UNIVERSIDAD AUTÓNOMA DE QUERÉTARO

Facultad de Informática

Nombre de la Tesina

“Programación PL/SQL Para Oracle 10g y Toad”

Que como parte de los requisitos para obtener el grado de Ingeniero en Computación

Presenta:

Cisneros Frias Azucena
Exp. 150080

Dirigido por:
M en A. Jabel Resendiz González

Querétaro, Qro. Febrero de 2013

ÍNDICE

CAPITULO 1

PROGRAMACIÓN PL/SQL PARA ORACLE 10G Y TOAD

MARCO TEÓRICO

Introducción

SQL es un lenguaje de consulta para los sistemas de bases de datos relaciónales, pero que no posee la potencia de los lenguajes de programación. Cuando se desea realizar una aplicación completa para el manejo de una base de datos relacional, resulta necesario utilizar alguna herramienta que soporte la capacidad de consulta del SQL y la versatilidad de los lenguajes de programación tradicionales. PL/SQL es el lenguaje de programación que proporciona Oracle para extender el SQL estándar con otro tipo de instrucciones.

¿Qué es PL/SQL?

PL/SQL provee una manera muy cómoda de relacionar los conceptos de bases de datos y manejarlos mediante ciertas estructuras de control, dentro del contexto de una herramienta netamente de programación.

Su utilización es dentro del administrador de bases de datos "Oracle" y sus principales características son la posibilidad que brinda de utilizar sentencias SQL para manipular datos en Oracle y sentencias de control de flujo para organizar esta manipulación de datos.

Dentro del lenguaje, es posible declarar constantes y variables, definir procedimientos y funciones y atrapar errores en tiempo de ejecución. Así visto, PL/SQL combina la el poder de la manipulación de datos, con SQL, y las facilidades del procesamiento de los mismos, tal como en los más modernos lenguajes de programación.

PL/SQL es el lenguaje procedimental de ORACLE. Es una extensión de SQL, el cual es, por su parte, un lenguaje estándar basado en la teoría de conjuntos.

El interés del lenguaje PL/SQL está en poder mezclar la potencia de las instrucciones SQL con la flexibilidad de un lenguaje procedimental en un mismo programa.

Estos programas pueden ser ejecutados directamente por las herramientas de ORACLE (bloques anónimos) o a partir de objetos de la base de datos (procedimientos almacenados y paquetes).

Inicialmente, y de un modo muy resumido, podemos decir que PL/SQL se compone de los siguientes elementos:

Bloques: unidades básicas en la programación de PL/SQL.

Unidades léxicas: secuencias de caracteres permitidos en PL/SQL que componen los programas.

Variables: espacios de memoria que pueden contener valores de datos.

Tipos: elementos que pueden usarse en las columnas de la base de datos y que definen la naturaleza de los datos permitidos en la base de datos y en el lenguaje PL/SQL. Los tipos de PL/SQL se definen en un paquete denominado STÁNDARD cuyos contenidos son accesibles desde cualquier bloque PL/SQL.

Expresiones y operadores: elementos que permiten unir las variables PL/SQL. Los operadores definen cómo se asignan los valores a las variables y cómo se manipulan dichos valores. Una expresión es una secuencia de variables y literales separados por operadores. El valor de una expresión se determina a partir de los valores de las variables y literales que la componen, y de la definición de los operadores.

Funciones: además de los tipos, el paquete STANDARD define las funciones predefinidas SQL y de conversión disponibles en PL/SQL.
Registros: los registros de PL/SQL son similares a las estructuras del lenguaje C. Un registro proporciona un mecanismo para tratar con variables diferentes, pero relacionadas, como si fueran una unidad.

Tablas y matrices: las tablas PL/SQL se asemejan a las matrices del lenguaje C. Sintácticamente se las trata de la misma forma que a las matrices, aunque su implementación es distinta. Para poder declarar una tabla en PL/SQL es necesario primero definir su tipo y luego una variable de dicho tipo.

Estructuras de control PL/SQL: permiten controlar el comportamiento del bloque a medida que este se ejecuta e incluyen las órdenes condicionales y los bucles. Las

Cursores: dentro de PL/SQL la orden SELECT no debe devolver más de una fila, pero si es necesario que SELECT devuelva más de una fila, hay que emplear un cursor para extraer individualmente cada fila.

Procedimientos: estructuras de bloques que pueden se almacenados en la base de datos para ser ejecutados cuando sea necesario.

Paquetes: estructuras de bloques PL/SQL que proporcionan un mecanismo para extender en todo momento el propio lenguaje PL/SQL.

Disparadores: estructuras de bloque que se ejecutan de forma implícita cada vez que tiene lugar el suceso de disparo. Los disparadores no admiten argumentos.

Las ventajas del lenguaje PL/SQL son diversas:

- Integración de SQL: se pueden utilizar las instrucciones DML, las de control de transacciones y las funciones SQL prácticamente con la misma sintaxis.

- Procesamiento procedimental: la gestión de variables y las estructuras de control (condiciones y bucles) incrementan las posibilidades de gestión de los datos.

- Funcionalidades suplementarias: la gestión de cursores y el tratamiento de errores ofrecen nuevas posibilidades de procesamiento.

- Mejora del rendimiento: se pueden agrupar varias instrucciones en una misma unidad (bloque) que sólo dará lugar a un "acceso" a la base de datos (en un lugar de un acceso por cada instrucción).

- Integración en los productos de ORACLE: los bloques o procedimientos PL/SQL son compilados y ejecutados por el "motor" de PL/SQL. Este motor está integrado en el motor de la base de datos, así como en determinadas herramientas de Oracle: Oracle*Forms, Oracle*Report, Oracle*Graphics.

Instrucciones SQL integradas en PL/SQL

Estas instrucciones se usan prácticamente con la misma sintaxis que en SQL.

- Para la realización de consultas: SELECT.
- Para la manipulación de datos: INSERT, UPDATE, DELETE.
- Para la gestión de transacciones: COMMIT, ROLLBACK, SAVEPOINT…
- Las funciones TO_CHAR, TO_DATE, UPPER, SUBSTR, ROUND…

Instrucciones específicas de PL/SQL

Las características procedimentales de PL/SQL aportan las siguientes posibilidades:

- Gestión de variables (declaración, asignación de valores y su uso),
- Estructuras de control (secuencias, condiciones, bucles).

Las funcionalidades suplementarias disponibles son las siguientes:

- Gestión de cursores (tratamiento del resultado de una consulta línea por línea),
- Tratamiento de errores (declaración, acción que hay que llevar a cabo).

Bloques PL/SQL

PL/SQL no interpreta un solo comando cada vez, sino un conjunto de comandos contenidos en un “bloque”.

PL/SQL es un lenguaje *estructurado en bloques*, lo que quiere decir que la unidad básica de codificación son bloques lógicos, los que a su vez pueden contener otros sub-bloques dentro de ellos, con las mismas características. Estos bloques puedes estar situados uno detrás de otro (*estructura secuencial*) o pueden estar uno dentro de otro (*estructura anidada*).

Un bloque (o sub-bloque) permite agrupar en forma lógica un grupo de sentencias. De esta manera se pueden efectuar declaraciones de variables que sólo tendrán validez en los bloques donde éstas se definan.

El motor PL/SQL del producto Oracle concreto o de la base de datos compila y ejecuta este bloque.

PL/SQL modulariza código mediante el uso de subprogramas que contienen un cuerpo en la forma estándar de un bloque PL/SQL. Los subprogramas son piezas manejables de código flexible y reusable. Usted puede hacer el código flexible utilizando subprogramas con parámetros. Estos parámetros hacen el mismo código reusable para diferentes valores de entrada. Esto se muestra en la Figura 1.

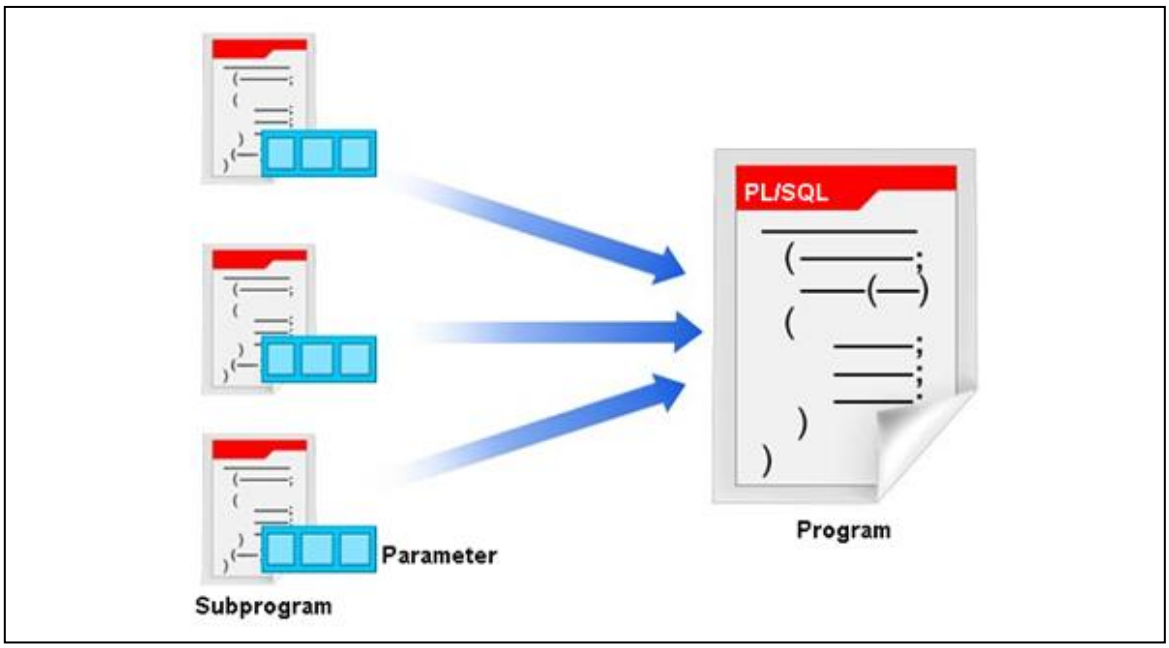

Figura 1. Uso de subprogramas y parámetros

Para modularizar el código existente, se localizan e identifican las secuencias repetitivas en el código y se mueven dentro de un subprograma PL/SQL. A continuación se reemplaza el código original repetitivo con las llamadas al recién creado subprograma PL/SQL como se muestra en la figura 2.

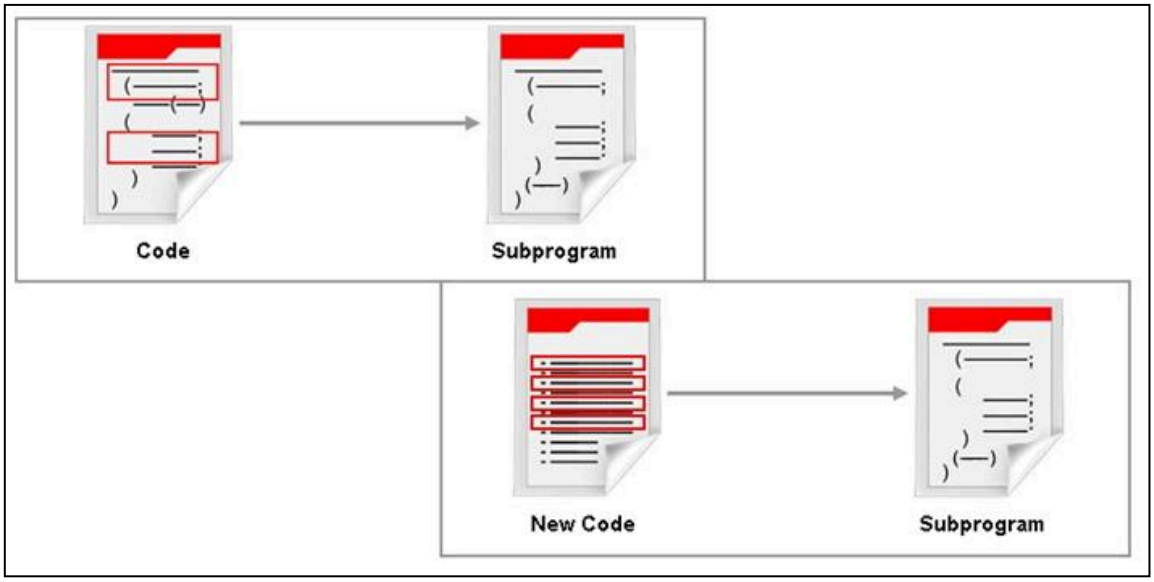

Figura 2. Modularizando el código

Estructura de un Bloque PL/SQL

Un bloque PL/SQL tiene tres secciones diferenciadas: una sección de *declaración*, una sección de *ejecución* y otra de manejo de *excepciones* (cómo se muestra en la Figura 3).

DECLARE

(Declaración de variables, constantes, excepciones y cursores).

BEGIN [nombre del bloque]

(Instrucciones SQL, PL/SQL, estructuras de control)

EXCEPTION

(tratamiento de errores)

END [nombre del bloque**];**

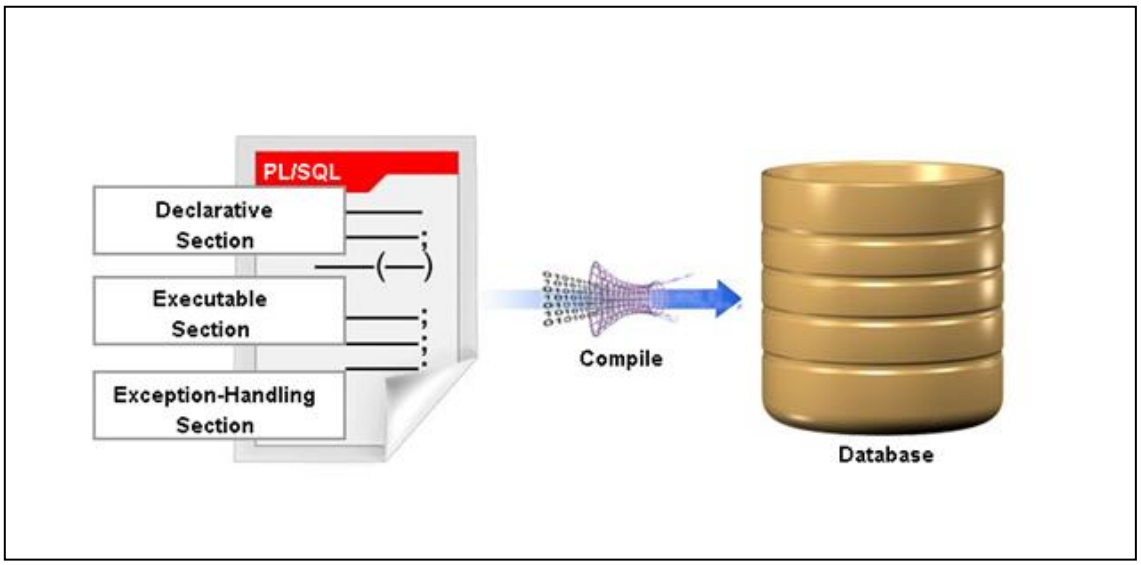

Figura3: Estructura de bloques de un programa PL/SQL

Es posible anidar sub-bloques en la sección ejecutable y de excepciones, pero no en la sección de declaraciones.

La única sección indispensable en un bloque es la sección ejecutable, siendo las otras dos opcionales. Si falta la sección declarativa, el bloque empieza con la palabra clave BEGIN. Si falta la sección de excepciones, no aparecerá la palabra clave EXCEPTION, y el bloque terminará con la palabra clave END, seguida de un punto y coma. Por tanto, un programa mínimo, es decir, un programa PL/SQL con solo la sección ejecutable, tiene la siguiente estructura:

```
BEGIN
/*Sección ejecutable*/
END;
```

Veamos un ejemplo de bloque PL/SQL muy genérico.

```
DECLARE
nombre_variable VARCHAR2(5);
nombre_excepcion EXCEPTION;
BEGIN
SELECT nombre_columna
INTO nombre_variable
FROM nombre_tabla;
EXCEPTION
WHEN nombre_excepcion THEN ...
END;
```

Un bloque de código PL / SQL ayuda a modularizar el código mediante el uso de bloques anónimos, procedimientos, funciones, paquetes y disparadores de base de datos (Figura 4). Usted puede convertir grandes bloques de código en grupos más pequeños de código llamados módulos. Estos módulos pueden ser reutilizados por el mismo programa o pueden ser compartidos con otros programas.

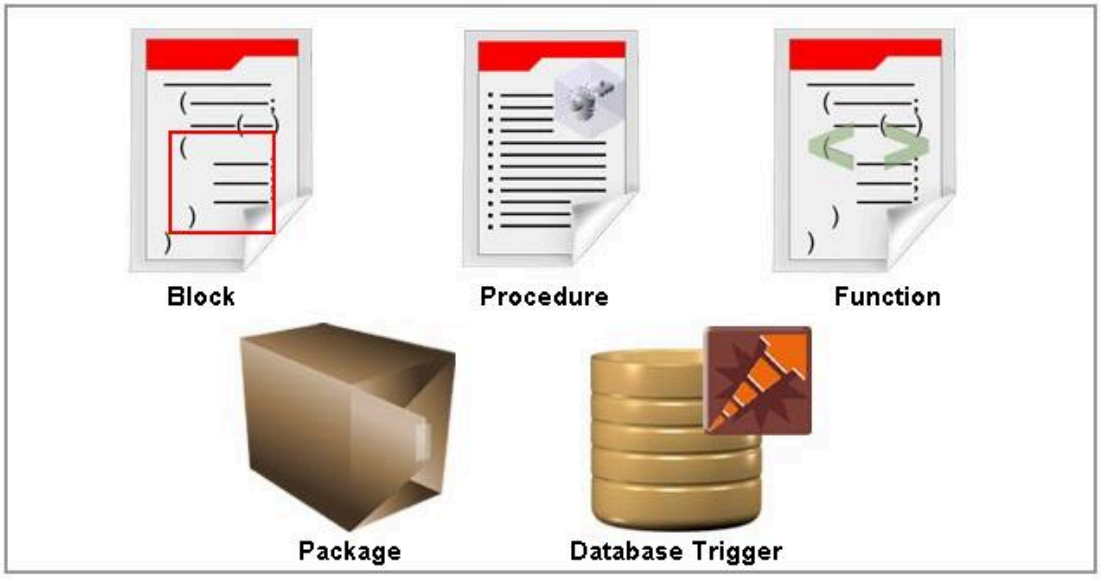

Figura4. Bloques, Procedimientos, Funciones, Paquetes y Disparadores de la Base de Datos

Beneficios de un Bloque PL/SQL dentro de un Subprograma

- **Fácil mantenimiento.** Los subprogramas se encuentran en un lugar único en una Base de Datos. Cualquier modificación requerida en múltiples aplicaciones necesita ser hecha solo en un único lugar. Esto a su vez, reduce la cantidad de pruebas.

- **Mejora de los datos de seguridad y la integridad.** Los subprogramas se ejecutan con los derechos del definidor de forma predeterminada. Un usuario que llama no tiene acceso a los objetos que son accesibles a un subprograma. Esto mejora la seguridad de los datos y los controles de acceso indirecto a objetos de la Base de Datos por los usuarios no privilegiados. Los subprogramas mejoran la integridad de los datos realizando las acciones relacionadas juntas o no realizándolas en absoluto.

- **Mejor rendimiento.** Usando subprogramas, puedes activar un mejor desempeño. Esto es así porque una vez ejecutado, el código analizado PL/SQL de un subprograma puede ser reutilizado y está disponible en el área de servidores de SQL compartido. Debido a que las declaraciones PL/SQL se analizan en tiempo de ejecución. El código puede ser escrito para reducir el número de llamadas de la red a la Base de Datos.

- **La claridad del código.** Usted puede mejorar la claridad del código mediante el uso apropiado de nombres y convenios que describen la acción de las rutinas. Así se reduce la necesidad de comentarios.

Tipos de Bloques

Se pueden distinguir varios tipos de bloques. En primer lugar, se pueden considerar los ***bloques anónimos,*** que son bloques que se ejecutan una sola vez y que por regla general se construyen de manera dinámica.

En segundo lugar, tenemos los ***bloques nominados,*** que no son más que bloques anónimos con una etiqueta que le da un nombre al bloque. Al igual que los bloques anónimos, se ejecuta una sola vez y suelen construirse de manera dinámica.

En tercer lugar tenemos los subprogramas, que son procedimientos**, paquetes y funciones almacenados en la base de datos.** Estos bloques suelen ejecutarse múltiples veces mediante una llamada al procedimiento, paquete o función que los constituyen.

Un último tipo de bloque lo constituyen los ***disparadores o triggers,*** que son bloques nominados que se almacenan en la base de datos y que se ejecutan de manera implícita cada vez que tiene lugar el suceso de disparo. El suceso de disparo es una orden SQL del lenguaje DML de manipulación de datos que se ejecuta sobre una tabla de la base de datos.

Es más fácil de mantener y depurar código compuesto de módulos más pequeños. Estos módulos pueden ser fácilmente personalizados mediante la incorporación de más funciones, sin perjuicio de los módulos restantes del programa.

Los bloques anónimos forman la estructura básica de bloques PL/SQL. Ellos inician el procesamiento PL/SQL de tareas de aplicaciones y pueden ser anidados dentro de la sección ejecutable de cualquier otro bloque PL/SQL.

Bloques Anónimos

Los bloques Anónimos son usados para activar los disparadores de código para los componentes de Oracle Forms y para aislar el manejo de excepciones dentro de un bloque de código. Además, estos se utilizan para iniciar las llamadas a procedimientos, funciones, y construye el paquete. Los bloques Anónimo también se utilizan para anidar otros bloques PL / SQL para manejar el código de control de flujo.

La sintaxis para declarar un bloque anónimo se muestra en la Figura 5. La palabra clave DECLARE es opcional. Se requiere para la declaración de variables, constantes y excepciones utilizados en el bloque PL / SQL. Las palabras clave BEGIN y END son obligatorios y requieren por lo menos un SQL o PL / SQL o ambos tipos de declaraciones entre ellos.

Figura 5. Sintaxis para declaración de un bloque anónimo

La sección de excepciones es opcional (Figura 6). Se utiliza para controlar los errores que están dentro del ámbito de aplicación de un bloque PL / SQL. Las excepciones que no se manejan en el bloque PL / SQL se propagan a la llamada del bloque PL / SQL mediante la asociación y el controlador de excepciones para la excepción específica. Por lo general, como una práctica de programación estándares, se recomienda utilizar una sección de control de excepciones.

Figura 6. Sección de excepciones en un bloque anónimo

Paquetes

Es posible almacenar lógicamente un conjunto de tipos de datos relacionados, variables, cursores e incluso subprogramas dentro de un *paquete*. Cada paquete involucra la definición y tratamiento de todos los elementos recién mencionados.

Un paquete es una estructura PL/SQL que permite almacenar juntos una serie de objetos relacionados. Un paquete tiene dos partes bien diferenciadas, la especificación y el cuerpo del paquete, los cuales se almacenan por separado en el diccionario de datos. Dentro de un paquete se pueden incluir procedimientos, funciones, cursores, tipos y variables. Posteriormente, estos elementos se pueden referenciar desde otros bloques PL/SQL, con lo que los paquetes permiten disponer de variables globales en PL/SQL.

La *especificación* (package specification) es idéntica a una sección de declaración de aplicaciones. En esta especie de encabezado es posible declarar tipos, constantes, variables, excepciones, cursores y subprogramas disponibles para su uso en el *cuerpo* del paquete. De esta manera, el cuerpo (package body) define la implementación de esos subprogramas declarados en el apartado anterior.

Los paquetes pueden ser compilados y almacenados en una base de datos Oracle y su contenido puede ser compartido por varias aplicaciones. Cuando un paquete es llamado para su ejecución, éste se almacena completamente en memoria la primera vez. Las siguientes llamadas no requieren efectuar este procedimiento cada vez y por esto aumentan la eficiencia de los programas.

La especificación del paquete se hace en un fichero y contiene información acerca de sus contenidos. El código de ejecución se almacena en otro fichero aparte con el cuerpo del paquete. Previamente a compilar el cuerpo y ejecutar sus procedimientos y funciones, hay que compilar la cabecera. Sus sintaxis respectivas son:

```
CREATE [OR REPLACE] PACKAGE nombre_paquete IS|AS
especificación de procedimiento o funcion
|declaración de variable
|declaración de tipo
|declaración de excepcion
|declaración de cursor
END nombre_paquete;

CREATE [OR REPLACE] PACKAGE BODY nombre_paquete IS|AS
cuerpos de procedimiento o funcion
END nombre_paquete;
```

Después de compilar la especificación y el cuerpo de un paquete, quedan almacenados en el diccionario de datos, y se pueden invocar en cualquier momento a sus elementos precediéndolos del nombre del paquete y un punto. Por ejemplo:

```
CREATE OR REPLACE PACKAGE paquete_cont IS
g_cont NUMBER := 0; -- se inicializa la variable global
PROCEDURE reset_cont (v_nuevovalor IN NUMBER);
END paquete_cont;

CREATE OR REPLACE PACKAGE BODY paquete_cont IS
PROCEDURE reset_cont (v_nuevovalor IN NUMBER) IS
begin
```

```
g_cont := v_nuevovalor;
end reset_cont;
END paquete_cont;
```

Ahora para actualizar el valor de la variable global desde cualquier bloque PL/SQL tenemos dos opciones: acceder directamente a la variable global definida en el paquete o ejecutar el procedimiento del paquete que también la actualiza.

```
EXECUTE paquete_cont.g_cont:=5;
EXECUTE dbms_output.put_line(paquete_cont.g_cont);
EXECUTE paquete_cont.reset_cont(5);

EXECUTE dbms_output.put_line(paquete_cont.g_cont);
```

Procedimientos y Funciones

Los bloques PL/SQL se pueden almacenar como objetos de la base de datos para ser ejecutados repetidamente. Los parámetros de entrada y salida permiten comunicarse con los procedimientos o funciones.

Creación de procedimientos almacenados

Para crear un procedimiento almacenado se usa la sintaxis general siguiente:

```
CREATE [OR REPLACE] PROCEDURE nombre_procedimiento

[(argumento [{IN|OUT|IN OUT}] tipo,
…
(argumento [{IN|OUT|IN OUT}] tipo)] {IS|AS}
Cuerpo_procedimiento
```

El nombre del procedimiento es *nombre_procedimiento, argumento* es el nombre de un parámetro del procedimiento, *tipo* es el tipo de parámetro asociado y *cuerpo_procedimiento* es el bloque PL/SQL que contiene el código del procedimiento. La cláusula OR REPLACE da un mensaje de aviso si se va a crear un procedimiento que ya existe. Para cambiar el

código de un procedimiento es necesario eliminarlo y volver a crearlo. Los procedimientos pueden tener *parámetros reales* (contienen los valores que se pasan al procedimiento cuando éste es invocado y reciben los resultados del procedimiento cuando éste termina) y *parámetros formales* (meros contenedores para los valores de los parámetros reales).

Cuando se llama al procedimiento (con CALL) se asignan el valor de los parámetros reales a los parámetros formales, dentro del procedimiento se hace referencia a dichos valores mediante los parámetros formales y cuando el procedimiento termina se asigna el valor de los parámetros formales a los parámetros reales.

Los parámetros formales pueden tener los modos IN (por defecto), OUT o IN OUT. El modo IN indica que el parámetro real se pasa al procedimiento cuando éste es invocado, dentro del procedimiento el parámetro formal se considera como de sólo lectura (no puede ser cambiado) y cuando termina el procedimiento el parámetro real no sufre cambios. El modo OUT indica que se ignora cualquier valor que tenga el parámetro real cuando se llama al procedimiento, dentro del procedimiento al parámetro real se considera como de sólo escritura (no puede ser leído peri si es posible asignarle valores) y cuando finaliza el procedimiento los contenidos del parámetro formal se pasan al parámetro real. El modo IN OUT indica que el parámetro real se pasa al procedimiento cuando éste es invocado, dentro del procedimiento el parámetro formal puede ser de lectura y escritura, y cuando termina el procedimiento los contenidos del parámetro formal se asignan al parámetro real.

El cuerpo de un procedimiento es un bloque PL/SQL con sus secciones declarativa, ejecutable y de manejo de excepciones. La sección declarativa se sitúa entre las palabras claves IS o AS y la palabra clave BEGIN, la ejecutable entre BEGIN y EXCEPTION y la de excepciones entre EXCEPTION y END (no existe la palabra clave DECLARE en un procedimiento y su lugar lo ocupa IS o AS). A veces, se suele incluir el nombre del propio procedimiento después de la orden END que cierra la declaración.

Los parámetros formales de un procedimiento pueden tener valores predeterminados que se declaran mediante la siguiente sintaxis:

```
Nombre_parámetro [modo] tipo_parámetro {:=|DEFAULT}valor_inicial
```

Creación de Funciones

El concepto de función es muy similar al concepto de procedimiento. Ambos aceptan argumentos, tanto en notación posicional como en notación nominal. Ambos son bloques PL/SQL con secciones declarativa, ejecutable y de excepciones. La diferencia estriba en que una llamada a un procedimiento es una orden PL/SQL en sí misma, mientras que una llamada a una función se realiza como parte de una expresión.

Para crear una función se utiliza la sintaxis general siguiente:

```
CREATE [OR REPLACE] FUNCTION nombre_functión
[(argument [{IN|OUT|IN OUT}] tipo,
…
(argument [{IN|OUT|IN OUT}] tipo)]
RETURN tipo_retorno {IS|AS}
Cuerpo_función
```

El nombre de la función es *nombre_función, argumento y tipo* juega el mismo papel que en un procedimiento, *tipo_retorno* es el tipo de valor que devuelve la función y *cuerpo_función* es un bloque PL/SQL que contiene el código de la función. La orden RETURN se emplea para devolver el control y un valor al entorno que realizó la llamada.

Eliminación de procedimientos y funciones

La sintaxis para borrar un procedimiento es la siguiente:

```
DROP PROCEDURE nombre_procedimiento
```

La sintaxis para borrar una función es la siguiente:

```
DROP FUNCTION nombre_procedimiento
```

Triggers (Disparadores de Base de Datos)

Un disparador es un bloque nominado que se asemeja a los procedimientos y a las funciones y tiene sección declarativa, ejecutable y de manejo de excepciones. Pero los

disparadores no admiten argumentos y se ejecutan de forma implícita cada vez que ocurre el suceso de disparo, que suele ser una operación DML (INSERT, UPDATE o DELETE) sobre una tabla de la base de datos. Los disparadores suelen utilizarse para restricciones de integridad complejas, auditoría de la información de una tabla o aviso a otros programas para ejecutar una acción.

La sintaxis general para crear un disparador es la siguiente:

```
CREATE [OR REPLACE] TRIGGER nombre_disparador
{BEFORE | AFTER} suceso_disparo ON referncia_tabla
[FOR EACH ROW [WHEN condición_disparo]]
Cuerpo_disparo
```

El nombre del disparador es *nombre_disparador, suceso_disparo* y especifica cuándo se activa el disparador, *referencia_tabla* es la tabla para la que se define el disparador y *cuerpo_disparador* es el código principal del disparador. El cuerpo del disparador se ejecuta cuando la *condición_disparo* incluida en la cláusula WHEN (si ésta existe) es verdadera.

Los componentes de un disparador son nombre, el suceso de disparo y su cuerpo, siendo opcional la cláusula WHEN. Nada impide que el nombre de un disparador sea el mismo que el de tablas y procedimiento, pero no es recomendable. Lo que sí es obligatorio es que dentro de un esquema todos los disparadores tengan nombre distinto.

El suceso de disparo determina el tipo de disparador. Los disparadores se definen para las operaciones INSERT, UPDATE, o DELETE y pueden dispararse antes o después de la operación. Los tipos de disparadores se resumen en la siguiente tabla:

Categoría	***Valores***	***Explicación***
Orden	INSERT, DELETE,UPDATE	Tipo de orden DML que lo activa
Temporización	BEFORE o AFTER	Se activa antes o después de la ejecución
Nivel	FILA u ORDEN	Los disparadores de fila se activan una vez por cada fila afectada por la orden de disparo, y los de orden se activan una vez, antes o después de la orden.

También existen disparadores de sustitución, que sólo puedes definirse sobre vistas que tienen que tener nivel de fila y que suelen actualizar vistas con uniones.

Para declarar el disparador, después de su nombre, hay que indicar si se tiene que ejecutar antes, después o en lugar de la ejecución del suceso (i.e. la sentencia SQL (`delete`, `insert`, `update`)) que ha causado su disparo. Un mismo disparador puede tener varios sucesos asociados, y una instrucción SQL sólo se corresponde con un suceso, aunque afecte a varias tuplas. Para diferenciar dentro del bloque PL/SQL cuál de los posibles sucesos es el que ha activado al disparador, se pueden utilizar los predicados condicionales `INSERTING`, `UPDATING` y `DELETING`.

```
IF INSERTING THEN
v_valor := 'I';
ELSIF UPDATING THEN
v_valor := 'U';
ELSE
v_valor := 'D';
END IF;
```

A continuación va la palabra `ON` y el nombre de la tabla o vista a la que se asocia el disparador. El cuerpo del disparador consiste en un bloque PL/SQL. Por defecto todos los disparadores son de tipo `for each statement`, de manera que se ejecutan una sola vez, antes o después de la instrucción que lanza su ejecución. Existen dos variables externas `old` y `new` que almacenan respectivamente los valores de cada tupla antes y después de ejecutar sobre ella la instrucción que lanza al disparador. Sin embargo, estas variables solamente se pueden utilizar cuando se escoge la opción de ejecutar el disparador una vez por cada tupla (`for each row`), es decir ejecutarlo separadamente para cada una de las tuplas afectadas por la instrucción SQL que ha activado al disparador. En este caso la cláusula optativa `WHEN` sirve para especificar una condición adicional que debe cumplir una tupla afectada por el disparador para que éste sea ejecutado sobre ella.

```
CREATE OR REPLACE TRIGGER Sal_Total
AFTER INSERT OR UPDATE OF salario ON empleado
FOR EACH ROW WHEN (new.dep_num IS NOT NULL)
```

```
/*El disparador solamente se va a ejecutar para aquellas
tuplas insertadas o modificadas que tengan un dep_num no
nulo*/
BEGIN
UPDATE departamento
SET sal_total= sal_total + :new.salario
WHERE dep_num = :new.dep_num;
END;
```

Nótese que las variables externas `old` y `new` deben ser precedidas por dos puntos para que puedan ser accedidas desde dentro del bloque PL/SQL. En el caso de que el disparador se tenga que ejecutar una sola vez para la instrucción que causa su disparo (`for each statement`), entonces la cláusula `WHEN` y las variables `old` y `new` no se pueden utilizar, ya que la instrucción puede afectar a varias tuplas.

```
CREATE OR REPLACE TRIGGER hacer_pedido
/*Disparador para vigilar que cuando la cantidad de piezas
caiga por
debajo del mínimo se haga un pedido de más piezas, si no se
ha hecho ya*/
AFTER UPDATE OF cantidad ON piezas
FOR EACH ROW WHEN (new.cantidad < new.cant_min)
DECLARE
v_pendientes NUMBER;
BEGIN
SELECT count(*) INTO v_pendientes FROM pedidos
WHERE numpie = :new.numpie;
IF v_pendientes = 0 THEN
INSERT INTO pedidos VALUES (:new.numpie, :new.cant_pedido,
sysdate);
END IF;
END;
```

Hay que tener en cuenta que desde el cuerpo de los disparadores de nivel de fila (`for each row`) no es posible ejecutar ordenes SQL de lectura o actualización sobre la tabla asociada al disparador, ni sobre las claves de las tablas referenciadas desde la tabla asociada al disparador por medio de una clave ajena. Oracle bloquea el acceso a estos datos durante la ejecución del disparador para poder asegurar la integridad de la tabla (error de las tablas mutantes).

Los disparadores de tipo `INSTEAD OF` solamente pueden definirse sobre vistas (no sobre tablas) y se ejecutan en lugar de la instrucción que los lance. Estos disparadores son siempre de tipo `for each row`, por lo que se pueden utilizar junto con la cláusula `when` y las variables `old` y `new`. Este tipo de disparadores se aplica para ejecutar operaciones sobre las tablas subyacentes a una vista. Por ejemplo, cuando se define una vista con varias tablas, Oracle no permite que la vista se modifique directamente, pero se puede implementar un disparador que se dispare en lugar de la operación de actualización de la vista, ejecutando las modificaciones adecuadas sobre las tablas que alimentan la vista.

Para activar un disparador se tiene que compilar con éxito (instrucción `start`), y se puede desactivar con la instrucción siguiente:

```
ALTER TRIGGER nombre_disparador [DISABLE|ENABLE]
```

Para borrar un disparador se ejecuta la acción:

```
DROP TRIGGER nombre_disparador
```

Para consultar la información de los disparadores se puede ejecutar la siguiente consulta sobre el diccionario de datos:

```
SELECT trigger_type, table_name, triggering_event
FROM user_triggers
WHERE trigger_name = '...';
```

Cuando se utilicen bases de datos con varios disparadores activados hay que tener en cuenta que éstos pueden interaccionar entre sí. Esto significa que una acción del usuario puede lanzar un disparador que a su vez ejecute acciones que lancen otros disparadores, y así sucesivamente. También es importante no definir conjuntos de disparadores que puedan lanzarse mutuamente en un bucle infinito, ni que se contradigan entre sí.

Los disparadores no pueden contener órdenes SQL de control de transacciones (COMMIT, ROLLBACK o SAVEPOINT) y ningún procedimiento o función llamado por el disparador puede emitir órdenes de control de transacciones.

Variables

Las variables son zonas de memoria nominadas que permiten almacenar un valor.

En PL/SQL, permiten almacenar valores procedentes de la base de datos o de cálculos, que se emplearán para comprobar condiciones, realizar cálculos o asignar valores a otras variables o datos de la base.

Las variables se caracterizan por:

- Su nombre, compuesto por letras, números y los caracteres $, _, o #. El nombre puede tener hasta un máximo de 30 caracteres y no debe ser una palabra reservada.

PL/SQL permite declarar constantes y variables para ser utilizadas en cualquier expresión dentro de un programa. La única condición exigida por PL/SQL es que cada variable (o constante) debe estar declarada antes de ser utilizada en una expresión.

Las variables pueden corresponder a cualquier tipo de dato de SQL, tal como *char*, *date* o *number*, o algún tipo de PL/SQL, como *boolean* o *binary_integer*.

Por ejemplo, si desea declarar una variable llamada "part_no" que almacene cuatro dígitos numéricos y otra variable "in_stock" de tipo booleano, es decir, que almacene solamente los valores True o False, la declaración se vería como sigue:

part_no number(4) ;

in_stock boolean ;

Cómo asignar valores a variables

Es posible asignar valores a las variables de dos formas. La primera utiliza el operador "**:=**". La variable se ubica al lado izquierdo y la expresión al lado derecho del símbolo.

Por ejemplo:

tax := price * tax_rate ;

bonus := current_salary * 0.10 ;

amount := TO_NUMBER(SUBSTR('750 dólares', 1, 3)) ;

valid := False ;

La segunda forma de asignar valores a variables es obtener valores directamente desde la base de datos, como en:

SELECT sal * 0.10 INTO bonus FROM emp WHERE empno = emp_id ;

Declaración de Constantes

En la declaración de una constante (muy similar a la de una variable), se debe incorporar la palabra reservada "constant" e inmediatamente asignar el valor deseado. En adelante, no se permitirán reasignaciones de valores para aquella constante que ya ha sido definida.

Ejemplo: *credit_limit CONSTANT real := 5000.00 ;*

Constantes y Variables

nombre_variable [CONSTANT] tipo [NOT NULL] [:= valor_inicial];

- Donde tipo es: tipo_escalar | identif%TYPE | identificador%ROWTYPE
- Donde tipo_escalar: NUMBER | DATE | CHAR | VARCHAR | BOOLEAN
- La cláusula CONSTANT indica la definición de una constante cuyo valor no puede ser modificado. Se debe incluir la inicialización de la constante en su declaración.
- La cláusula NOT NULL impide que a una variable se le asigne el valor nulo, y por tanto debe inicializarse a un valor diferente de NULL.
- Las variables que no son inicializadas toman el valor inicial NULL.
- La inicialización puede incluir cualquier expresión legal de PL/SQL, que lógicamente debe corresponder con el tipo del identificador definido.
- Los tipos escalares incluyen los definidos en SQL más los tipos VARCHAR y BOOLEAN. Este último puede tomar los valores TRUE, FALSE y NULL, y se suele utilizar para almacenar el resultado de alguna operación lógica. Por su parte,

VARCHAR es un sinónimo de CHAR, siendo más conveniente la utilización del tipo CHAR.

- También es posible definir el tipo de una variable o constante, dependiendo del tipo de otro identificador, mediante la utilización de las cláusulas %TYPE y %ROWTYPE. Mediante la primera opción se define una variable o constante escalar, y con la segunda se define una variable fila, donde identificador puede ser otra variable fila o una tabla.

Ejemplos:

```
DECLARE
v_location VARCHAR2(15) := 'Granada';
c_comm CONSTANT NUMBER(3) := 160;

v_nombre tabla_empleados.nombre%TYPE;
```

Tipos de datos estándar

Los diferentes tipos de datos estándar en Oracle SQL son los siguientes:

- NUMBER(p [,e]), números donde p es la precisión, e es opcional y representa la escala para decimales.
- VARCHAR2(limite), cadena de caracteres de longitud variable, limite representa el número máximo de caracteres.
- DATE, fecha, puede contener día, mes, año, hora, minutos y segundos

Tipos de datos de una tabla

También se pueden realizar la declaración de variables usando como origen de la declaración una tabla de la base de datos, por ejemplo, si tenemos una tabla en nuestra base de datos llamada usuarios y con la columna nombre VARCHAR2(100), podemos realizar la declaración de una variable del mismo tipo que nombre de esta manera: Vlnombre usuarios.nombre%TYPE. De esta manera garantizamos la coherencia entre los datos de la tabla y el código PL/SQL.

Podemos declarar también la estructura completa de una tabla con %ROWTYPE. La estructura de una tabla es el conjunto de columnas y sus correspondientes tipos, si queremos realizar una declaración para todas las columnas de la Personal lo haremos con la declaración Vlpersonal usuarios%ROWTYPE.

Cursores

El resultado de una consulta puede almacenarse en variables, en las que se almacenan cada una de las tuplas del resultado, o bien en una variable de tupla que sea compatible con el resultado de la consulta. Si aparece más de una fila como resultado de una consulta, resulta conveniente la utilización de cursores que permiten recorrer todas sus filas.

```
CURSOR nombre_cursor [parámetros] IS consulta_SQL;
```

Los cursores son áreas de trabajo que permiten ejecutar sentencias SQL y procesar la información obtenida de ellos.

Hay dos tipos de cursores: implícitos y explícitos. PL/SQL declara implícitamente un cursor para todas las sentencias de manipulación de datos, incluyendo las consultas que retornan sólo una fila. Para consultas que devuelven más de una fila, es posible declarar explícitamente un cursor que procese las filas en forma individual.

Cursores explícitos

Un cursor explícito se declara mediante la sintaxis siguiente:

```
CURSOR nombre_cursor IS orden_SELECT
```

El cursor explícito se abre mediante la siguiente sintaxis:

```
OPEN nombre_cursor
```

La extracción de los datos y su recogida en variables PL/SQL se realiza mediante la siguiente sintaxis:

```
FETCH nombre_cursor INTO lista_variables;
FETCH nombre_cursor INTO registro_PL/SQL;
```

La sintaxis *nombre_cursor* identifica un cursor previamente declarado y abierto, *lista_variables* es una lista de variables PL/SQL previamente declaradas y separadas por comas y *registro_PL/SQL* es un registro PL/SQL previamente declarado.

El cierre de un cursor explícito se realiza mediante la sintaxis:

```
CLOSE nombre_cursor
```

Entre los atributos de los cursores explícitos tenemos %FOUND y %NOT FOUND para controlar si la última orden FETCH devolvió o no una fila, %ISOPEN para ver si el cursor está o no abierto y %ROWCOUNT que devuelve el número de filas extraídas por el cursor hasta ahora.

También existe un tipo de cursor que admite parámetros como argumentos al estilo de los procedimientos de los lenguajes de programación procedimentales. Este tipo de cursores se denomina cursores parametrizados.

Cursores implícitos

Los cursores implícitos se utilizan para procesar las órdenes INSERT, UPDATE, DELETE, y SELECT…INTO de una sola fila. Este tipo de cursores admiten los mismos atributos que los cursores explícitos.

Cuando PL/SQL procesa cualquier orden SQL, abre el cursor SQL relativo a esa orden de modo implícito y lo cierra automáticamente después de procesar la orden. Toda orden SQL tiene asociado de forma automática un cursor PL/SQL implícito.

Ejemplos:

```
DECLARE
CURSOR curs_01 IS
SELECT empno, ename, job FROM emp WHERE deptno=20;
```

```
DECLARE
CURSOR emp_cursor IS SELECT empno,ename FROM empleados;
```

Los parámetros de un cursor se pueden utilizar para definir variables con valores de entrada que determinen el resultado de cada ejecución de la consulta SQL asociada.

El conjunto de filas retornado se denomina "set de resultados". Su tamaño está determinado por el número de filas que calzan con el criterio de selección de la query que implementa el cursor. Las filas son procesadas de a una cada vez. En la Figura 7 se muestra un ejemplo de recuperación de filas por medio de un cursor

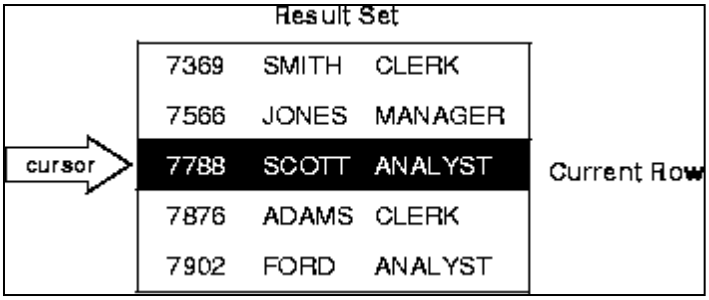

Figura 7: Recuperación de filas a través de un cursor

Estructuras de Control en PL/SQL

Tipos de Estructuras de control en PL/SQL

Las estructuras de control permiten controlar el comportamiento del bloque a medida que éste se ejecuta. Las más importantes son las estructuras condicionales y los bucles.

Estructuras condicionales

Las estructuras condicionales se utilizan para la realización de acciones dependiendo del cumplimiento o no de determinadas condiciones. Las estructuras condicionales más comunes son IF-THEN, IF-THEN-ELSE, IF-THEN-ELSIF y CASE.

IF-THEN

Se trata de una estructura condicional en la que se ejecuta una secuencia de instrucciones si la condición es cierta. Su sintaxis es la siguiente:

```
IF condición THEN
     Secuencia_de_instrucciones
END IF;
```

También puede escribirse toda la estructura condicional sobre la misma línea tal y como se observa a continuación:

```
IF x > y THEN high :=x; END IF;
     Secuencia_de_instrucciones
END IF;
```

IF-THEN-ELSE

Se trata de una estructura condicional en la que se ejecuta una primera secuencia de instrucciones si la condición es cierta, y se ejecuta una segunda secuencia de instrucciones si la condición es falsa. Su sintaxis es la siguiente:

```
IF condición THEN
     Secuencia_de_instrucciones1
ELSE
     Secuencia_de_instrucciones2
END IF;
```

IF-THEN-ELSIF

Estructura condicional que se utiliza para seleccionar una acción entre varias alternativas mutuamente excluyentes. Si la primera condición es falsa o nula, se pasa a testear la condición de la cláusula ELSIF comenzando una nueva estructura condicional IF-THEN-ELSE. Su sintaxis es la siguiente:

```
IF condición THEN
     Secuencia_de_instrucciones1
ELSIF condición2 THEN
     Secuencia_de_instrucciones2
ELSE
     Secuencia_de_instrucciones3
END IF;
```

CASE

Al igual que IF, la sentencia CASE selecciona una secuencia de sentencias a ejecutar, sin embargo, para realizar la selección utiliza un selector y no una expresión booleana. La sentencia CASE es más legible y más eficiente que la sentencia IF. La sintaxis de CASE podría expresarse como sigue:

```
 [<<nombre etiqueta>>]
CASE selector
     WHEN expresión1 THEN sentencia_de_instrucciones1;
     WHEN expresión1 THEN sentencia_de_instrucciones2;
     …
     WHEN expresiónN THEN sentencia_de_instruccionesN;
     [ELSE secuencia de instruccionesN+1;]
END CASE [nombre_etiqueta];
```

BUCLES

PL/SQL utiliza los bucles para ejecutar órdenes de forma repetida. Existen distintos tipos de bucles que se analizarán en párrafos posteriores.

Bucles Simples

Los bucles simples tienen la siguiente sintaxis:

```
LOOP
     secuencia_de_instrucciones;
     EXIT [WHEN condición];
END LOOP;
```

La sintaxis EXIT [WHEN condición] es equivalente a:

```
IF condición THEN
     EXIT;
END IF;
```

Por lo tanto, la sintaxis completa del bucle simple es la siguiente:

```
LOOP
     secuencia_de_instrucciones;
     IF condición THEN
     EXIT;
END IF;
END LOOP;
```

Bucles WHILE

La sintaxis de un bucle WHILE es la siguiente:

```
WHILE condición LOOP
     secuencia_de_instrucciones;
END LOOP;
```

Se produce una evaluación de la condición previa a cada iteración del bucle. Si la condición es verdadera, se ejecuta la secuencia de órdenes, pero si es falsa, el bucle termina y se transfiere el control a las instrucciones posteriores a END LOOP.

También se pueden utilizar las órdenes EXIT o EXIT WHEN dentro de un bucle WHILE para salir del bucle de forma prematura. La sintaxis será:

```
LOOP
     secuencia_de_instrucciones;
     EXIT WHEN expresión booleana;
END LOOP;
```

Bucles FOR numéricos

Los bucles FOR numéricos disponen de un número de iteraciones definido. Ya hemos visto que en los bucles simples y WHILE el número de iteraciones depende de la condición del bucle y no se conoce de antemano. La sintaxis es la siguiente:

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

El contador del bucle se declara de modo explícito, el límite inferior y el límite superior acotan el número de iteraciones (sólo se declaran una vez) y la secuencia de órdenes es el contenido del bucle.

Órdenes GOTO y etiquetas

PL/SQL dispone de la orden de salto GO TO. Para identificar el lugar al que se tiene que realizar el salto, se utiliza una etiqueta que suele encerrarse entre corchetes angulares dobles. La sintaxis es la siguiente:

```
GOTO etiqueta;
```

En el ejemplo siguiente la orden GOTO pasa el control a una inserción de fila en la tabla TEMP.

```
…
GOTO insertar_fila;
…
<<insertar_fila>>
INSERT INTO emp VALUES…
END;
```

Orden NULL

La orden NULL se utiliza dentro de los programas PL/SQL para indicar que no se realice ninguna acción.

REGISTROS EN PL/SQL

El paquete estándar PL/SQL tiene predefinidos los tipos escalares, sin embrago, los tipos compuestos deben ser definidos por el usuario antes de ser asignados a alguna variable. Los registros, junto con las tablas, forman los tipos compuestos de PL/SQL. Los registros proporcionan un mecanismo para tratar con variables distintas que estén relacionadas entre sí, para tratarlas como una unidad. Un registro es una estructura de datos en PL/SQL, almacenados en campos, cada uno de los cuales tiene su propio nombre y tipo y que se tratan como una sola unidad lógica. Los registros de PL/SQL son similares a las estructuras del leguaje C.

La sintaxis general para definir un tipo de registro es la siguiente:

```
TYPE tipo_registro IS RECORD(
     Campo1 tipo1 [NOT NULL] [:=expr1],
     Campo2 tipo2 [NOT NULL] [:=expr2],
     Campon tipon [NOT NULL] [:=exprn]);
```

El nombre del Nuevo registro es *tipo_registro* y los nombres de los campos que lo componen son *campo1, campo2, ..., campon.* Los posibles valores iniciales de los campos son *expr1, expr2, ..., exprn.* Cada declaración de campo se asemeja a una declaración de variable.

Utilización de %TYPE y %ROWTYPE

Uso de %TYPE

El atributo %TYPE define el tipo de una variable utilizando una definición previa de otra variable o columna de la base de datos.

Ejemplo:

```
DECLARE
credito REAL(7,2);
debito credito%TYPE;
…
```

También se podría declarar una variable siguiendo el tipo de un campo de alguna tabla, como por ejemplo en:

```
debito cuenta.debe%TYPE;
```

La ventaja de esta última forma es que no es necesario conocer el tipo de dato del campo "debe" de la tabla "emp", manteniendo la independencia necesaria para proveer más flexibilidad y rapidez en la construcción de los programas.

Uso de %ROWTYPE

El atributo %ROWTYPE precisa el tipo de un registro (*record*) utilizando una definición previa de una tabla o vista de la base de datos, se utiliza para definir un registro con el mismo tipo que una fila de la base de datos. También se puede asociar a una variable como del tipo de la estructura retornada por un cursor.

Ejemplo:

```
DECLARE
emp_rec emp%ROWTYPE;
CURSOR c1 IS SELECT deptno, dname, loc FROM dept;
dept_rec c1%ROWTYPE;
```

En este ejemplo la variable *emp_rec* tomará el formato de un registro completo de la tabla *emp* y la variable *dept_rec* se define por una estructura similar a la retornada por el cursor *c1*.

Ventajas en la utilización de PL/SQL

SQL se ha convertido en el lenguaje estándar de bases de datos por su flexibilidad de uso y facilidad de aprenderlo. Unos pocos comandos permiten la fácil manipulación de prácticamente toda la información almacenada en una base de datos.

SQL es no-procedural, lo cual significa que es Oracle quien se preocupará de cómo ejecutar de la mejor manera un requerimiento señalado en una sentencia SQL. No es necesaria la conexión entre varias sentencias porque Oracle las ejecuta de a una a la vez.

PL/SQL le permite a usted una completa manipulación de los datos almacenados en una base Oracle, proporciona comandos de control de transacciones y permite utilizar las funciones de SQL, operadores y pseudocolumnas. Así, usted puede manipular los datos en Oracle de una manera flexible y segura. Además, PL/SQL soporta tipos de datos de SQL, lo que reduce la necesidad de convertir los datos al pasar de una a otra aplicación.

PL/SQL también soporta SQL dinámico, una avanzada técnica de programación que convierte a sus aplicaciones en más flexibles y versátiles.

PL/SQL es un lenguaje de procesamiento de transacciones completamente portable y con un alto rendimiento, que proporciona las siguientes ventajas al ser utilizado:

Soporte para Programación Orientada a Objetos

Los objetos se han convertido en una herramienta ideal para modelar situaciones de la vida real. Con su utilización es posible reducir el costo y tiempo de construcción de aplicaciones complejas. Otra ventaja es que utilizando una metodología de este tipo es posible mantener diferentes equipos de programadores construyendo aplicaciones basadas en el mismo grupo de objetos.

Permitir el encapsulamiento del código en bloques es el primer paso para la implementación de métodos asociados a diferentes tipos de objetos construidos también con PL/SQL.

Mejor rendimiento

Sin PL/SQL, Oracle tendría que procesar las instrucciones una a una. Cada llamada produciría un overhead considerable, sobre todo si consideramos que estas consultas viajan a través de la red.

Por el contrario, con PL/SQL, un bloque completo de sentencias puede ser enviado cada vez a Oracle, lo que reduce drásticamente la intensidad de comunicación con la base de datos. Los procedimientos almacenados escritos con PL/SQL son compilados una vez y almacenados en formato ejecutable, lo que produce que las llamadas sean más rápidas y eficientes. Además, ya que los procedimientos almacenados se ejecutan en el propio servidor, el tráfico por la red se reduce a la simple llamada y el envío de los parámetros necesarios para su ejecución.

El código ejecutable se almacena en caché y se comparte a todos los usuarios, redundando en mínimos requerimientos de memoria y disminuyendo el overhead al mínimo.

Alta productividad

Si se decide utilizar otros productos de Oracle como *Oracle Forms* y *Oracle Reports*, es posible integrar bloques completos de PL/SQL en un trigger de Oracle Forms, debido a que PL/SQL es el mismo en todos los ambientes.

Completa portabilidad

Las aplicaciones escritas con PL/SQL son portables a cualquier sistema operativo y plataforma en la cual se encuentre corriendo Oracle. En otras palabras, PL/SQL corre dondequiera que se encuentre corriendo Oracle también. Esto significa que se pueden codificar librerías que podrán ser reutilizadas en otros ambientes.

Integración con Oracle

PL/SQL y los lenguajes SQL en general se encuentran perfectamente integrados. PL/SQL soporta todos los tipos de datos de SQL. Los atributos %TYPE y %ROWTYPE integran PL/SQL con SQL, permitiendo la declaración de variables basado en tipos de columnas de tablas de la base de datos. Lo anterior provee independencia de los datos, reduce costos de mantención y permite a los programas adaptarse a los cambios en la base de datos para cumplir con las nuevas necesidades del negocio.

Seguridad

Los procedimientos almacenados construidos con PL/SQL habilitan la división de la lógica del cliente con la del servidor. De esta manera, se previene que se efectúe manipulación de los datos desde el cliente. Además, se puede restringir el acceso a los datos de Oracle, permitiendo a los usuarios la ejecución de los procedimientos almacenados para los cuales tengan privilegios solamente.

MARCO METODOLOGICO

Ejemplos Declaración de variables:

En la Figura 8 se muestra la declaración de la variable V_MUNDO de tipo varchar2 que almacena el mensaje 'Hola Mundo de Ázu' y le damos salida con un DBMS_OUTPUT.PUT_LINE.

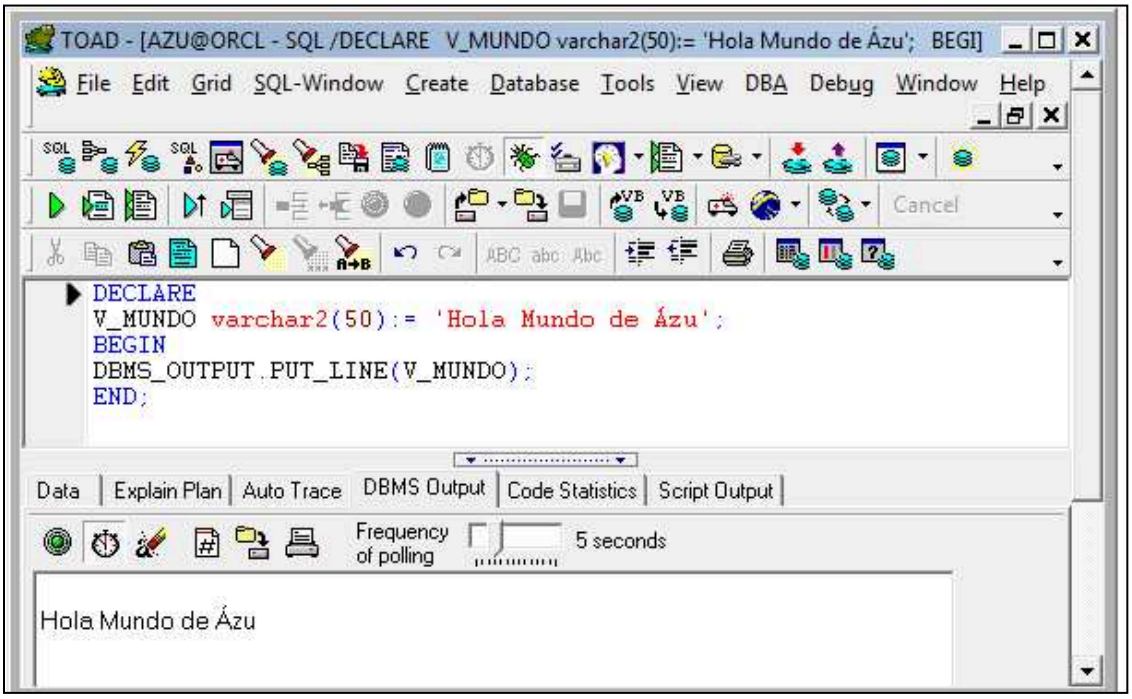

Figura 8. Declaración de la variable V_MUNDO

En la Figura 9 se muestra la declaración de la variable *amount* de tipo integer que tiene el valor de 100 y le damos salida con un DBMS_OUTPUT.PUT_LINE

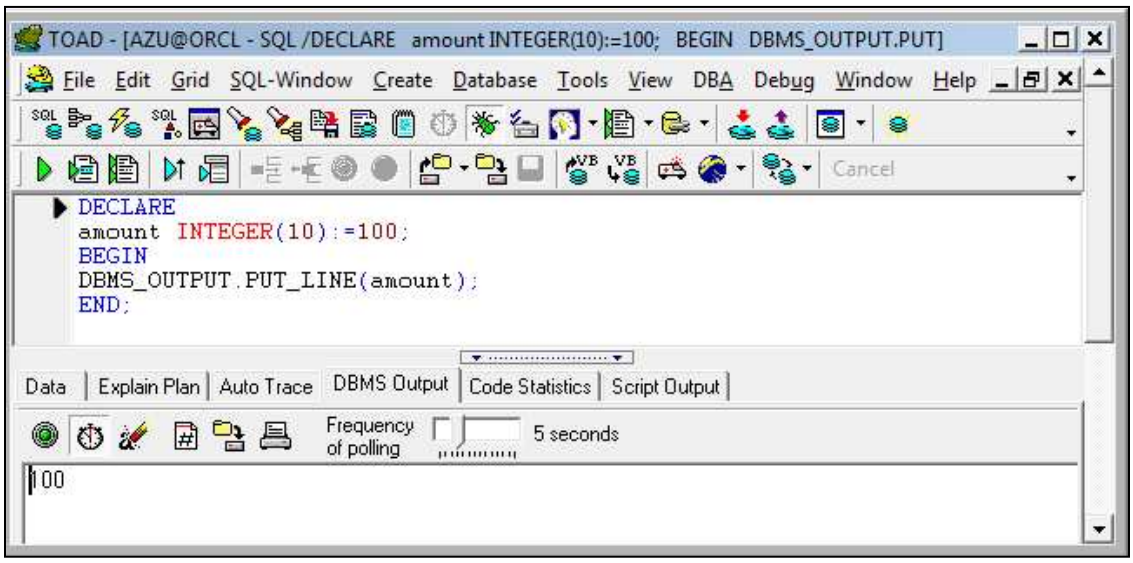

Figura 9. Declaración de la variable *amount*

En la Figura 10 se muestra un ejemplo de la declaración de variables y constantes. Se declaran dos variables una de tipo NUMBER y la otra de tipo VARCHAR, y la declaración de una constante de tipo NUMBER. Para poder ver en pantalla lo que estamos haciendo utilizamos un DBMS_OUTPUT.PUT_LINE en donde concatenamos todas las variables.

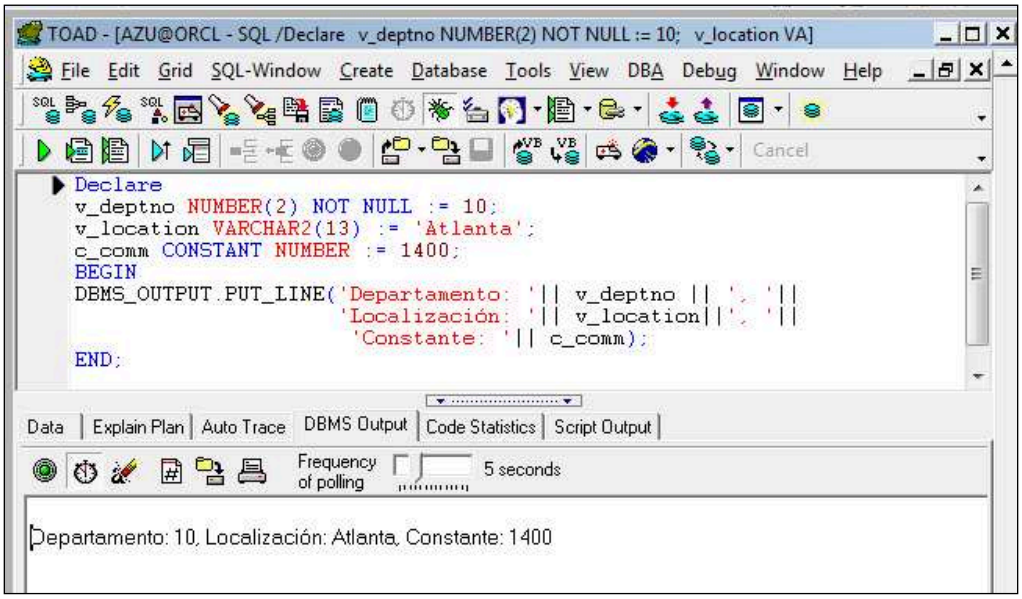

Figura 10. Declaración de variables y constantes

Ejemplos de Registros:

En la Figura 11 se presenta un ejemplo de un programa PL/SQL que cambia la especialidad de un determinado estudiante, de modo que si el estudiante no existiera se crea un nuevo registro. Primero hacemos la declaración de variables donde incluimos especialidad, nombre, apellido y Id del alumno. Actualizamos la tabla estudiantes y hacemos una consulta para colocar la especialidad de acuerdo al nombre, apellido y Id en donde se verifica si existe el registro y en caso de no existir se crea.

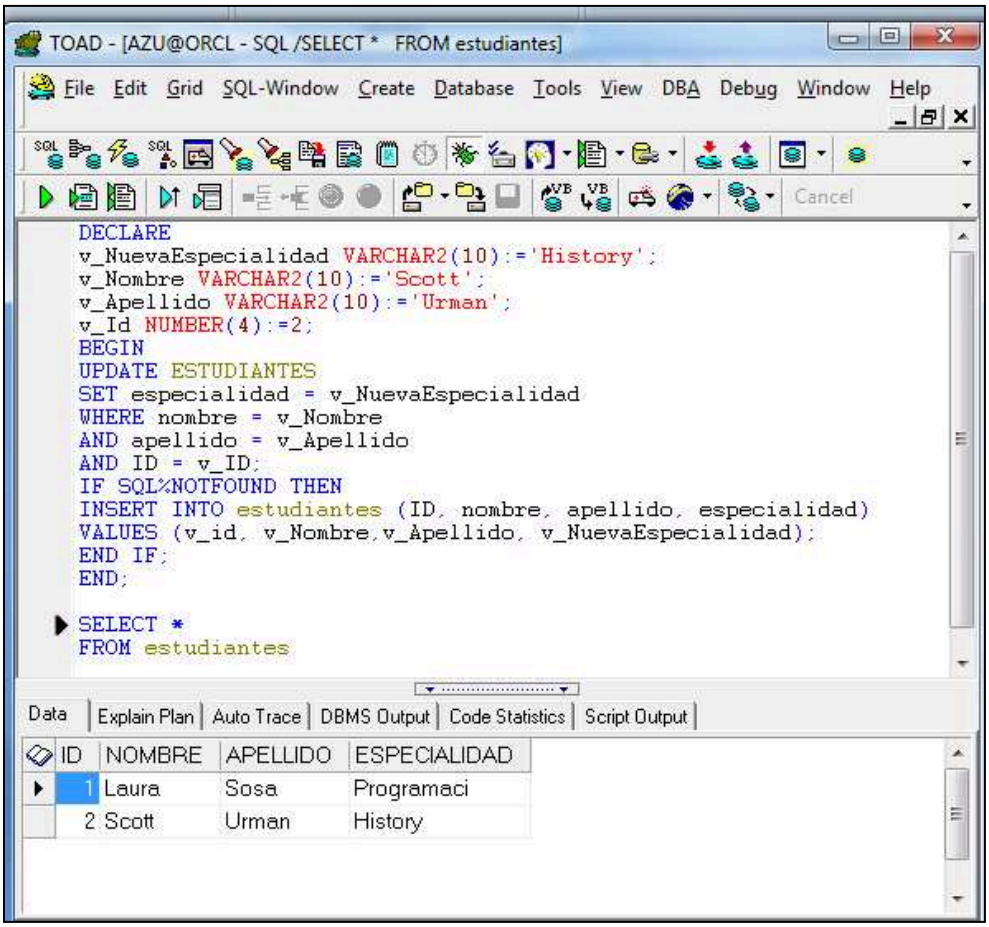

Figura 11. Programa PL/SQL para cambiar la especialidad de un estudiante

En la Figura 12 se muestra un ejemplo con %TYPE. En donde declaramos la variable tomorrow la cual utiliza la definición previa de la variable today de la base de datos, esto se hace por medio de today%TYPE.

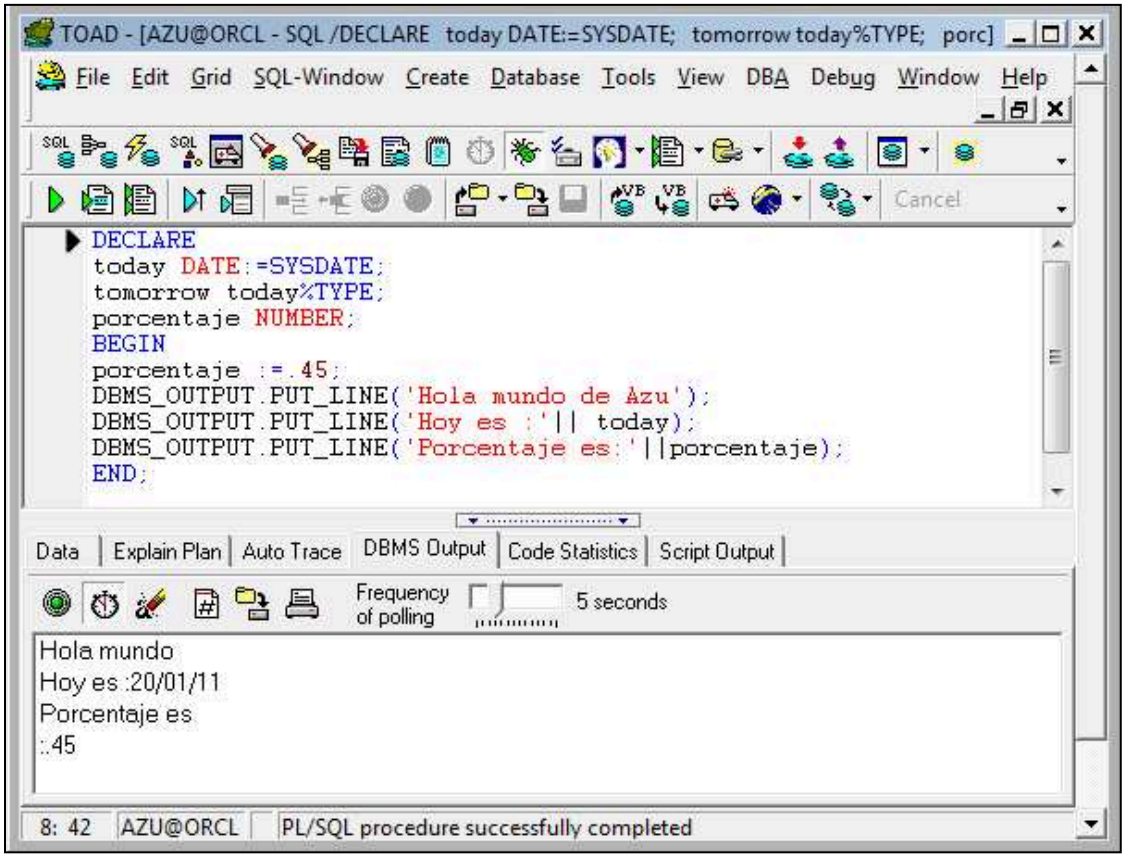

Figura 12. Utilizando %TYPE

Ejemplos de Bloques

A continuación, se presenta un ejemplo de bloque PL/SQL anónimo (Figura 13) que inserta dos filas en la tabla *temporal* de la base de datos del ejemplo, posteriormente las selecciona y las muestra en pantalla. Primero se hace la declaración de las variables a utilizar en el bloque, después insertamos las dos filas en la tabla temporal usando los valores de las variables y finalmente consultamos la tabla temporal para ver las dos filas que insertamos utilizando en paquete DBMS_OUTPUT.

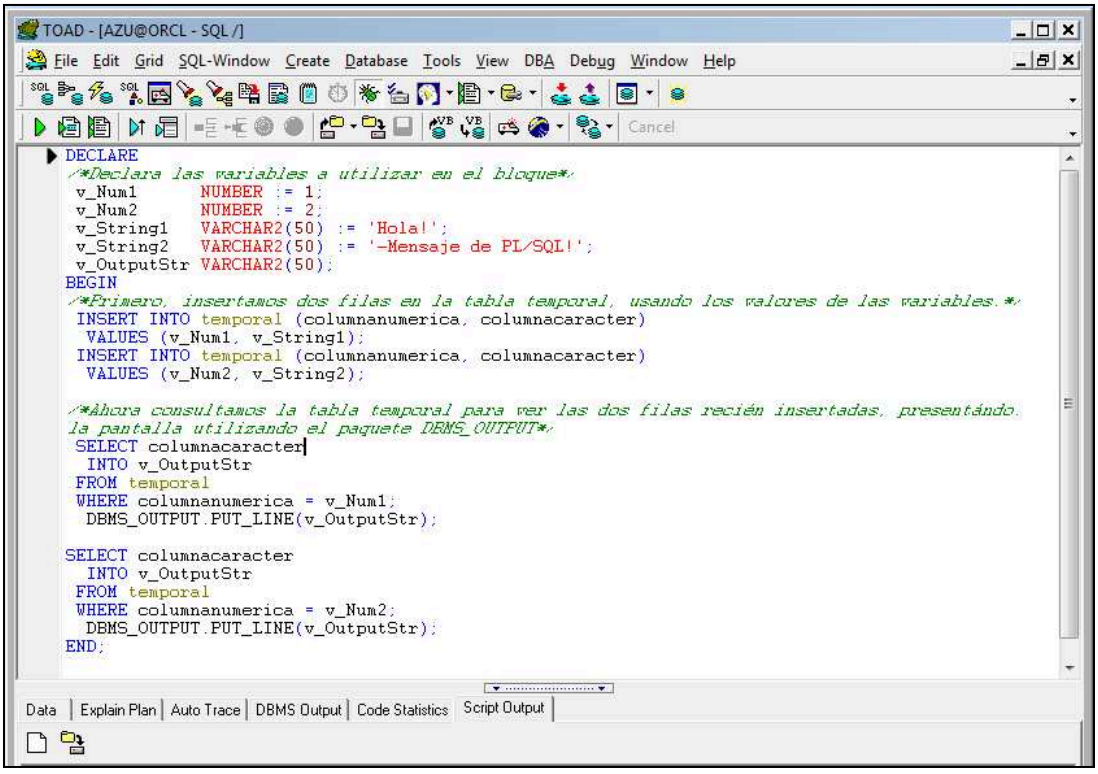

Figura 13. Ejemplo de un Bloque

Con la sentencia SELECT de SQL también podemos ver los nuevos valores introducidos en las columnas de la tabla temporal (Figura 14).

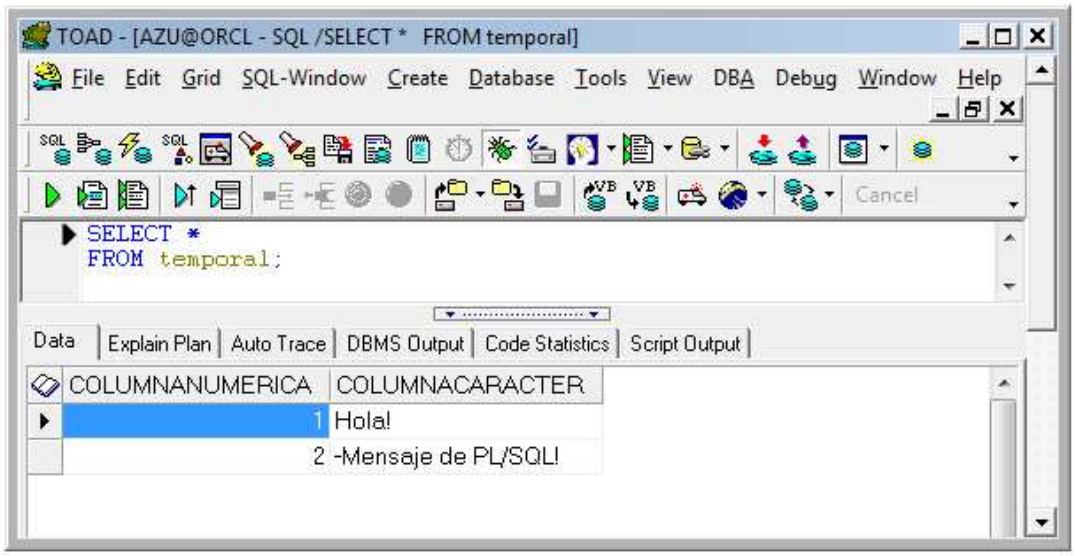

Figura 14. Valores introducidos en las columnas de la tabla temporal

Ejemplos de Procedimientos

A continuación, se presenta un procedimiento para añadir a un nuevo estudiante a la tabla estudiantes (Figura 15). Creamos el procedimiento *AñadeNuevoEstudiante* el cual inserta una nueva fila en la tabla estudiantes, usando una secuencia para generar un nuevo ID de estudiante.

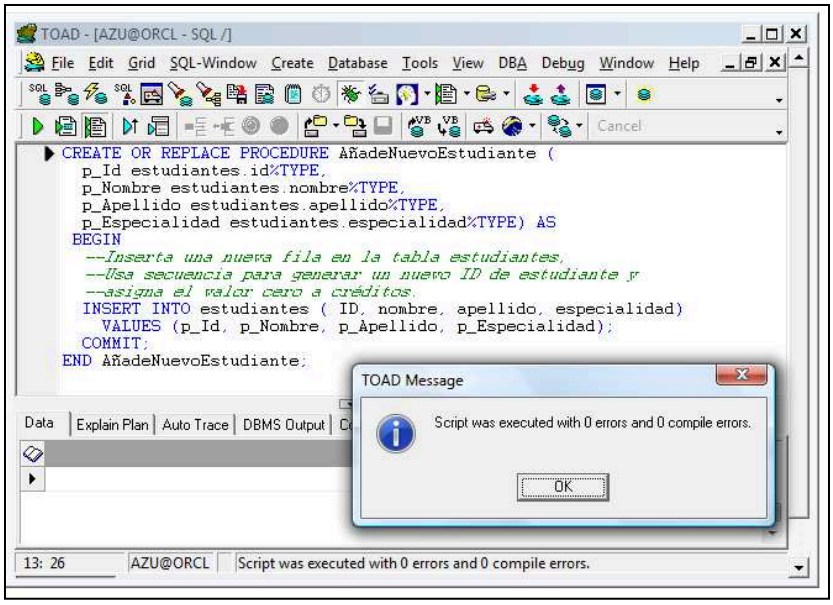

Figura 15. Procedimiento para añadir un nuevo estudiante

Una vez creado este procedimiento, puede invocarse cuando se quiera desde otro bloque PL/SQL pasándole sus parámetros formales correspondientes, tal como se muestra en la Figura 16. En donde añadimos a un nuevo estudiante.

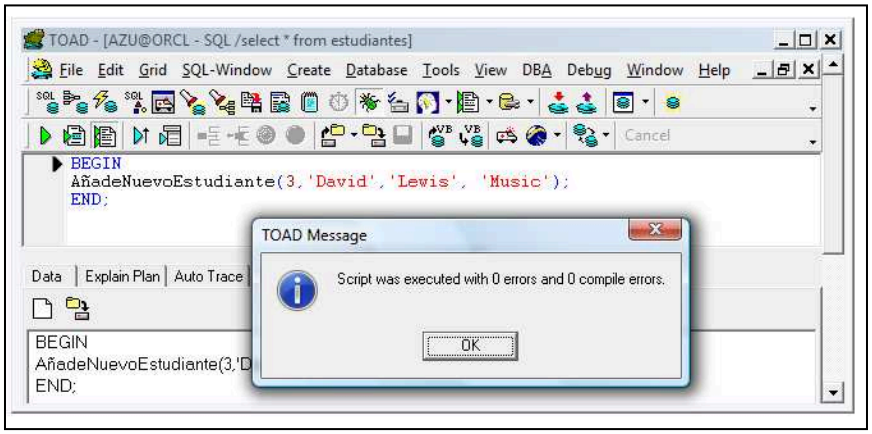

Figura 16. Invocando un procedimiento

Podemos asegurarnos que funciono el procedimiento haciendo un SELECT a la tabla estudiantes (Figura 17).

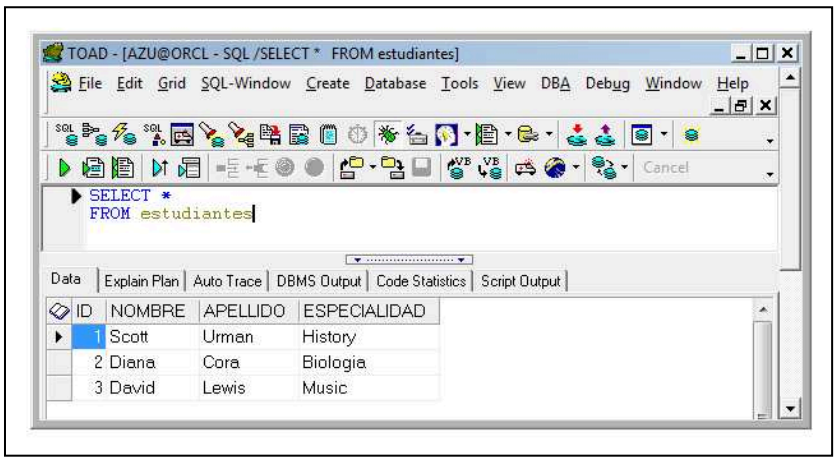

Figura 17. Select de la tabla *estudiantes*

El ejemplo de la Figura 13 del Bloque anónimo puede convertirse en un procedimiento almacenado mediante el cambio de la palabra clave DECLARE por las palabras clave CREATE OR REPLACE PROCEDURE y situando el nombre del procedimiento después de la palabra clave END como se muestra en la Figura 18.

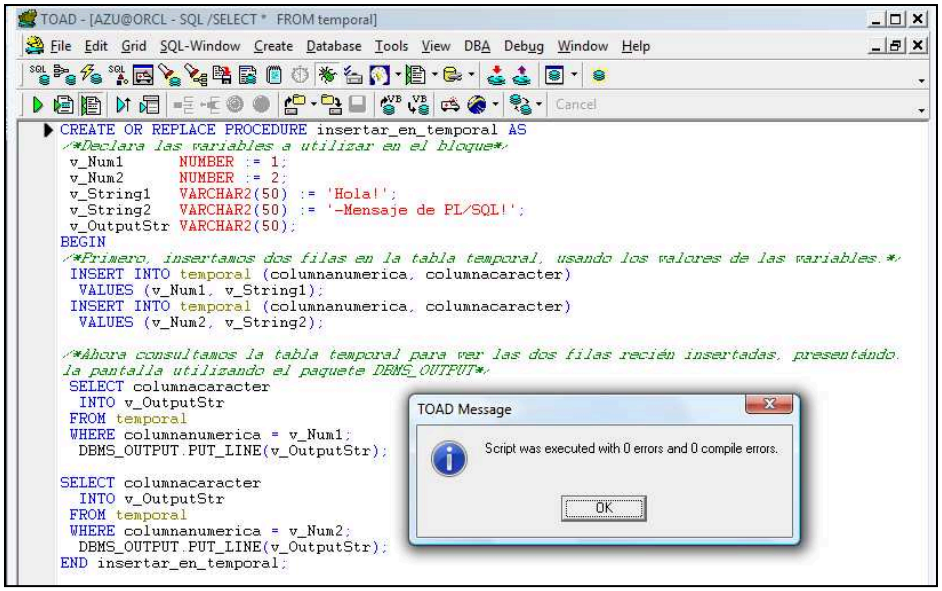

Figura 18. Procedimiento almacenado

Ejemplos de Triggers o Disparadores

A continuación, construiremos un disparador sobre la tabla *temporal* para contrastar que sólo se introducen valores positivos en la columna *columnanumerica* (ver Figura 19).

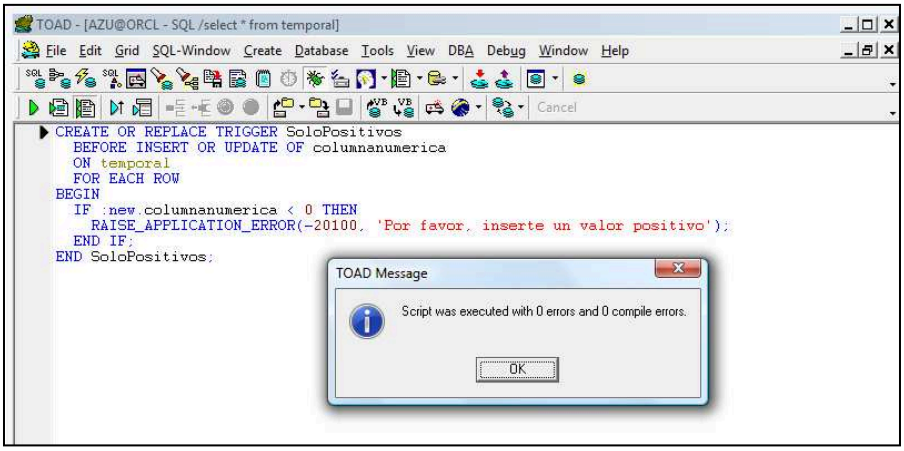

Figura 19. Creación del trigger *SoloPositivos*

Este disparador se activará cuando se inserte una nueva columna en la tabla temporal o cuando se actualice una fila ya existente, esto se muestra en la Figura 20.

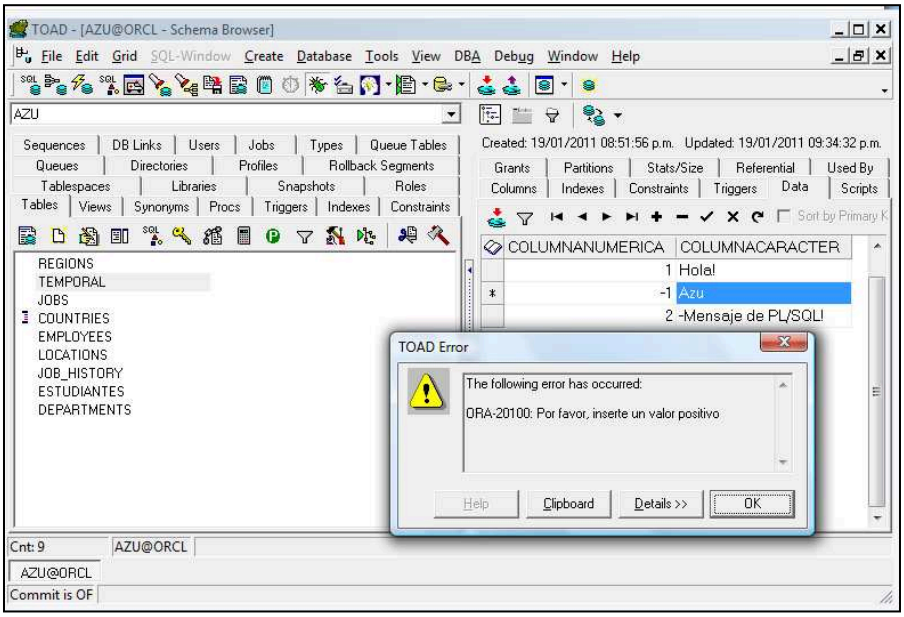

Figura 20. Activando el trigger *SoloPositivos*

Ejemplos de Paquetes

Como ejemplo crearemos un paquete que implementa una función de generación de números aleatorios (Figuras 21 y 22).

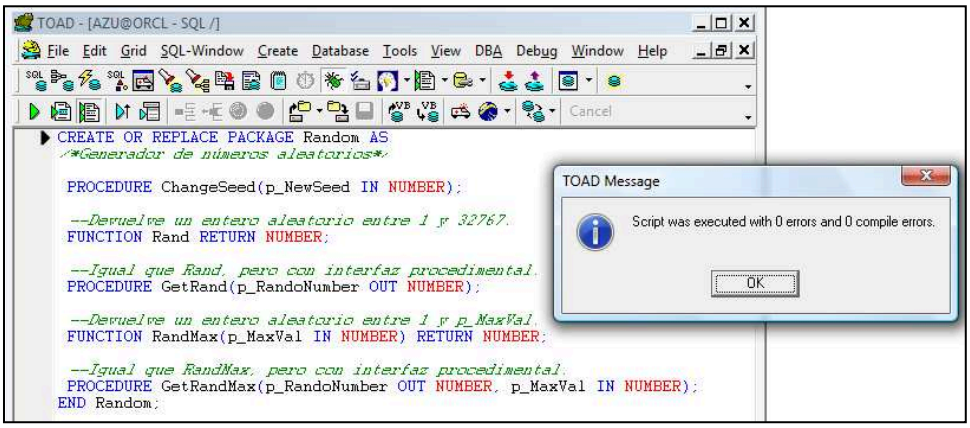

Figura 21. Creación del paquete Body

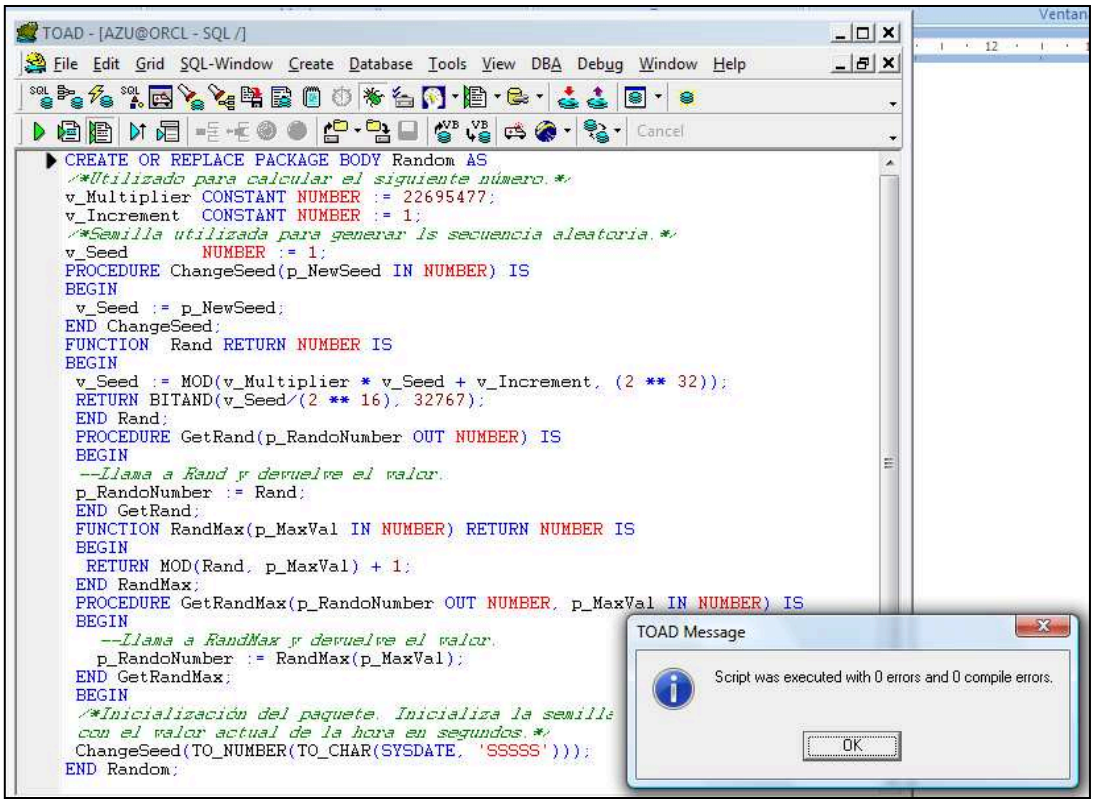

Figura 22. Creación del cuerpo del paquete Random

Para obtener un número aleatorio basta con llamar a la función Random.Rand, como se muestra en la Figura 23.

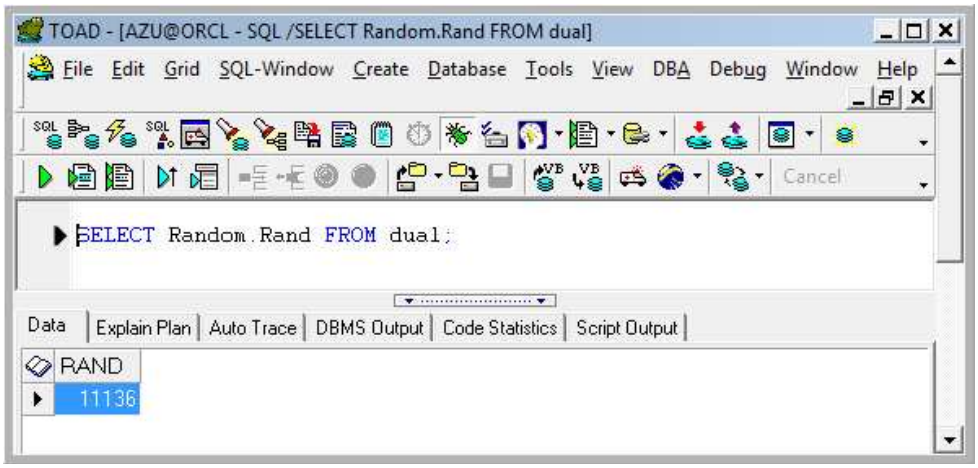

Figura 23. Llamando a la función *Random.Rand*

Ejemplos de Funciones

A continuación en la Figura 24, se construye la función *NombreCompleto* que toma como entrada el número de identificación de un estudiante y devuelve el nombre y apellido concatenados.

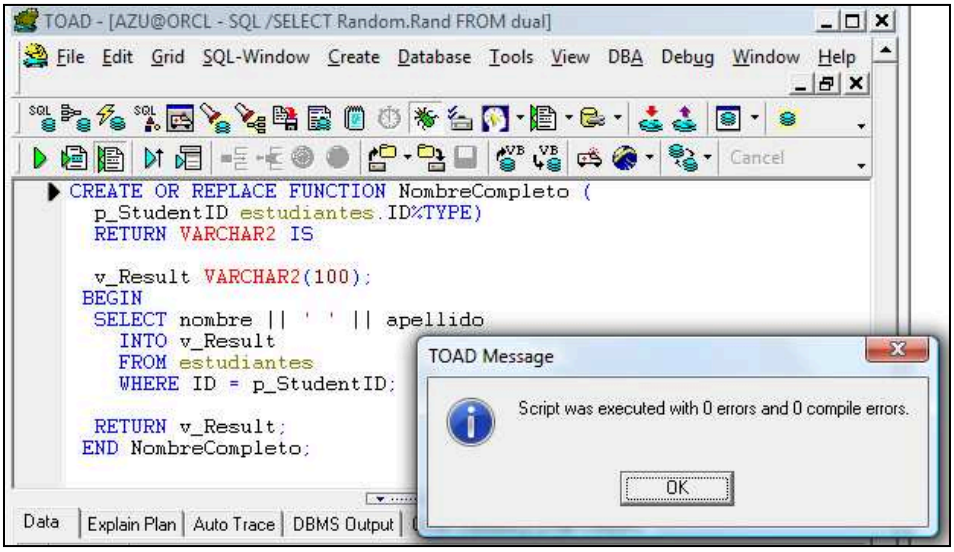

Figura 24. Creación de la función *NombreCompleto*

En la figura 25 veamos como ejecutamos la función recién creada incluyéndola en una orden SQL. Con la sentencia SELECT NombreCompleto (ID) de la tabla estudiantes.

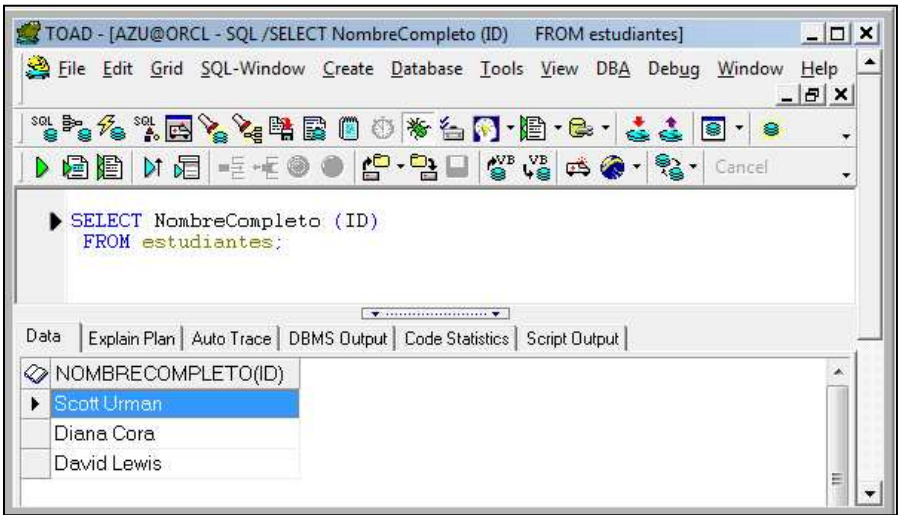

Figura 25. Ejecución de la función NombreCompleto

Ejemplos de Estructuras de Control en PL/SQL

En la Figura 26, se presenta un ejemplo que contiene las estructura condicional IF-THEN-ELSE. En donde queremos saber cuál de los dos empleados gana más. Para esto primero declaramos las variables, y luego escribimos las sentencias select con las estructura condicional if-then-else para asegurarnos que nos arroje el resultado de quien gana más utilizando DBMS_OUTPUT.PUT_LINE.

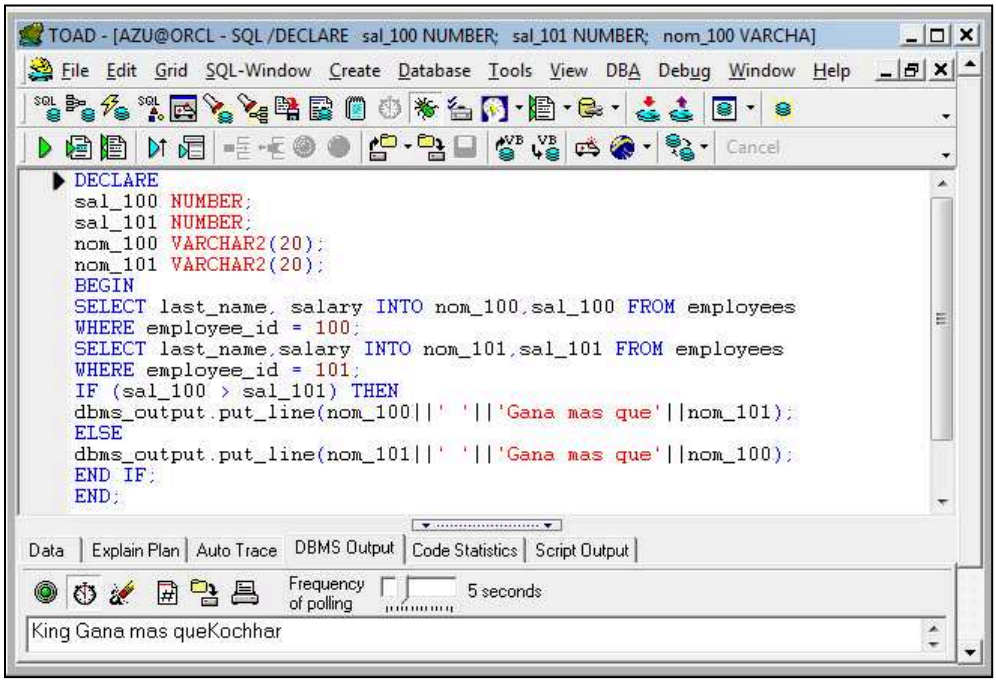

Figura 26. Estructura condicional IF-THEN-ELSE

En la Figura 27, se presenta un ejemplo que al igual que la figura anterior nos arroja cual de dos empleados gana más. Pero en esta ocasión utilizamos la estructura condicional IF-THEN-ELSIF.

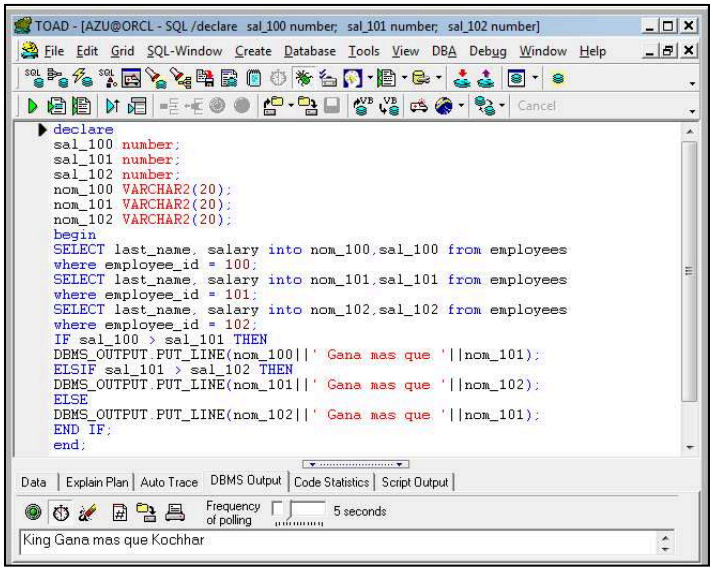

Figura 27. Estructura condicional IF-THEN-ELSIF

En la Figura 28, se presenta un ejemplo que contiene las estructuras condicionales IF-THEN-ELSE, ELSIF y ELSE. Aquí utilizamos las estructuras condicionales antes mencionadas para saber el tamaño de un aula de acuerdo al número de asientos que tiene.

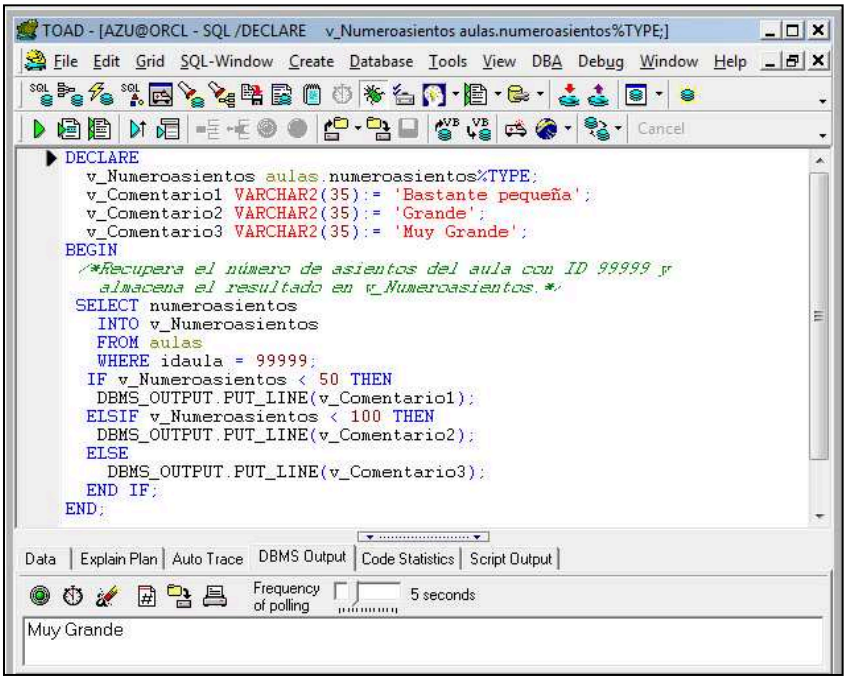

Figura 28. Estructuras condicionales

En la Figura 29, se presenta un ejemplo de bucle simple en el que se inserta una fila en la tabla temporal con el valor actual del contador del bucle y se sale del bucle cuando su contador supera el valor 50.

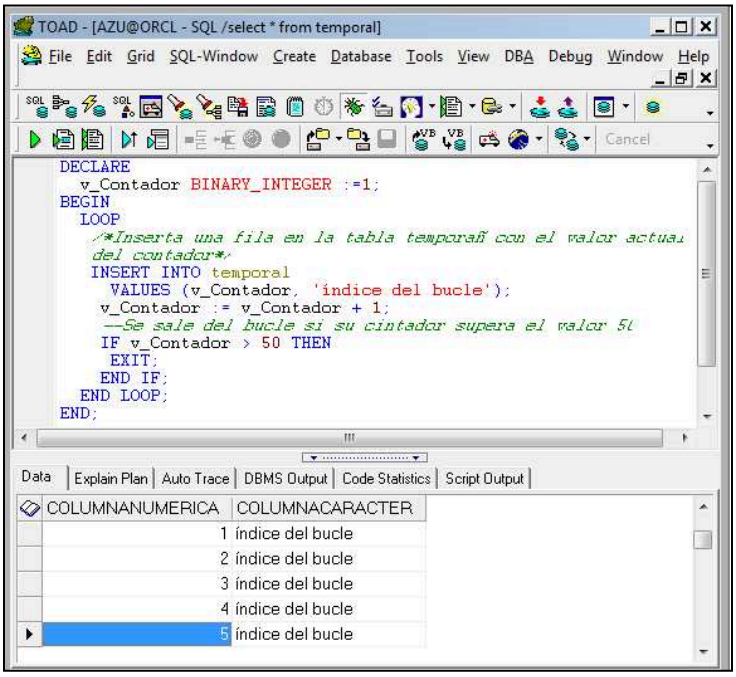

Figura 29. Ejemplo de un Bucle simple

A continuación en la Figura 30, se presenta un ejemplo de bucle WHILE que comprueba el contador del bucle antes de cada iteración para asegurar que es menor que 50. Si este hecho ocurre, se inserta una fila en la tabla temporal con el valor actual del contador del bucle y se sale del bucle cuando su contador supera el valor 50.

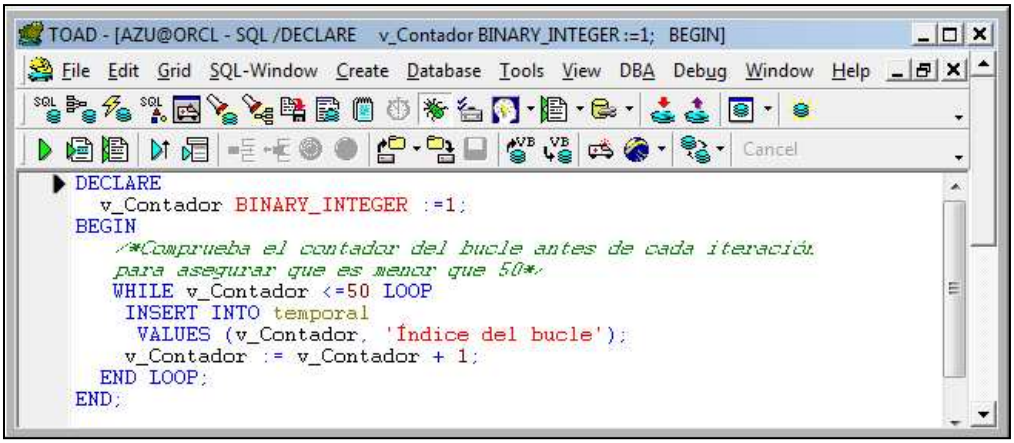

Figura 30. Ejemplo Bule WHILE

En la figura 31, podemos ver los valores recién insertados en la tabla temporal haciendo un select a dicha tabla.

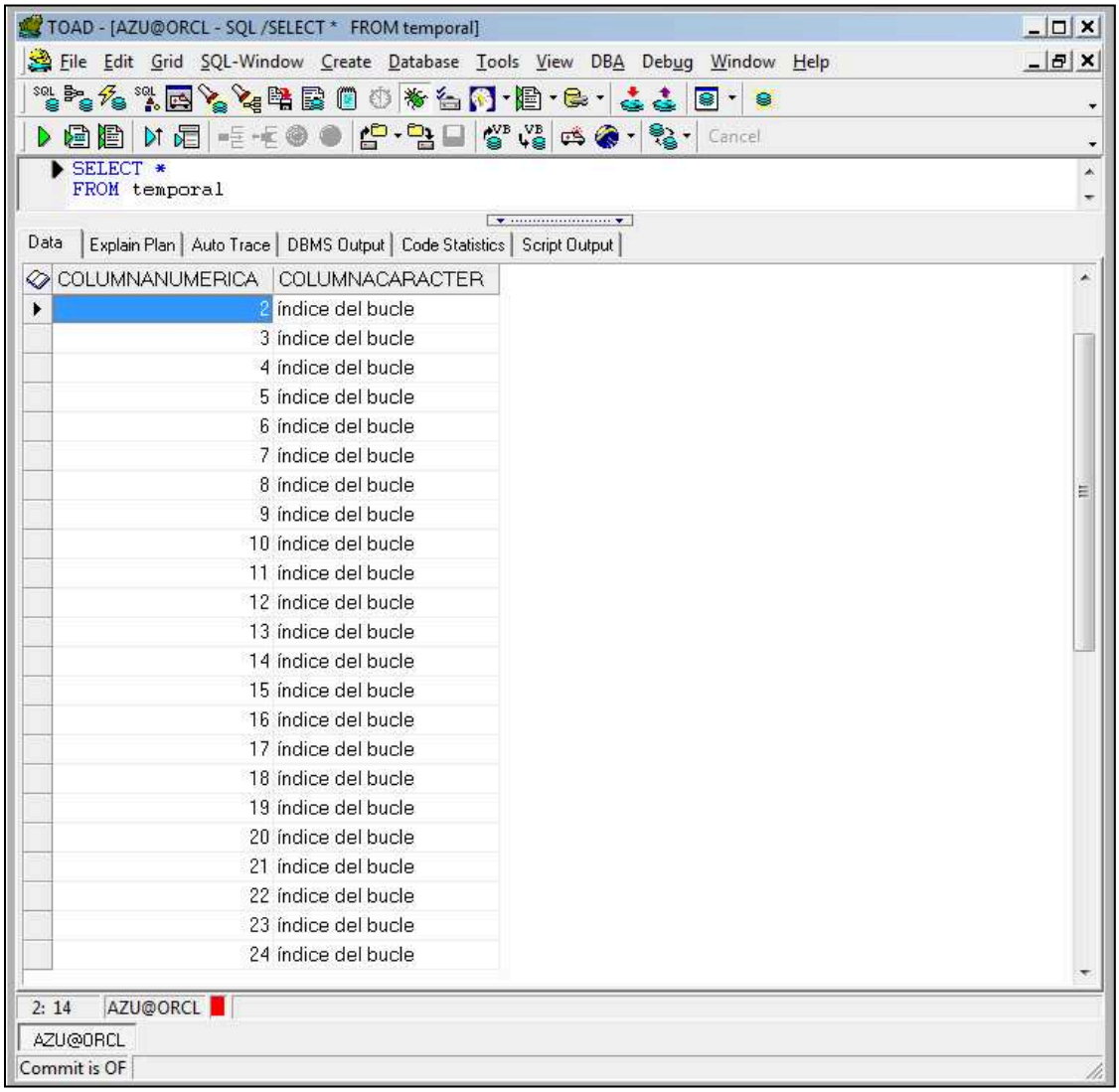

Figura 31. Select de la Tabla temporal para ver resultados del bucle While

En la siguiente figura (Figura 32) se presenta un ejemplo de un FOR en donde insertamos el número 7 en la tabla temporal.

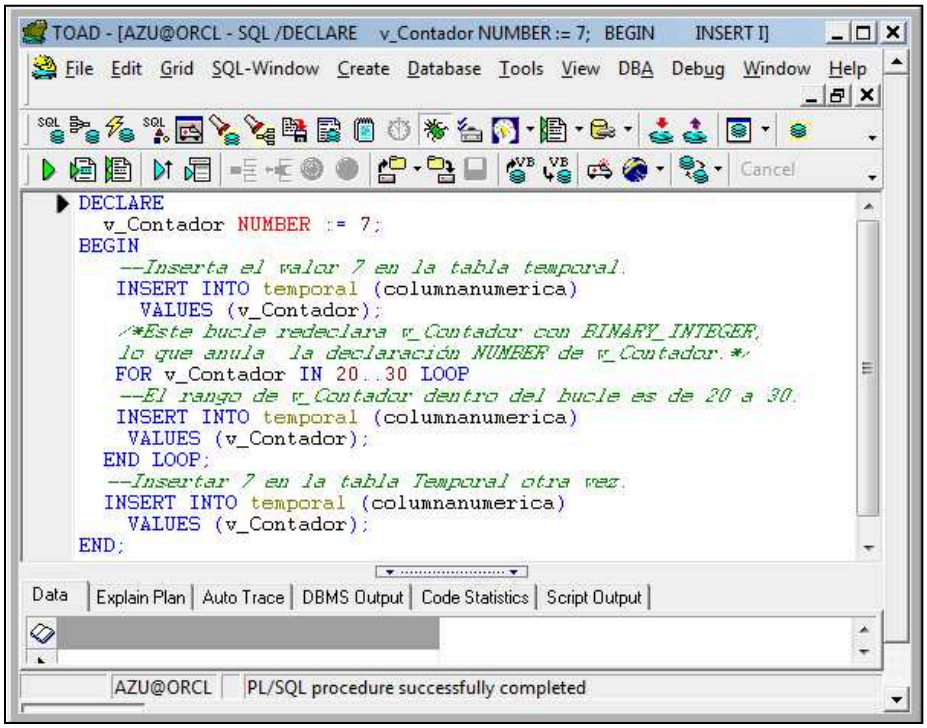

Figura 32. Ejemplo de un For

Por medio de un SELECT a la tabla temporal vemos los resultados del ejemplo anterior (ver Figura 33).

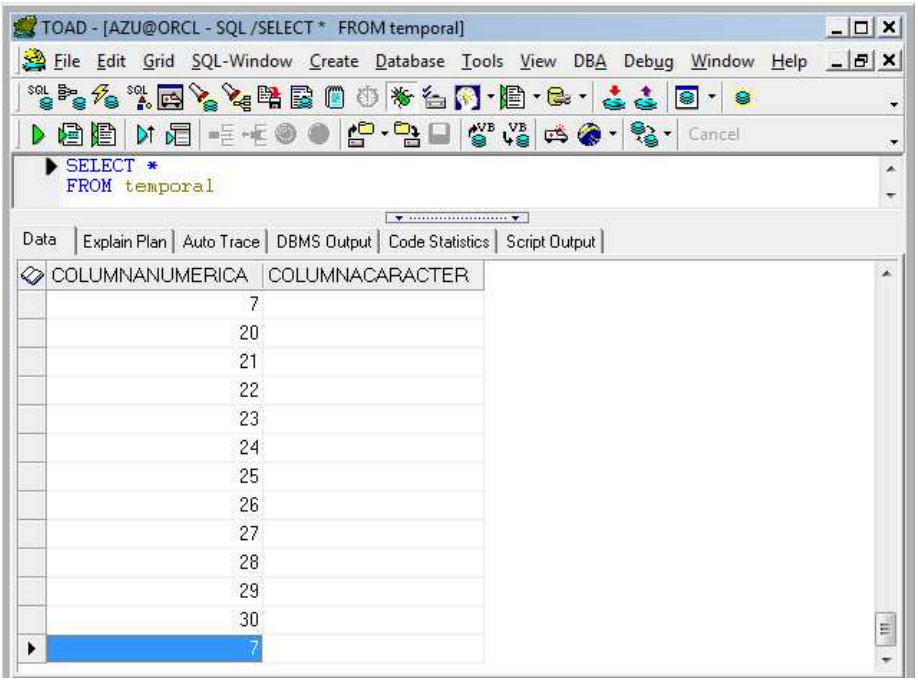

Figura 33. Select a la tabla Temporal para ver los resultados del For

En la siguiente figura (Figura 34) se presenta un ejemplo de la Orden NULL, cuando no se cumple la condición de IF no se realiza ninguna acción. Aquí declaramos la variable v_TempVar a la cual está asignado el número 7. Si esta variable es menor a 50 entonces se inserta en la variable temporal en la columna columnacacarter con el mensaje 'Demasiado pequeño', si v_TempVar es menor a 10 entonces se inserta el mensaje 'Justo a la derecha' y de no cumplir con algunas de estas condiciones no se ejecuta ninguna acción, es decir le damos la orden NULL.

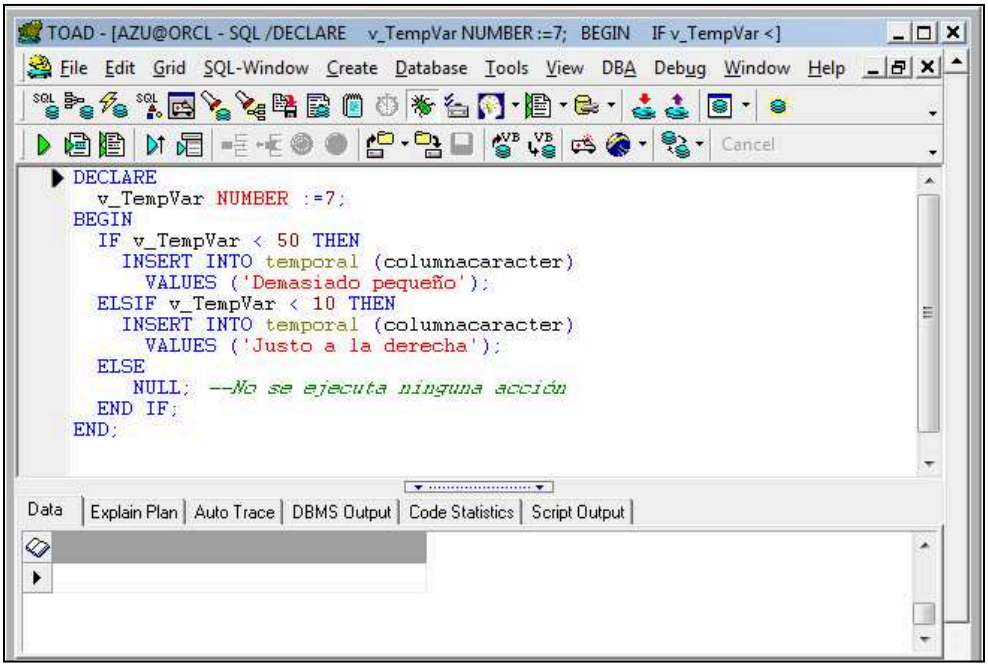

Figura 34. Orden NULL

Por medio de un SELECT a la tabla temporal vemos los resultados del ejemplo anterior (ver Figura 35).

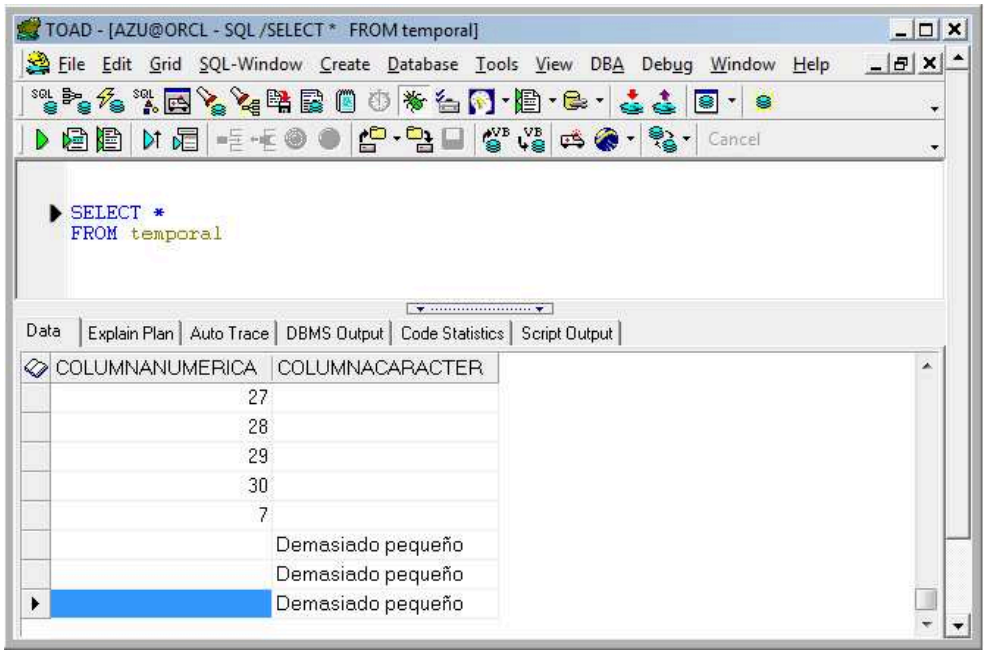

Figura 35. Select de la tabla temporal

En el ejemplo siguiente (Figura 36) se introduce el valor *'contar bucle'* en la tabla temporal para valores del contador menores de 50. Para valores mayores que 50 se utiliza una orden GOTO que pasa el control a la inserción de la cadena *'Hecho'* en el campo *columnacaracter* de la tabla temporal.

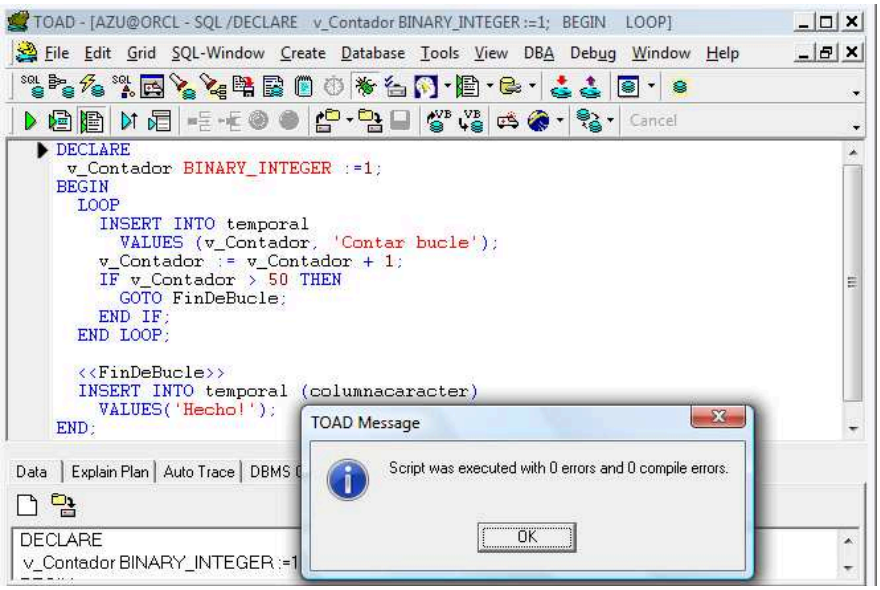

Figura 36. Utilización de GOTO

Con un SELECT a la tabla temporal confirmamos lo que se hizo en el ejemplo anterior (Ver Figura 37).

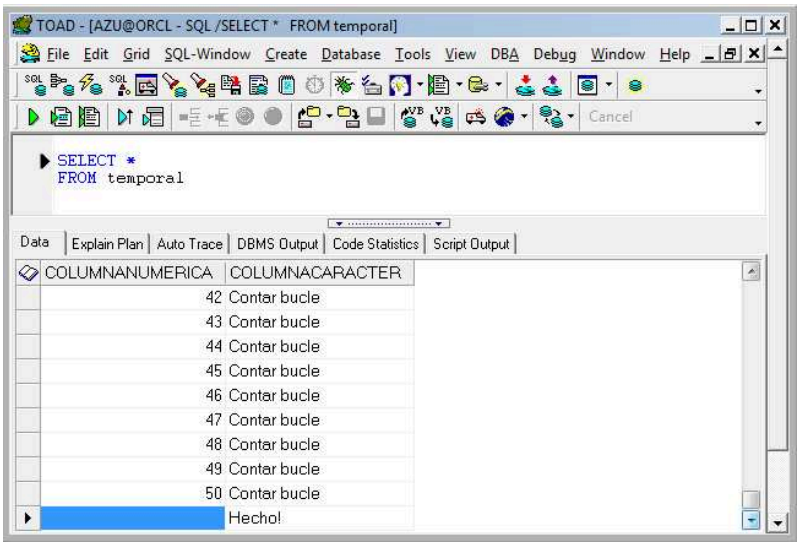

Figura 37. Select a la tabla temporal para ver resultados de utilizar un GOTO

CONCLUSIONES

En un mundo donde predominan las aplicaciones distribuidas, cada vez se ha dado más importancia a la capacidad de programar la lógica de negocio en la base de datos. En este escenario, PL/SQL se ha convertido en el lenguaje de base de datos más solicitado en la actualidad y es ahí donde entra la importancia de este trabajo.

A lo largo de esta Tesina conocí sobre lo que es la programación de PL/SQL y comprendí las ventajas de este lenguaje de programación.

Aprendí a crear bloques PL/SQL de código de aplicación que se puede compartir en varias pantallas, informes y aplicaciones de gestión de datos.

Aprendí la descripción de la base de la sintaxis de PL/SQL y de las distintas unidades de programa que se pueden desarrollar. Esto incluye la programación de procedimientos, funciones, paquetes, triggers, bucles de control, declaración de las variables, entre otros.

En general fue un gran trabajo y aprendí muchas cosas nuevas, que espero aplicar en un futuro muy cercano ya en una vida laboral.

Bibliografía

- ✓ César, Pérez. ORACLE PL/SQL. Alfaomega Ra-Ma

- ✓ Jérôme, GABILLAUD. Recursos Informáticos Oracle 10g SQL, PL/SQL, SQL*Plus. INFORMÁTICA TÉCNICA

- ✓ Oracle Database 10g: PL/SQL Fundamentals Part 2
  THOMSON
  NETg