos de datos variables y el atributo `%TYPE`
- Examen de variables `bind`

1 - 35

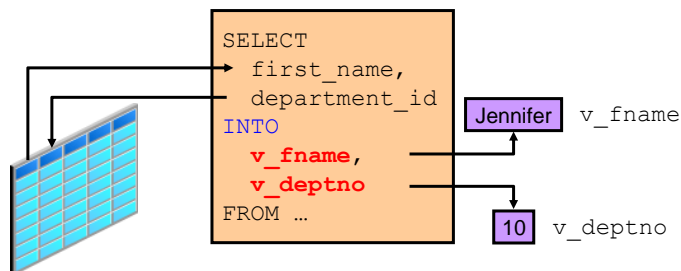
## Uso de Variables

- Con PL/SQL, puede declarar variables y luego usarlas en sentencias SQL y procedurales.
- Las variables se utilizan principalmente para el almacenamiento de datos y la manipulación de los valores almacenados.
  - Puede utilizar el valor almacenado en estas variables para procesar y manipular datos.
- Las variables pueden almacenar cualquier objeto PL/SQL como: **variables**, **tipos**, **cursores** y **subprogramas**.
- Una vez declaradas las variables, puede utilizarlas repetidamente en una aplicación haciendo referencia a ellas varias veces en varias sentencias.

1 - 36

## Uso de Variables

- Considere la instrucción PL / SQL en la diapositiva.
- La instrucción recupera **first\_name** y **department\_id** de la tabla.
- Si tiene que manipular **first\_name** o **department\_id**, tiene que almacenar el valor recuperado.
- Las variables se utilizan para almacenar temporalmente el valor.



1 - 37

## Requisitos para Nombres de Variable

Un nombre de variable:

- Debe comenzar con una letra
- Puede incluir letras o números
- Puede incluir caracteres especiales (como \$, \_ y #)
- No debe contener más de 30 caracteres
- No debe incluir palabras reservadas



1 - 38

## Manejo de variables en PL/SQL

Las variables son:

- Declaradas y (opcionalmente) inicializadas en la sección declarativa
  - Puede declarar variables en la parte declarativa de cualquier bloque, subprograma o paquete de PL / SQL
- Valores nuevos utilizados y asignados en la sección ejecutable
  - En la sección ejecutable, el valor existente de la variable se puede reemplazar con un nuevo valor.
- Pasadas como parámetros a los subprogramas PL/SQL
  - Los subprogramas pueden tomar parámetros. Puede pasar variables como parámetros a subprogramas
- Utilizadas para mostrar la salida de un subprograma PL/SQL
  - Se pueden utilizar variables para contener el valor devuelto por una función.

1 - 39

## Declaración e inicialización de variables PL/SQL

- Debe declarar todos los identificadores PL/SQL en la sección de declaración antes de referenciarlos en el bloque PL/SQL
  - Tiene la opción de asignar un valor inicial a una variable

Sintaxis:

```
identifier [CONSTANT] datatype [NOT NULL]
[:= | DEFAULT expr];
```

<b>Identifier</b>	Nombre de la variable
<b>datatype</b>	Es un tipo de datos escalar, compuesto, REF o LOB
<b>CONSTANT</b>	Restringe la variable para que su valor no pueda cambiar (las constantes deben ser inicializadas).
<b>NOT NULL</b>	Restringe la variable para que contenga un valor (las variables NOT NULL deben ser inicializadas).
<b>Expr</b>	Es cualquier expresión de PL/SQL que puede ser una expresión literal, otra variable o una expresión que implique operadores

1 - 40

## Declaración e inicialización de variables PL/SQL

Ejemplos:

```
DECLARE
  v_hiredate    DATE;
  v_location    VARCHAR2(13) := 'Atlanta';
  v_deptno     NUMBER(2) NOT NULL := 10;
  c_comm        CONSTANT NUMBER := 1400;
```

- Las variables de tipo cadena deben de estar entre comillas simples.
- El operador de asignación es: " := "
- Si no asigna un valor inicial, la nueva variable contiene NULL de forma predeterminada hasta que asigne un valor

1 - 41

## Declaración e inicialización de variables PL/SQL

1

```
DECLARE
  v_myName VARCHAR(20);
BEGIN
  DBMS_OUTPUT.PUT_LINE('My name is: '||v_myName);
  v_myName := 'John';
  DBMS_OUTPUT.PUT_LINE('My name is: '||v_myName);
END;
/
```

2

```
DECLARE
  v_myName VARCHAR2(20) := 'John';
BEGIN
  v_myName := 'Steven';
  DBMS_OUTPUT.PUT_LINE('My name is: '|| v_myName);
END;
/
```

1 - 42

## Delimitadores en variables de caracteres

- Los delimitadores por defecto de las cadenas de caracteres es la comilla simple '.
- Las necesitamos introducir éste carácter dentro de la cadena, deberemos duplicarla. Ésta comilla actuará como **carácter de escape**.

```
v_event VARCHAR2(15) := 'Father's day';
```

- Puede especificar cualquier carácter que no esté presente en la cadena como **delimitador**. → q + comilla + delimitador

```
v_event := q'!Father's day!';
```

```
v_event := q'[Father's day]';
```

1 - 43

## Delimitadores en variables de caracteres

Ejemplos:

```
DECLARE
    v_event VARCHAR2(15);
BEGIN
    v_event := q'!Father's day!';
    DBMS_OUTPUT.PUT_LINE('3rd Sunday in June is :
    || v_event );
    v_event := q'[Mother's day]';
    DBMS_OUTPUT.PUT_LINE('2nd Sunday in May is :
    || v_event );
END;
/
```

Salida  
resultante

```
anonymous block completed
3rd Sunday in June is : Father's day
2nd Sunday in May is : Mother's day
```

1 - 44

## Agenda

- Introducción a las variables
- Examinar los tipos de datos variables y el atributo %TYPE
- Examen de variables bind

1 - 45

## Tipos de variables

- Cada variable `PL/SQL` tiene un tipo de datos, que especifica un formato de almacenamiento, restricciones y un rango válido de valores
- Variables `PL/SQL`:
  - **Escalar**
    - Los tipos de datos escalares contienen un solo valor. `CHAR`, `VARCHAR2`, etc
  - **Referencia**
    - Contienen valores, denominados punteros, que apuntan a una ubicación de almacenamiento
  - **Objeto grande (LOB)**
    - Contienen valores, llamados localizadores, que especifican la ubicación de objetos grandes (como imágenes gráficas) almacenados fuera de la tabla
  - **Compuesto**
    - Las colecciones y registros de `PL/SQL` contienen elementos internos que se pueden tratar como variables individuales

1 - 46

## Tipos de variables



1 - 47

## Directrices para declarar e inicializar variables PL/SQL

- Siga las convenciones de nomenclatura coherentes.
- Utilizar identificadores significativos para las variables.
- Inicialice las variables que se designan como `NOT NULL` y `CONSTANT`.
- Inicializar variables con el operador de asignación **(:=)** o la palabra clave `DEFAULT` :

```
v_myName VARCHAR2(20) := 'John';
```

```
v_myName VARCHAR2(20) DEFAULT 'John';
```

- Declare un identificador por línea para una mejor legibilidad y mantenimiento del código.

```
sal CONSTANT NUMBER := 50000.00;
```

1 - 48

## Directrices para declarar variables PL/SQL

- Evite utilizar nombres de columnas como identificadores.

```
DECLARE
  employee_id NUMBER(6);
BEGIN
  SELECT employee_id
  INTO   employee_id
  FROM   employees
  WHERE  last_name = 'Kochhar';
END;
/
```



### NOTAS:

- Utilice la restricción `NOT NULL` cuando la variable debe contener un valor.
- Dos objetos pueden tener el **mismo nombre sólo si están definidos en bloques diferentes**

1 - 49



## Convenciones de nomenclatura de estructuras PL/SQL utilizadas en este curso

Estructura PL/SQL	Convention	Ejemplo
Variable	<code>v_variable_name</code>	<code>v_rate</code>
Constant	<code>c_constant_name</code>	<code>c_rate</code>
Subprogram parameter	<code>p_parameter_name</code>	<code>p_id</code>
Bind (host) variable	<code>b_bind_name</code>	<code>b_salary</code>
Cursor	<code>cur_cursor_name</code>	<code>cur_emp</code>
Record	<code>rec_record_name</code>	<code>rec_emp</code>
Type	<code>type_name_type</code>	<code>ename_table_type</code>
Exception	<code>e_exception_name</code>	<code>e_products_invalid</code>
File handle	<code>f_file_handle_name</code>	<code>f_file</code>

1 - 50

## Tipos de datos escalares

- PL/SQL proporciona una variedad de tipos de datos predefinidos.
  - Por ejemplo, puede elegir entre entero, punto flotante, carácter, booleano, fecha, colección y tipos de LOB.
- Un tipo de datos escalares tiene un valor único y no tiene componentes internos.
  - Se pueden clasificar en cuatro categorías: **número**, **carácter**, **fecha** y **booleano**

**TRUE**

**256120.08**

El alma del perezoso desea, y no tiene nada; Pero el alma del diligente se enriquecerá.

**15-JAN-09**

**Atlanta**

1 - 51

## Tipos de datos escalares básicos

Tipo de Datos	Descripción
<b>CHAR</b> [(maximum_length)]	Tipo de base para datos de caracteres de <b>longitud fija</b> de hasta 32.767 bytes. Si no especifica una longitud máxima, la longitud predeterminada se establece en 1 byte.
<b>VARCHAR2</b> (maximum_length)	Tipo base para datos de caracteres de <b>longitud variable</b> hasta 32.767 bytes. Es obligatorio definir el tamaño
<b>NUMBER</b> [(precisión, escala)]	Número que tiene precisión <b>p</b> y escala <b>s</b> . La precisión <b>p</b> puede variar de 1 a 38. La escala <b>s</b> puede variar de -84 a 127. Oracle garantiza la portabilidad de números con una precisión igual o inferior a <b>38 dígitos</b> . Puede especificar una escala y sin precisión:
<b>BINARY_INTEGER</b>	Tipo base para <b>enteros</b> entre -2.147.483.647 y 2.147.483.647

1 - 52

## Tipos de datos escalares básicos

Tipo de Datos	Descripción
<b>PLS_INTEGER</b>	Tipo base para <b>enteros</b> entre -2.147.483.647 y 2.147.483.647 Los valores de <b>PLS_INTEGER</b> requieren menos almacenamiento y son más rápidos que <b>NUMBER</b> valores. En Oracle Database 11g y Oracle Database 12c, los tipos de datos <b>PLS_INTEGER</b> y <b>BINARY_INTEGER</b> son idénticos
<b>BOOLEAN</b>	Tipo base para datos que almacena uno de los tres valores posibles utilizados para cálculos lógicos: <b>TRUE, FALSE y NULL</b>
<b>BINARY_FLOAT</b>	Representa el número de coma flotante en formato IEEE 754. Tipo de datos de precisión simple de 32 bits. Requiere 5 bytes para almacenar el valor.
<b>BINARY_DOUBLE</b>	Representa el número de coma flotante en formato IEEE 754. Tipo de datos de precisión simple de 64 bits. Requiere 9 bytes para almacenar el valor.

1 - 53

## Tipos de datos escalares básicos

Tipo de Datos	Descripción
<b>DATE</b>	Tipo base para fechas y horas. Los valores de DATE incluyen la hora del día en segundos desde la medianoche. El rango de fechas es entre 4712 a.d. hasta 9999.
<b>TIMESTAMP</b> [(precision)]	TIMESTAMP amplía el tipo de datos DATE, almacena el año, mes, día, hora, minuto, segundo y fracción de segundo. La precisión es un parámetro opcional que especifica el número de dígitos en la parte fraccionaria en el rango 0-9. <b>El valor predeterminado es 6.</b>
<b>TIMESTAMP WITH TIME ZONE</b> [(precision)]	Extiende el tipo de datos TIMESTAMP, incluye un desplazamiento de zona horaria. El desplazamiento de la zona horaria es la diferencia (en horas y minutos) entre la hora local y la hora universal coordinada (UTC), La precisión es un parámetro opcional que especifica el número de dígitos en la parte fraccionaria en el rango 0-9. <b>El valor predeterminado es 6.</b>

1 - 54

## Tipos de datos escalares básicos

Tipo de Datos	Descripción
<b>TIMESTAMP WITH LOCAL TIME ZONE</b> [(precision)]	Similar a <b>TIMESTAMP WITH TIME ZONE</b> pero difiere de este en que al insertar un valor en una columna de base de datos, el valor se normaliza en la zona horaria de la base de datos y el desplazamiento de zona horaria no se almacena en la columna. Cuando recupera el valor, el servidor Oracle devuelve el valor en la zona horaria de su sesión local.
<b>INTERVAL YEAR TO MONTH</b> [(precision)]	Este tipo almacena y manipular intervalos de años y meses. La precisión especifica el número de dígitos en el campo de años. No puede utilizar una constante simbólica o una variable para especificar la precisión; Debe utilizar un literal entero en el rango 0-4. <b>El valor predeterminado es 2.</b>
<b>INTERVAL DAY</b> [(precision1)] <b>TO SECOND</b> [(precision2)]	Tipo utilizado para almacenar y manipular intervalos de días, horas, minutos y segundos. La precision1 y precision2 especifican el número de dígitos en el campo días y segundos, respectivamente. Debe utilizar un literal entero en el rango 0-9. Los valores por defecto son 2 y 6, respectivamente.

1 - 55

## Declaración de variables escalares

Examples:

```
DECLARE
  v_emp_job          VARCHAR2(9);
  v_count_loop       BINARY_INTEGER := 0;
  v_dept_total_sal    NUMBER(9,2) := 0;
  v_orderdate         DATE := SYSDATE + 7;
  c_tax_rate          CONSTANT NUMBER(3,2) := 8.25;
  v_valid             BOOLEAN NOT NULL := TRUE;
  ...
```

1 - 56

## Atributo %TYPE

- Las variables PL/SQL se declaran generalmente para almacenar y manipular datos almacenados en una base de datos.
  - Cuando declara que las variables PL/SQL contienen valores de columna, debe asegurarse de que la variable sea del tipo y precisión de datos correctos
  - Si no es así, se produce un error PL / SQL durante la ejecución
- Podemos utilizar el atributo **%TYPE** en los siguientes escenarios:
  - Cuando queremos declarar una variable de acuerdo con una variable previamente definida.
  - Cuando queremos declarar una variable de acuerdo con una columna de una tabla de base de datos

1 - 57

## Atributo %TYPE

- El atributo %TYPE se utiliza con mayor frecuencia cuando el valor almacenado en la variable se deriva de una tabla de la base de datos.
- Cuando utiliza el atributo %TYPE para declarar una variable, debe prefijarla con la tabla de base de datos y el nombre de columna.  
`Variable Tabla.Columna%TYPE`
- Una restricción de columna NOT NULL **no se aplica a las variables que se declaran utilizando %TYPE.**
  - Por lo tanto, si declara una variable utilizando el atributo %TYPE que utiliza una columna de base de datos definida como NOT NULL, puede asignar el valor NULL a la variable.

1 - 58

## Atributo %TYPE

### Ventajas del atributo %TYPE

- Puede evitar errores causados por el tipo de datos no coincidentes o por una precisión incorrecta.
- Puede evitar codificar el tipo de datos de una variable.
- No es necesario cambiar la declaración de la variable si cambia la definición de la columna.
  - Si ya ha declarado algunas variables para una tabla en particular sin utilizar el atributo %TYPE, el bloque PL/SQL puede generar errores si se modifica la columna para la que se declara la variable.
- Cuando se utiliza el atributo %TYPE, PL/SQL determina el tipo de datos y el tamaño de la variable cuando se compila el bloque. Esto asegura que dicha variable sea siempre compatible con la columna que se utiliza para rellenarla.

1 - 59

## Declaring Variables with the %TYPE Attribute

Sintaxis:

```
identifier      table.column_name%TYPE;
```

Ejemplos:

```
...  
  v_emp_lname      employees.last_name%TYPE;  
...
```

```
...  
  v_balance        NUMBER(7,2);  
  v_min_balance    v_balance%TYPE := 1000;  
...
```

1 - 60

## Declaración de variables booleanas

- Sólo los valores **TRUE**, **FALSE** y **NULL** se pueden asignar a una variable booleana.
- Las expresiones condicionales utilizan los operadores lógicos **AND** y **OR** y el operador unario **NOT** para comprobar los valores de la variable.
- Las variables siempre producen **TRUE**, **FALSE** o **NULL**.
- Las expresiones aritméticas, de carácter y de fecha se pueden utilizar para devolver un valor booleano.
- En una instrucción procedural, las expresiones booleanas son la base para el **control condicional**

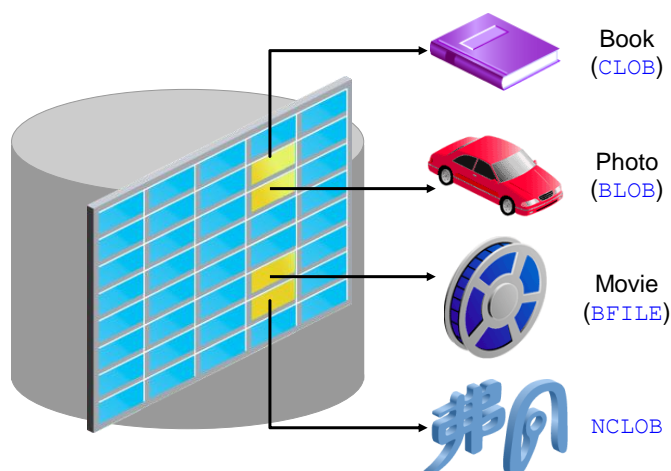
1 - 61

## Variables de tipo de datos LOB

- Objetos grandes (LOB) están destinados a almacenar una gran cantidad de datos.
- Una columna de base de datos puede ser de la categoría LOB.
  - Con la categoría LOB de tipos de datos (BLOB, CLOB, etc.), puede almacenar bloques de datos no estructurados (como texto, imágenes gráficas, videoclips y formas de ondas de sonido) de hasta **128 terabytes**
- El tipo de datos de objeto binario grande (BLOB) se utiliza para almacenar **objetos binarios** grandes.
  - Cuando inserta o recupera dichos datos en o desde la base de datos, la base de datos **no interpreta los datos**. Las aplicaciones externas que utilizan estos datos deben interpretar los datos.
- El tipo de datos de archivo binario (BFILE) se utiliza para almacenar archivos binarios grandes fuera de la BBDD.

1 - 62


## Variables de tipo de datos LOB



1 - 63

## Tipos de datos compuestos: registros y colecciones

- Los tipos de datos compuestos, llamados **registros PL/SQL** y **colecciones PL/SQL**, tienen componentes internos que pueden tratarse como variables individuales.
- Registros PL/SQL**
  - Los componentes internos pueden ser de diferentes tipos de datos, y se llaman **campos**.
  - Se accede a cada campo con esta sintaxis:  
**record\_name.field\_name.**
  - Una variable de registro puede contener una fila de tabla o algunas columnas de una fila de tabla.
  - Cada campo de registro corresponde a una columna de tabla.

TRUE	23-DEC-98	ATLANTA	
------	-----------	---------	---

1 - 64

## Tipos de datos compuestos: registros y colecciones

- Colecciones PL/SQL**
  - los componentes internos son siempre del mismo tipo de datos y se llaman **elementos**.
  - Se accede a cada elemento por su único subíndice.
  - Existen tres tipos de colecciones PL/SQL:  
**Matrices asociativas, Tablas anidadas y tipos VARRAY.**

1	SMITH	1	5000
2	JONES	2	2345
3	NANCY	3	12
4	TIM	4	3456

↑
↑
↑
↑

PLS\_INTEGER
VARCHAR2
PLS\_INTEGER
NUMBER

1 - 65



## Agenda

- Introducción a las variables
- Examinar los tipos de datos variables y el atributo %TYPE
- Examen de variables `bind`

1 - 66

## Variables Bind

- Las variables **bind** son variables que se crean en un entorno de host.
- Las variables de enlace se crean en el entorno y no en la sección declarativa de un bloque PL/SQL.
  - Por lo tanto, las variables de enlace son accesibles incluso después de ejecutar el bloque
- Cuando se crean, las variables de enlace pueden ser utilizadas y manipuladas por varios subprogramas.
- Pueden usarse en sentencias SQL y bloques PL/SQL como cualquier otra variable.
  - Estas variables se pueden pasar como valores de tiempo de ejecución dentro o fuera de los subprogramas PL / SQL.

1 - 67

## Variables Bind

- Para crear una variable de enlace en SQL Developer, utilice el comando **VARIABLE**.
- Por ejemplo, se declara una variable de tipo **NUMBER** y **VARCHAR2** como sigue:

```
VARIABLE return_code NUMBER  
VARIABLE return_msg VARCHAR2(30)
```

- Puede hacer referencia a la variable de enlace mediante SQL Developer y ver su valor mediante el comando **PRINT**.

1 - 68

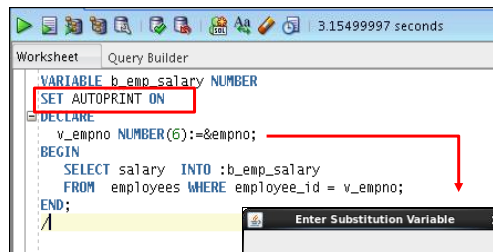
## Variables Bind

### Ejemplo

```
VARIABLE b_result NUMBER  
BEGIN  
    SELECT (SALARY*12) + NVL(COMMISSION_PCT,0) INTO :b_result  
    FROM employees WHERE employee_id = 144;  
END;  
/  
PRINT b_result
```

1 - 69

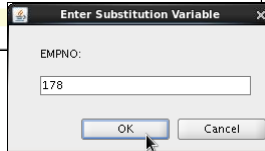
## Uso de AUTOPRINT con variables Bind



```
Worksheet Query Builder
3.15499997 seconds

VARIABLE b_emp_salary NUMBER
SET AUTOPRINT ON
DECLARE
  v_empno NUMBER(6):=&empno;
BEGIN
  SELECT salary INTO :b_emp_salary
  FROM employees WHERE employee_id = v_empno;
END;
```

Utilice el comando **SET AUTOPRINT ON** para mostrar automáticamente las variables bind usadas en un bloque PL/SQL exitoso.

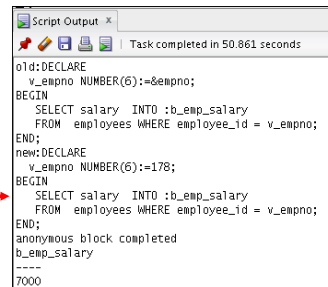


Enter Substitution Variable

EMPNO:

178

OK Cancel



```
Script Output
Task completed in 50.861 seconds

old:DECLARE
  v_empno NUMBER(6):=&empno;
BEGIN
  SELECT salary INTO :b_emp_salary
  FROM employees WHERE employee_id = v_empno;
END;
new:DECLARE
  v_empno NUMBER(6):=178;
BEGIN
  SELECT salary INTO :b_emp_salary
  FROM employees WHERE employee_id = v_empno;
END;
anonymous block completed
b_emp_salary
7000
```

1 - 71

## Quiz

El atributo %TYPE: (3)

- a. Se utiliza para declarar una variable según una definición de columna de base de datos
- b. Se utiliza para declarar una variable de acuerdo con una colección de columnas en una tabla o vista de una base de datos
- c. Se utiliza para declarar una variable de acuerdo con la definición de otra variable declarada
- d. Tiene prefijo con el nombre de la tabla y la columna de la base de datos o el nombre de la variable declarada

1 - 72

## Resumen

En esta lección, debes haber aprendido a:

- Reconocer identificadores válidos e inválidos
- Enumerar los usos de las variables
- Declarar e inicializar variables
- Enumerar y describir varios tipos de datos
- Identificar los beneficios de usar el atributo `%TYPE`
- Declarar, utilizar e imprimir variables `bind`

1 - 73

## Práctica 3:

En esta lección, realiza las siguientes prácticas:

- Determinación de identificadores válidos
- Determinación de declaraciones de variables válidas
- Declaración de variables dentro de un bloque anónimo
- Utilizar el atributo `%TYPE` para declarar variables
- Declaración e impresión de una variable `bind`
- Ejecución de un bloque `PL/SQL`

1 - 74

# 4

## Escribir declaraciones ejecutables



ORACLE®

### Objetivos

Después de completar esta lección, usted debería ser capaz de:

- Identificar unidades léxicas en un bloque `PL/SQL`
- Utilizar funciones `Built-in` de `SQL` en `PL/SQL`
- Describa cuándo se producen conversiones implícitas y cuándo deben tratarse las conversiones explícitas
- Escribir bloques anidados y calificar variables con etiquetas
- Escribir código legible con la sangría adecuada
- Utilizar secuencias en expresiones `PL/SQL`

## Agenda

- Escribir instrucciones ejecutables en un bloque PL/SQL
- Escribir bloques anidados
- Uso de operadores y desarrollo de códigos legibles

1 - 77

## Unidades léxicas en un bloque PL / SQL

- Las unidades léxicas incluyen **letras, números, caracteres especiales, tabulados, espacios, returns y símbolos.**
- Los **identificadores** son los nombres dados a los objetos PL/SQL.
  - Recuerde que las palabras clave no se pueden utilizar como identificadores.
- Los **delimitadores** son símbolos que tienen un significado especial.
  - El punto y coma ( ; ) se utiliza para finalizar una instrucción SQL o PL / SQL.

1 - 78

## Unidades léxicas en un bloque PL / SQL

Las unidades léxicas incluyen

- Pueden clasificarse como:
  - Identificadores: `v_fname, c_percent`
  - Delimitadores: `;, +, -`
  - Literales: `John, 428, True`
  - Comentarios: `--, /* */`

1 - 79

## Sintaxis de Bloque PL/SQL

- Uso de literales
  - Los literales de caracteres incluyen todos los caracteres imprimibles del conjunto de caracteres PL / SQL: letras, números, espacios y símbolos especiales.
  - Los literales de caracteres y fechas deben estar encerrados entre comillas simples.

```
v_name := 'Henderson';
```

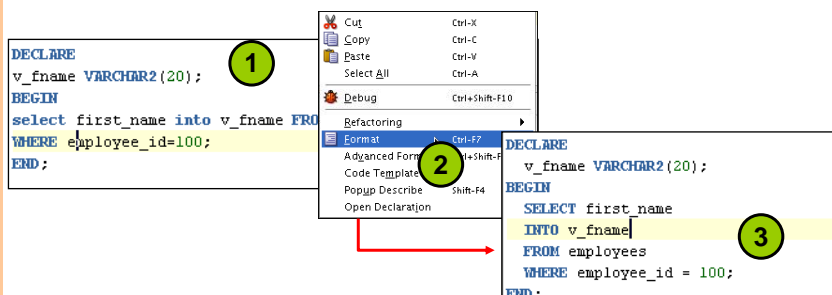
- Los literales numéricos pueden representarse por un valor simple (por ejemplo, -32.5) o en notación científica (por ejemplo, 2E5 significa  $2 * 10^5 = 200.000$ ).

```
v_name := 2E5;
```

1 - 80

## Sintaxis de Bloque PL/SQL

- Código de formato:
  - En un bloque PL/SQL, una instrucción SQL puede abarcar varias líneas (como se muestra en el ejemplo 3 de la diapositiva).
  - Puede formatear una instrucción SQL sin formatear (como se muestra en el ejemplo 1 de la diapositiva) como se muestra en 2 y 3 o mediante **Ctrl + F7**



1 - 81

## Comentando el código

- Si desea comentarios de una sola línea utilice dos guiones (--)
- Coloque un comentario de bloque entre los símbolos /\* \*/
- Ejemplo:

```
DECLARE
...
v_annual_sal NUMBER (9,2);
BEGIN
/* Compute the annual salary based on the
   monthly salary input from the user */
v_annual_sal := monthly_sal * 12;
--The following line displays the annual salary
DBMS_OUTPUT.PUT_LINE(v_annual_sal);
END;
/
```

1 - 82



## Funciones de SQL en PL/SQL

- SQL proporciona varias funciones predefinidas que se pueden utilizar en sentencias PL/SQL.
  - La mayoría de estas funciones son válidas en expresiones PL/SQL.
  - Funciones de una fila
- Las funciones siguientes **no están disponibles** en declaraciones **procedimentales**
  - `DECODE`
  - Funciones de Grupo:
    - `AVG`, `MIN`, `MAX`, `COUNT`, `SUM`, `STDDEV` y `VARIANCE`
    - Las funciones de grupo se aplican a grupos de filas de una tabla y, por lo tanto, sólo están disponibles en sentencias SQL en un bloque PL/SQL

1 - 83

## Funciones de SQL en PL/SQL: Ejemplos

- Obtener la longitud de una cadena:

```
v_desc_size INTEGER(5);  
v_prod_description VARCHAR2(70):='You can use this  
product with your radios for higher frequency';  
  
-- get the length of the string in prod description  
v_desc_size:= LENGTH(v_prod_description);
```

- Obtenga el número de meses que un empleado ha trabajado:

```
v_tenure:= MONTHS_BETWEEN (CURRENT_DATE, v_hiredate);
```

1 - 84

## Uso de secuencias en expresiones PL/SQL

- Antes de **Oracle Database 11g**, estábamos obligados a escribir una sentencia `SQL` para utilizar un valor de objeto de secuencia en una subrutina `PL/SQL`  

```
SELECT my_seq.NEXTVAL INTO v_new_id FROM Dual;
```
- En **Oracle Database 11g** y **posterior**, puede utilizar las pseudocolumnas `NEXTVAL` y `CURRVAL` en cualquier contexto `PL/SQL`, donde una expresión del tipo de datos `NUMBER` puede aparecer legalmente.  

```
v_new_id := my_seq.NEXTVAL;
```
- De esta forma:
  - Se mejora la usabilidad de la secuencia
  - El desarrollador tiene que escribir menos
  - El código resultante es más claro

1 - 85

## Uso de secuencias en expresiones PL/SQL

Comenzando en 11g:

```
DECLARE
  v_new_id NUMBER;
BEGIN
  v_new_id := my_seq.NEXTVAL;
END;
/
```

Antes de 11g:

```
DECLARE
  v_new_id NUMBER;
BEGIN
  SELECT my_seq.NEXTVAL INTO v_new_id FROM Dual;
END;
/
```

1 - 86

## Conversión de tipo de datos

- En cualquier lenguaje de programación, convertir un tipo de datos a otro es un requisito común.
- PL/SQL puede manejar tales conversiones con tipos de datos escalares.
- Las conversiones de tipo de datos pueden ser de dos tipos:
  - Conversiones **implícitas**:
    - PL/SQL intenta convertir tipos de datos de forma dinámica.
    - Las conversiones implícitas pueden ser entre:
      - Caracteres y numeros
      - Caracteres y Fechas

1 - 87

## Conversión de tipo de datos

- Conversiones **explicitas**:
  - Son utilizada para convertir valores de un tipo de datos a otro
  - Se utilizan las funciones BUILT-in.
  - Por ejemplo, para convertir un valor **CHAR** a un valor **DATE** o **NUMBER**, utilice **TO\_DATE** o **TO\_NUMBER**, respectivamente.
- Funciones:
  - **TO\_CHAR**
  - **TO\_DATE**
  - **TO\_NUMBER**
  - **TO\_TIMESTAMP**

1 - 88

## Conversión de tipo de datos

1

```
-- conversión implícita de tipo de datos  
v_date_of_joining DATE:= '02-Feb-2000';
```

2

```
-- Error en la conversión de tipo de datos  
v_date_of_joining DATE:= 'February 02,2000';
```

3

```
-- Conversión explícita de tipo de datos  
v_date_of_joining DATE:= TO_DATE('February  
02,2000','Month DD, YYYY');
```

1 - 89

## Agenda

- Escribir instrucciones ejecutables en un bloque PL/SQL
- Escribir bloques anidados
- Uso de operadores y desarrollo de códigos legibles

1 - 90

## Bloques anidados

Los bloques PL/SQL pueden anidarse.

- Una sección ejecutable (BEGIN ... END)
- Una sección de excepciones
- Puede anidar bloques donde quiera que se permita una instrucción ejecutable.
  - Si su sección ejecutable tiene código para muchas funcionalidades, puede dividir la sección ejecutable en bloques más pequeños.



1 - 91

## Bloques anidados: Ejemplos

```
DECLARE
  v_outer_variable VARCHAR2(20):='GLOBAL VARIABLE';
BEGIN
  DECLARE
    v_inner_variable VARCHAR2(20):='LOCAL VARIABLE';
  BEGIN
    DBMS_OUTPUT.PUT_LINE(v_inner_variable);
    DBMS_OUTPUT.PUT_LINE(v_outer_variable);
  END;
  DBMS_OUTPUT.PUT_LINE(v_outer_variable);
END;
```

```
anonymous block completed
LOCAL VARIABLE
GLOBAL VARIABLE
GLOBAL VARIABLE
```

- La variable `v_outer_variable` es local al bloque externo pero global al bloque interno
- la variable `v_outer_variable` se considera que es la variable global para todos los bloques internos
- La variable `v_inner_variable` es local al bloque interno y no es global porque el bloque interno no tiene ningún bloque anidado

1 - 92

## Alcance y visibilidad variables

```
DECLARE
  v_father_name VARCHAR2(20):='Patrick';
  v_date_of_birth DATE:='20-Apr-1972';
BEGIN
  DECLARE
    v_child_name VARCHAR2(20):='Mike';
    v_date_of_birth DATE:='12-Dec-2002';
  BEGIN
    DBMS_OUTPUT.PUT_LINE('Father's Name: '||v_father_name);
    DBMS_OUTPUT.PUT_LINE('Date of Birth: '||v_date_of_birth);
    DBMS_OUTPUT.PUT_LINE('Child's Name: '||v_child_name);
  END;
  DBMS_OUTPUT.PUT_LINE('Date of Birth: '||v_date_of_birth);
END;
/
```

Father's Name: Patrick  
Date of Birth: 12-DEC-02  
Child's Name: Mike  
Date of Birth: 20-APR-72

1 - 93

## Alcance y visibilidad variables

- **Scope: (Alcance)**
  - Las variables tienen el **alcance del bloque en el que se declaran**.
  - Las variables `v_child_name` y `v_date_of_birth` se declaran en el bloque interno o en el bloque anidado.
    - Estas variables son accesibles sólo dentro del bloque anidado y no son accesibles en el bloque externo.
    - Cuando una variable está fuera del alcance, PL/SQL libera la memoria utilizada para almacenar la variable; Por lo tanto, estas variables no pueden ser referenciadas.
- **Visibility: (Visibilidad)**
  - Las variables del bloque externo tiene visibilidad dentro de todos los bloques anidados, a menos que exista una variable en dicho bloque con el mismo nombre.
  - La variable `v_date_of_birth` declarada en el bloque externo tiene alcance incluso en el bloque interno.
    - Sin embargo, esta variable **no es visible** en el bloque interno porque el bloque interno tiene una variable local con el mismo nombre.

1 - 94

## Utilizar un calificador con bloques anidados

- Un **calificador** es una etiqueta dada a un bloque.
- Puede utilizar un **calificador** para acceder a las variables que tienen alcance pero no son visibles.

```
BEGIN <<outer>>
DECLARE
  v_father_name VARCHAR2(20):='Patrick';
  v_date_of_birth DATE:='20-Apr-1972';
BEGIN
  DECLARE
    v_child_name VARCHAR2(20):='Mike';
    v_date_of_birth DATE:='12-Dec-2002';
  BEGIN
    DBMS_OUTPUT.PUT_LINE('Father's Name: '||v_father_name);
    DBMS_OUTPUT.PUT_LINE('Date of Birth: '||outer.v_date_of_birth);
    DBMS_OUTPUT.PUT_LINE('Child's Name: '||v_child_name);
    DBMS_OUTPUT.PUT_LINE('Date of Birth: '||v_date_of_birth);
  END;
END;
END outer;
```

1 - 95

## Desafío: Determinación del alcance variable

```
BEGIN <<outer>>
DECLARE
  v_sal      NUMBER(7,2) := 60000;
  v_comm     NUMBER(7,2) := v_sal * 0.20;
  v_message  VARCHAR2(255) := ' eligible for commission';
BEGIN
  DECLARE
    v_sal      NUMBER(7,2) := 50000;
    v_comm     NUMBER(7,2) := 0;
    v_total_comp NUMBER(7,2) := v_sal + v_comm;
  BEGIN
    v_message := 'CLERK not'||v_message;
    outer.v_comm := v_sal * 0.30;
  END;
  v_message := 'SALESMAN'||v_message;
END;
END outer;
/
```

Evalúe el bloque PL/SQL en la diapositiva. Determine Siguiente Transparencia:

1 - 96

## Desafío: Determinación del alcance variable

1. Valor de `v_message` en la posición 1
2. Valor de `v_total_comp` en la posición 2
3. Valor de `v_comm` en la posición 1
4. Valor de `outer.v_comm` en la posición 1
5. Valor de `v_comm` en la posición 2
6. Valor de `v_message` en la posición 2

1 - 97

## Desafío: Determinación del alcance variable

- 1.- Valor de `v_message` en la posición 1  
**CLERK not eligible for commission**
- 2.- Valor de `v_total_comp` en la posición 2  
**Error. v\_total\_comp is not visible here because it is defined within the inner block.**
- 3.- Valor de `v_comm` en la posición 1  
**0**
- 4.- Valor de `outer.v_comm` en la posición 1  
**15000**
- 5.- Valor de `v_comm` en la posición 2  
**12000**
- 6.- Valor de `v_message` en la posición 2  
**SALESMANCLERK not eligible for commission**

1 - 98



## Agenda

- Escribir instrucciones ejecutables en un bloque PL/SQL
- Escribir bloques anidados
- Uso de operadores y desarrollo de códigos legibles

1 - 99

## Operadores en PL/SQL

- Las operaciones en una expresión se realizan en un orden particular dependiendo de su precedencia (prioridad).
- La siguiente tabla muestra el orden predeterminado de las operaciones de alta prioridad a baja prioridad:

Operator	Operation
**	Exponentiation
+, -	Identity, negation <b>a:=-5; b:=-a</b>
*, /	Multiplication, division
+, -,	Addition, subtraction, concatenation
=, <, >, <=, >=, <>, !=, ~=, ^=, IS NULL, LIKE, BETWEEN, IN	Comparison
NOT	Logical negation
AND	Conjunction
OR	Inclusion

1 - 100

## Operadores en PL/SQL: Ejemplos

- Incrementar el contador para un bucle.

```
loop_count := loop_count + 1;
```

- Establezca el valor de un indicador booleano.

```
good_sal := sal BETWEEN 50000 AND 150000;
```

- Valide si un número de empleado contiene un valor.

```
valid := (empno IS NOT NULL);
```

Cuando trabaja con **valores nulos**, puede evitar algunos errores comunes teniendo en cuenta las siguientes reglas:

- Las comparaciones con valores nulos siempre dan **NULL**.
- Aplicar el operador lógico **NOT** a un **null** produce **NULL**.
- En declaraciones de control condicional, si la condición produce **NULL**, su secuencia asociada de sentencias no se ejecuta.

1 - 101

## Directrices de programación

- Siga las pautas de programación mostradas a continuación para producir código claro y reducir el mantenimiento al desarrollar un bloque PL/SQL.

Facilite el mantenimiento del código mediante:

- Documentando el código con comentarios
- Desarrollo de una convención para el código
- Desarrollo de convenciones de nomenclatura para identificadores y otros objetos
- Mejora de la legibilidad mediante indentación

1 - 102

## Identación de código

Para mayor claridad, indente cada nivel de código.

```
BEGIN
  IF x=0 THEN
    y:=1;
  END IF;
END;
/
```

```
DECLARE
  v_deptno      NUMBER(4);
  v_location_id NUMBER(4);
BEGIN
  SELECT  department_id,
          location_id
  INTO    v_deptno,
          v_location_id
  FROM    departments
  WHERE   department_name
          = 'Sales';
  ...
END;
/
```

1 - 103

## Quiz

¿Puede utilizar la mayoría de las funciones de una sola fila de SQL, como las funciones de una sola fila de número, carácter, conversión y fecha en expresiones PL/SQL?

- a. True
- b. False

1 - 104

## Resumen

En esta lección, debes haber aprendido a:

- Identificar unidades léxicas en un bloque `PL/SQL`
- Utilizar funciones `Built-in de SQL` en `PL/SQL`
- Describa cuándo se producen conversiones implícitas y cuándo deben tratarse las conversiones explícitas
- Escribir bloques anidados y calificar variables con etiquetas
- Escribir código legible con la sangría adecuada
- Utilizar secuencias en expresiones `PL/SQL`

1 - 105

## Práctica 4

En esta lección, realiza las siguientes prácticas:

- Revisión de las reglas de definición y definición de anidaciones
- Escribir y probar bloques `PL/SQL`

1 - 106

# 5

## Uso de sentencias SQL dentro de un bloque PL/SQL



ORACLE®

### Objetivos

Después de completar esta lección, usted debería ser capaz de:

- Determinar las sentencias de SQL que se pueden incluir directamente en un bloque ejecutable de PL/SQL
- Manipular datos con instrucciones DML en PL/SQL
- Utilizar instrucciones de control de transacciones en PL/SQL
- Hacer uso de la cláusula `INTO` para mantener los valores devueltos por una instrucción SQL
- Diferenciar entre cursores implícitos y cursores explícitos
- Utilizar atributos de cursor SQL

## Agenda

- Recuperación de datos con PL/SQL
- Manipulación de datos con PL/SQL
- Presentación de cursores SQL

1 - 109

## Sentencias SQL en PL/SQL

- En un bloque PL/SQL, se utilizan sentencias SQL para recuperar y modificar datos de la tabla de la base de datos.
  - PL/SQL soporta el lenguaje de manipulación de datos (**DML**) y los comandos de control de transacciones **COMMIT**, **ROLLBACK**, **SAVEPOINT**
- Sin embargo, recuerde los siguientes puntos mientras usa sentencias **DML** y comandos de control de transacciones en bloques PL/SQL:
  - La palabra clave **END** indica el final de un bloque PL/SQL, no el final de una transacción.
  - PL/SQL no soporta **directamente** declaraciones de lenguaje de definición de datos (**DDL**) como **CREATE TABLE**, **ALTER TABLE** o **DROP TABLE**
    - Debe utilizar **SQL dinámico** para ejecutar las instrucciones DDL en PL/SQL
  - PL/SQL no admite directamente instrucciones de lenguaje de control de datos (**DCL**) como **GRANT** o **REVOKE**

1 - 110

## Sentencia SELECT en PL/SQL

- Utilice la instrucción `SELECT` para recuperar datos de la base de datos.

```
SELECT  select_list
INTO    {variable_name[, variable_name]...
        | record_name}
FROM    table
[WHERE  condition];
```

Directrices:

- Finalizar cada instrucción SQL con un punto y coma (;).
- Cada valor recuperado debe almacenarse en una variable utilizando la cláusula `INTO` y debe devolver 1 y sólo 1 fila (sino CURSORES)
- Especifique el mismo número de variables en la cláusula `INTO` como el número de columnas de la base de datos en la cláusula `SELECT`.
- Podemos utilizar funciones de grupo en la `SELECT`

1 - 111

## Sentencia SELECT en PL/SQL

- Se requiere la cláusula `INTO`.
- Las consultas deben devolver sólo una fila.

```
DECLARE
  v_fname VARCHAR2(25);
BEGIN
  SELECT first_name INTO v_fname
  FROM employees WHERE employee_id=200;
  DBMS_OUTPUT.PUT_LINE(' First Name is : '||v_fname);
END;
/
```

anonymous block completed  
First Name is : Jennifer

- `SELECT ... INTO` puede provocar excepciones  
`NO_DATA_FOUND` y `TOO_MANY_ROWS`

1 - 112

## Recuperación de datos en PL/SQL: Ejemplo

Recuperar hire\_date y salary para el empleado especificado.

```
DECLARE
  v_emp_hiredate  employees.hire_date%TYPE;
  v_emp_salary    employees.salary%TYPE;
BEGIN
  SELECT  hire_date, salary
  INTO    v_emp_hiredate, v_emp_salary
  FROM    employees
  WHERE   employee_id = 100;
  DBMS_OUTPUT.PUT_LINE ('Hire date is :'|| v_emp_hiredate);
  DBMS_OUTPUT.PUT_LINE ('Salary is :'|| v_emp_salary);
END;
/
```

```
anonymous block completed
Hire date is :17-JUN-03
Salary is :24000
```

1 - 113

## Recuperación de datos en PL/SQL: Ejemplo

Devolver la suma de los salarios de todos los empleados del departamento especificado.

```
DECLARE
  v_sum_sal  NUMBER(10,2);
  v_deptno  NUMBER NOT NULL := 60;
BEGIN
  SELECT  SUM(salary) -- group function
  INTO    v_sum_sal  FROM employees
  WHERE   department_id = v_deptno;
  DBMS_OUTPUT.PUT_LINE ('The sum of salary is ' || v_sum_sal);
END;
```

```
anonymous block completed
The sum of salary is 28800
```

1 - 114



## Ambigüedades

- En las sentencias SQL potencialmente ambiguas, los nombres de las **columnas** de la base de datos tienen prioridad sobre los nombres de las **variables locales**.

```
DECLARE
    hire_date      employees.hire_date%TYPE;
    sysdate        hire_date%TYPE;
    employee_id    employees.employee_id%TYPE := 176;
BEGIN
    SELECT  hire_date, sysdate
    INTO    hire_date, sysdate
    FROM    employees
    WHERE   employee_id = employee_id;
END;
/
```

OBJETIVO: Recuperar la fecha de contratación y la fecha actual en la tabla de empleados para **employee\_id=176**.

**ERROR** excepción de tiempo de ejecución en el WHERE pues nombre variables igual a las columnas

1 - 115

## Ambigüedades

```
DECLARE
    hire_date      employees.hire_date%TYPE;
    sysdate        hire_date%TYPE;
    employee_id    employees.employee_id%TYPE := 176;
BEGIN
    SELECT  hire_date, sysdate
    INTO    hire_date, sysdate
    FROM    employees
    WHERE   employee_id = employee_id;
END;
/
```

Error report:  
ORA-01422: exact fetch returns more than requested number of rows  
ORA-06512: at line 6  
01422. 00000 - "exact fetch returns more than requested number of rows"  
\*Cause: The number specified in exact fetch is less than the rows returned.  
\*Action: Rewrite the query or change number of rows requested

1 - 116

## Convenciones de nombres

- Utilice una convención de nomenclatura para evitar la ambigüedad en la cláusula `WHERE` .
- Evite el uso de nombres de columna de base de datos como identificadores.
- Los **nombres de las columnas de la tabla** de base de datos **tienen prioridad sobre los nombres de las variables locales**.
- Los **nombres de las variables** **tienen precedencia sobre los nombres de las funciones**.
- Los nombres de las **variables locales y los parámetros formales** **tienen prioridad sobre los nombres de las tablas** de la base de datos.

1 - 117

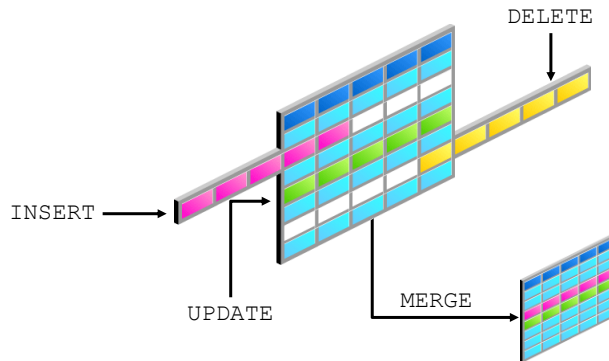
## Agenda

- Recuperación de datos con PL/SQL
- Manipulación de datos con PL/SQL
- Presentación de cursores SQL

1 - 118

## Uso PL/SQL para manipular datos

- Puede manipular datos en la base de datos mediante comandos **DML**.
- Puede emitir comandos **DML** como `INSERT`, `UPDATE`, `DELETE`, `MERGE` sin restricción en PL/SQL



1 - 119

## Uso PL/SQL para manipular datos

- Los bloqueos de fila (y bloqueos de tabla) se liberan cuando se ejecutan las instrucciones `COMMIT` o `ROLLBACK` en el código PL/SQL.
- La instrucción **MERGE** selecciona filas de una tabla para actualizar o insertar en otra tabla.
  - La decisión de actualizar o insertar en la tabla de destino se basa en una condición en la cláusula **ON**.
- **MERGE** es una orden determinista.
  - Es decir, no puede actualizar la misma fila de la tabla de destino varias veces en la misma instrucción **MERGE**.
  - Debe tener privilegios de objeto `INSERT` y `UPDATE` en la tabla de **destino** y privilegios `SELECT` en la tabla de origen

1 - 120

## Inserción de datos: Ejemplo

En el ejemplo de la diapositiva, se utiliza una sentencia `INSERT` dentro de un bloque `PL/SQL` para insertar un registro en la tabla de empleados

```
BEGIN
  INSERT INTO employees
    (employee_id, first_name, last_name, email,
     hire_date, job_id, salary)
  VALUES (employees_seq.NEXTVAL, 'Ruth', 'Cores',
           'RCORES', CURRENT_DATE, 'AD_ASST', 4000);
END;
/
```

1 - 121

## Actualización de datos: Ejemplo

Aumentar el salario de todos los empleados que son empleados de bolsa.

```
DECLARE
  sal_increase employees.salary%TYPE := 800;
BEGIN
  UPDATE      employees
  SET         salary = salary + sal_increase
  WHERE job_id = 'ST_CLERK';
END;
/
```

```
anonymous block completed
FIRST_NAME      SALARY
-----
Julia            4000
Irene            3500
James            3200
Steven           3000
. . .
Curtis           3900
Randall          3400
Peter            3300
20 rows selected
```

Puede haber ambigüedad en la cláusula **SET** de la instrucción **UPDATE** porque, aunque el identificador a la izquierda del operador de asignación es siempre una columna de base de datos, el identificador de la derecha puede ser una columna de base de datos o una variable `PL / SQL`

1 - 122

## Eliminación de datos: Ejemplo

Elimine las filas que pertenecen al departamento 10 de la tabla de employees.

```
DECLARE
  deptno employees.department_id%TYPE := 10;
BEGIN
  DELETE FROM employees
  WHERE department_id = deptno;
END;
/
```

Si no se utiliza la cláusula WHERE, todas las filas de una tabla se pueden eliminar si no hay restricciones de integridad.

1 - 123

## Combinación de filas (MERGE)

Inserte o actualice filas en la tabla copy\_emp para que coincida con la tabla de empleados.

```
BEGIN
MERGE INTO copy_emp c
  USING employees e
  ON (e.employee_id = c.empno)
  WHEN MATCHED THEN
    UPDATE SET
      c.first_name = e.first_name,
      c.last_name  = e.last_name,
      c.email      = e.email,
      . . .
  WHEN NOT MATCHED THEN
    INSERT VALUES (e.employee_id, e.first_name, e.last_name,
      . . ., e.department_id);
END;
/
```

Cada fila se inserta o actualiza en la tabla de destino dependiendo de una condición de equijoin ON

1 - 124

## Agenda

- Recuperación de datos con PL/SQL
- Manipulación de datos con PL/SQL
- Presentación de cursores SQL

1 - 125

## SQL Cursor

- Hemos visto que podemos incluir instrucciones `SQL` que devuelven una sola fila en un bloque `PL/SQL`.
- Los datos recuperados por la sentencia `SQL` deben mantenerse en variables utilizando la cláusula `INTO`.

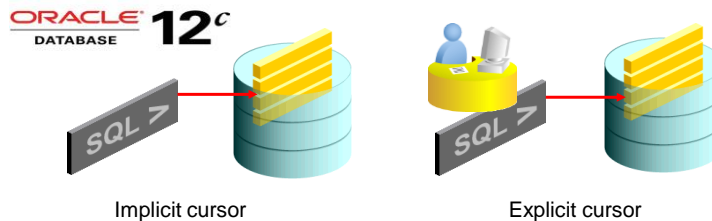
¿Qué realiza el Servidor Oracle cuando ejecuta una sentencia `SQL`?

- El servidor Oracle asigna un **área de memoria privada** denominada área de contexto para procesar sentencias `SQL`
  - La instrucción `SQL` se analiza y procesa en esta área
- Un **cursor** es un puntero al área de contexto. Sin embargo, este cursor es un **cursor implícito** y es administrado automáticamente por el servidor Oracle.
- Cuando el bloque ejecutable emite una sentencia `SQL`, `PL/SQL` crea un **cursor implícito**.

1 - 126

## SQL Cursor

- Hay dos tipos de cursores:
  - **Implicito:**
    - Creado y administrado internamente por Oracle Server para procesar sentencias SQL
  - **Explicito:**
    - Declarado explícitamente por el programador.
    - Es utilizado para recuperar varias filas de una tabla de base de datos, y solucionar el inconveniente de **SELECT .. INTO**



1 - 127

## Atributos de SQL Cursor para los cursores implícitos

Mediante los atributos de cursor SQL, puede probar el resultado de sus sentencias SQL.

<b>SQL%FOUND</b>	Atributo booleano que se evalúa como TRUE si la sentencia SQL más reciente <b>afectó</b> al menos una fila
<b>SQL%NOTFOUND</b>	Atributo booleano que se evalúa como TRUE si la sentencia SQL más reciente <b>no afectó</b> ni siquiera a una fila
<b>SQL%ROWCOUNT</b>	Un valor entero que representa el número de filas afectadas por la sentencia SQL más reciente

- Puede probar los atributos SQL% ROWCOUNT, SQL% FOUND y SQL% NOTFOUND en la sección ejecutable de un bloque para recopilar información **después de ejecutar el comando DML apropiado**

1 - 128

## Atributos de SQL Cursor para los cursores implícitos

Elimine las filas que tengan la ID de empleado especificada de la tabla de `employees` . Imprimir el número de filas eliminadas.

```
DECLARE
  v_rows_deleted VARCHAR2(30);
  v_empno employees.employee_id%TYPE := 176;
BEGIN
  DELETE FROM employees
  WHERE employee_id = v_empno;
  v_rows_deleted := (SQL%ROWCOUNT ||
                    ' row deleted. ');
  DBMS_OUTPUT.PUT_LINE (v_rows_deleted);
END;
```

1 - 129

## Quiz

Cuando se utiliza la instrucción `SELECT` en `PL/SQL`, se requiere la cláusula `INTO` y las consultas pueden devolver una o más filas.

- a. True
- b. False

1 - 130



## Resumen

En esta lección, debes haber aprendido a:

- Determinar las sentencias de SQL que se pueden incluir directamente en un bloque ejecutable de PL/SQL
- Manipular datos con instrucciones DML en PL/SQL
- Utilizar instrucciones de control de transacciones en PL/SQL
- Hacer uso de la cláusula `INTO` para mantener los valores devueltos por una instrucción SQL
- Diferenciar entre cursores implícitos y cursores explícitos
- Utilizar atributos de cursor SQL

1 - 131

## Prácticas 5

En esta lección, realiza las siguientes prácticas:

- Selección de datos de una tabla
- Inserción de datos en una tabla
- Actualización de datos en una tabla
- Eliminar un registro de una tabla

1 - 132

# 6

## Estructuras de control



ORACLE®

### Objetivos

Después de completar esta lección, usted debería ser capaz de:

- Identificar los usos y tipos de estructuras de control
- Construir una declaración **IF**
- Utilizar instrucciones **CASE** y expresiones **CASE**
- Construir e identificar los enunciados del bucle
- Utilice pautas cuando use estructuras de control condicional

## Control del flujo de ejecución

- Puede cambiar el flujo lógico de sentencias dentro del bloque PL/SQL con varias estructuras de control
- Esta lección aborda cuatro tipos de estructuras de control PL/SQL:

- Condicionales IF
- Expresiones CASE
- Estructuras LOOP
- EXIT y CONTINUE

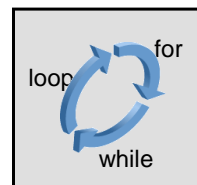
```
IF...  
THEN..  
END IF;
```

```
IF...  
THEN..  
ELSE..  
END IF;
```

```
IF...  
THEN..  
ELSIF..  
THEN..  
END IF;
```

```
IF...  
THEN..  
ELSIF..  
THEN..  
ELSE..  
END IF;
```

```
CASE  
WHEN... THEN..  
WHEN... THEN..  
WHEN... THEN..  
ELSE  
END CASE;
```



1 - 135

## Agenda

- Uso de instrucciones IF
- Utilizar instrucciones CASE y expresiones CASE
- Construir e identificar las sentencias para bucles

1 - 136

## Sentencia IF

- La estructura IF de PL/SQL es similar a la estructura de las sentencias IF en otros lenguajes procedimentales.
- Permite PL/SQL realizar acciones selectivamente basadas en condiciones.
- Sintaxis

```
IF condition THEN
    statements;
[ELSIF condition THEN
    statements;]
[ELSE
    statements;]
END IF;
```

**Note:** ELSIF y ELSE son opcionales en una instrucción IF.

1 - 137

## Simple Sentencia IF

```
DECLARE
    v_myage number:=31;
BEGIN
    IF v_myage < 11
    THEN
        DBMS_OUTPUT.PUT_LINE(' I am a child ');
    END IF;
END;
/
```

anonymous block completed

- Una instrucción IF puede tener múltiples expresiones condicionales relacionadas con operadores lógicos como AND, OR y NOT.

```
IF (myfirstname = 'Christopher' AND v_myage <11) ...
```

1 - 138

## Sentencia IF THEN ELSE

- La cláusula **THEN** sólo se ejecutan si la condición devuelve **TRUE**.
- La cláusula **ELSE** sólo se ejecuta si la condición devuelve **FALSE**

```
DECLARE
  v_myage  number:=31;
BEGIN
  IF v_myage < 11 THEN
    DBMS_OUTPUT.PUT_LINE(' I am a child ');
  ELSE
    DBMS_OUTPUT.PUT_LINE(' I am not a child ');
  END IF;
END;
/
```

anonymous block completed  
I am not a child

1 - 139

## Clausula IF ELSIF ELSE

- La cláusula **IF** puede contener varias cláusulas **ELSIF** y una cláusula **ELSE**.
- Las cláusulas **ELSIF** pueden tener condiciones, a diferencia de la cláusula **ELSE** que no puede tener condiciones.
- La condición para **ELSIF** debe ser seguida por la cláusula **THEN**, que se ejecuta si la condición para **ELSIF** devuelve **TRUE**.
- Cuando tiene varias cláusulas **ELSIF**, si la primera condición es **FALSE** o **NULL**, el control cambia a la siguiente cláusula **ELSIF**.
- Las condiciones se evalúan una por una desde arriba.
- Si todas las condiciones son **FALSE** o **NULL**, las sentencias de la cláusula **ELSE** se ejecutan.

1 - 140

## Clausula IF ELSIF ELSE

```
DECLARE
  v_myage number:=31;
BEGIN
  IF v_myage < 11 THEN
    DBMS_OUTPUT.PUT_LINE(' I am a child ');
  ELSIF v_myage < 20 THEN
    DBMS_OUTPUT.PUT_LINE(' I am young ');
  ELSIF v_myage < 30 THEN
    DBMS_OUTPUT.PUT_LINE(' I am in my twenties');
  ELSIF v_myage < 40 THEN
    DBMS_OUTPUT.PUT_LINE(' I am in my thirties');
  ELSE
    DBMS_OUTPUT.PUT_LINE(' I am always young ');
  END IF;
END;
/
```

```
anonymous block completed
I am in my thirties
```

1 - 141

## Valor NULL en la declaración IF

- La condición en la instrucción IF devuelve NULL, el control pasa a la instrucción ELSE.
- En el ejemplo **v\_myage** no esta inicializada por lo tanto NULL

```
DECLARE
  v_myage number;
BEGIN
  IF v_myage < 11 THEN
    DBMS_OUTPUT.PUT_LINE(' I am a child ');
  ELSE
    DBMS_OUTPUT.PUT_LINE(' I am not a child ');
  END IF;
END;
/
```

```
anonymous block completed
I am not a child
```

1 - 142

## Agenda

- Uso de instrucciones IF
- Utilizar instrucciones CASE y expresiones CASE
- Construir e identificar las sentencias para bucles

1 - 143

## Expresiones CASE

- Una expresión **CASE** devuelve un resultado basado en una o más alternativas
- Para devolver el resultado, la expresión **CASE** utiliza un *selector*, que es una expresión o valor.
- El *selector* es seguido por una o más cláusulas **WHEN** que se comprueban **secuencialmente**
- Si el valor del *selector* es igual al valor de **WHEN** , se ejecuta esa cláusula **WHEN** y se devuelve ese resultado
- Si el valor del *selector* no es encontrado, se ejecuta la cláusula **ELSE** si existe

1 - 144

## Expresiones CASE

- Sintaxis

```
CASE selector
  WHEN expression1 THEN sentencias1
  [WHEN expression2 THEN sentencias2
  ...
  WHEN expressionN THEN sentenciasN]
  [ELSE sentenciasN+1]
END;
```

1 - 145

## Expresiones CASE

- Sintaxis

```
V_devuelto := CASE selector
  WHEN expression1 THEN 'valor1'.
  ...
  [ELSE sentenciasN+1]
END;
```

- La expresión CASE se puede asignar a una variable y devolver un valor en función del WHEN

1 - 146



## Expresiones CASE: Ejemplos

```
SET VERIFY OFF
DECLARE
    v_grade CHAR(1) := UPPER('&grade');
    v_appraisal VARCHAR2(20);
BEGIN
    v_appraisal := CASE v_grade
        WHEN 'A' THEN 'Excellent'
        WHEN 'B' THEN 'Very Good'
        WHEN 'C' THEN 'Good'
        ELSE 'No such grade'
    END;
    DBMS_OUTPUT.PUT_LINE ('Grade: ' || v_grade ||
                          'Appraisal' || v_appraisal);
END;
/
```

1 - 147

## Expresiones CASE: Desigualdad

- Como norma general, las expresiones CASE utilizan el selector para ser comparado con los diferentes valores.
- Esta comparación se realiza mediante la igualdad.
- También se puede utilizar CASE para realizar comparaciones de desigualdad y diferentes condiciones de comparación.
- En este tipo de CASE, no tiene una selector. En su lugar, la cláusula WHEN contiene una expresión que da como resultado un valor booleano

1 - 148

## Expresiones CASE: Desigualdad

```
DECLARE
    v_grade CHAR(1) := UPPER('&grade');
    v_appraisal VARCHAR2(20);
BEGIN
    v_appraisal := CASE
        WHEN v_grade = 'A' THEN 'Excellent'
        WHEN v_grade IN ('B','C') THEN 'Good'
        ELSE 'No such grade'
    END;
    DBMS_OUTPUT.PUT_LINE ('Grade: ' || v_grade ||
                          ' Appraisal ' || v_appraisal);
END;
/
```

1 - 149

## Sentencia CASE: Ejemplo

```
DECLARE
    v_deptid NUMBER;
    v_deptname VARCHAR2(20);
    v_emps NUMBER;
    v_mgrid NUMBER:= 108;
BEGIN
    CASE v_mgrid
    WHEN 108 THEN
        SELECT department_id, department_name
        INTO v_deptid, v_deptname FROM departments
        WHERE manager_id=108;
        SELECT count(*) INTO v_emps FROM employees
        WHERE department_id=v_deptid;
    WHEN 200 THEN
        ...
    END CASE;
    DBMS_OUTPUT.PUT_LINE ('You are working in the ' || v_deptname ||
                          ' department. There are ' || v_emps || ' employees in this
                          department');
END;
/
```

1 - 150

## Manejo de valores nulos

Cuando trabaja con valores nulos, puede evitar algunos errores comunes teniendo en cuenta las siguientes reglas:

- Las comparaciones simples que implican valores nulos siempre producen `NULL`.
- Aplicar el operador lógico `NOT` a un `NULL` produce `NULL`.
- Si la condición produce `NULL` en sentencias de control condicional (`IF`), su secuencia asociada de sentencias no se ejecuta, se ejecutaría el `ELSE`

1 - 151

## Tablas de lógica

Crear una condición Booleana simple con un operador de comparación.

AND	TRUE	FALSE	NULL	OR	TRUE	FALSE	NULL	NOT	
TRUE	TRUE	FALSE	NULL	TRUE	TRUE	TRUE	TRUE	TRUE	FALSE
FALSE	FALSE	FALSE	FALSE	FALSE	TRUE	FALSE	NULL	FALSE	TRUE
NULL	NULL	FALSE	NULL	NULL	TRUE	NULL	NULL	NULL	NULL

1 - 152

## Expresión booleana o expresión lógica?

¿Cuál es el valor de `flag` en cada caso?

```
flag := reorder_flag AND available_flag;
```

REORDER_FLAG	AVAILABLE_FLAG	FLAG	
TRUE	TRUE	? (1)	TRUE
TRUE	FALSE	? (2)	FALSE
NULL	TRUE	? (3)	NULL
NULL	FALSE	? (4)	FALSE

1 - 153

## Agenda

- Uso de instrucciones IF
- Utilizar instrucciones CASE y expresiones CASE
- Construir e identificar las sentencias para bucles

1 - 154

## Control iterativo: instrucciones LOOP

- PL/SQL proporciona varias facilidades para crear bucles con el objetivo de repetir una sentencia o secuencia de sentencias varias veces
- Es obligatorio tener una condición de salida en un bucle; De lo contrario, el bucle es infinito.
- PL/SQL proporciona los siguientes tipos de bucles:
  - Bucle **básico (LOOP)**:
    - Realiza acciones repetitivas sin condiciones generales
  - Bucles **FOR**
    - Realizan acciones iterativas basadas en un contador
  - Bucles **WHILE**
    - Realizan acciones iterativas basadas en una condición



1 - 155

## Bucle Básico (LOOP)

- El bucle básico es una instrucción **LOOP**, que encierra una secuencia de instrucciones entre las palabras clave **LOOP** y **END LOOP**
- ```
LOOP
    statement1;
    . . .
    EXIT [WHEN condition];
END LOOP;
```
- Cada vez que el flujo de ejecución llega a la instrucción **END LOOP**, el control se devuelve a la instrucción **LOOP**
  - El bucle básico debe de incluir al menos una instrucción **EXIT** para posibilitar la salida del bucle.
    - Sin la instrucción **EXIT**, el bucle **sería infinito**.

1 - 156

## Bucle Básico: Ejemplo

- Un bucle básico permite la ejecución de sus sentencias hasta que se cumple la condición **EXIT WHEN**.

```
DECLARE
  v_countryid    locations.country_id%TYPE := 'CA';
  v_loc_id       locations.location_id%TYPE;
  v_counter      NUMBER(2) := 1;
  v_new_city     locations.city%TYPE := 'Montreal';
BEGIN
  SELECT MAX(location_id) INTO v_loc_id FROM locations
  WHERE country_id = v_countryid;
  LOOP
    INSERT INTO locations(location_id, city, country_id)
    VALUES((v_loc_id + v_counter), v_new_city, v_countryid);
    v_counter := v_counter + 1;
    EXIT WHEN v_counter > 3;
  END LOOP;
END;
/
```

1 - 157

## Bucle WHILE

- Puede utilizar el bucle **WHILE** para repetir una secuencia de instrucciones mientras la condición es **TRUE**:

```
WHILE condition LOOP
  statement1;
  statement2;
  . . .
END LOOP;
```

- La condición se evalúa al inicio de cada iteración.
- El bucle finaliza cuando la condición es **FALSE** o **NULL**.
  - Se puede ejecutar entre 0 y N veces
    - Si la condición es **FALSE** o **NULL** al inicio del bucle, no se realizan más iteraciones.
- Si la condición produce **NULL**, el bucle se pasa por alto y el control pasa a la siguiente instrucción.

1 - 158

## Bucle WHILE: Ejemplo

```
DECLARE
  v_countryid  locations.country_id%TYPE := 'CA';
  v_loc_id     locations.location_id%TYPE;
  v_new_city   locations.city%TYPE := 'Montreal';
  v_counter    NUMBER := 1;
BEGIN
  SELECT MAX(location_id) INTO v_loc_id FROM locations
  WHERE country_id = v_countryid;
  WHILE v_counter <= 3 LOOP
    INSERT INTO locations(location_id, city, country_id)
    VALUES((v_loc_id + v_counter), v_new_city, v_countryid);
    v_counter := v_counter + 1;
  END LOOP;
END;
/
```

1 - 159

## Bucles FOR

- Los bucles FOR tienen la misma estructura general que el bucle básico.
- Se define una instrucción de control antes de la palabra clave LOOP para establecer el número de iteraciones que realiza el PL/SQL.

```
FOR counter IN [REVERSE] lower_bound..upper_bound
LOOP
  statement1;
  statement2;
  . . .
END LOOP;
```

- El contador no se declara, es declarado implícitamente.

1 - 160

## Bucles FOR

- La secuencia de instrucciones se ejecuta cada vez que el contador se incrementa, según lo determinado por los **dos límites**
- Los límites inferiores y superiores del intervalo de bucle pueden ser **literales, variables o expresiones**, pero deben evaluarse como **enteros**
  - Los límites se redondean a números enteros sino lo son
- El límite inferior y el límite superior están incluidos dentro del intervalo de bucle.
  - Si el límite inferior del intervalo de bucle se evalúa a un entero más grande que el límite superior, la secuencia de sentencias no se ejecuta.
- El bucle se incrementa siempre de 1 en 1 unidad ( no step by ).

1 - 161

## Bucles FOR: Ejemplos

```
DECLARE
  v_countryid  locations.country_id%TYPE := 'CA';
  v_loc_id     locations.location_id%TYPE;
  v_new_city   locations.city%TYPE := 'Montreal';
BEGIN
  SELECT MAX(location_id) INTO v_loc_id
  FROM locations
  WHERE country_id = v_countryid;
  FOR i IN 1..3 LOOP
    INSERT INTO locations(location_id, city, country_id)
    VALUES((v_loc_id + i), v_new_city, v_countryid );
  END LOOP;
END;
/
```

1 - 162



## Bucles FOR: Reglas

- Referencia del contador sólo dentro del bucle; Es indefinido fuera del bucle.
- No haga referencia al contador como el objetivo de una asignación.
- Ningún límite de bucle debe ser `NULL`.
- Los límites inferior y superior de una instrucción `LOOP` no necesitan ser literales numéricos. Pueden ser expresiones que se convierten a valores numéricos

1 - 163

## Uso Sugerido de bucles

- Utilice el bucle básico (`LOOP`) cuando las sentencias dentro del bucle deben ejecutarse al menos una vez.
  - Sin la instrucción `EXIT`, el bucle sería infinito
- Utilice el bucle `WHILE` si la condición debe ser evaluada al inicio de cada iteración.
  - El bucle termina cuando la condición es `FALSE`.
  - Si la condición es `FALSE` al inicio del bucle, no se realizan más iteraciones.
- Utilice un bucle `FOR` si se conoce el número de iteraciones.

1 - 164

## Bucles anidados y etiquetas

- Puede anidar los bucles `FOR`, `WHILE` y básicos (`LOOP`) dentro de otro bucles.
- La terminación de un bucle interno no termina el bucle externo a menos que se genere una excepción.
  - Sin embargo, puede **etiquetar** bucles y salir del bucle externo con la instrucción `EXIT`.
- Una **etiqueta** se coloca antes de una declaración, ya sea en la misma línea o en una línea separada, dentro de los delimitadores de etiquetas `<< label >>`
  - **Básicos** : Antes de la palabra `LOOP`
  - **FOR y WHILE** : Antes de las palabras `FOR` o `WHILE`

1 - 165

## Bucles anidados y etiquetas: Ejemplo

```
...
BEGIN
  <<Outer_loop>>
  LOOP
    v_counter := v_counter+1;
    EXIT WHEN v_counter>10;
    <<Inner_loop>>
    LOOP
      ...
      EXIT Outer_loop WHEN total_done = 'YES';
      -- Leave both loops
      EXIT WHEN inner_done = 'YES';
      -- Leave inner loop only
      ...
    END LOOP Inner_loop;
    ...
  END LOOP Outer_loop;
END;
/
```

1 - 166

## Sentencia CONTINUE

- La instrucción **CONTINUE** permite transferir el control dentro de un bucle a una nueva iteración o dejar el bucle.
- Definición
  - Añade la funcionalidad para iniciar la siguiente iteración de bucle
  - Proporciona a los programadores la capacidad de transferir el control a la siguiente iteración de un bucle
  - Usa estructura y semántica paralelas a la sentencia **EXIT**
- Beneficios
  - Facilita el proceso de programación
  - Puede proporcionar una pequeña mejora de rendimiento sobre las soluciones anteriores de programación para simular la instrucción **CONTINUE**



1 - 167

## Sentencia CONTINUE: Ejemplo1

```
DECLARE
  v_total SIMPLE_INTEGER := 0;
BEGIN
  FOR i IN 1..10 LOOP
    1 v_total := v_total + i;
    dbms_output.put_line
      ('Total is: ' || v_total);
    2 CONTINUE WHEN i > 5;
    v_total := v_total + i;
    dbms_output.put_line
      ('Out of Loop Total is:
        ' || v_total);
    END LOOP;
  END;
/
```

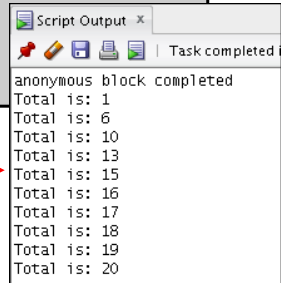
Script Output x  
Task compl

anonymous block completed  
Total is: 1  
Out of Loop Total is:  
2  
Total is: 4  
Out of Loop Total is:  
6  
Total is: 9  
Out of Loop Total is:  
12  
Total is: 16  
Out of Loop Total is:  
20  
Total is: 25  
Out of Loop Total is:  
30  
Total is: 36  
Total is: 43  
Total is: 51  
Total is: 60  
Total is: 70

1 - 168

## Sentencia CONTINUE: Ejemplo2

```
DECLARE
v_total NUMBER := 0;
BEGIN
<<BeforeTopLoop>>
FOR i IN 1..10 LOOP
v_total := v_total + 1;
dbms_output.put_line
('Total is: ' || v_total);
FOR j IN 1..10 LOOP
CONTINUE BeforeTopLoop WHEN i + j > 5;
v_total := v_total + 1;
END LOOP;
END LOOP BeforeTopLoop;
END;
```



Script Output x

Task completed

anonymous block completed

Total is: 1  
Total is: 6  
Total is: 10  
Total is: 13  
Total is: 15  
Total is: 16  
Total is: 17  
Total is: 18  
Total is: 19  
Total is: 20

1 - 169

## Quiz

Hay tres tipos de bucles: básico, FOR y WHILE.

- a. True
- b. False

1 - 170

## Resumen

En esta lección, debería haber aprendido a cambiar el flujo lógico de sentencias utilizando las siguientes estructuras de control:

- Condicional (declaración `IF`)
- Expresiones `CASE` y sentencias `CASE`
- Bucles
  - Basic loop
  - `FOR` loop
  - `WHILE` loop
- Instrucción `EXIT`
- Instrucción `CONTINUE`

1 - 171

## Práctica 6

En esta lección, realiza las siguientes prácticas:

- Realización de acciones condicionales mediante el uso de instrucciones `IF`
- Realización de pasos iterativos mediante el uso de estructuras `LOOP`

1 - 172

# 7

## Trabajo con tipos de datos compuestos



ORACLE®

### Objetivos

Después de completar esta lección, usted debería ser capaz de:

- Describir las colecciones y registros de PL/SQL
- Crear registros PL/SQL definidos por el usuario
- Cree un registro PL/SQL con el atributo `%ROWTYPE`

## Agenda

- Examinando los tipos de datos compuestos
- Uso de registros PL/SQL
  - Manipulación de datos con registros PL/SQL
  - Ventajas del atributo %ROWTYPE

1 - 175

## Tipos de datos compuestos

- Como hemos visto anteriormente, las variables del tipo de datos **escalares** pueden contener sólo un valor
- Las variables del tipo de **datos compuesto** pueden contener varios valores del tipo de **datos escalares** o del **tipo de datos compuesto**.
- ¿Por qué utilizar los tipos de datos compuestos?
  - Disponemos de todos los datos bajo una sola una sola unidad.
  - Puede acceder y modificar datos fácilmente.
  - Los datos son más fáciles de manejar, relacionar y transportar.

Una analogía es tener una sola bolsa para todos los componentes de su computadora portátil en lugar de una bolsa separada para cada componente.

1 - 176


# Tipos de datos compuestos

- Hay dos tipos de tipos de datos compuestos:
  - **Registros PL/SQL**
    - Los registros se usan para agrupar bajo un mismo nombre un conjunto de datos relacionados entre si ( Datos de un cliente)
    - Un registro PL/SQL puede tener variables de diferentes tipos.
  - **Colecciones PL/SQL**
    - Las colecciones se utilizan para tratar los datos como una sola unidad. (Array)
    - Las colecciones son de tres tipos:
      - `Array asociativo`
      - `Tabla anidada`
      - `VARRAY`

## registros PL/SQL o colecciones?

- Utilice **registros PL/SQL** cuando desee almacenar valores de diferentes tipos de datos, pero sólo una ocurrencia a la vez.
- Utilice **colecciones PL/SQL** cuando desee almacenar valores del mismo tipo de datos y muchas ocurrencias.

Registro PL/SQL :

|      |           |         |                                                                                     |
|------|-----------|---------|-------------------------------------------------------------------------------------|
| TRUE | 23-DEC-98 | ATLANTA |  |
|------|-----------|---------|-------------------------------------------------------------------------------------|

Colección PL/SQL :

|   |         |
|---|---------|
| 1 | SMITH   |
| 2 | JONES   |
| 3 | BENNETT |
| 4 | KRAMER  |

Diagram illustrating data storage structures:

- Registro PL/SQL :** A single record containing four fields: TRUE, 23-DEC-98, ATLANTA, and a 3D purple flower icon.
- Colección PL/SQL :** A collection of four records, each with an index and a name:
  - 1 SMITH
  - 2 JONES
  - 3 BENNETT
  - 4 KRAMER

Annotations for the collection:

- Red arrows point from the text **PLS\_INTEGER** to the index column (1, 2, 3, 4).
- Red arrows point from the text **VARCHAR2** to the name column (SMITH, JONES, BENNETT, KRAMER).



## Agenda

- Examinando los tipos de datos compuestos
- Uso de registros PL/SQL
  - Manipulación de datos con registros PL/SQL
  - Ventajas del atributo %ROWTYPE

1 - 179

## Registros PL/SQL

- Un registro es un grupo de elementos de datos relacionados almacenados en **campos**, cada uno con su propio nombre y tipo de datos.
  - Son similares a las estructuras en la mayoría de los lenguajes de tercera generación (incluyendo C y C ++)
- Los registros pueden asignarse a valores iniciales y se pueden definir como NOT NULL.
  - Los campos sin valores iniciales se inicializan en NULL.
- La palabra clave **DEFAULT** así como **:=** se puede utilizar en la inicialización de campos.
- Puede definir los tipos **RECORD** y declarar los registros definidos por el usuario en la parte **declarativa**, **subprograma** o **paquete**

1 - 180

## Creación de un registro PL/SQL

Sintaxis:

```
1  TYPE type_name IS RECORD
    (field_declaration[, field_declaration]...);

2  identifier type_name;

    field_declaration:

    field_name {field_type | variable%TYPE
               | table.column%TYPE | table%ROWTYPE}
               [[NOT NULL] {:= | DEFAULT} expr]
```

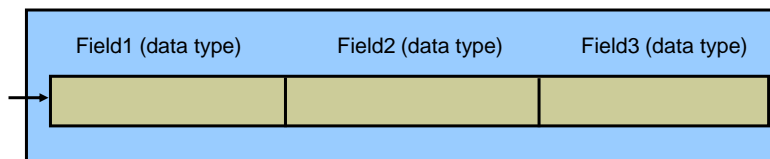
Type\_name      Es el nombre del tipo RECORD  
Field\_name      Es el nombre de un campo dentro del registro  
Field\_type      Es el tipo de datos del campo

1 - 181

## Estructura del registro PL/SQL

- Los campos de un registro se acceden con el nombre del registro.
- Para referenciar o inicializar un campo individual, utilice la notación de puntos:

`Nombre_record.nombre_campo`



1 - 182

## Estructura del registro PL/SQL

Ejemplo:

| Field1 (data type)    | Field2 (data type)     | Field3 (data type)  |
|-----------------------|------------------------|---------------------|
| employee_id number(6) | last_name varchar2(25) | job_id varchar2(10) |
| 100                   | King                   | AD_PRES             |

- Por ejemplo, hace referencia al campo `job_id` en el registro `emp_record` como sigue:  
`Emp_record.job_id`
- A continuación, puede asignar un valor al campo de registro:  
`Emp_record.job_id := 'ST_CLERK';`
- En un bloque o subprograma, los registros definidos por el usuario se instancian cuando se introduce el bloque o subprograma.

1 - 183

## Creación de un registro PL/SQL: Ejemplo

```
DECLARE
  TYPE t_rec IS RECORD
    (v_sal number(8),
     v_minsal number(8) default 1000,
     v_hire_date employees.hire_date%type,
     v_recl employees%rowtype);
  v_myrec t_rec;
BEGIN
  v_myrec.v_sal := v_myrec.v_minsal + 500;
  v_myrec.v_hire_date := sysdate;
  SELECT * INTO v_myrec.v_recl
    FROM employees WHERE employee_id = 100;
  DBMS_OUTPUT.PUT_LINE(v_myrec.v_recl.last_name || ' ' ||
    to_char(v_myrec.v_hire_date) || ' ' || to_char(v_myrec.v_sal));
END;
```

```
anonymous block completed
King 16-OCT-12 1500
```

1 - 184

## Atributo %ROWTYPE

- Se ha aprendido que **%TYPE** se utiliza para declarar una variable del tipo de columna.
  - La variable tiene el mismo tipo de datos y el mismo tamaño que la columna de la tabla.
- El beneficio de **%TYPE** es que no es necesario cambiar la variable si se modifica la columna
- El atributo **%ROWTYPE** se utiliza para declarar un registro que puede contener una fila entera de una tabla o vista.
  - Los campos del registro toman sus nombres y tipos de datos de las columnas de la tabla o vista.
  - El registro también puede almacenar una fila entera de datos extraídos de una variable de cursor o cursor.

1 - 185

## Atributo %ROWTYPE

Sintaxis:

```
DECLARE  
  identificador      Table%ROWTYPE;
```

- Declare una variable de acuerdo con el conjunto de columnas en una tabla o vista de base de datos.
- Prefijo **%ROWTYPE** deberemos anteponerle la tabla o vista de base de datos.
- Los campos del registro toman sus nombres y tipos de datos de las columnas de la tabla o vista.
- Puede asignar una lista de valores comunes a un registro mediante la instrucción **SELECT** o **FETCH**
- También puede asignar un registro a otro si ambos tienen los mismos tipos de datos correspondientes

1 - 186

## Ventajas de usar el atributo %ROWTYPE

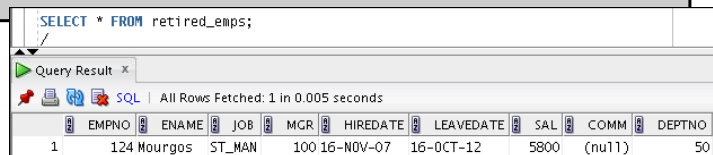
### Ventajas

- No es necesario conocer el número y los tipos de datos de las columnas de la base de datos subyacente.
- El uso de %ROWTYPE garantiza que los tipos de datos de las variables declaradas con este atributo cambian dinámicamente cuando se altera la tabla subyacente
- El atributo %ROWTYPE es útil cuando desea recuperar una fila con:
  - La sentencia SELECT \*
  - Declaraciones INSERT y UPDATE de nivel de fila
- El uso de %ROWTYPE simplifica el mantenimiento de código.

1 - 187

## Atributo %ROWTYPE : ejemplo

```
DECLARE
  v_employee_number number:= 124;
  v_emp_rec employees%ROWTYPE;
BEGIN
  SELECT * INTO v_emp_rec FROM employees
  WHERE employee_id = v_employee_number;
  INSERT INTO retired_emps(empno, ename, job, mgr,
                          hiredate, leavedate, sal, comm, deptno)
  VALUES (v_emp_rec.employee_id, v_emp_rec.last_name,
          v_emp_rec.job_id, v_emp_rec.manager_id,
          v_emp_rec.hire_date, SYSDATE,
          v_emp_rec.salary, v_emp_rec.commission_pct,
          v_emp_rec.department_id);
END;
```



The screenshot shows a SQL query window with the command `SELECT * FROM retired_emps;` and a 'Query Result' window displaying the following data:

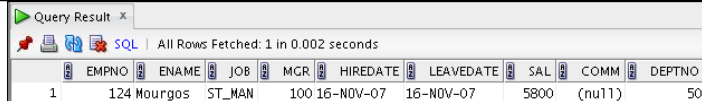
| EMPNO | ENAME       | JOB    | MGR | HIREDATE  | LEAVEDATE | SAL  | COMM   | DEPTNO |
|-------|-------------|--------|-----|-----------|-----------|------|--------|--------|
| 1     | 124 Mourgos | ST_MAN | 100 | 16-NOV-07 | 16-OCT-12 | 5800 | (null) | 50     |

1 - 188

## Insertar un registro utilizando %ROWTYPE

```
...  
DECLARE  
    v_employee_number number:= 124;  
    v_emp_rec retired_emps%ROWTYPE;  
BEGIN  
    SELECT employee_id, last_name, job_id, manager_id,  
           hire_date, hire_date, salary, commission_pct,  
           department_id INTO v_emp_rec FROM employees  
    WHERE employee_id = v_employee_number;  
    INSERT INTO retired_emps VALUES v_emp_rec;  
END;  
/  
SELECT * FROM retired_emps;
```

El número de campos en el registro debe ser igual al número de nombres de campo en la cláusula INTO



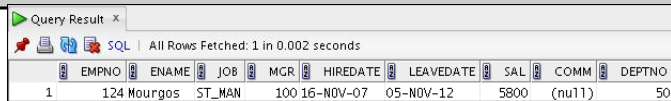
| EMPNO | ENAME       | JOB    | MGR | HIREDATE  | LEAVEDATE | SAL  | COMM   | DEPTNO |
|-------|-------------|--------|-----|-----------|-----------|------|--------|--------|
| 1     | 124 Mourgos | ST_MAN | 100 | 16-NOV-07 | 16-NOV-07 | 5800 | (null) | 50     |

1 - 189

## Actualizar un registro utilizando %ROWTYPE

```
DECLARE  
    v_employee_number number:= 124;  
    v_emp_rec retired_emps%ROWTYPE;  
BEGIN  
    SELECT * INTO v_emp_rec FROM retired_emps WHERE  
    empno = v_employee_number;  
    v_emp_rec.leavedate:= CURRENT_DATE;  
    UPDATE retired_emps SET ROW = v_emp_rec  
    WHERE empno=v_employee_number;  
END;  
/  
SELECT * FROM retired_emps;
```

La palabra clave ROW se utiliza para representar toda la fila.



| EMPNO | ENAME       | JOB    | MGR | HIREDATE  | LEAVEDATE | SAL  | COMM   | DEPTNO |
|-------|-------------|--------|-----|-----------|-----------|------|--------|--------|
| 1     | 124 Mourgos | ST_MAN | 100 | 16-NOV-07 | 05-NOV-12 | 5800 | (null) | 50     |

1 - 190

## Agenda

- Examinando los tipos de datos compuestos
- Uso de registros PL/SQL
  - Manipulación de datos con registros PL/SQL
  - Ventajas del atributo %ROWTYPE
- Uso de colecciones PL/SQL
  - Examinar arrays asociativos
  - Introducción a las tablas anidadas
  - Introducción a las VARRAY

1 - 191

## Arrays Asociativos (Tablas INDEX BY)

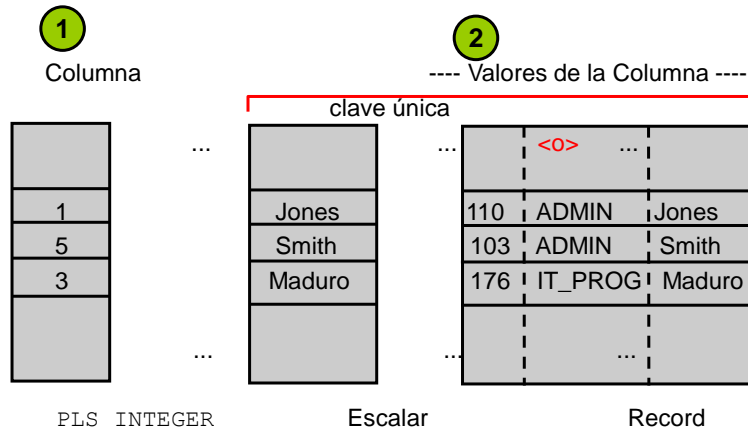
- Una `array asociativo` es un tipo de colección PL/SQL
  - Es un tipo de **datos compuesto** y está **definido por el usuario**
- Los `arrays asociativos` son conjuntos de pares **clave-valor**
  - Pueden almacenar datos utilizando un valor de clave principal como índice, donde los valores de clave no son necesariamente secuenciales.
- Los `arrays asociativos` también se conocen como tablas **INDEX BY**
- Las matrices asociativas tienen sólo dos columnas:
  - La primera columna
    - de tipo entero o de cadena, actúa como **clave principal**.
  - La segunda columna
    - de tipo escalar o de datos de registro, **contiene valores**.

| Key | Values |
|-----|--------|
| 1   | JONES  |
| 2   | HARDEY |
| 3   | MADURO |
| 4   | KRAMER |

1 - 192

## Estructura de Arrays Asociativos

- La **primera** columna contiene una clave única
- La **segunda** columna contiene un único valor o un conjunto de valores



1 - 193

## Estructura de Arrays Asociativos

### Columna clave única

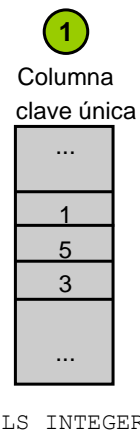
- El tipo de datos de la columna clave puede ser:

- Numéricos
  - `BINARY_INTEGER`
  - `PLS_INTEGER`

Estos dos tipos de datos numéricos requieren menos almacenamiento que `NUMBER`.

Las operaciones aritméticas en estos tipos de datos son más rápidas que la aritmética `NUMBER`.

- `VARCHAR2` o un subtipo suyo



1 - 194



## Estructura de Arrays Asociativos

### Columna VALOR

- La columna de valores puede ser un tipo:

- **Escalar**

Solo contiene un valor por fila

- **Tipo registro**

puede contener varios valores por fila.

#### NOTA:

- Una matriz asociativa no se rellena en el momento de la declaración
- Se requiere una instrucción explícita para rellenar la matriz asociativa

2

---- Valores de la Columna ----

|        |
|--------|
|        |
| Jones  |
| Smith  |
| Maduro |
|        |

Escalar

|     |         |        |
|-----|---------|--------|
| ... | <0>     | ...    |
| 110 | ADMIN   | Jones  |
| 105 | ADMIN   | Smith  |
| 170 | IT_PROG | Maduro |
| ... | ...     | ...    |

Record

1 - 195

## Pasos para crear un Array asociativo

- 1.- Declare un **tipo** de datos TABLE utilizando la opción INDEX BY:

```
1  TYPE type_name IS TABLE OF
   { column_type [NOT NULL] | variable%TYPE [NOT NULL]
   | table.column%TYPE [NOT NULL]
   | table%ROWTYPE }
   INDEX BY { PLS_INTEGER | BINARY_INTEGER
   | VARCHAR2(<size>) } ;
2  identifier type_name;
```

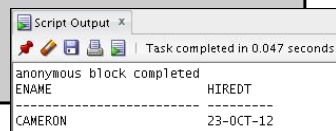
- 2.- Declare una variable de ese **tipo** de datos.

```
...
TYPE ename_table_type IS TABLE OF
employees.last_name%TYPE
INDEX BY PLS_INTEGER;
...
ename_table ename_table_type;
```

1 - 196

## Creación y accediendo un Array asociativo

```
...  
DECLARE  
  TYPE ename_table_type IS TABLE OF  
    employees.last_name%TYPE  
    INDEX BY PLS_INTEGER;  
  TYPE hiredate_table_type IS TABLE OF DATE  
    INDEX BY PLS_INTEGER;  
  ename_table      ename_table_type;  
  hiredate_table   hiredate_table_type;  
BEGIN  
  ename_table(1)   := 'CAMERON';  
  hiredate_table(8) := SYSDATE + 7;  
  IF ename_table.EXISTS(1) THEN  
    INSERT INTO ...  
    ...  
END;  
/  
...
```



Script Output x

Task completed in 0.047 seconds

anonymous block completed

| ENAME   | HIREDT    |
|---------|-----------|
| CAMERON | 23-OCT-12 |

1 - 197

## Uso de los métodos de INDEX BY

- Un método de tabla INDEX BY es un procedimiento o función incorporado que opera en el array asociativo creado.
- Son llamados mediante la utilización de la notación de puntos.

```
Array.method_name[ (parameters) ]
```

Los métodos existentes son:

- EXISTS
- COUNT
- FIRST
- LAST
- PRIOR
- NEXT
- DELETE

1 - 198

## Tabla INDEX BY de Registros de Filas

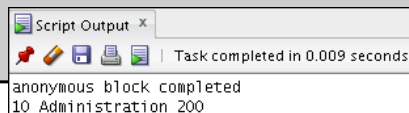
- Como se comentó anteriormente, un array asociativo se puede crear para guardar tipos de datos escalares.
- Sin embargo, a menudo es necesario almacenar **todas las columnas recuperadas por una consulta**.
- La opción **TABLE OF Tipo** permite definir un array asociativo que contenga información sobre todos los campos de una tabla de base de datos.
- Esto se puede conseguir utilizando el atributo **%ROWTYPE** como componente del array asociativo, según ejemplo siguiente.

1 - 199

## Tabla INDEX BY de Registros de Filas

Definir un array asociativo para guardar filas completas de una tabla

```
DECLARE
  TYPE dept_table_type IS TABLE OF departments%ROWTYPE
  INDEX BY VARCHAR2(20);
  dept_table dept_table_type;
  -- Each element of dept_table is a record
BEGIN
  SELECT * INTO dept_table(1) FROM departments
  WHERE department_id = 10;
  DBMS_OUTPUT.PUT_LINE(dept_table(1).department_id || ' ' ||
    dept_table(1).department_name || ' ' ||
    dept_table(1).manager_id);
END;
/
```



Script Output x

Task completed in 0.009 seconds

anonymous block completed  
10 Administration 200

1 - 200

## Tabla INDEX BY de Registros de Filas: Ejemplo 2

```
DECLARE
  TYPE emp_table_type IS TABLE OF
    employees%ROWTYPE INDEX BY PLS_INTEGER;
  my_emp_table          emp_table_type;
  max_count             NUMBER(3) := 104;
BEGIN
  FOR i IN 100..max_count
  LOOP
    SELECT * INTO my_emp_table(i) FROM employees
    WHERE employee_id = i;
  END LOOP;
  FOR i IN my_emp_table.FIRST..my_emp_table.LAST
  LOOP
    DBMS_OUTPUT.PUT_LINE(my_emp_table(i).last_name);
  END LOOP;
END;
/
```

1 - 201

## Quiz

Identifique las situaciones en las que puede utilizar el atributo %ROWTYPE.

- a. Cuando no está seguro acerca de la estructura de la tabla de base de datos subyacente
- b. Cuando quiera recuperar una fila entera de una tabla
- c. Cuando desea declarar una variable de acuerdo con otra columna previamente declarada de variable o base de datos

1 - 212

## Resumen

En esta lección, debes haber aprendido a:

- Describir las colecciones y registros de PL/SQL
- Crear registros PL/SQL definidos por el usuario
- Cree un registro PL/SQL con el atributo `%ROWTYPE`

1 - 213

## Práctica 7

En esta lección, realiza las siguientes prácticas:

- Declaración de matrices asociativas
- Procesamiento de datos mediante el uso de matrices asociativas
- Declaración de un registro PL/SQL
- Procesamiento de datos mediante un registro PL/SQL

1 - 214

# 8

## Usando Cursores explícitos



ORACLE®

### Objetivos

Después de completar esta lección, usted debería ser capaz de:

- Distinguir entre cursores implícitos y explícitos
- Discutir las razones para usar cursores explícitos
- Declarar y controlar los cursores explícitos
- Utilice bucles simples y cursores con bucles `FOR` para obtener datos
- Declara y usa cursores con parámetros
- Bloquear filas con la cláusula `FOR UPDATE`
- Haga referencia a la fila actual con la cláusula `WHERE CURRENT OF`

1 - 216

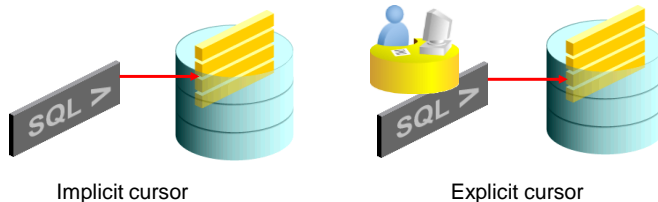
## Agenda

- ¿Qué son los cursores explícitos?
- Usando cursores explícitos
- Uso de cursores con parámetros
- Bloquear filas y hacer referencia a la fila actual

1 - 217

## Cursores

- Cuando el Servidor de Oracle ejecuta una instrucción SQL, ésta se ejecuta en un área de trabajo y es utilizada para almacenar información de proceso
- Esta área de trabajo es denominado cursor y hay 2 tipos:
  - Cursores **implícitos**: declarados y administrados por PL/SQL para todas las sentencias `SELECT` de DML y PL/SQL
  - Cursores **explícitos**: Declarado y administrado por el programador



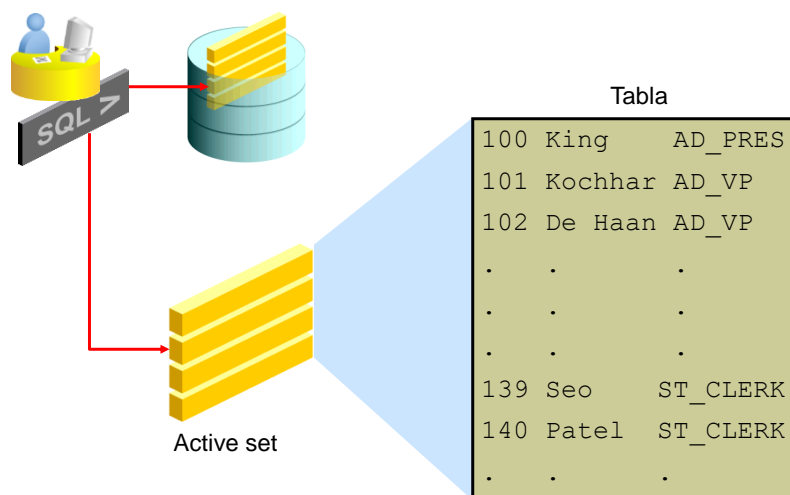
1 - 218

## Usos de los Cursores Explícitos

- Uno de los usos mas claros de su utilización es cuando tiene una instrucción `SELECT` que devuelve varias filas.
- Podemos crear un cursor explícito con el objetivo de procesar cada fila devuelta por la instrucción `SELECT` .
- El tamaño del cursor se corresponde con el número de filas que cumplen con los criterios de búsqueda.
- Los cursores disponen de un puntero a la fila actual del cursor, para permitir su recorrido y que su programa procese las filas una a la vez.

1 - 219

## Usos de los Cursores Explícitos

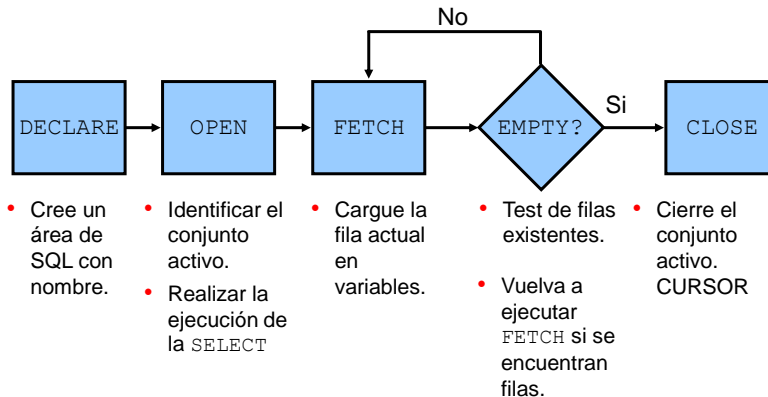


1 - 220



## Controlando Cursores Explícitos

- Pasos para la utilización de los cursores Explícitos



1 - 221

## Agenda

- ¿Qué son los cursores explícitos?
- Usando cursores explícitos
- Uso de cursores con parámetros
- Bloquear filas y hacer referencia a la fila actual

1 - 222

## Declaración del cursor

Sintaxis:

```
CURSOR cursor_name IS  
    select_statement;
```

- **Cursor\_name**
  - Es un identificador de PL/SQL indicando el nombre del Cursor
- **Select\_statement**
  - Es una instrucción SELECT sin una cláusula INTO
- La instrucción SELECT en la declaración de cursor **no puede tener una cláusula INTO**.
  - Esto es porque sólo está definiendo un cursor en la sección declarativa y no recuperando ninguna fila en el cursor.
  - La cláusula INTO aparece más adelante en la instrucción FETCH

1 - 223

## Declaración del cursor

Ejemplos:

```
DECLARE  
CURSOR c_emp_cursor IS  
    SELECT employee_id, last_name FROM employees  
    WHERE department_id =30;  
...
```

```
DECLARE  
    v_locid NUMBER:= 1700;  
CURSOR c_dept_cursor IS  
    SELECT * FROM departments  
    WHERE location_id = v_locid;  
...
```

1 - 224

## Apertura del cursor

```
DECLARE
  CURSOR c_emp_cursor IS
    SELECT employee_id, last_name FROM employees
    WHERE department_id = 30;
  ...
BEGIN
  OPEN c_emp_cursor;
```

- La instrucción **OPEN**
  1. Asigna dinámicamente memoria para un área de contexto
  2. Ejecuta la consulta asociada con el cursor
  3. Identifica el conjunto activo y
  4. Posiciona el puntero del cursor en la primera fila
- La instrucción **OPEN** se debe de incluir en la sección ejecutable del bloque PL/SQL.
- **NOTA:** Si una consulta no devuelve ninguna fila cuando se abre el cursor, PL / SQL no genera una excepción.

1 - 225

## Fetching Data from the Cursor

```
:
BEGIN
  OPEN c_emp_cursor;
  FETCH c_emp_cursor INTO v_empno, v_lname;
  DBMS_OUTPUT.PUT_LINE( v_empno || ' ' || v_lname);
END;
/
```

- La instrucción **FETCH** recupera las filas del cursor de una en una.
  - Debe de tener definida la cláusula **INTO** y debe de definir también tantas variables como valores se esperan recibir de la fila.
  - Alternativamente, también puede definir un registro para el cursor y hacer referencia al registro en la cláusula **FETCH INTO**
- Después de cada **FETCH**, el cursor avanza a la siguiente fila del conjunto activo.
- Puede utilizar el atributo **%NOTFOUND** para determinar si se ha recuperado todo el conjunto activo

1 - 226

## Recolección de datos desde el cursor

```
DECLARE
  CURSOR c_emp_cursor IS
    SELECT employee_id, last_name FROM employees
    WHERE department_id = 30;
  v_empno employees.employee_id%TYPE;
  v_lname employees.last_name%TYPE;
BEGIN
  OPEN c_emp_cursor;
  LOOP
    FETCH c_emp_cursor INTO v_empno, v_lname;
    EXIT WHEN c_emp_cursor%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE( v_empno || ' ' || v_lname);
  END LOOP;
END;
/
```

Observe que un simple LOOP se utiliza para buscar todas las filas  
El atributo de cursor `%NOTFOUND` se utiliza para probar la condición de salida

1 - 227

## Cierre del cursor

```
...
  LOOP
    FETCH c_emp_cursor INTO empno, lname;
    EXIT WHEN c_emp_cursor%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE( v_empno || ' ' || v_lname);
  END LOOP;
  CLOSE c_emp_cursor;
END;
/
```

- La instrucción `CLOSE` elimina el cursor y libera el área de contexto.
- El cursor debe de ser cerrado después de completar el procesamiento de la instrucción `FETCH`
  - Un cursor puede reabrirse sólo si está cerrado.
- Si intenta obtener datos de un cursor después de que se cierra, se genera una excepción `INVALID_CURSOR`.

1 - 228

## Cursores y Registros

Procese las filas del conjunto activo extrayendo valores en un registro PL/SQL.

```
DECLARE
  CURSOR c_emp_cursor IS
    SELECT employee_id, last_name FROM employees
    WHERE department_id = 30;
  v_emp_record c_emp_cursor%ROWTYPE;
BEGIN
  OPEN c_emp_cursor;
  LOOP
    FETCH c_emp_cursor INTO v_emp_record;
    EXIT WHEN c_emp_cursor%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE( v_emp_record.employee_id
                          || ' ' || v_emp_record.last_name);
  END LOOP;
  CLOSE c_emp_cursor;
END;
```

1 - 229

## Cursores en bucles FOR

- PL/SQL también permite la utilización de los bucles FOR sobre cursores
- El bucle FOR del cursor es un atajo para procesar cursores explícitos.
  - Realizar una apertura automática
  - Un recorrido automático y
  - Un cierre directo.

### Sintaxis

```
FOR record_name IN cursor_name LOOP
  statement1;
  . . .
END LOOP;
```

1 - 230

## Cursores en bucles FOR

```
FOR record_name IN cursor_name LOOP
    statement1;
    . . .
END LOOP;
```

- **Record\_name**
  - Es la variable utilizada para guardar las filas recuperadas del cursor
  - No es necesario declararlo, es declarado implícitamente
- **Cursor\_name**
  - Es un identificador PL/SQL para el cursor previamente declarado.

### Directrices

- No declare el registro que controla el bucle; Se declara implícitamente.
- El contador del bucle no se puede incrementar ni decrementar.
- Suministre los parámetros de un cursor, si es necesario, entre paréntesis siguiendo el nombre del cursor en la instrucción FOR.

1 - 231

## Cursores en bucles FOR

```
DECLARE
    CURSOR c_emp_cursor IS
        SELECT employee_id, last_name FROM employees
        WHERE department_id = 30;
BEGIN
    FOR emp_record IN c_emp_cursor
    LOOP
        DBMS_OUTPUT.PUT_LINE( emp_record.employee_id
        || ' ' || emp_record.last_name);
    END LOOP;
END;
```

```
anonymous block completed
114 Raphaely
115 Khoo
116 Baida
117 Tobias
118 Himuro
119 Colmenares
```

1 - 232

## Atributos de los Cursores Explícitos

- Podemos utilizar los atributos de los cursores explícitos para obtener información de estado sobre un cursor.
- Debemos añadir el nombre del cursor a estos atributos

**Cursor1%ISOPEN, Cursor1%NOTFOUND**

| Attribute | Type    | Description                                                       |
|-----------|---------|-------------------------------------------------------------------|
| %ISOPEN   | Boolean | Devuelve TRUE si el cursor está abierto                           |
| %NOTFOUND | Boolean | Devuelve TRUE si el FETCH más reciente no devuelve una fila       |
| %FOUND    | Boolean | Devuelve TRUE si el FETCH más reciente devuelve una fila;         |
| %ROWCOUNT | Number  | Devuelve el número de filas que se han obtenido hasta ese momento |

1 - 233

## Atributo %ISOPEN

- Sólo puede buscar filas cuando el cursor está abierto.
- Utilice el atributo %ISOPEN antes de realizar una búsqueda para comprobar si el cursor está abierto.

Example:

```
IF NOT c_emp_cursor%ISOPEN THEN
  OPEN c_emp_cursor;
END IF;
LOOP
  FETCH c_emp_cursor...
```

1 - 234

## %ROWCOUNT y %NOTFOUND: Ejemplo

```
DECLARE
  CURSOR c_emp_cursor IS SELECT employee_id,
    last_name FROM employees;
  v_emp_record  c_emp_cursor%ROWTYPE;
BEGIN
  OPEN c_emp_cursor;
  LOOP
    FETCH c_emp_cursor INTO v_emp_record;
    EXIT WHEN c_emp_cursor%ROWCOUNT > 10 OR
              c_emp_cursor%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE( v_emp_record.employee_id
      || ' ' || v_emp_record.last_name);
  END LOOP;
  CLOSE c_emp_cursor;
END ; /
```

anonymous block completed

174 Abel  
166 Ande  
130 Atkinson  
105 Austin  
204 Baer  
116 Baida  
167 Banda  
172 Bates  
192 Bell  
151 Bernstein

recupera uno a uno los  
primeros 10 empleados

1 - 235

## Cursores con FOR Loops usando Subconsultas In-line

- Podemos utilizar **subconsultas In-Line** como cursor y así eliminar la necesidad de declarar el cursor.

```
BEGIN
  FOR emp_record IN (SELECT employee_id, last_name
    FROM employees WHERE department_id =30)
  LOOP
    DBMS_OUTPUT.PUT_LINE( emp_record.employee_id
      || ' ' || emp_record.last_name);
  END LOOP;
END;
/
```

anonymous block completed

114 Raphaely  
115 Khoo  
116 Baida  
117 Tobias  
118 Himuro  
119 Colmenares

1 - 236



## Agenda

- ¿Qué son los cursores explícitos?
- Usando cursores explícitos
- **Uso de cursores con parámetros**
- Bloquear filas y hacer referencia a la fila actual

1 - 237

## Cursores con parámetros

- PL/SQL permite pasar parámetros a un cursor.
- Esto significa que puede abrir y cerrar un cursor explícito varias veces en un bloque, devolviendo un conjunto activo diferente en cada ocasión
  - Para cada ejecución, el cursor anterior se cierra y se reabre con un nuevo conjunto de parámetros.
- Sintaxis

```
CURSOR cursor_name [(parameter_name datatype, ...)]  
IS  
    select_statement;
```

```
OPEN cursor_name (parameter_value, .....);
```

1 - 238

## Cursores con parámetros

- Cada **parámetro formal** en la declaración del cursor debe tener un parámetro real correspondiente en la instrucción `OPEN`.
- Los tipos de datos de parámetros son los mismos que los de las variables escalares, pero **sin tamaño**.
- La utilización de cursores con parámetros es útil cuando se hace referencia repetidamente al mismo cursor.

1 - 239

## Cursores con parámetros: Ejemplo

```
DECLARE
  CURSOR c_emp_cursor (deptno NUMBER) IS
    SELECT employee_id, last_name
    FROM   employees
    WHERE  department_id = deptno;
  ...
BEGIN
  OPEN c_emp_cursor (10);
  ...
  CLOSE c_emp_cursor;
  OPEN c_emp_cursor (20);
  ...
```

```
anonymous block completed
200 Whalen
201 Hartstein
202 Fay
```

1 - 240

## Agenda

- ¿Qué son los cursores explícitos?
- Usando cursores explícitos
- Uso de cursores con parámetros
- Bloquear filas y hacer referencia a la fila actual

1 - 241

## Clausula FOR UPDATE

- Si hay varias sesiones para una sola base de datos, existe la posibilidad de que las filas de una tabla en particular se hayan actualizado después de abrir el cursor.
- Verá los datos actualizados sólo cuando vuelva a abrir el cursor
- Por lo tanto, es mejor tener bloqueos en las filas antes de actualizar o eliminar filas.
- Puede bloquear las filas con la cláusula `FOR UPDATE` en la consulta de cursor.
  - La cláusula `FOR UPDATE` es la última cláusula en una sentencia `SELECT`

1 - 242

## Clausula FOR UPDATE

Sintaxis:

```
CURSOR name IS  
SELECT      ...  
FROM        ...  
FOR UPDATE [OF column_reference] [NOWAIT | WAIT n];
```

- **OF Columna**
  - Cuando desea consultar varias tablas, puede utilizar la cláusula FOR UPDATE columna(s) para bloquear las filas sólo en tablas que contienen nombre(s) col.
- La palabra clave **NOWAIT** le dice al servidor de Oracle que no espere si las filas solicitadas han sido bloqueadas por otro usuario.
  - El control se devuelve inmediatamente a su programa para que pueda hacer otro trabajo antes de intentar de nuevo adquirir el bloqueo.
  - Si omite la palabra clave **NOWAIT**, el servidor Oracle esperará hasta que las filas estén disponibles.

1 - 243

## Clausula WHERE CURRENT OF

- La cláusula WHERE CURRENT OF se utiliza junto con la cláusula FOR UPDATE para referirse a la fila actual en un cursor explícito.
- La cláusula:
  - WHERE CURRENT OF se utiliza en la instrucción UPDATE o DELETE
  - FOR UPDATE se especifica en la declaración del curso
- Puede utilizar la combinación de ambas para actualizar/eliminar la fila actual del cursor en la tabla de base de datos correspondiente.

1 - 244

## Clausula WHERE CURRENT OF

Syntax:

```
WHERE CURRENT OF cursor ;
```

```
CURSOR c_emp_cursor IS  
SELECT    ...  
FROM      ...  
FOR UPDATE [OF column_reference] [NOWAIT | WAIT n];  
:
```

```
FOR indice in c_emp_cursor LOOP  
    UPDATE employees  
    SET    salary = ...  
    WHERE CURRENT OF c_emp_cursor;  
:
```

1 - 245

## Quiz

Las funciones de los cursores explícitos permiten al programador controlar manualmente los cursores explícitos en el bloque PL/SQL.

- a. True
- b. False

1 - 246

## Resumen

En esta lección, debes haber aprendido a:

- Distinguir entre cursores implícitos y explícitos
- Discutir las razones para usar cursores explícitos
- Declarar y controlar los cursores explícitos
- Utilice bucles simples y cursores con bucles `FOR` para obtener datos
- Declara y usa cursores con parámetros
- Bloquear filas con la cláusula `FOR UPDATE`
- Haga referencia a la fila actual con la cláusula `WHERE CURRENT OF`

1 - 247

## Práctica 8

En esta lección, realiza las siguientes prácticas:

- Declaración y uso de cursores explícitos para consultar las filas de una tabla
- Utilizar un bucle `FOR` del cursor
- Aplicación de atributos de cursor para probar el estado del cursor
- Declaración y uso de cursores con parámetros
- Utilizar las cláusulas `FOR UPDATE` y `WHERE CURRENT OF`

1 - 248

# 9

## Manejo de excepciones



ORACLE®

### Objetivos

Después de completar esta lección, usted debería ser capaz de:

- Definir excepciones de PL/SQL
- Reconocer excepciones no tratadas
- Lista y uso de diferentes tipos de manejadores de excepciones de PL/SQL
- Atrapa errores imprevistos
- Describir el efecto de la propagación de excepciones en bloques anidados
- Personalizar mensajes de excepción de PL/SQL

1 - 250

## Agenda

- Descripción de las excepciones de PL/SQL
- Intercepción de excepciones

1 - 251

## Introducción a las excepciones

- Anteriormente, hemos aprendido a escribir bloques PL/SQL con:
  - una sección **declarativa** (comenzando con la palabra clave **DECLARE**) y una sección **ejecutable** (comenzando y terminando con las palabras clave **BEGIN** y **END**).
- Para el tratamiento de excepciones, incluye otra sección opcional denominada **EXCEPTION**.
- Una excepción es un error en PL/SQL que se plantea durante la ejecución de un bloque.
- Un bloque siempre termina cuando PL/SQL genera una excepción.

1 - 252



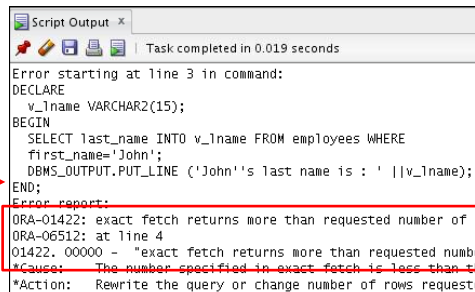
## Introducción a las excepciones

- Podemos realizar un tratamiento de las excepciones para realizar acciones finales antes de que el bloque termine.
- Esta sección comienza con la palabra clave **EXCEPTION**, es opcional y si está presente, debe ser la última sección de un bloque PL/SQL.
- Disponemos de varios tipos de excepciones:
  - Predefinidas
  - De usuario
  - Personalizadas

1 - 253

## Revisemos el siguiente código

```
DECLARE
  v_lname VARCHAR2(15);
BEGIN
  SELECT last_name INTO v_lname
  FROM employees
  WHERE first_name='John';
  DBMS_OUTPUT.PUT_LINE ('John's last name is : ' || v_lname);
END;
```



Script Output x

Task completed in 0.019 seconds

Error starting at line 3 in command:

```
DECLARE
  v_lname VARCHAR2(15);
BEGIN
  SELECT last_name INTO v_lname FROM employees WHERE
  first_name='John';
  DBMS_OUTPUT.PUT_LINE ('John's last name is : ' || v_lname);
END;
```

Error report:

```
ORA-01422: exact fetch returns more than requested number of rows
ORA-06512: at line 4
01422. 00000 - "exact fetch returns more than requested number of rows"
*Cause: The number specified in exact fetch is less than the rows returned.
*Action: Rewrite the query or change number of rows requested
```

No hay errores de sintaxis en el código, Sin embargo, se produce un error al ejecutar el código

1 - 254

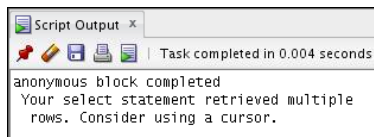
## Introducción a las excepciones

- Podemos reescribir el código anterior para manejar la excepción que se produjo.  
`EXCEPTION`  
`WHEN TOO_MANY_ROWS THEN`
- Al agregar la sección `EXCEPTION` del código, el programa PL/SQL no termina abruptamente.
- Cuando se produce la excepción, el control cambia a la sección de excepciones y todas las sentencias de la sección de excepciones se ejecutan.
- El bloque PL/SQL finaliza con una finalización normal.

1 - 255

## Manejo de una Excepción: Ejemplo

```
DECLARE
  v_lname VARCHAR2(15);
BEGIN
  SELECT last_name INTO v_lname
  FROM employees
  WHERE first_name='John';
  DBMS_OUTPUT.PUT_LINE ('John's last name is : ' || v_lname);
EXCEPTION
  WHEN TOO_MANY_ROWS THEN
    DBMS_OUTPUT.PUT_LINE (' Your select statement retrieved
      multiple rows. Consider using a cursor. ');
END;
/
```



1 - 256

## Entendiendo las excepciones con PL/SQL

- En PL/SQL se dispone de 2 métodos para lanzar una excepción:
  - **Implícitamente** por el servidor Oracle
    - Se produce un error Oracle y la excepción asociada se genera automáticamente.
    - Por ejemplo, si el error `ORA-01403` se produce cuando no se recuperan filas de la base de datos en una instrucción `SELECT`, PL/SQL genera la excepción `NO_DATA_FOUND`.
    - Estos errores se convierten en excepciones predefinidas.
  - **Explícitamente** por el programa
    - Dependiendo de la funcionalidad de negocio implementada por su programa, puede que tenga que plantear una excepción explícitamente.
    - Se produce una excepción explícitamente emitiendo la instrucción **RAISE** en el bloque.

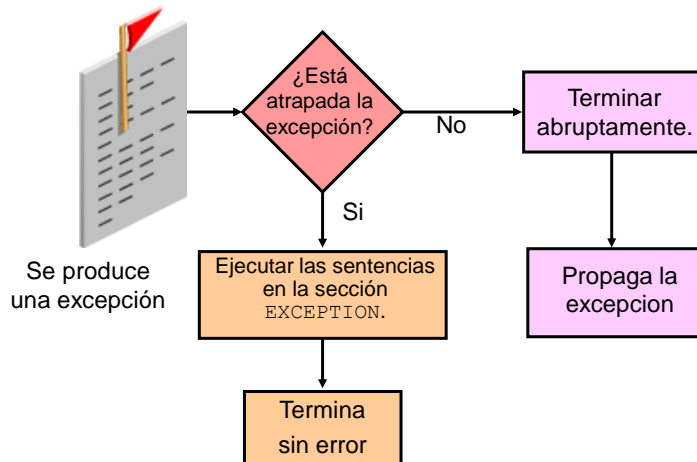
1 - 257

## Entendiendo las excepciones con PL/SQL

- Se puede manejar una excepción:
  - **Atrapándolo con un manejador**
    - Incluya una sección **EXCEPTION** en su programa PL/SQL para interceptar excepciones.
    - Si se genera la excepción en la sección ejecutable del bloque, el procesamiento se deriva a la sección de excepciones del bloque.
    - Si PL/SQL gestiona correctamente la excepción, la excepción no se propagará y el bloque PL/SQL finaliza correctamente.
  - **Al propagarlo al entorno llamante**
    - Si se genera la excepción en la sección ejecutable del bloque y no hay un manejador de excepciones correspondiente, el bloque PL/SQL termina con un error y la excepción se propaga a un bloque adjunto o al entorno llamante.
    - El entorno llamante puede ser cualquier aplicación (como SQL\*Plus que invoca el programa PL/SQL).

1 - 258

## Manejo de excepciones



1 - 259

## Tipos de excepciones

- Predefinidas por Oracle
  - No predefinidas por Oracle (Personalizadas)
- } Lanzadas de forma implícita
- Definidas por el Usuario
- Lanzadas de forma explícita

1 - 260

## Agenda

- Descripción de las excepciones de PL/SQL
- Intercepción de excepciones

1 - 261

## Sintaxis para atrapar excepciones

- Puede interceptar cualquier error incluyendo un **manejador** correspondiente dentro de la sección de gestión de excepciones.
- Cada **manejador** consiste en una cláusula `WHEN`, que especifica:
  - Un nombre de excepción
  - Seguido por una conjunto de sentencias que se ejecutará cuando se genere dicha excepción.
- Puede incluir cualquier número de **manejadores** dentro de una sección `EXCEPTION` para manejar excepciones específicas.
- La sintaxis de captura de excepciones incluye los siguientes elementos:

1 - 262

## Sintaxis para atrapar excepciones

- La sintaxis de captura de excepciones incluye los siguientes elementos:

### EXCEPTION

```
WHEN exception1 [OR exception2 . . .] THEN
    statement1;
    statement2;
    . . .
[WHEN exception3 [OR exception4 . . .] THEN
    statement1;
    statement2;
    . . .]
[WHEN OTHERS THEN
    statement1;
    statement2;
    . . .]
```

**Exception1** se corresponde con el **nombre** de la excepción y no con el **número** de error producido

1 - 263

## WHEN OTHERS Exception Handler

- La sección de `EXCEPTION` sólo atrapa las excepciones que se especifican.
- Para interceptar las excepciones que no se especifican, utilice el controlador de excepciones `OTHERS`.
- Esta opción intercepta cualquier excepción aún no tratada.
- Por este motivo, si se utiliza el controlador `OTHERS`, debe ser el último controlador de excepciones que se define.
- El controlador `OTHERS` captura todas las excepciones que no están ya atrapadas.

1 - 264

## Directrices para la captura de excepciones

- La sección de gestión de excepciones comience con la palabra clave `EXCEPTION`.
- Podemos definir varios **manejadores** de excepciones, cada uno con su propio conjunto de acciones.
- Cuando se produce una excepción, PL/SQL sólo procesa el primer **manejadores** que se corresponde con la excepción producida y luego abandonar el bloque.
- Coloque la cláusula `OTHERS` después de todas las demás cláusulas de tratamiento de excepciones, si deseamos gestionar excepciones no tratadas

1 - 265

## Excepciones Predefinidas por Oracle

- Oracle define un conjunto de excepciones predefinidas con el objeto de facilitar el desarrollo de código.
- Estas excepciones son definidas en el paquete `STANDARD`.
- De forma implícita asignan un `ALIAS` a errores comúnmente generados en programación
- Ejemplos de excepciones predefinidas:
  - `NO_DATA_FOUND`
  - `TOO_MANY_ROWS`
  - `INVALID_CURSOR`
  - `ZERO_DIVIDE`
  - `DUP_VAL_ON_INDEX`

1 - 266

## Excepciones Predefinidas por Oracle: Ejemplo

```
DECLARE
  v_lname VARCHAR2 (15);
BEGIN
  SELECT last_name INTO v_lname
  FROM employees
  WHERE first_name = 'John';
  DBMS_OUTPUT.PUT_LINE ('Last name is :'|| v_lname);
EXCEPTION
  WHEN TOO_MANY_ROWS THEN
    DBMS_OUTPUT.PUT_LINE (' Your SELECT statement
      retrieved multiple rows. Consider using a cursor. ');
  WHEN NO_DATA_FOUND THEN
    DBMS_OUTPUT.PUT_LINE (' Not Data Found');
END;
/
```

1 - 267

## Excepciones NO PreDefinidas por Oracle

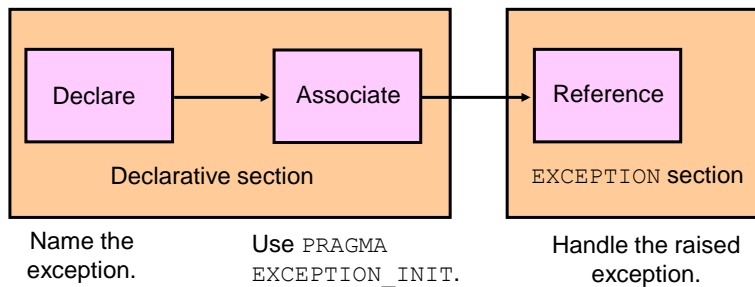
- No todos los errores tienen asociada una excepción predefinida.
- Para los casos en los cuales un error se produce de forma habitual, podemos crear una excepción No preDefinidas.
- Estas son similares a las excepciones predefinidas, excepto que no están definidas dentro del Servidor de Oracle, se definen como requisito de programación
- Puede crear excepciones y asociarlas a errores estándar de Oracle mediante la función **PRAGMA EXCEPTION\_INIT**.

1 - 268



## Excepciones NO PreDefinidas por Oracle

- Con la sentencia `PRAGMA EXCEPTION_INIT` en la zona declarativa, le indica al compilador que asocie un **nombre de excepción** con un **número** de error de Oracle.
- Cuando en ejecución se produzca un error con ese número, el código buscara resolver una excepción con el nombre de excepción indicado



1 - 269

## Excepciones NO PreDefinidas por Oracle

Para interceptar el error del servidor Oracle 01400 ("cannot insert NULL"):

```
DECLARE
e_insert_excep EXCEPTION;
PRAGMA EXCEPTION_INIT(e_insert_excep, -01400);
BEGIN
    INSERT INTO departments
    (department_id, department_name) VALUES (280, NULL);
EXCEPTION
    WHEN e_insert_excep THEN
        DBMS_OUTPUT.PUT_LINE('INSERT OPERATION FAILED');
        DBMS_OUTPUT.PUT_LINE(SQLERRM);
END;
/
```

Script Output x

Task completed in 0.164 seconds

anonymous block completed  
INSERT OPERATION FAILED  
ORA-01400: cannot insert NULL into ('ORA41', 'DEPARTMENTS', 'DEPARTMENT\_NAME')

1 - 270

## Funciones para capturar excepciones

- Cuando se produce una excepción, puede identificar el código de error asociado o mensaje de error utilizando dos funciones.
- Con base en los valores del código o del mensaje, puede decidir qué acciones posteriores tomar.
- `SQLCODE`
  - Devuelve el número de error de Oracle para las excepciones internas.
- `SQLERRM`
  - Devuelve el mensaje asociado con el número de error.
- El manejador de excepciones `WHEN OTHERS`, puede utilizar estas funciones genéricas para identificar el error producido

1 - 271

## Funciones para capturar excepciones

```
DECLARE
    error_code      NUMBER;
    error_message   VARCHAR2(255);
BEGIN
    ...
EXCEPTION
    ...
    WHEN OTHERS THEN
        ROLLBACK;
        error_code := SQLCODE ;
        error_message := SQLERRM ;
        INSERT INTO errors (e_user, e_date, error_code,
            error_message) VALUES (USER, SYSDATE, error_code,
            error_message);
END;
/
```

No puede utilizar `SQLCODE` o `SQLERRM` directamente en una sentencia SQL. En su lugar, debe asignar sus valores a variables locales

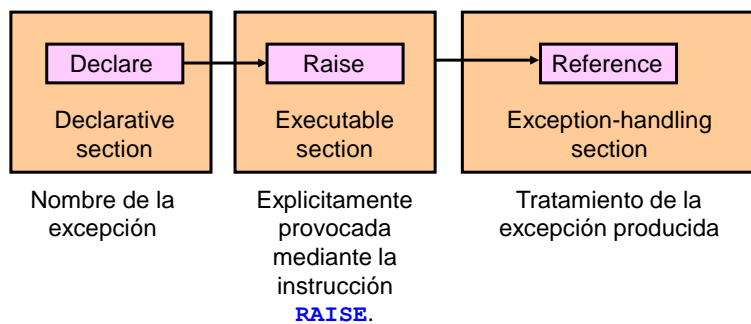
1 - 272

## Excepciones definidas por el usuario

- PL/SQL le permite definir sus propias excepciones en función de los requisitos de su aplicación.
  - Por ejemplo, puede pedirle al usuario que introduzca un número de departamento.
  - Defina una excepción.
  - Compruebe si existe el número de departamento.
  - Si no lo hace, puede que tenga que aumentar la excepción definida por el usuario.
- Estas excepciones deben de ser:
  - Declaradas en la sección declarativa de un bloque PL/SQL, como una variable de tipo **EXCEPTION**
  - Provocada de forma explícita mediante la sentencia **RAISE**
  - Gestionada en la sección **EXCEPTION**

1 - 273

## Excepciones definidas por el usuario



1 - 274

## Excepciones definidas por el usuario

```
DECLARE
  v_deptno NUMBER := 500;
  v_name VARCHAR2(20) := 'Testing';
  e_invalid_department EXCEPTION;
BEGIN
  UPDATE departments
  SET department_name = v_name
  WHERE department_id = v_deptno;
  IF SQL%NOTFOUND THEN
    RAISE e_invalid_department;
  END IF;
  COMMIT;
EXCEPTION
  WHEN e_invalid_department THEN
    DBMS_OUTPUT.PUT_LINE('No such department id.');
```

```

/
```

1 - 275

## Propagando excepciones en un subbloque

- Cuando un bloque embebido termina de forma correcta, el control se reanuda en el bloque padre que lo contiene.

```
BEGIN
  FOR c_record IN emp_cursor LOOP
    BEGIN
      SELECT ...
      UPDATE ...
      IF SQL%NOTFOUND THEN
        RAISE e_no_rows;
      END IF;
    END;
  END;
```

- Sin embargo, si un bloque embebido, produce una excepción y no es tratada en el bloque, la excepción se propaga a los bloques superiores.
- Si ninguno de estos bloques gestiona la excepción, se produce una **excepción no controlada** en el entorno del host

1 - 276

## Propagando excepciones en un subbloque

- Cuando la excepción se propaga a un bloque superior, las acciones ejecutables restantes del bloque embebido se anulan.

```
DECLARE
    . . .
    e_no_rows      exception;
    e_integrity     exception;
    PRAGMA EXCEPTION_INIT (e_integrity, -2292);
BEGIN
    FOR c record IN emp_cursor LOOP
        BEGIN
            SELECT ...
            UPDATE ...
            IF SQL%NOTFOUND THEN
                RAISE e_no_rows;
            END IF;
        END;
    END LOOP;
EXCEPTION
    WHEN e_integrity THEN ...
    WHEN e_no_rows THEN ...
END;
/
```

1 - 277

## Sentencia RAISE

- La sentencia **RAISE**, detiene la ejecución normal de un bloque o subprograma PL/SQL y transfiere el control a un manejador de excepciones

```
RAISE exception_name ;
```

- Si la llamada a la sentencia RAISE se produce en una sección **ejecutable**, es obligatorio poner el **nombre de la excepción**.
- Si la sentencia RAISE se produce en la sección de **excepciones** podemos:
  - Indicar el nombre de la excepción a producir
  - No indicar el nombre de la excepción y se utilizar el nombre de la excepción actual que provoco el salto a la sección de excepciones

1 - 278

## Procedimiento RAISE\_APPLICATION\_ERROR

- Utilice el procedimiento `RAISE_APPLICATION_ERROR` para comunicar **interactivamente** una excepción devolviendo un **código de error** y un **mensaje de error no estándar**
- Puede utilizar este procedimiento para emitir mensajes de error definidos por el usuario desde **subprogramas almacenados**.

Sintaxis

```
raise_application_error (error_number,  
                        message[, {TRUE | FALSE}]);
```

1 - 279

## Procedimiento RAISE\_APPLICATION\_ERROR

- El procedimiento `RAISE_APPLICATION_ERROR` se puede utilizar en
  - La sección **ejecutable**
  - La sección de **excepciones**
- El error devuelto es idéntico al que utiliza el servidor de Oracle produce un error predefinido.
- El número de error y el mensaje se muestran al usuario.
  - El código de error deberá estar comprendido entre 20000 y 20999

1 - 280

## Procedimiento RAISE\_APPLICATION\_ERROR

Sección ejecutable:

```
BEGIN
...
DELETE FROM employees
WHERE manager_id = v_mgr;
IF SQL%NOTFOUND THEN
    RAISE_APPLICATION_ERROR(-20202,
        'This is not a valid manager');
END IF;
...
```

Sección de excepción:

```
...
EXCEPTION
WHEN NO_DATA_FOUND THEN
    RAISE_APPLICATION_ERROR (-20201,
        'Manager is not a valid employee.');
```

END;

1 - 281

## Quiz

Puede interceptar cualquier error dentro de la sección de gestión de excepciones del bloque PL/SQL.

- a. True
- b. False

1 - 282

## Resumen

En esta lección, debes haber aprendido a:

- Definir excepciones de PL/SQL
- Reconocer excepciones no tratadas
- Lista y uso de diferentes tipos de manejadores de excepciones de PL/SQL
- Atrapa errores imprevistos
- Describir el efecto de la propagación de excepciones en bloques anidados
- Personalizar mensajes de excepción de PL/SQL

1 - 283

## Práctica 9

En esta lección, realiza las siguientes prácticas:

- Creación e invocación de excepciones definidas por el usuario
- Manejo de excepciones de Oracle Server con nombre

1 - 284