|  |
| --- |
|  |

Programación de Base de Datos (MDY3231)

**Experiencia 1**

**Semana 1: Construyendo mis primeras funciones almacenadas en PL/SQL**

Temáticas

[Introducción a la semana 3](#_Toc126162516)

[Resultado de Aprendizaje (RA) – Indicador de Logro (IL) 3](#_Toc126162517)

[Palabras claves 4](#_Toc126162518)

[Preguntas gatillantes 4](#_Toc126162519)

[Conceptos previos 5](#_Toc126162520)

[Lenguaje de Programación PL/SQL 5](#_Toc126162521)

[Beneficios de PL/SQL 6](#_Toc126162522)

[Conociendo PL/SQL 8](#_Toc126162523)

[Bloque anónimo 8](#_Toc126162524)

[Declaración de variables 9](#_Toc126162525)

[Cuerpo del bloque 10](#_Toc126162526)

[Construcción de una función almacenada 11](#_Toc126162527)

[Función almacenada SIN parámetros 11](#_Toc126162528)

[Función almacenada CON parámetros 26](#_Toc126162529)

[Cierre de la semana 32](#_Toc126162530)

[Referencias 33](#_Toc126162531)

[Bibliografía de la semana 33](#_Toc126162532)

[Apuntes 34](#_Toc126162533)

# **Introducción a la semana**

En esta primera semana deberás hacer una actividad formativa llamada "Construyendo mis primeras funciones almacenadas", que es la entrega de un producto a través de un encargo, el cual consta de construir funciones que den solución a las problemáticas indicadas en ejercicios, utilizando el lenguaje de programación PL/SQL.

# **Resultado de Aprendizaje (RA) – Indicador de Logro (IL)**

|  |  |
| --- | --- |
| **Resultado de Aprendizaje (RA)** | **Indicador de Logro (IL)** |
| **RA1.** Construye funciones y procedimientos almacenados para responder a requerimientos de información y/o lógica de programación, de acuerdo a. reglas de negocio, sintaxis del lenguaje PL/SQL, restricciones del sistema de base de datos y buenas prácticas de la industria. | **IL1.** Construye función almacenada con y sin parámetros utilizando sentencias y operadores SQL y PL/SQL para responder a los requerimientos de información y/o de la lógica de programación de acuerdo a reglas de negocio, sintaxis del lenguaje, restricciones del SGBD y buenas prácticas de la industria. |

# **Palabras claves**

* Argumento o parámetro
* Cursor implícito
* Cursor explícito
* Dato
* Función almacenada
* PL/SQL
* Tipo de dato
* Variable

# **Preguntas gatillantes**

* ¿Qué rol juega una función en el contexto de un lenguaje de programación?
* ¿Puede una función almacenada en la base de datos ser llamada por otras funciones?
* ¿Puede una función almacenada en la base datos llamar a otras funciones y o procedimientos?
* ¿De qué forma se logra recuperar información de la base de datos desde una función almacenada?

# **Conceptos previos**

## Lenguaje de Programación PL/SQL

Las sentencias SQL son nativas e integradas en el lenguaje PL/SQL. Este fue desarrollado por Oracle a finales de los 80 y originalmente tenía capacidades limitadas, sin embargo, eso cambió a principios de la década de 1990.

PL/SQL es un lenguaje de procedimiento diseñado específicamente para abarcar sentencias SQL dentro de su sintaxis. El servidor de base de datos Oracle compila las unidades de programa PL/SQL y las almacena dentro de ella para que, en tiempo de ejecución, tanto PL/SQL como SQL se ejecutan dentro del mismo proceso del servidor, brindando una eficiencia óptima. De esta forma, PL/SQL hereda automáticamente la robustez, la seguridad y la portabilidad de la base de datos Oracle.

Se puede ejecutar PL/SQL directamente desde la línea de comandos de la Interfaz SQL \* Plus, desde herramientas de desarrollos que se conecten a una base de datos Oracle y desde cualquier aplicación construida en otro de lenguaje de programación que se conecte a una base de datos Oracle (ejemplo: Java, HTML, Angular, entre otros).

Además, este lenguaje permite construir las unidades de programación de la Base de Datos Oracle tales como: Procedimientos, Funciones, Triggers, Packages y Scripts.

Figura 1: Unidades de programación en PL/SQL.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Nota: Esquema de las posibles unidades de programación que pueden ser escritas en PL/SQL. Captura de pantalla, (2022). SQL Developer.

Las unidades de programación construidas en la base de datos pueden ser usadas por otras unidades de programación construidas en el mismo lenguaje o desde aplicaciones (construidas en otros lenguajes de programación) que se conecten a una base de datos Oracle.

## Beneficios de PL/SQL

1. **Integración de lenguaje procedimental con SQL**: SQL es un lenguaje no procedimental, es decir, que cuando se ejecuta un comando SQL, este le dice al servidor de base de datos qué hacer. Sin embargo, no se puede especificar cómo hacerlo, por lo que PL/SQL integra elementos propios de los lenguajes de programación con SQL que permiten construir unidades de programación que, en forma individual o integral, permiten el procesamiento de información.
2. **Mejora el rendimiento**: sin PL/SQL no se podría combinar lógicamente sentencias SQL como una unidad. Las sentencias SQL se envían de una en una a la base de datos. Esto se traduce en una llamada a la base de datos para cada sentencia de SQL, lo que aumenta el tráfico de red y la reducción del rendimiento (especialmente en un modelo cliente/servidor). Con PL/SQL se pueden combinar todas estas sentencias SQL en una sola unidad de programación, enviando todo el bloque a la base de datos en lugar de enviar las sentencias SQL de una en una, reduciendo significativamente el número de llamadas a la base de datos.
3. **Desarrollo de Programas Modulares**: la unidad básica en un programa PL/SQL es un bloque, por lo tanto, todos los programas PL/SQL contienen bloques los que pueden estar en secuencia o anidados en otros bloques. El desarrollo de programas modularizados tiene la ventaja de que:
   * 1. Se pueden agrupar lógicamente sentencias relacionadas dentro de bloques.
     2. Se pueden anidar bloques, dentro de bloques grandes, para construir programas poderosos.
     3. Se pueden mantener fácilmente y optimizar el código.
     4. Integración con herramientas de Oracle como Oracle Forms, Reports de Oracle entre otras. Al utilizar estas herramientas, el motor de PL/SQL localmente procesa las sentencias procedimentales; sólo las sentencias SQL se ejecutan en la base de datos.
     5. Portabilidad, un programa PL/SQL se puede ejecutar en cualquier Servidor Oracle independiente del Sistema Operativo y la Plataforma.
     6. Manejo de errores, PL/SQL permite manejar errores o excepciones de manera eficiente. Se pueden definir bloques separados para tratar las excepciones. Las excepciones pueden ser definidas por el usuario o propios de Oracle (predefinidas).

Figura 2: Características PL/SQL

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Nota: Muestra las características de PL/SQL. Captura de pantalla, (2022). SQL Developer.

# **Conociendo PL/SQL**

Es el lenguaje procedimental de Oracle y permite combinar la potencialidad de SQL con las estructuras de control propias de todo lenguaje de programación: secuencia, condición y repetición.

## Bloque anónimo

Lo más básico que vamos a construir es un bloque anónimo que permita: declarar 2 variables e imprimir su contenido en la consola.

#### Ejemplo: Primer bloque anónimo

#### En la Figura 3 - Primer bloque anónimo se aprecia el primer bloque anónimo que trabaja con 2 variables e imprime su contenido en la salida estándar. Vamos a analizar línea por línea el contenido del bloque.

#### Figura 3 - Primer bloque anónimo

#### Texto Descripción generada automáticamente

#### Nota: Sintaxis de un bloque anónimo. Captura de pantalla, (2022). SQL Developer.

#### El bloque comienza en la línea 2 y termina en la línea 10 y está formado por 2 secciones: declaración de variables y cuerpo del bloque.

#### Cuenta con 3 palabras reservadas: DECLARE, BEGIN y END.



En las secciones posteriores vamos a usar un bloque anónimo para poder comprobar la correcta ejecución de una función almacenada. Por ahora, solo iniciaremos con los elementos básicos.

## Declaración de variables

En PL/SQL las variables se deben declarar UNA POR UNA y cada declaración necesita definir: el nombre de la variable y su tipo de dato.

#### Ejemplo: Bloque anónimo - declaración

#### Figura 1 - Unidades de programación en PL/SQL. Interfaz de usuario gráfica, Aplicación Descripción generada automáticamente

#### Nota: Esquema de las posibles unidades de programación que pueden ser escritas en PL/SQL. Captura de pantalla, (2022). SQL Developer.



En la Figura 3 - Primer bloque anónimo se aprecia que la línea 3 declara una variable llamada v\_valor que es de tipo NUMBER y la línea 4 declara una variable llamada v\_cadena que es de tipo cadena de caracteres de longitud 30.

## Cuerpo del bloque

El cuerpo de un bloque anónimo define las instrucciones que se ejecutarán respetando el orden en el que aparecen (secuencia). En el ejemplo que se está analizando el cuerpo del bloque cuenta con 4 instrucciones: 2 de asignación y 2 de llamadas a un procedimiento existente en el lenguaje.

#### Ejemplo: Bloque anónimo - cuerpo

#### Figura 1 - Unidades de programación en PL/SQL. Interfaz de usuario gráfica, Aplicación Descripción generada automáticamente

#### Nota: Esquema de las posibles unidades de programación que pueden ser escritas en PL/SQL. Captura de pantalla, (2022). SQL Developer.

#### PL/SQL. Captura de pantalla, (2022). SQL Developer.



Retomando la Figura 3 – Primer bloque anónimo, la línea 6 permite asignar, usando el operador de asignación :=, un valor a la variable v\_valor que fue previamente declarada. En este caso se asigna un 29, en este punto se debe tener en cuenta que el valor a asignar debe ser del mismo tipo con el cual fue declarada la variable.

De forma análoga, la línea 7 permite asignar un valor a la variable v\_cadena.

Las líneas 8 y 9 realizan una llamada a un procedimiento (unidad de programación) que permite imprimir en la consola del servidor de base de datos. La pregunta es: ¿qué imprimirá?, la respuesta: el contenido de lo que se entregue en el argumento al momento de llamar.

La línea 8 solo entrega el valor de la variable y la línea 9 utiliza el operador de concatenación para unir una cadena de texto con el valor de una variable y será eso lo que entregue al procedimiento (llamado PUT\_LINE) para que pueda imprimir.

# **Construcción de una función almacenada**

## Función almacenada SIN parámetros

Para crear una nueva función se debe usar la sentencia **CREATE FUNCTION** seguido del nombre de la función, una lista de parámetros opcionales, el tipo de dato a retornar y la palabra clave IS o AS; y se debe definir las acciones a ser realizadas (lógica del negocio), a través de un bloque PL/SQL estándar.

#### Ejemplo: Función almacenada SIN parámetros

#### Figura 4 - Función almacenada.

#### Diagrama Descripción generada automáticamente

#### Nota: Sintaxis de una función almacenada. Capturas de pantalla, (2022). SQL Developer.

#### El bloque PL/SQL en la función comienza con BEGIN, opcionalmente precedida por la declaración de variables locales, y termina con END o END seguido del nombre de la función.

#### Se debe declarar una sentencia RETURN para retornar un valor con un tipo de dato que sea consistente con la declaración de la función.

#### En el cuerpo de la Función se debe tener a lo menos una expresión RETURN (puede haber múltiples sentencias RETURN).

#### La opción OR REPLACE indica que, si la función existe, ésta será eliminada y reemplazada con la nueva versión.



En la sintaxis que se presenta en el ejemplo anterior, aparecen los elementos claves para definir una función almacenada:

* ***nombre\_función*:** es el nombre con que se crea la función.
* ***parámetro*:** es el nombre que se le da al parámetro. Se pueden tener más de un parámetro separados por coma.
* ***modo*:** aunque el modo OUT o IN OUT se puede usar no es una buena práctica usarlos en funciones. Si se debe retornar más de un valor se debe considerar retornar el valor en una estructura de dato compuesto.
* ***tipo\_dato*:** especifica el tipo de dato del parámetro sin precisión (sin el largo).
* **RETURN *tipo\_dato*:** especifica el tipo de dato que la función retornará. No se especifica su precisión.
* ***declaración\_variables\_locales*:** variables que serán utilizadas en el bloque PL/SQL.
* **RETURN *expresión*:** especifica el valor que retornará la función (variable, expresión, constante etc) que debe coincidir con el tipo de dato que retorna la función especificada en el encabezado (RETURN tipo\_dato).

#### *Creando una función almacenada*

Existen dos opciones para crear una función Almacenada usando SQL Developer. La primera opción es:

1. Abrir una Hoja de Trabajo de SQL.

Figura 5: Paso 1

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Nota: Abrir una hoja de trabajo. Captura de pantalla, (2022). SQL Developer.

1. Escribir el código. Una vez que se ha escrito todo el código se debe elegir la opción Ejecutar Script.

Figura 6: Paso 2

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Nota: Escribir y ejecutar el código. Captura de pantalla, (2022). SQL Developer.

1. Si la función no tiene errores, quedará almacenada en la base de datos y podrá ser usada por los usuarios.

Figura 7: Paso 3

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Nota: Situación en caso de que el código compile. Captura de pantalla, (2022). SQL Developer.

1. Si la función tiene errores de compilación quedará almacenada en la base de datos, pero no podrá ser ejecutada hasta que se corrijan todos los errores.

Figura 8: Paso 4

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

Descripción generada automáticamente

Nota: Situación en caso de que el código no compile. Captura de pantalla, (2022). SQL Developer.

La segunda opción es:

1. Elegir funciones y con el botón derecho del mouse seleccionar nueva Función.

Figura 9: Paso 1

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamenteInterfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Nota: Creando la nueva función. Captura de pantalla, (2022). SQL Developer.

1. Se abrirá una ventana, en donde se le puede dar el nombre a la función (en este caso FN\_OBT\_ANNOS\_TRAB). Además, se debe indicar el tipo de dato que va a retornar la función. Opcionalmente, además, se pueden definir los parámetros que usará. En este caso, la función requiere de un parámetro para poder obtener los años del empleado un empleado.

Figura 10: Paso 2

Interfaz de usuario gráfica, Aplicación, Word

Descripción generada automáticamente

Nota: Datos de la función. Captura de pantalla, (2022). SQL Developer.

1. Posteriormente, al dar clic en aceptar se abrirá el área de trabajo para escribir el código de la Función para finalmente compilarla para que quede almacenada en la base de datos.

Si la función no tiene errores, quedará almacenado en la base de datos y podrá ser usada por los usuarios. Si tiene errores de compilación, quedará almacenada en la base de datos, pero, como ya se mencionó anteriormente, no podrá ser ejecutada hasta que se corrijan todos los errores.

Figura 11: Paso 3

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Nota: Escribir el código y compilar. Captura de pantalla, (2022). SQL Developer.

#### *Variables y tipos de datos en PL/SQL*

Para explicar el uso de las variables y tipos de datos se va a usar una tabla llamada CARGO cuya estructura y datos se muestran a continuación:

Figura 12: Estructura

Logotipo, nombre de la empresa

Descripción generada automáticamente

Nota: Estructura de la tabla.

Figura : Datos

Tabla

Descripción generada automáticamente

Nota: Datos de la tabla. Captura de pantalla, (2022). SQL Developer.

Vamos a construir dos funciones almacenadas con el fin de explicar el concepto de variables, tipo de datos y cursor implícito. Las funciones serán:

* Función que permita retornar el total de cargos registrados.
* Función que permita, dado el ID de un cargo, retornar el nombre del cargo asociado a ese ID.

#### *Cursor implícito*

Al intentar construir la primera función que retorna la cantidad de cargos registrados es posible notar que lo primero que se necesita es la forma de recuperar el total de cargos registrados.

Figura : Sentencia para calcular el total de registros



Nota: Sentencia para calcular el total de registros. Captura de pantalla, (2022). SQL Developer.

La sentencia para calcular el total de registros se debe incorporar en la función que vamos a construir para poder retornar el valor que esta sentencia entrega. La forma de lograr esto es usando un cursor implícito que permitirá dos acciones:

* La ejecución de la sentencia.
* El almacenamiento del resultado en una variable.

Por lo tanto, en este caso, el cursor estará definido por una sentencia SQL y una variable que almacene el resultado de esa sentencia.

#### Importante

#### Un cursor implícito que usa una sentencia SELECT está formado por la sentencia SELECT y la(s) variables donde se almacenarán las columnas que retorna la sentencia SELECT.



Al igual que en todos los lenguajes de programación la forma de guardar los resultados es a través del uso de variables, que son espacios de memoria que, identificados por un nombre, permiten a la máquina guardar el resultado de las operaciones que va realizando. En este caso, nos interesa guardar el valor que la sentencia está retornando. Esta variable formará parte del cursor implícito junto con la sentencia que lo define.

Ahora vamos a construir la función que, usando este cursor implícito, permita retornar el total de cargos registrados.

#### Ejemplo: Función con curso implícito

#### Figura 15 – Función.

#### Diagrama Descripción generada automáticamente

#### Nota: Función para retornar el total de cargos registrados. Captura de pantalla, (2022). SQL Developer.

#### Acá se muestra la implementación de la función para cumplir con el requerimiento de retornar el total de cargos registrados. Ahora, analicemos ahora en detalle la implementación:

#### Las líneas 4, 7 y 11 son comentarios que, como en todos los lenguajes, ayudan a comprender un poco mejor la implementación. Es considerada una buena práctica documentar nuestros códigos.

#### Las líneas 1 y 2 muestran la firma de la función, esto es su nombre y su tipo de dato de retorno. Acá de forma implícita se muestra que uno de los tipos de datos de PL/SQL es el NUMBER.

#### Las líneas 8 a la 10 definen lo que se conoce como el cursor implícito que en este caso se trata de una sentencia SELECT (más adelante veremos que es posible usar sentencias DML como parte de un cursor implícito).

#### Continúa...



#### Ejemplo: Función con curso implícito

#### Cuando el cursor implícito está definido en base a una sentencia SELECT se debe incluir una cláusula que permita guardar el resultado de la sentencia, esta cláusula es la que aparece en la línea 9 (INTO) y va acompañada de una variable (en este caso llamada v\_total), la cual debe ser declarada. La sección de la función donde se deben declarar las variables es entre la palabra reservada IS y la palabra reservada BEGIN.

#### Dado el punto anterior, se explica el contenido de la línea 5 que muestra la forma de declarar una variable: primero va su nombre y luego su tipo de dato.

#### Al igual que la gran mayoría de los lenguajes las instrucciones en PL/SQL terminan con el símbolo “;”.

#### Finalmente, la línea 12 contiene la instrucción que permite devolver (retornar) el resultado calculado.



#### Enlace

#### Tipos de datos soportados:

#### Para el detalle de los tipos de datos soportados en PL/SQL visitar:

#### <https://docs.oracle.com/es-ww/iaas/data-safe/doc/supported-data-types.html>



Como ya se mencionó con anterioridad, luego de escribir la implementación de la función existen 2 posibles caminos: **la función compila o marca errores**. En caso de que no se hayan registrado errores de sintaxis, entonces la función debería aparecer en el listado de funciones de la base de datos:

Figura 16: Función almacenada

Gráfico, Histograma

Descripción generada automáticamente

Nota: Función almacenada en la base de datos. Captura de pantalla, (2022). SQL Developer.

#### *Probando la función*

Para probar la función es necesario llamarla o invocarla, para lo que se va a construir un bloque anónimo que es una unidad de programación que puede llamar a otras unidades (funciones y/o procedimientos).

#### Ejemplo: Probando la función

#### Figura 17 - Probar la función.

#### Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico Descripción generada automáticamente

#### Nota: Bloque anónimo para probar la función. Capturas de pantalla, (2022). SQL Developer

#### Acá se muestra el bloque que permite comprobar la ejecución de la función almacenada que hemos construido. Analicemos las líneas de código.

#### El bloque está definido entre las líneas 16 y 24. La línea 15 sirve para habilitar la salida por pantalla.

#### Un bloque anónimo está formado por 3 palabras reservadas DECLARE, BEGIN y END.

#### Entre el DECLARE y el BEGIN estarán las variables que el bloque va a utilizar

#### Entre el BEGIN y el END estarán las instrucciones que se ejecutarán; en este caso con las instrucciones necesarias para comprobar el resultado de la ejecución de la función almacenada).

#### Continúa...



#### Ejemplo: Probando la función

#### La línea 21 realiza la llamada a la función; dado que la función retorna un valor es necesario recibir el resultado y almacenarlo en una variable (en este caso total\_cargos); esta variable DEBE ser declarada (línea 18) indicando su nombre y tipo de dato.

#### La línea 23 permite imprimir el contenido de la variable para poder comprobar que el valor retornado es el correcto. Acá se está llamando a un procedimiento (PUT\_LINE) que está disponible en el lenguaje y que permite, como ya se mencionó, interactuar con la salida estándar (impresión en pantalla). Utiliza el operador de concatenación revisado en el módulo anterior de Consultas de base de datos ( || ) para poder imprimir un texto junto al valor de una variable.



Al ejecutar las sentencias que aparecen en el ejemplo se obtiene la salida de lo siguiente:

Figura 18: Ejecución VSC

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Nota: Ejecución salida desde Visual Studio Code. Captura de pantalla, (2022). Visual Studio Code.

Figura 19: Ejecución SQL developer

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Nota: Ejecución salida desde SQL developer. Captura de pantalla, (2022). SQL Developer.

## Función almacenada CON parámetros

Las funciones pueden recibir argumentos o parámetros que son variables que la función va a utilizar para completar el objetivo que debe cumplir. Para explicar este concepto se va a construir una función almacenada que, dado el ID de un cargo, permita retornar el nombre del cargo asociado a ese ID.

#### Ejemplo: Función almacenada CON parámetros

#### Figura 20 - Con parámetros.

#### Interfaz de usuario gráfica, Texto, Aplicación Descripción generada automáticamente

#### Nota: Función almacenada con parámetros. Captura de pantalla, (2022). SQL Developer.

#### La figura muestra la implementación para cumplir con el requerimiento. Analicemos las instrucciones de esta implementación:

#### Las líneas 27 y 28 muestran la firma de la función donde en esta oportunidad se define un parámetro o argumento que representa el ID del cargo del cual se desea conocer su nombre. Un parámetro, como ya se mencionó, es una variable y por ende debe tener asociado un tipo de dato; en este caso se utiliza el atributo %TYPE que permite declarar una variable considerando el tipo de dato de otra variable declarada previamente; acá se está utilizando el tipo de dato del campo car\_id de la tabla cargo.

#### Para usar el atributo %TYPE se debe considerar el operador. que es un operador binario que contiene a su izquierda el nombre de la tabla y a su derecha el nombre del campo de la tabla.

#### Continúa...



#### Ejemplo: Función almacenada CON parámetros

#### La línea 28 (que forma parte de la firma de la función) indica el tipo de dato de retorno de la función, en este caso, una cadena de caracteres.

#### Las líneas 33 a la 35 incluye el cursor implícito que permite rescatar el nombre del cargo cuyo ID ha sido recibido como argumento. Hay que recordar que el cursor implícito está pensado para retornar a los más un registro; en ese sentido, eso acá se cumple dado que estamos buscando por ID que es la clave primaria de la tabla, por lo tanto, los posibles resultados son 0 o 1 registro. Por ahora, solo vamos a considerar que el ID entregado existe en la tabla; en las siguientes semanas aprenderemos cómo manejar la particular situación que implica entregar un ID que no está registrado.



#### *Probando la función*

Para probar la función se construye un bloque anónimo que considera la llamada a la función, la recepción de su respuesta y la impresión de ella.

#### Ejemplo: Probando la función

#### Figura 21 - Probando.

#### 

#### Nota: Bloque anónimo para probar la función. Captura de pantalla, (2022). SQL Developer.

#### Acá se muestra la implementación del bloque anónimo y en ella se aprecian muchas instrucciones que ya fueron explicadas en la sección anterior. Lo nuevo es:

* Línea 42 que declara una variable, esta vez, usando el atributo %TYPE para rescatar el tipo de dato del campo nombre de la tabla cargo que, en definitiva, es el valor que va a almacenar la variable nombre\_cargo que se está declarando.
* La línea 44 declara una variable de tipo NUMBER a la cual se le asigna (usando el operador de asignación :=) el valor de una variable cuyo valor viene desde el usuario (usando el operador &)
* La línea 47 permite la llamada a la función y, en este caso, se debe indicar el valor del argumento que la función requiere (el ID del cargo). El pasaje de parámetros se hace entre un par de paréntesis y en caso de que sea más de uno se separan por una coma ( , ).



Al ejecutar el bloque anónimo (que implica la ejecución de la función) se abre un cuadro de diálogo que solicita el ID el cargo para el cual se desea saber su nombre.

Figura 22: Ejecución del bloque anónimo

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

Descripción generada automáticamente

Nota: Muestra el resultado de la ejecución. Captura de pantalla, (2022). SQL Developer.

Luego, al ingresar un 100 en el cuadro de diálogo y hacer clic en Aceptar se obtiene el siguiente resultado:

Figura 23: Resultado

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Nota: Resultado al ejecutar el bloque anónimo. Captura de pantalla, (2022). SQL Developer.

Finalmente, al revisar en la sección Salida de Script es posible notar que aparece un Antiguo:DECLARE y un Nuevo:DECLARE; esto se debe al uso de la variable de sustitución ID\_CARGO; para evitar esto debería ejecutar la instrucción que aparece a continuación:

Figura 24: Instrucción



Nota: Instrucción para evitar el que aparezca el código del bloque al momento de ejecutar. Captura de pantalla, (2022). SQL Developer

# **Cierre de la semana**

Durante esta semana aprendiste a implementar dos unidades de programación: bloque anónimo y función almacenada. Por ahora, los requerimientos son relativamente sencillos por lo que la modularización de la solución resulta ser simple.

Si bien existe similitud entre un bloque anónimo y una función almacenada -dado que ambas unidades de programación cuentan con una secuencia de instrucciones que se deben incluir dentro de la sección BEGIN y END- la diferencia entre ambas es que la función quedará almacenada en la base de datos (de ahí el nombre de la unidad de programación), mientras que en el bloque es anónimo y no quedará de forma persistente en la base de datos.

Tus primeras funciones almacenadas tienen la característica de ser con y sin parámetros; y utilizan cursores implícitos y cursores explícitos. Además, aprendiste a usar -para construir tus funciones almacenadas- otras funciones creadas por ti, así como también usaste un procedimiento definido dentro del lenguaje PL/SQL.

De igual forma, aprendiste que para recuperar información desde la base de datos se utiliza un cursor, el cual puede ser implícito o explícito. Ambos han sido definidos en esta primera etapa en base a una sentencia SELECT.

También pudiste aprender que los pasos para resolver un problema a través de una función almacenada son:

1. Declarar la firma de la función: esto es definir un nombre, una eventual lista de argumentos y un valor de retorno.
2. Escribir las sentencias que te permiten cumplir con el objetivo planteado; todas estas sentencias van dentro de la sección ubicada entre el BEGIN y el END
3. Probar o comprobar la implementación de la función se debe construir un bloque anónimo que posee una estructura más o menos similar a la que tiene una función almacenada.

Con esto ya estás en condiciones de construir soluciones modulares que permitan dar respuesta a ciertos requerimientos planteados.

# **Referencias**

Gupta, S. (2016). Advanced oracle PL/SQL developer’s guide. Segunda edición. Packt Publishing

Jayapalan, L., Morin, L., Alpern, D., Belden, E., Agrawal, S., Baer, H., … Yang, M. (n.d.). Oracle database database PL/SQL language reference, 18c. Recuperado el 30 de noviembre, 2022 desde: <https://docs.oracle.com/en/database/oracle/oracle-database/18/lnpls/database-pl-sql-language-reference.pdf>

Muñoz Chaparro, A. (2018). Oracle 12c PL/SQL1: Curso práctico de formación. España: RC Libros

Murray, C., Belden, E., Engsig, B., Greenberg, N., Huey, P., Jones, C., … Moore, S. (n.d.). Oracle database 2 day developer’s guide, 21. Recuperado el 30 de noviembre, 2022 desde: <https://docs.oracle.com/en/database/oracle/oracle-database/21/tdddg/2-day-developers-guide.pdf>

Uso de Oracle Data Safe. (s. f.). Recuperado de <https://docs.oracle.com/es-ww/iaas/data-safe/doc/supported-data-types.html>   

# **Bibliografía de la semana**

Oracle PL/SQL Languaje.

**Fuente:** Feuerstein, S., Pribyl, B y Dawes, C (2015). Oracle PL/SQL language: pocket reference. Quinta edición. O’Reilly Media*.* Páginas:1-19.

Conditional and Sequential Control / Sequential Control Statements / Loop / Statement.

**Fuente:** Feuerstein, S., Pribyl, B y Dawes, C (2015). Oracle PL/SQL language: pocket reference. Quinta edición. O’Reilly Media*.* Páginas:24-34.

Cursors in PL/SQL.

**Fuente:** Feuerstein, S., Pribyl, B y Dawes, C (2015). Oracle PL/SQL language: pocket reference. Quinta edición. O’Reilly Media*.* Páginas:40-46.Dibujo de la tierra

Descripción generada automáticamente con confianza baja

# **Apuntes**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

***Facilitador disciplinar:*** Jazna Meza  
***Asesor par:*** Ricardo Pino

Reservados todos los derechos Fundación Instituto Profesional Duoc UC. No se permite copiar, reproducir, reeditar, descargar, publicar, emitir, difundir, de forma total o parcial la presente obra, ni su incorporación a un sistema informático, ni su transmisión en cualquier forma o por cualquier medio (electrónico, mecánico, fotocopia, grabación u otros) sin autorización previa y por escrito de Fundación Instituto Profesional Duoc UC La infracción de dichos derechos puede constituir un delito contra la propiedad intelectual.