4

Escritura de Estructuras de Control

ORACLE

Objetivos

Al finalizar este curso, debería estar capacitado para hacer lo siguiente:

- **Identificar los usos y tipos de estructuras de control**
- **Construir una sentencia `IF`**
- **Utilizar expresiones `CASE`**
- **Construir e identificar distintas sentencias de bucle**
- **Utilizar tablas lógicas**
- **Controlar el flujo de bloques utilizando etiquetas y bucles anidados**

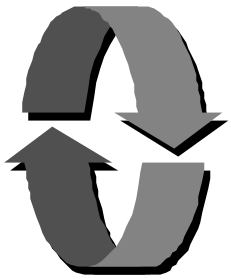
ORACLE

Finalidad de la Lección

En esta lección, aprenderá a utilizar el control condicional en el interior de los bloques PL/SQL utilizando bucles y sentencias `IF`.

Control del Flujo de Ejecución de PL/SQL

- **Se puede cambiar la ejecución lógica de las sentencias utilizando sentencias `IF` condicionales y estructuras de control de bucles.**
- **Sentencias `IF` condicionales:**
  - **`IF-THEN-END IF`**
  - **`IF-THEN-ELSE-END IF`**
  - **`IF-THEN-ELSIF-END IF`**



ORACLE

Control del Flujo de Ejecución de PL/SQL

Para cambiar el flujo lógico de las sentencias en el interior de los bloques PL/SQL, puede utilizar una serie de *estructuras de control*. Esta lección explica tres tipos de estructuras de control PL/SQL: construcciones condicionales con la sentencia IF, expresiones CASE y estructuras de control `LOOP` (que se explicarán más adelante, en esta misma lección).

Existen tres tipos de sentencias `IF`:

- `IF-THEN-END IF`
- `IF-THEN-ELSE-END IF`
- `IF-THEN-ELSIF-END IF`

Sentencias IF

Sintaxis:

```
IF condición THEN
  sentencias;
[ELSIF condición THEN
  sentencias;]
[ELSE
  sentencias;]
END IF;
```

Si el empleado se llama Gietz, defina el identificador del director como 102.

```
IF UPPER(v_last_name) = 'GIETZ' THEN
  v_mgr := 102;
END IF;
```

ORACLE

Sentencias IF

La estructura de la sentencia IF de PL/SQL es similar a la estructura de las sentencias IF de otros lenguajes procedurales. Permite a PL/SQL realizar acciones selectivamente basándose en condiciones.

En la sintaxis:

condición	es una variable o expresión booleana (TRUE, FALSE o NULL). (Se asocia a una secuencia de sentencias, que sólo se ejecuta si la expresión devuelve TRUE.)
THEN	es una cláusula que asocia la expresión booleana que la precede con la secuencia de sentencias que la sigue.
sentencias	pueden ser una o varias sentencias PL/SQL o SQL. (Pueden incluir otras sentencias IF que contienen varias sentencias IF, ELSE y ELSIF anidadas.)
ELSIF	es una palabra clave que introduce una expresión booleana. (Si la primera condición devuelve FALSE o NULL, la palabra clave ELSIF introduce condiciones adicionales.)
ELSE	es una palabra clave que ejecuta la secuencia de sentencias que le sigue si el control llega hasta ella.

Sentencias IF Simples

Si el apellido es Vargas:

- **Defina el identificador del trabajo como SA_REP**
- **Defina el número del departamento como 80**

```
. . .
IF v_ename      = 'Vargas' THEN
   v_job       := 'SA_REP';
   v_deptno    := 80;
END IF;
. . .
```

ORACLE

Sentencias IF Simples

En el ejemplo de la transparencia, PL/SQL asigna valores a las siguientes variables, sólo si la condición es TRUE:

`v_job` y `v_deptno`

Si la condición es FALSE o NULL, PL/SQL ignora las sentencias del bloque IF. En cualquiera de los casos, el control continúa en la siguiente sentencia del programa después de END IF.

Instrucciones

- Se pueden realizar acciones selectivamente basándose en las condiciones que se van cumpliendo.
- Al escribir el código, no se olvide de la ortografía de las palabras clave:
  - ELSIF tiene una palabra.
  - END IF tiene dos palabras.
- Si la condición booleana de control es TRUE, se ejecuta la secuencia asociada de sentencias; si la condición booleana de control es FALSE o NULL, la secuencia asociada de sentencias se pasa por alto. Se permite cualquier número de cláusulas ELSIF.
- Utilice sangrados en las sentencias que se ejecutan condicionalmente para que sean más claras.

Sentencias IF Compuestas

Si el apellido es Vargas y el sueldo es mayor de 6.500:

Defina el número del departamento como 60.

```
. . .
IF v_ename  = 'Vargas' AND salary > 6.500 THEN
   v_deptno := 60;
END IF;
. . .
```

ORACLE

Sentencias IF Compuestas

Las sentencias IF compuestas utilizan operadores lógicos como AND y NOT. En el ejemplo de la transparencia, la sentencia IF tiene que evaluar dos condiciones:

- El apellido debería ser Vargas
- El sueldo debería ser mayor de 6.500

Sólo en el caso de que las dos condiciones anteriores se evalúen como TRUE, v_deptno se definirá como 60.

Observe el siguiente ejemplo:

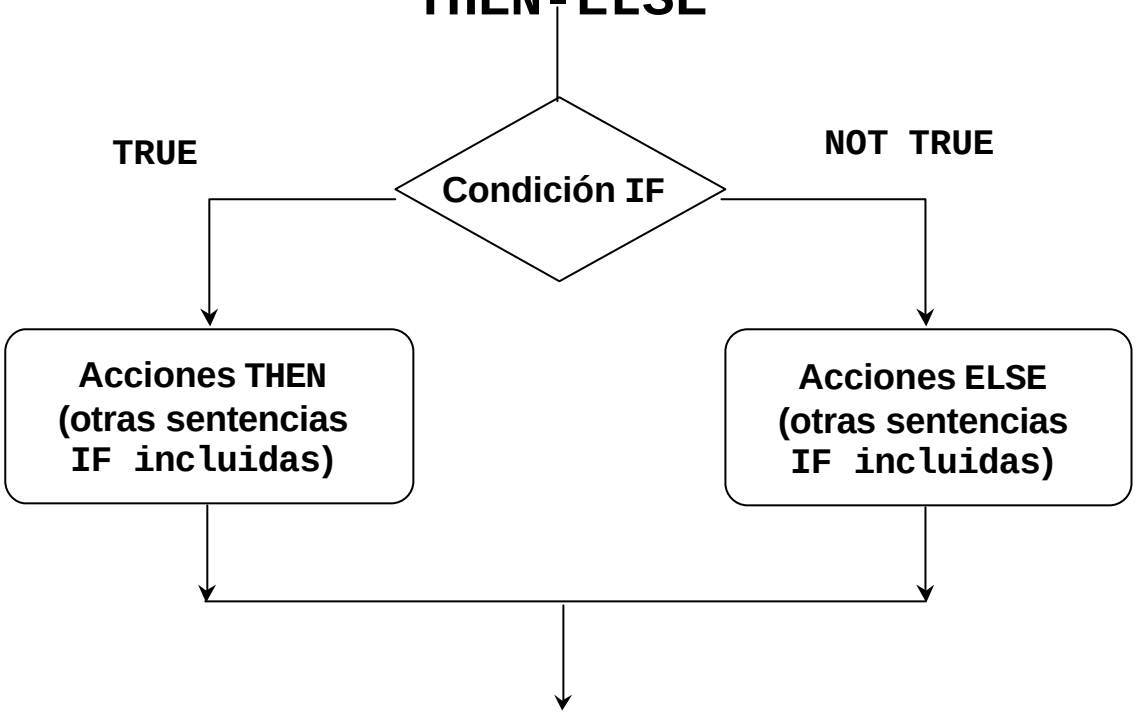
```
. . .
IF v_department = '60' OR v_hiredate > '01-Dec-1999' THEN
      v_mgr := 101;
END IF;
. . .
```

En el ejemplo anterior, la sentencia IF tiene que evaluar dos condiciones:

- El identificador del departamento debería ser 60
- La fecha de contratación debería ser posterior a 01-Dic-1999

En el caso de que cualquiera de las dos condiciones anteriores se evalúen como TRUE, v_mgr se definirá como 101.

Flujo de Ejecución de Sentencias IF-THEN-ELSE



ORACLE

Flujo de Ejecución de Sentencias IF-THEN-ELSE

Al escribir una construcción IF, si la condición es FALSE o NULL, puede usar la cláusula ELSE para realizar otras acciones. Al igual que en las sentencias IF simples, el control continúa en el programa desde la cláusula END IF. Por ejemplo:

```
IF condición1 THEN
  sentencia1;
ELSE
  sentencia2;
END IF;
```

Sentencias IF Anidadas

Cualquier conjunto de acciones del resultado de la primera sentencia IF puede incluir otras sentencias IF, antes de realizar acciones específicas. Las cláusulas THEN y ELSE pueden incluir sentencias IF. Cada sentencia IF anidada debe terminar con su cláusula END IF correspondiente.

```
IF condición1 THEN
  sentencia1;
ELSE
  IF condición2 THEN
    sentencia2;
  END IF;
END IF;
```

Sentencias IF-THEN-ELSE

Defina un indicador booleano como TRUE si la fecha de contratación es mayor de cinco años; de lo contrario, defina el indicador booleano como FALSE.

```
DECLARE
   v_hire_date  DATE := '12-Dec-1990';
   v_five_years BOOLEAN;
BEGIN
. . .
IF MONTHS_BETWEEN(SYSDATE,v_hire_date)/12 > 5 THEN
   v_five_years := TRUE;
ELSE
   v_five_years := FALSE;
END IF;
...
```

ORACLE

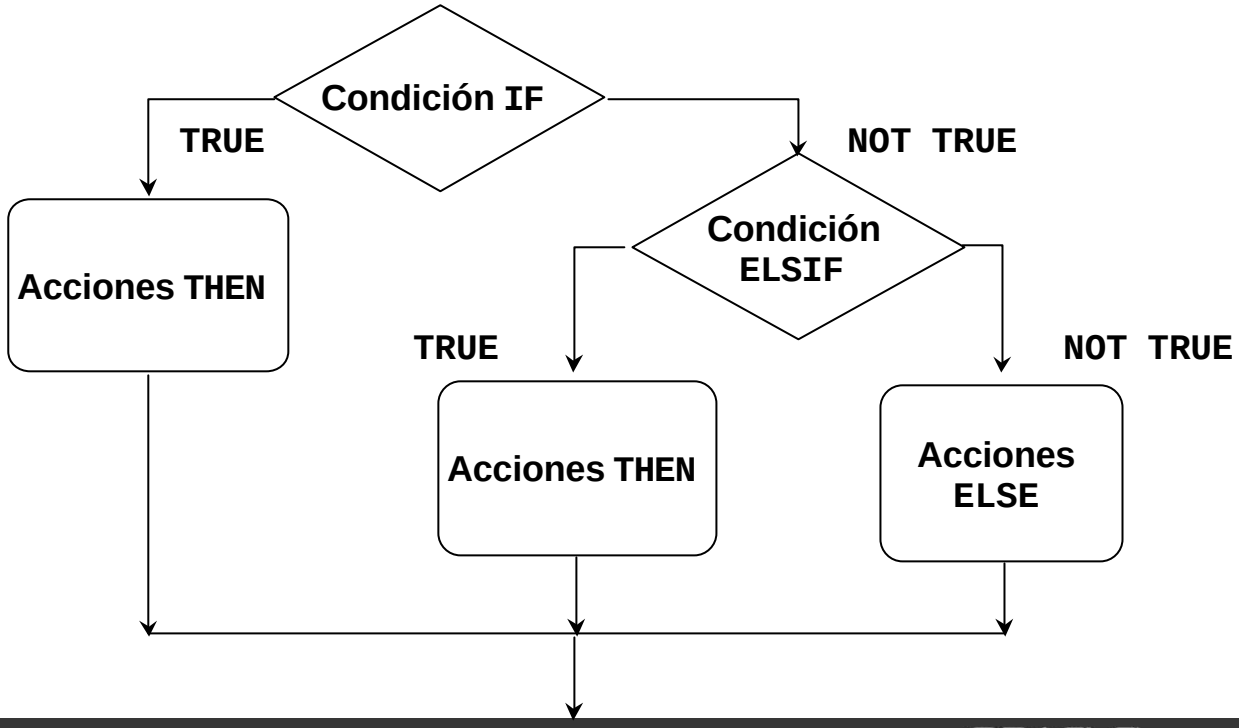
Sentencias IF-THEN-ELSE: Ejemplo

En el ejemplo de la transparencia, la función MONTHS_BETWEEN sirve para calcular la diferencia de meses entre la fecha actual y la variable _hire_date. Dado que el resultado es la diferencia entre las dos fechas en meses, hay que dividir el valor resultante entre 12 para convertir el resultado en años. Si el valor resultante es mayor de 5, el indicador booleano se define como TRUE; de lo contrario, se define como FALSE.

Observe el siguiente ejemplo: Compruebe el valor de la variable v_ename. Si el valor es King, defina la variable v_job como AD_PRES. De lo contrario, defina la variable v_job como ST_CLERK.

```
IF v_ename  = 'King' THEN
   v_job   := 'AD_PRES';
ELSE
   v_job   := 'ST_CLERK';
END IF;
```

Flujo de ejecución de sentencias IF-THEN-ELSIF



ORACLE

Flujo de Ejecución de Sentencias IF-THEN-ELSIF

En ocasiones, hay que seleccionar una acción entre varias alternativas que son mutuamente excluyentes. El tercer tipo de sentencia `IF` utiliza la palabra clave `ELSIF` (no `ELSEIF`) para introducir condiciones adicionales, de la siguiente manera:

```
IF condición1 THEN
   secuencia_de_sentencias1;
ELSIF condición2 THEN
   secuencia_de_sentencias2;
ELSE
   secuencia_de_sentencias3;
END IF;
```

Flujo de Ejecución de Sentencias `IF-THEN-ELSIF` (continuación)

Si la primera condición es falsa o nula, la cláusula `ELSIF` prueba otra condición. Una sentencia `IF` puede tener cualquier número de cláusulas `ELSIF`; la cláusula `ELSE` final es opcional. Las condiciones se evalúan de una en una desde arriba hacia abajo. Si alguna de las condiciones es `TRUE`, se ejecuta su secuencia asociada de sentencias y el control pasa a la siguiente sentencia. Si todas las condiciones son `FALSE` o `NULL`, se ejecuta la secuencia de la cláusula `ELSE`. Observe el siguiente ejemplo: Determine la bonificación de un empleado en base al departamento al que pertenece.

```
IF v_deptno = 10 THEN
  v_bonus := 5000;
ELSIF v_deptno = 80 THEN
  v_bonus := 7500;
ELSE
  v_bonus := 2000;
END IF;
```

Nota: En caso de que haya múltiples sentencias `IF–ELSIF`, sólo se procesará la primera sentencia verdadera.

Sentencias IF-THEN-ELSIF

Calcule un porcentaje de un valor determinado basándose en una condición.

Ejemplo:

```
. . .
IF    v_start > 100 THEN
      v_start := 0.2 * v_start;
ELSIF v_start >= 50 THEN
      v_start := 0,5 * v_start;
ELSE
      v_start := 0.1 * v_start;
END IF;
. . .
```

ORACLE

Sentencias `IF-THEN-ELSIF`

Cuando sea posible, utilice la cláusula `ELSIF` en lugar de sentencias `IF` anidadas. La razón es que el código es más fácil de leer y de comprender y la lógica está claramente identificada. Si la acción de la cláusula `ELSE` consta únicamente de otra sentencia `IF`, es más cómodo utilizar la cláusula `ELSIF`. De esta manera, el código es más claro, ya que no hay ninguna necesidad de utilizar sentencias `END IF` al final de cada conjunto de condiciones y acciones adicionales.

Ejemplo

```
IF condición1 THEN
  sentencia1;
ELSIF condición2 THEN
  sentencia2;
ELSIF condición3 THEN
  sentencia3;
END IF;
```

El ejemplo de la sentencia `IF-THEN-ELSIF` anterior se define de la siguiente manera:

Calcular un porcentaje de un valor determinado original. Si el valor es mayor de 100, entonces el valor calculado es el doble del valor inicial. Si el valor se encuentra entre 50 y 100, entonces el valor calculado es la mitad del valor inicial. Si el valor introducido es menor de 50, entonces el valor calculado es el 10% del valor inicial.

Nota: Cualquier expresión aritmética que contenga valores nulos se evalúa como nula.

Expresiones CASE

- **La expresión CASE selecciona un resultado y lo devuelve.**
- **Para seleccionar el resultado, la expresión CASE utiliza una expresión cuyo valor se usa para seleccionar una de varias alternativas.**

```
CASE selector
   WHEN expresión1 THEN resultado1
   WHEN expresión2 THEN resultado2
   ...
   WHEN expresiónN THEN resultadoN
  [ELSE resultadoN+1;]
END;
```


Expresiones CASE

La expresión CASE selecciona un resultado y lo devuelve. Para seleccionar el resultado, la expresión CASE utiliza un selector, que es una expresión cuyo valor sirve para seleccionar una de varias alternativas. El selector va seguido de una o varias cláusulas WHEN, que se comprueban secuencialmente. El valor del selector determina qué cláusula se ejecuta. Si el valor del selector es igual al valor de una expresión de cláusula WHEN, se ejecuta esa cláusula WHEN.

PL/SQL también proporciona una expresión CASE de búsqueda, que tiene la siguiente forma:

```
CASE
   WHEN condición1_búsqueda THEN resultado1
   WHEN condición2_búsqueda THEN resultado2
   ...
   WHEN condiciónN_búsqueda THEN resultadoN
  [ELSE resultadoN+1;]
END;
/
```

La expresión CASE de búsqueda no tiene selector. Además, las cláusulas WHEN contienen condiciones de búsqueda que devuelven un valor booleano, no expresiones que pueden devolver un valor de cualquier tipo.

Expresiones CASE: Ejemplo

```
SET SERVEROUTPUT ON
DECLARE
   v_grade CHAR(1) := UPPER('&p_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;
/
```

ORACLE

Expresiones CASE: Ejemplo

En el ejemplo de la transparencia, la expresión CASE utiliza el valor de la variable `v_grade` como expresión. El usuario acepta este valor utilizando una variable de sustitución. En función del valor introducido por el usuario, la expresión CASE evalúa el valor de la variable `v_appraisal` basándose en el valor de `v_grade`. El resultado del ejemplo anterior sería el siguiente:

```
old 2: v_grade CHAR(1) := UPPER('&p_grade');
new 2: v_grade CHAR(1) := UPPER('a');
Grade: A Appraisal Excellent
PL/SQL procedure successfully completed.
```

Expresiones CASE: Ejemplo (continuación)

Si el ejemplo de la transparencia se escribe utilizando una expresión CASE buscada, tendría el siguiente aspecto:
REM Cuando se solicite, proporcione p_grade = a en el siguiente código.

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

Gestión de Valores Nulos

A la hora de trabajar con valores nulos, se puede evitar cometer algunos de los errores más comunes si se tienen en cuenta las siguientes reglas:

- **Las comparaciones simples en las que se utilizan valores nulos siempre devuelven NULL.**
- **Si se aplica el operador lógico `NOT` a un valor nulo, se devolverá `NULL`.**
- **En las sentencias de control condicional, si la condición devuelve `NULL`, su secuencia de sentencias asociada no se ejecuta.**

ORACLE

Gestión de Valores Nulos

En el siguiente ejemplo, es de esperar que se ejecute la secuencia de sentencias porque *x* e *y* parecen diferentes. No obstante, los valores nulos son indeterminados. No se sabe si *x* es igual a *y*. Por lo tanto, la condición `IF` devuelve `NULL` y se ignora la secuencia de sentencias.

```
x := 5;
y := NULL;
...
IF x != y THEN  -- devuelve NULL, no TRUE
   secuencia_de_sentencias;  -- no se ejecuta
END IF;
```

En el siguiente ejemplo, es de esperar la secuencia de sentencias que se ejecute porque *a* y *b* parecen iguales. Pero, una vez más, es imposible saberlo, así que la condición `IF` devuelve `NULL` y se ignora la secuencia de sentencias.

```
a := NULL;
b := NULL;
...
IF a = b THEN  -- devuelve NULL, no TRUE
   secuencia_de_sentencias;  -- no se ejecuta
END IF;
```

Tablas Lógicas

Cree una condición booleana simple con un operador de comparación.

AND	*TRUE*	*FALSE*	*NULL*
TRUE	TRUE	FALSE	NULL
FALSE	FALSE	FALSE	FALSE
NULL	NULL	FALSE	NULL

OR	*TRUE*	*FALSE*	*NULL*
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	NULL
NULL	TRUE	NULL	NULL

NOT	
TRUE	FALSE
FALSE	TRUE
NULL	NULL

ORACLE

Condiciones Booleanas con Operadores Lógicos

Se puede crear una condición booleana simple combinando expresiones de número, carácter o fecha con operadores de comparación.

Se puede crear una condición booleana compleja combinando condiciones booleanas simples con los operadores lógicos AND, OR y NOT. En las tablas lógicas de la transparencia:

- FALSE tiene precedencia en las condiciones AND, y TRUE tiene precedencia en las condiciones OR.
- AND devuelve TRUE sólo si sus dos operandos son TRUE.
- OR devuelve FALSE sólo si sus dos operandos son FALSE.
- NULL AND TRUE siempre se evalúa como NULL porque no se sabe si el segundo operando se evalúa como TRUE o no.

Nota: La negación de NULL (NOT NULL) produce un valor nulo porque los valores nulos son indeterminados.

Condiciones Booleanas

¿Qué valor tiene V_FLAG en cada caso?

```
v_flag := v_reorder_flag AND v_available_flag;
```

V_REORDER_FLAG	V_AVAILABLE_FLAG	V_FLAG
TRUE	TRUE	?
TRUE	FALSE	?
NULL	TRUE	?
NULL	FALSE	?

ORACLE

Creación de Condiciones Lógicas

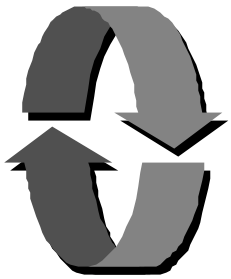
La tabla lógica AND le ayudará a evaluar las posibilidades de la condición booleana de la transparencia.

Respuestas

1. TRUE
2. FALSE
3. NULL
4. FALSE

Control Iterativo: Sentencias LOOP

- **Los bucles repiten múltiples veces una sentencia o una secuencia de sentencias.**
- **Existen tres tipos de bucles:**
  - **Bucle básico**
  - **Bucle FOR**
  - **Bucle WHILE**



ORACLE

Control Iterativo: Sentencias LOOP

PL/SQL ofrece una serie de facilidades para estructurar bucles que repiten múltiples veces una sentencia o una secuencia de sentencias.

Las construcciones de bucles son el segundo tipo de estructura de control. PL/SQL proporciona los siguientes tipos de bucles:

- Un bucle básico que realiza acciones repetitivas sin condiciones globales
- Bucles FOR que realizan controles iterativos de acciones basándose en un contador
- Bucles `WHILE` que realizan controles iterativos de acciones basándose en una condición

Utilice la sentencia `EXIT` para finalizar los bucles.

Para obtener más información, consulte el capítulo "Control Structures" de *PL/SQL User's Guide and Reference*.

Nota: En una lección posterior, se explicará otro tipo de bucle `FOR LOOP`, el bucle `FOR LOOP` para un cursor.

Bucles Básicos

Sintaxis:

```
LOOP                                    -- delimitador
  sentencia1;                           -- sentencias
  . . .
  EXIT [WHEN condición];                -- sentencia EXIT
END LOOP;                               -- delimitador
```

condición es una expresión o variable booleana (TRUE, FALSE o NULL);

ORACLE

Bucles Básicos

El tipo más sencillo de sentencia `LOOP` es el bucle básico (o infinito), que encierra una secuencia de sentencias entre las palabras clave `LOOP` y `END LOOP`. Cada vez que el flujo de ejecución llega a la sentencia `END LOOP`, el control se devuelve a la sentencia `LOOP` correspondiente por encima de ella. Un bucle básico permite la ejecución de su sentencia al menos una vez, aunque la condición ya se haya cumplido antes de entrar en el bucle. Sin la sentencia `EXIT`, el bucle sería infinito.

La Sentencia EXIT

Utilice la sentencia `EXIT` para finalizar los bucles. El control pasa a la sentencia que hay después de la sentencia `END LOOP`. Puede emitir `EXIT` como una acción en el interior de una sentencia `IF`, o como una sentencia autónoma en el interior del bucle. La sentencia `EXIT` se debe colocar en el interior de un bucle. En este caso, puede añadir una cláusula `WHEN` para permitir que el bucle termine de manera condicional. Cuando se encuentra con la sentencia `EXIT`, la condición de la cláusula `WHEN` se evalúa. Si esta condición devuelve `TRUE`, el bucle finaliza y el control pasa a la siguiente sentencia que hay después del bucle. Un bucle básico puede contener múltiples sentencias `EXIT`.

Bucles Básicos

Ejemplo:

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

ORACLE

Bucles Básicos (continuación)

El ejemplo de bucle básico que se muestra en la transparencia se define de la siguiente manera: Insertar tres nuevos identificadores de ubicaciones para el código de país de CA y la ciudad de Montreal.

Nota: Los bucles básicos permiten ejecutar sus sentencias al menos una vez, aunque la condición ya se haya cumplido al entrar en el bucle, siempre y cuando la condición se coloque en el bucle, para que no se compruebe hasta después de estas sentencias. Sin embargo, si la condición de salida se sitúa en la parte superior del bucle, antes de cualquiera de las otras sentencias ejecutables, y esa condición es verdadera, el bucle finalizará y las sentencias nunca se ejecutarán.

Bucles WHILE

Sintaxis:

```
WHILE condición LOOP
  sentencia1;
  sentencia2;
  . . .
END LOOP;
```

La condición se evalúa al principio de cada iteración.

Utilice el bucle WHILE para repetir sentencias mientras una condición sea TRUE.

ORACLE

Bucles WHILE

El bucle WHILE sirve para repetir una secuencia de sentencias hasta que la condición de control ya no sea TRUE. La condición se evalúa al principio de cada iteración. El bucle termina cuando la condición es FALSE. Si la condición es FALSE al principio del bucle, no se realiza ninguna iteración más.

En la sintaxis:

condición es una variable o expresión booleana (TRUE, FALSE o NULL).

sentencia *puede ser una o varias sentencias PL/SQL o SQL.*

Si las variables que se utilizan en las condiciones no cambian a lo largo del cuerpo del bucle, la condición sigue siendo TRUE y el bucle no termina.

Nota: Si la condición devuelve NULL, se ignora el bucle y el control pasa a la siguiente sentencia.

Bucles WHILE

Ejemplo:

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

ORACLE

Bucles WHILE (continuación)

En el ejemplo de la transparencia, se están agregando tres nuevos identificadores de ubicaciones para el código de país de CA y la ciudad de Montreal.

Con cada iteración a través del bucle WHILE, se incrementa un contador (v_counter). Si el número de iteraciones es menor o igual al número 3, se ejecuta el código que hay en el interior del bucle y se inserta una fila en la tabla LOCATIONS. Después de que el contador supere el número de elementos de esta ubicación, la condición que controla el bucle se evalúa como FALSE y el bucle termina.

Bucles FOR

Sintaxis:

```
FOR contador IN [REVERSE]
    límite_inferior..límite_superior LOOP
  sentencia1;
  sentencia2;
  . . .
END LOOP;
```

- **Utilice un bucle FOR para abreviar la prueba del número de iteraciones.**
- **No declare el contador, ya que se declara implícitamente.**
- **La sintaxis requerida es 'límite_inferior .. límite_superior'.**

ORACLE

Bucles FOR

Los bucles FOR tienen la misma estructura general que los bucles básicos. Pero, además, poseen una sentencia de control antes de la palabra clave LOOP para determinar el número de iteraciones que realiza PL/SQL. En la sintaxis:

contador	es un número entero declarado implícitamente cuyo valor aumenta o disminuye automáticamente (disminuye si se utiliza la palabra clave REVERSE) por 1 en cada iteración del bucle hasta que se llega al límite inferior o superior.
REVERSE	hace que el contador disminuya con cada iteración desde el límite superior hasta el límite inferior. (Tenga en cuenta que el límite inferior se sigue referenciando en primer lugar).
límite_inferior	especifica el límite inferior del rango de valores del contador.
límite_superior	especifica el límite superior del rango de valores del contador.

No declare el contador, ya que se declara implícitamente como un entero.

Nota: La secuencia de sentencias se ejecuta cada vez que aumenta el contador, tal y como determinan los dos límites. El límite inferior y el límite superior del rango del bucle pueden ser literales, variables o expresiones, pero se deben evaluar como enteros. El límite inferior y el límite superior están incluidos en el rango del bucle. Si el límite inferior del rango del bucle se evalúa como un entero mayor que el límite superior, la secuencia de sentencias no se ejecutará, siempre y cuando no se haya utilizado REVERSE. Por ejemplo, la siguiente sentencia sólo se ejecuta una vez:

```
FOR i IN 3..3 LOOP sentencia1; END LOOP;
```

Bucles FOR

Inserte tres nuevos identificadores de ubicaciones para el código de país de CA y la ciudad de Montreal.

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

ORACLE

Bucles FOR (continuación)

El ejemplo de la transparencia se define de la siguiente manera: Insertar tres nuevos identificadores de ubicaciones para el código de país de CA y la ciudad de Montreal.

Para ello, se utiliza un bucle FOR.

Bucles FOR

Instrucciones

- **Referencie el contador únicamente en el interior del bucle; es indefinido en el exterior del bucle.**
- ***No* referencie el contador como destino de una asignación.**

ORACLE

Bucles FOR (continuación)

La transparencia muestra las instrucciones que hay que seguir para escribir un bucle FOR.

Nota: Para escribir un bucle FOR, no es necesario que los límites inferiores y superiores de una sentencia LOOP sean literales numéricos. Pueden ser expresiones que se convierten en valores numéricos.

Ejemplo

```
DECLARE
  v_lower          NUMBER := 1;
  v_upper          NUMBER := 100;
BEGIN
  FOR i IN v_lower..v_upper LOOP
  ...
  END LOOP;
END;
```

Instrucciones para Utilizar Bucles

- **Utilice el bucle básico cuando las sentencias que hay en el interior de dicho bucle se deban ejecutar al menos una vez.**
- **Utilice el bucle `WHILE` si hay que evaluar la condición al principio de cada iteración.**
- **Utilice el bucle `FOR` si conoce el número de iteraciones.**

ORACLE

Instrucciones para Utilizar Bucles

Un bucle básico permite la ejecución de su sentencia al menos una vez, aunque la condición ya se haya cumplido antes de entrar en el bucle. Sin la sentencia `EXIT`, el bucle sería infinito.

El bucle `WHILE` sirve para repetir una secuencia de sentencias hasta que la condición de control ya no sea `TRUE`. La condición se evalúa al principio de cada iteración. El bucle termina cuando la condición es `FALSE`. Si la condición es `FALSE` al principio del bucle, no se realiza ninguna iteración más.

Los bucles `FOR` poseen una sentencia de control antes de la palabra clave LOOP para determinar el número de iteraciones que realiza PL/SQL. Utilice el bucle `FOR` si el número de iteraciones está predeterminado.

Etiquetas y Bucles Anidados

- **Anide bucles en múltiples niveles.**
- **Utilice etiquetas para distinguir los bloques de los bucles.**
- **Salga del bucle exterior con la sentencia `EXIT` que referencia la etiqueta.**

ORACLE

Etiquetas y Bucles Anidados

Puede anidar bucles en múltiples niveles. Se pueden anidar bucles `FOR`, `WHILE` y básicos unos en el interior de los otros. La terminación de un bucle anidado no supone la terminación del bucle que lo contiene, a menos que se haya emitido una excepción. Sin embargo, se pueden etiquetar los bucles y salir del bucle exterior con la sentencia `EXIT`.

Los nombres de las etiquetas siguen las mismas reglas que el resto de los identificadores. La etiqueta se coloca antes de la sentencia, ya sea en la misma línea o en una línea diferente. Etiquete los bucles colocando la etiqueta antes de la palabra LOOP entre delimitadores (<<*etiqueta*>>).

Si el bucle tiene una etiqueta, se puede incluir su nombre después de la sentencia `END LOOP` para que quede más claro.

Etiquetas y Bucles Anidados

```
...
BEGIN
  <<Outer_loop>>
  LOOP
    v_counter := v_counter+1;
  EXIT WHEN v_counter>10;
    <<Inner_loop>>
    LOOP
      ...
      EXIT Outer_loop WHEN total_done = 'YES';
      -- Salir de ambos bucles
      EXIT WHEN inner_done = 'YES';
      -- Salir sólo del bucle interior
      ...
    END LOOP Inner_loop;
    ...
  END LOOP Outer_loop;
END;
```

ORACLE

Etiquetas y Bucles Anidados (continuación)

En el ejemplo de la transparencia, hay dos bucles. El bucle exterior se identifica por la etiqueta `<<Outer_Loop>>` y el bucle interior se identifica por la etiqueta `<<Inner_Loop>>`. Los identificadores se colocan antes de la palabra `LOOP` entre delimitadores (*<<etiqueta>>*). El bucle interior se anida en el interior del bucle exterior. Para que el código quede más claro, los nombres de las etiquetas se incluyen después de la sentencia `END LOOP`.

Resumen

En esta lección, ha aprendido a:

Cambiar el flujo lógico de sentencias utilizando estructuras de control.

- **Sentencias condicionales (sentencias `IF`)**
- **Expresiones `CASE`**
- **Bucles:**
  - **Bucle básico**
  - **Bucle `FOR`**
  - **Bucle `WHILE`**
- **Sentencias `EXIT`**

ORACLE

Resumen

Las construcciones de control condicionales comprueban la validez de una condición y realizan la acción correspondiente. Hay que utilizar la construcción `IF` para realizar una ejecución condicional de sentencias.

Las construcciones de control iterativas ejecutan una secuencia de sentencias repetidamente, mientras la condición especificada siga siendo `TRUE`. Hay que utilizar las diversas construcciones de bucles para realizar operaciones iterativas.

Visión General de la Práctica 4

Esta práctica cubre los siguientes temas:

- **Realización de acciones condicionales utilizando la sentencia `IF`**
- **Realización de pasos iterativos utilizando la estructura de bucles**

ORACLE

Visión General de la Práctica 4

En esta práctica, hay que crear bloques PL/SQL que incorporen bucles y estructuras de control condicionales. Las prácticas ponen a prueba el conocimiento del alumno acerca de la creación de las diversas sentencias `IF` y construcciones `LOOP`.

Práctica 4

1. Ejecute el comando del archivo `lab04_1.sql` para crear la tabla `MESSAGES`. Escriba un bloque PL/SQL para insertar números en la tabla `MESSAGES`.
   a. Inserte los números del 1 al 10, excluyendo el 6 y el 8.
   b. Valide antes del final de bloque.
   c. Realice una selección en la tabla `MESSAGES` para verificar que el bloque PL/SQL funciona.

RESULTS
1
2
3
4
5
7
9
10

8 rows selected.

2. Cree un bloque PL/SQL que calcule la comisión de un empleado determinado basándose en el sueldo de dicho empleado.
   a. Utilice el comando `DEFINE` para proporcionar el identificador del empleado. Pase el valor al bloque PL/SQL mediante una variable de sustitución de *i*SQL*Plus.

      ```
      DEFINE p_empno = 100
      ```

   b. Si el sueldo del empleado es menor de $5.000, muestre la bonificación del empleado como un 10% del sueldo.
   c. Si el sueldo del empleado está entre $5.000 y $10.000, muestre la bonificación del empleado como un 15% del sueldo.
   d. Si el sueldo del empleado supera los $10.000, muestre la bonificación del empleado como un 20% del sueldo.
   e. Si el sueldo del empleado es NULL, muestre la bonificación del empleado como 0.
   f. Pruebe el bloque PL/SQL en cada caso utilizando los siguientes casos de prueba y compruebe cada bonificación.

Nota: Incluya `SET VERIFY OFF` en su solución.

Número de Empleado	Sueldo	Bonificación Resultante
100	24000	4800
149	10500	2100
178	7000	1050

Práctica 4 (continuación)

Si le queda tiempo, realice los siguientes ejercicios:

3. Cree una tabla EMP que sea una réplica de la tabla EMPLOYEES. Para ello, ejecute el archivo de comandos
   lab04_3.sql. Añada una nueva columna, STARS, del tipo de dato VARCHAR2 y con una longitud de valor 50 a la tabla EMP para guardar asteriscos (*).

   ```
   Table altered.
   ```

4. Cree un bloque PL/SQL que recompense al empleado añadiendo un asterisco a la columna STARS por cada $1000 de sueldo de dicho empleado. Guarde dicho bloque PL/SQL en un archivo con el nombre p4q4.sql haciendo clic en el botón Save Script. No se olvide de guardar el archivo de comandos con la extensión .sql.
   a. Utilice el comando DEFINE para proporcionar el identificador del empleado. Pase el valor al bloque PL/SQL mediante una variable de sustitución de *i*SQL*Plus.
      ```
      DEFINE p_empno=104
      ```
   b. Inicialice una variable v_asterisk que contenga un valor NULL.
   c. Añada un asterisco a la cadena por cada $1000 del sueldo. Por ejemplo, si el empleado tiene un sueldo de $8000, la cadena de asteriscos debería contener ocho asteriscos. Y, si el empleado tiene un sueldo de $12500, la cadena de asteriscos debería contener 13 asteriscos.
   d. Actualice la columna STARS del empleado con la cadena de asteriscos.
   e. Valide.
   f. Pruebe los siguientes valores del bloque:
      ```
      DEFINE p_empno=174
      DEFINE p_empno=176
      ```
   g. Muestre las filas de la tabla EMP para verificar si el bloque PL/SQL se ha ejecutado con éxito.

EMPLOYEE_ID	SALARY	STARS
104	6000	******
174	11000	**********
176	8600	********

Nota: Especifique SET VERIFY OFF en el bloque PL/SQL.

5

Trabajo con Tipos de Dato Compuestos

ORACLE

Objetivos

Al finalizar este curso, debería estar capacitado para hacer lo siguiente:

- **Crear registros PL/SQL definidos por el usuario**
- **Crear un registro con el atributo `%ROWTYPE`**
- **Crear una tabla `INDEX BY`**
- **Crear una tabla de registros `INDEX BY`**
- **Describir la diferencia entre registros, tablas y tablas de registros**

ORACLE

Finalidad de la Lección

En esta lección, aprenderá más en el tema de los tipos de dato compuestos y sus usos.

Tipos de Dato Compuestos

- **Existen dos tipos:**
  - **Registros (RECORD) PL/SQL**
  - **Recopilaciones PL/SQL**
    - **Tablas INDEX BY**
    - **Tablas anidadas**
    - **VARRAY**
- **Contienen componentes internos**
- **Son reutilizables**

ORACLE

Tipos de Dato RECORD y TABLE

Al igual que las variables escalares, las variables compuestas también poseen un tipo de dato. Los tipos de dato compuestos (también llamados recopilaciones) son RECORD, TABLE, NESTED TABLE y VARRAY. El tipo de dato RECORD sirve para tratar los datos relacionados, pero distintos, como una unidad lógica. El tipo de dato TABLE sirve para hacer referencia y manipular las recopilaciones de datos como un objeto completo. Los tipos de dato NESTED TABLE y VARRAY se explican en el curso *Advanced PL/SQL*.

Un registro es un grupo de datos relacionados que se almacenan como campos, cada uno de ellos con su propio nombre y tipo de dato. Una tabla contiene una columna y una clave primaria, que le permiten tener acceso de tipo matriz a las filas. Una vez definidos, las tablas y los registros se pueden volver a utilizar.

Para obtener más información, consulte el capítulo "Collections and Records" de *PL/SQL User's Guide and Reference*.

Registros PL/SQL

- **Deben contener uno o varios componentes de cualquier tipo de dato de tabla `INDEX BY`, `RECORD` o escalar, que se denominan campos**
- **Su estructura es similar a la de los registros de un lenguaje de tercera generación (3GL)**
- **No son iguales que las filas de una tabla de base de datos**
- **Tratan las recopilaciones de campos como unidades lógicas**
- **Son prácticos para recuperar una fila de datos de una tabla para realizar el procesamiento**

ORACLE

Registros PL/SQL

Un *registro* es un grupo de datos relacionados que se almacenan en *campos*, cada uno con su propio nombre y tipo de dato. Suponga, por ejemplo, que posee diferentes tipos de datos acerca de un empleado como, por ejemplo, su nombre, su sueldo, su fecha de contratación, etc. Estos datos son de tipo diferente pero están relacionados de manera lógica. Un registro que contiene campos como, por ejemplo, el nombre, el sueldo y la fecha de contratación de un empleado permite tratar los datos como una unidad lógica. Cuando se declara un tipo de registro para estos campos, se pueden manipular como una unidad.

- Cada registro definido puede tener todos los campos que sean necesarios.
- A los registros se les pueden asignar valores iniciales y se pueden definir como `NOT NULL`.
- Los campos que carecen de valores iniciales se inicializan como `NULL`.
- A la hora de definir campos, también se puede utilizar la palabra clave `DEFAULT`.
- Se pueden definir tipos de `RECORD` y declarar registros definidos por el usuario en la parte declarativa de cualquier bloque, subprograma o paquete.
- Se puede declarar y hacer referencia a registros anidados. Un registro puede ser un componente de otro registro.

Creación de un Registro PL/SQL

Sintaxis:

```
TYPE nombre_tipo IS RECORD
     (declaración_campo[, declaración_campo]…);
identificador nombre_tipo;
```

Donde *declaración_campo* es:

```
nombre_campo {tipo_campo | variable%TYPE
           | tabla.columna%TYPE | tabla%ROWTYPE}
           [[NOT NULL] {:= | DEFAULT} expr]
```

ORACLE

Definición y Declaración de un Registro PL/SQL

Para crear un registro, hay que definir un tipo `RECORD` y luego declarar registros de ese tipo.

En la sintaxis:

nombre_tipo	es el nombre del tipo `RECORD`. (Este identificador sirve para declarar registros.)
nombre_campo	es el nombre de un campo situado en el interior del registro.
tipo_campo	es el tipo de dato del campo. (Representa cualquier tipo de dato PL/SQL excepto `REF CURSOR`. Puede utilizar los atributos
`%TYPE` y	`%ROWTYPE`.)
expr	es el *tipo_campo* o un valor inicial.

La restricción de No Nulo (`NOT NULL`) evita que se asignen valores nulos a esos campos. Asegúrese de inicializar campos `NOT NULL`.

Creación de un Registro PL/SQL

Declare variables para almacenar el nombre, el puesto y el sueldo de un nuevo empleado.

Ejemplo:

```
...
  TYPE emp_record_type IS RECORD
    (last_name   VARCHAR2(25),
     job_id      VARCHAR2(10),
     salary      NUMBER(8,2));
  emp_record     emp_record_type;
...
```

ORACLE

Creación de un Registro PL/SQL

Las declaraciones de campo son como las declaraciones de variables. Cada campo posee un nombre y un tipo de dato específico únicos. Los registros PL/SQL no tienen tipos de dato predefinidos como las variables escalares. Por lo tanto, primero debe crear el tipo de registro y luego declarar un identificador utilizando ese tipo.

En el ejemplo de la transparencia, se ha definido un tipo de registro `EMP_RECORD_TYPE` para guardar los valores de `last_name`, `job_id` y `salary`. En el siguiente paso, se declara el registro `EMP_RECORD` del tipo `EMP_RECORD_TYPE`.

El siguiente ejemplo demuestra que se puede utilizar el atributo `%TYPE` para especificar un tipo de dato de campo:

```
DECLARE
    TYPE emp_record_type IS RECORD
      (employee_id      NUMBER(6) NOT NULL := 100,
       last_name        employees.last_name%TYPE,
       job_id           employees.job_id%TYPE);
       emp_record       emp_record_type;
       ...
```

Nota: Se puede agregar la restricción de No Nulo (`NOT NULL`) a cualquier declaración de campo para evitar asignar valores nulos a ese campo. Recuerde que hay que inicializar los campos declarados como `NOT NULL`.

Estructura de Registros PL/SQL

Ejemplo:

ORACLE

Estructura de Registros PL/SQL

Para acceder a los campos de un registro, se utiliza el nombre de dicho campo. Para hacer referencia o inicializar un campo individual, utilice la notación de puntos y la siguiente sintaxis:

```
nombre_registro.nombre_campo
```

Por ejemplo, para hacer referencia al campo `job_id` del registro `emp_record`, hay que hacer lo siguiente:

```
emp_record.job_id ...
```

A continuación, puede asignar un valor al campo del registro de la siguiente manera:

```
emp_record.job_id := 'ST_CLERK';
```

En los bloques o los subprogramas, los registros definidos por el usuario se instancian cuando se entra en el bloque o en el subprograma y dejan de existir cuando se sale del bloque o del subprograma.

El Atributo %ROWTYPE

- **Declare una variable de acuerdo con una recopilación de columnas de una tabla o una vista de base de datos.**
- **Prefije %ROWTYPE con la tabla de base de datos.**
- **Los campos del registro obtienen sus nombres y tipos de dato de las columnas de la tabla o la vista.**

ORACLE

Declaración de Registros con el Atributo %ROWTYPE

Para declarar un registro basándose en una recopilación de columnas de una tabla o una vista de base de datos, hay que utilizar el atributo %ROWTYPE. Los campos del registro obtienen sus nombres y tipos de dato de las columnas de la tabla o la vista. El registro también puede almacenar una fila completa de datos que se ha recuperado desde un cursor o una variable de cursor.

En el siguiente ejemplo, se ha declarado un registro utilizando %ROWTYPE como especificador de tipo de dato.

```
DECLARE
      emp_record     employees%ROWTYPE;
      ...
```

La estructura del registro emp_record constará de los siguientes campos, cada uno de los cuales representa una columna de la tabla EMPLOYEES.

Nota: Esto no es un ejemplo de código, sino simplemente la estructura de la variable compuesta.

```
  (employee_id       NUMBER(6),
   first_name        VARCHAR2(20),
   last_name         VARCHAR2(20),
   email             VARCHAR2(20),
   phone_number      VARCHAR2(20),
   hire_date         DATE,
   salary            NUMBER(8,2),
   commission_pct    NUMBER(2,2),
   manager_id        NUMBER(6),
   department_id     NUMBER(4))
```

Declaración de Registros con el Atributo %ROWTYPE (continuación)

Sintaxis

```
DECLARE
  identificador referencia%ROWTYPE;
```

donde: *identificador* es el nombre elegido para el registro como una unidad.

referencia es el nombre de la tabla, la vista, el cursor, o la variable de cursor en el cual se va a basar el registro. Para que esta referencia sea válida, la tabla o la vista deben existir.

Para hacer referencia o inicializar un campo individual, utilice la notación de puntos y la siguiente sintaxis:

```
nombre_registro.nombre_campo
```

Por ejemplo, para hacer referencia al campo `commission_pct` del registro `emp_record`, hay que hacer lo siguiente:

```
emp_record.commission_pct
```

Luego, puede asignar un valor al campo de registros de la siguiente manera:

```
emp_record.commission_pct:= .35;
```

Asignación de Valores a Registros

Para asignar una lista de valores comunes a un registro, puede utilizar la sentencia `SELECT` o `FETCH`. Asegúrese de que los nombres de columna aparecen en el mismo orden que los campos del registro. También se puede asignar un registro a otro registro si los dos tienen el mismo tipo de dato. Un registro definido por el usuario y un registro `%ROWTYPE` *nunca* tienen el mismo tipo de dato.

Ventajas del Uso de %ROWTYPE

- **No es necesario conocer el número y los tipos de dato de las columnas de base de datos subyacentes.**
- **El número y los tipos de dato de las columnas de base de datos subyacentes pueden cambiar en tiempo de ejecución.**
- **El atributo resulta útil para recuperar una fila con la sentencia SELECT *.**

ORACLE

Ventajas del Uso de %ROWTYPE

En la transparencia se enumeran las ventajas de utilizar el atributo %ROWTYPE. Utilice este atributo si no está seguro de cuál es la estructura de la tabla de base de datos subyacente. Este atributo también garantiza que los tipos de dato de las variables que se han declarado con él cambian dinámicamente en el caso de que se modifique la tabla subyacente. Este atributo es particularmente útil cuando se desea recuperar una fila entera de una tabla. Si dicho atributo no estuviera presente, no le quedaría más remedio que declarar una variable por cada una de las columnas recuperadas por la sentencia SELECT *.

El Atributo %ROWTYPE

Ejemplos:

Declare una variable para almacenar la información acerca de un departamento de la tabla DEPARTMENTS.

```
dept_record    departments%ROWTYPE;
```

Declare una variable para almacenar la información acerca de un empleado de la tabla EMPLOYEES.

```
emp_record     employees%ROWTYPE;
```

ORACLE

El Atributo %ROWTYPE

En la primera declaración de la transparencia se crea un registro con los mismos nombres de campo y tipos de dato de campo que los de una fila de la tabla DEPARTMENTS. Los campos son DEPARTMENT_ID, DEPARTMENT_NAME, MANAGER_ID y LOCATION_ID. En la segunda declaración se crea un registro con los mismos nombres de campo, tipos de dato de campo y orden que una fila de la tabla EMPLOYEES. Los campos son EMPLOYEE_ID, FIRST_NAME, LAST_NAME, EMAIL, PHONE_NUMBER, HIRE_DATE, JOB_ID, SALARY, COMMISSION_PCT, MANAGER_ID, DEPARTMENT_ID.

El Atributo %ROWTYPE (continuación)

En el siguiente ejemplo, un empleado se va a jubilar. La información acerca de los empleados jubilados se agrega a una tabla en la que se guarda este tipo de información. El usuario proporciona el número de empleado. El registro del empleado especificado por el usuario se recupera de EMPLOYEES y se almacena en la variable emp_rec, que se declara con el atributo %ROWTYPE.

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

A continuación se muestra el registro que se ha insertado en la tabla RETIRED_EMPS:

```
SELECT * FROM RETIRED_EMPS;
```

EMPNO	ENAME	JOB	MGR	HIREDATE	LEAVEDATE	SAL	COMM	DEPTNO
124	Mourgos	ST_MAN	100	16-NOV-99	24-SEP-01	5800		50

Tablas INDEX BY

- **Poseen dos componentes:**
  - **Clave primaria del tipo de dato BINARY_INTEGER**
  - **Columna de tipo de dato escalar o de registro**
- **Pueden aumentar de tamaño dinámicamente porque no están restringidas**

ORACLE

Tablas INDEX BY

Los objetos del tipo TABLE se denominan tablas INDEX BY. Tienen un modelo parecido (aunque no igual) que el de las tablas de base de datos. Las tablas INDEX BY utilizan una clave primaria para proporcionarle un acceso de tipo matriz a las filas.

Las tablas INDEX BY:

- Son parecidas a las matrices
- Deben tener dos componentes:
  - Una clave primaria de tipo de dato BINARY_INTEGER que indexe la tabla INDEX BY
  - Una columna de tipo de dato escalar o de registro que almacene los elementos de la tabla INDEX BY
- Pueden aumentar dinámicamente porque no están restringidas

Creación de una Tabla INDEX BY

Sintaxis:

```
TYPE nombre_tipo IS TABLE OF
     {tipo_columna | variable%TYPE
     | tabla.columna%TYPE} [NOT NULL]
     | tabla.%ROWTYPE
     [INDEX BY BINARY_INTEGER];
identificador nombre_tipo;
```

Declare una tabla INDEX BY para almacenar nombres.

Ejemplo:

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

ORACLE

Creación de una Tabla INDEX BY

La creación de una tabla INDEX BY consta de dos pasos.

1. Declare un tipo de dato TABLE.
2. Declare una variable de ese tipo de dato.

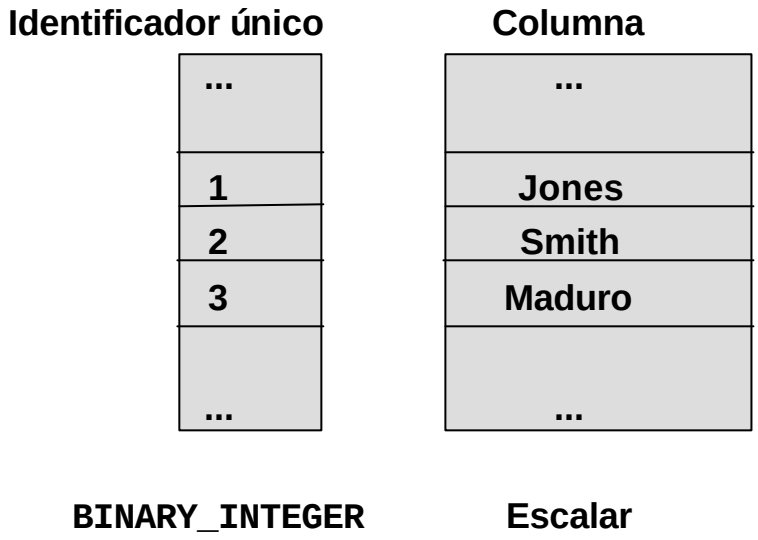
En la sintaxis:

nombre_de_tipo	es el nombre del tipo TABLE. (Es un especificador de tipo que se utiliza en posteriores declaraciones de tablas PL/SQL)
tipo_de_columna	es cualquier tipo de dato escalar (escalar y compuesto), como VARCHAR2, DATE, NUMBER o %TYPE. (Puede utilizar el atributo %TYPE para proporcionar el tipo de dato de la columna.)
identificador	es el nombre del identificador que representa toda una tabla PL/SQL.

La restricción de No Nulo (NOT NULL) impide que los valores nulos se asignen a la tabla PL/SQL de ese tipo. No inicialice la tabla INDEX BY.

Las tablas INDEX-BY pueden tener los siguientes tipos de elementos: BINARY_INTEGER, BOOLEAN, LONG, LONG RAW, NATURAL, NATURALN,PLS_INTEGER, POSITIVE, POSITIVEN, SIGNTYPE y STRING. Inicialmente, las tablas INDEX-BY son escasas. Eso permite, por ejemplo, almacenar datos de referencia en una tabla INDEX-BY utilizando una clave primaria numérica como índice.

Estructura de las Tablas INDEX BY



ORACLE

Estructura de las Tablas INDEX BY

Al igual que en las tablas de base de datos, el tamaño de las tablas INDEX BY no está restringido. Dicho de otro modo, el número de filas de las tablas INDEX BY puede aumentar dinámicamente, de manera que la tabla INDEX BY va creciendo a medida que se van agregando nuevas filas.

Las tablas INDEX BY pueden tener una columna y un identificador único para esa columna y a ninguno de los dos se le puede asignar un nombre. La columna puede pertenecer a cualquier tipo de dato escalar o de registro, pero la clave primaria debe pertenecer al tipo BINARY_INTEGER. Las tablas INDEX BY no se pueden inicializar en su declaración. En el momento de declararlas, las tablas INDEX BY no están llenas. No contienen claves ni valores. Es necesario utilizar una sentencia ejecutable explícita para inicializar (rellenar) una tabla INDEX BY.

Creación de una Tabla INDEX BY

```
DECLARE
  TYPE ename_table_type IS TABLE OF
       employees.last_name%TYPE
       INDEX BY BINARY_INTEGER;
  TYPE hiredate_table_type IS TABLE OF DATE
       INDEX BY BINARY_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;
/
```

ORACLE

Referencia a una Tabla INDEX BY

Sintaxis:

nombre_tabla_INDEX_BY(valor_clave_primaria)

donde: *valor_clave_primaria* pertenece al tipo BINARY_INTEGER.

Haga referencia a la tercera fila de la tabla INDEX BY ENAME_TABLE:

```
ename_table(3) ...
```

El rango de magnitud de un tipo BINARY_INTEGER es -2147483647 ... 2147483647, por lo que el valor de clave primaria puede ser negativo. La indexación no tiene que comenzar necesariamente por el 1.

Note: La sentencia *table*.EXISTS(i) devuelve TRUE si se devuelve una fila con el índice *i*. Utilice la sentencia EXISTS para evitar la emisión de errores en referencia a un elemento de tabla que no existe.

Uso de Métodos de Tabla INDEX BY

Los siguientes métodos facilitan el uso de las tablas INDEX BY:

- **EXISTS**
- **COUNT**
- **FIRST y LAST**
- **PRIOR**
- **NEXT**
- **TRIM**
- **DELETE**

ORACLE

Uso de Métodos de Tabla INDEX BY

Los métodos de tabla INDEX BY son procedimientos o funciones incorporadas que funcionan en las tablas y que se llaman utilizando la notación de puntos.

Sintaxis: *nombre_tabla.nombre_método*[(*parámetros*)]

Método	Descripción
EXISTS(*n*)	Devuelve TRUE si existe el elemento *n* de una tabla PL/SQL
COUNT	Devuelve el número de elementos que contiene actualmente una tabla PL/SQL
FIRST LAST	Devuelve los primeros y los últimos (los menores y los mayores) números de índice de una tabla PL/SQL. Devuelve NULL si la tabla PL/SQL está vacía.
PRIOR(*n*)	Devuelve el número de índice que precede al índice *n* en una tabla PL/SQL.
NEXT(*n*)	Devuelve el número de índice que sigue al índice *n* en una tabla PL/SQL.
TRIM	TRIM elimina un elemento del final de una tabla PL/SQL. TRIM(n) elimina los elementos *n* del final de una tabla PL/SQL.
DELETE	DELETE elimina todos los elementos de una tabla PL/SQL. DELETE(n) elimina el elemento *n* de una tabla PL/SQL. DELETE(m, n) elimina todos los elementos que se encuentran en el rango *m* ... *n* de una tabla PL/SQL.

Tablas de Registros INDEX BY

- **Defina una variable TABLE con un tipo de dato PL/SQL permitido.**
- **Declare una variable PL/SQL para guardar la información del departamento.**

Ejemplo:

```
DECLARE
  TYPE dept_table_type IS TABLE OF
       departments%ROWTYPE
       INDEX BY BINARY_INTEGER;
  dept_table dept_table_type;
  -- Cada elemento de dept_table es un registro
```

ORACLE

Tablas de Registros INDEX BY

En un punto determinado del tiempo, la tabla INDEX BY puede almacenar solamente los detalles de una de las columnas de una tabla de base de datos. Siempre existe la necesidad de almacenar todas las columnas que ha recuperado una consulta. La tabla de registros INDEX BY es la solución a este problema. Dado que sólo se necesita una definición de tabla para guardar la información acerca de todos los campos de una tabla de base de datos, la tabla aumenta enormemente la funcionalidad de las tablas INDEX BY.

Referencias a Tablas de Registros

En el ejemplo que se muestra en esta transparencia, se puede hacer referencia a los campos del registro DEPT_TABLE porque todos los elementos de esta tabla son registros.

Sintaxis:

```
tabla(índice).campo
```

Ejemplo:

```
dept_table(15).location_id := 1700;
```

LOCATION_ID representa un campo de la tabla DEPT_TABLE.

Nota: Se puede utilizar el atributo %ROWTYPE para declarar un registro que represente una fila de una tabla de base de datos. La diferencia entre el atributo %ROWTYPE y el tipo de dato compuesto RECORD es que RECORD le permite especificar los tipos de dato de los campos del registro o declarar campos propios.

Ejemplo de Tablas de Registros INDEX BY

```
SET SERVEROUTPUT ON
DECLARE
   TYPE emp_table_type is table of
      employees%ROWTYPE INDEX BY BINARY_INTEGER;
   my_emp_table  emp_table_type;
   v_count       NUMBER(3):= 104;
BEGIN
  FOR i IN 100..v_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;
```

ORACLE

Ejemplo de Tablas de Registros INDEX BY

En el ejemplo de la transparencia se declara la tabla de registros `INDEX BY` `emp_table_type` para almacenar temporalmente los detalles de los empleados cuyo `EMPLOYEE_ID` se encuentre entre 100 y 104. La información acerca de los empleados de la tabla `EMPLOYEES` se recupera y se almacena en la tabla `INDEX BY` utilizando un bucle. Se utiliza otro bucle para imprimir la información acerca de los apellidos desde la tabla `INDEX BY`. Observe cómo se utilizan los métodos `FIRST` y `LAST` en el ejemplo.

Resumen

En esta lección, ha aprendido a:

- **Definir y hacer referencia a variables PL/SQL de tipos de dato compuestos:**
  - **Registros PL/SQL**
  - **Tablas `INDEX BY`**
  - **Tablas de registros `INDEX BY`**
- **Definir un registro PL/SQL utilizando el atributo `%ROWTYPE`**

ORACLE

Resumen

Un registro PL/SQL es una colección de campos individuales que representa una fila de una tabla. Al utilizar registros, se pueden agrupar los datos en una estructura y luego manipular esta estructura como una sola entidad o una unidad lógica. Esto ayuda a reducir el código y facilita su mantenimiento y comprensión.

Al igual que los registros PL/SQL, la tabla es otro tipo de dato compuesto. Las tablas `INDEX BY` son objetos del tipo `TABLE` y se parecen a las tablas de bases de datos, excepto por una pequeña diferencia. Las tablas `INDEX BY` utilizan una clave primaria para proporcionarle acceso de tipo matriz a las filas. El tamaño de las tablas `INDEX BY` no está restringido. Las tablas `INDEX BY` pueden tener una columna y una clave primaria, y a ninguna de las dos se le puede asignar un nombre. La columna puede pertenecer a cualquier tipo de dato, pero la clave primaria debe ser del tipo `BINARY_INTEGER`.

Las tablas de registros `INDEX BY` mejoran la funcionalidad de las tablas `INDEX BY`, porque sólo es necesaria una definición de tabla para guardar la información acerca de todos los campos.

Los siguientes métodos de recopilación ayudan a mejorar el código, facilitan el uso de las recopilaciones y facilitan el mantenimiento de las aplicaciones:

`EXISTS`, `COUNT`, `LIMIT`, `FIRST` y `LAST`, `PRIOR` y `NEXT`, `TRIM`, y `DELETE`

El atributo `%ROWTYPE` sirve para declarar una variable compuesta cuyo tipo es el mismo que el de una fila de una tabla de base de datos.

Visión General de la Práctica 5

Esta práctica cubre los siguientes temas:

- **Declaración de tablas `INDEX BY`**
- **Procesamiento de datos utilizando tablas `INDEX BY`**
- **Declaración de un registro PL/SQL**
- **Procesamiento de datos utilizando un registro PL/SQL**

ORACLE

Visión General de la Práctica 5

En esta práctica, deberá definir, crear y utilizar tablas `INDEX BY` y un registro PL/SQL.

Práctica 5

1. Escriba un bloque PL/SQL para imprimir la información acerca de un país determinado.
   a. Declare un registro PL/SQL basándose en la estructura de la tabla `COUNTRIES`.
   b. Utilice el comando `DEFINE` para proporcionar el identificador del país. Transfiera el valor al bloque PL/SQL mediante una variable de sustitución de *i*SQL*Plus.
   c. Utilice `DBMS_OUTPUT.PUT_LINE` para imprimir la información seleccionada acerca del país. A continuación se muestra un resultado de ejemplo:

      ```
      Country Id: CA Country Name: Canada Region: 2
      PL/SQL procedure successfully completed.
      ```

   d. Ejecute y pruebe el bloque PL/SQL para los países que tienen los identificadores CA, DE, UK, US.
2. Cree un bloque PL/SQL para recuperar el nombre de todos los departamentos de la tabla `DEPARTMENTS` e imprima el nombre de cada departamento en la pantalla, incorporando una tabla `INDEX BY`. Guarde el código en un archivo con el nombre `p5q2.sql` haciendo clic en el botón `Save Script`. Guarde el archivo de comandos con la extensión `.sql`.
   a. Declare una tabla `INDEX BY`, `MY_DEPT_TABLE`, para almacenar temporalmente el nombre de los departamentos.
   b. Con un bucle, recupere el nombre de todos los departamentos que se encuentran actualmente en la tabla `DEPARTMENTS` y almacénelos en la tabla `INDEX BY`. Utilice la siguiente tabla para asignar el valor de `DEPARTMENT_ID` basándose en el valor del contador que se ha utilizado en el bucle.

COUNTER	DEPARTMENT_ID
1	10
2	20
3	50
4	60
5	80
6	90
7	110

   c. Con otro bucle, recupere los nombres de los departamentos de la tabla `INDEX BY` e imprímalos en pantalla utilizando `DBMS_OUTPUT.PUT_LINE`. En la siguiente página se muestra el resultado del programa.

Práctica 5 (continuación)

```
Administration
Marketing
Shipping
IT
Sales
Executive
Accounting
Accounting
Accounting
Accounting
Accounting
Accounting
Accounting
Accounting
Accounting
Accounting
Accounting
Accounting
Accounting
Accounting
Accounting
Accounting
Accounting
Accounting
Accounting
Accounting
Accounting
PL/SQL procedure successfully completed.
```

Práctica 5 (continuación)

Si le queda tiempo, realice el siguiente ejercicio.

3. Modifique el bloque que creó en la práctica 2 para recuperar toda la información acerca de todos los departamentos de la tabla `DEPARTMENTS` e imprima la información en pantalla, incorporando una tabla de registros `INDEX BY`.

   a. Declare una tabla `INDEX BY`, `MY_DEPT_TABLE`, para almacenar temporalmente el número, el nombre y la ubicación de todos los departamentos.

   b. Con un bucle, recupere la información acerca de todos los departamentos que se encuentran actualmente en la tabla `DEPARTMENTS` y almacénela en la tabla `INDEX BY`. Utilice la siguiente tabla para asignar el valor de `DEPARTMENT_ID` basándose en el valor del contador que se ha utilizado en el bucle. Salga del bucle cuando el contador llegue al valor 7.

COUNTER	DEPARTMENT_ID
1	10
2	20
3	50
4	60
5	80
6	90
7	110

   c. Con otro bucle, recupere la información acerca de los departamentos de la tabla `INDEX BY` e imprímala en pantalla utilizando `DBMS_OUTPUT.PUT_LINE`. A continuación se muestra un resultado de ejemplo.

```
Department Number: 10 Department Name: Administration Manager Id: 200 Location Id: 1700
Department Number: 20 Department Name: Marketing Manager Id: 201 Location Id: 1800
Department Number: 50 Department Name: Shipping Manager Id: 121 Location Id: 1500
Department Number: 60 Department Name: IT Manager Id: 103 Location Id: 1400
Department Number: 80 Department Name: Sales Manager Id: 145 Location Id: 2500
Department Number: 90 Department Name: Executive Manager Id: 100 Location Id: 1700
Department Number: 110 Department Name: Accounting Manager Id: 205 Location Id: 1700
PL/SQL procedure successfully completed.
```