

PL/SQL

¿Qué es PL/SQL?, PL/SQL significa Procedural Language extensions to the Structured Query Language, es decir, PL es Lenguaje Procedural y SQL es el famoso SQL. Básicamente esto significa que estamos ante la utilización de SQL (que entiendo que todos dominamos) dentro de un lenguaje procedural. Y ¿qué es un lenguaje procedural?, pues un lenguaje de toda la vida, que vamos de arriba abajo de forma secuencial, al contrario del SQL que es no procedural donde nosotros solo debemos decir qué debe hacer pero no el como, en el procedural nosotros debemos decir qué queremos y como.

Como sabemos SQL no tiene ningún elemento de programación, por lo que cada sistema gestor desarrolla su propio lenguaje para suplir esta carencia y enriquecer el SQL. PL/SQL incluye todos los comandos y operaciones de SQL pero le añade todo el paradigma de programación y paulatinamente ha ido añadiendo elementos más complejos que distinguen al PL/SQL como el lenguaje más potente del mercado.

Algunas de sus características más destacadas son:

Amplia variedad de tipos de datos para declarar caracteres, números, registros, arreglos (colecciones en Oracle).

Control condicional

Bucles

Manejo de excepciones

Manejo cursores

Utilización de XML

Reutilización de código con procedimientos y funciones, triggers, objetos (POO) y packages.

Programación dinámica, que se forma el código en tiempo de ejecución.

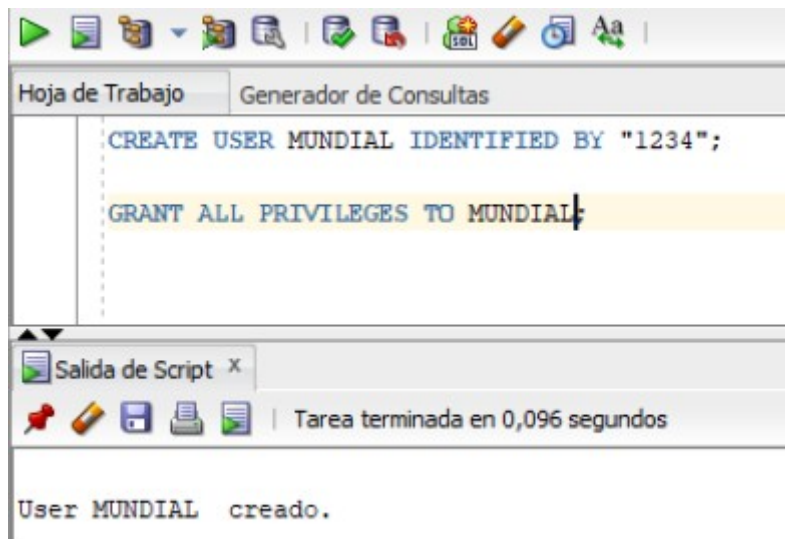
etc

PL/SQL es un lenguaje procedural y de transacciones, muy potente y de fácil manejo, está basado en otro lenguaje de programación llamado ADA, el cual fue un lenguaje diseñado por el departamento de defensa de Estados Unidos. Ambos, tanto ADA como PL/SQL tienen como antecesor común el PASCAL, sin embargo, PL/SQL dispone de una estructura novedosa, el Package, que permite a los programadores definir tipos de datos públicos y privados, contantes y variables estáticas en estos paquetes PL/SQL.

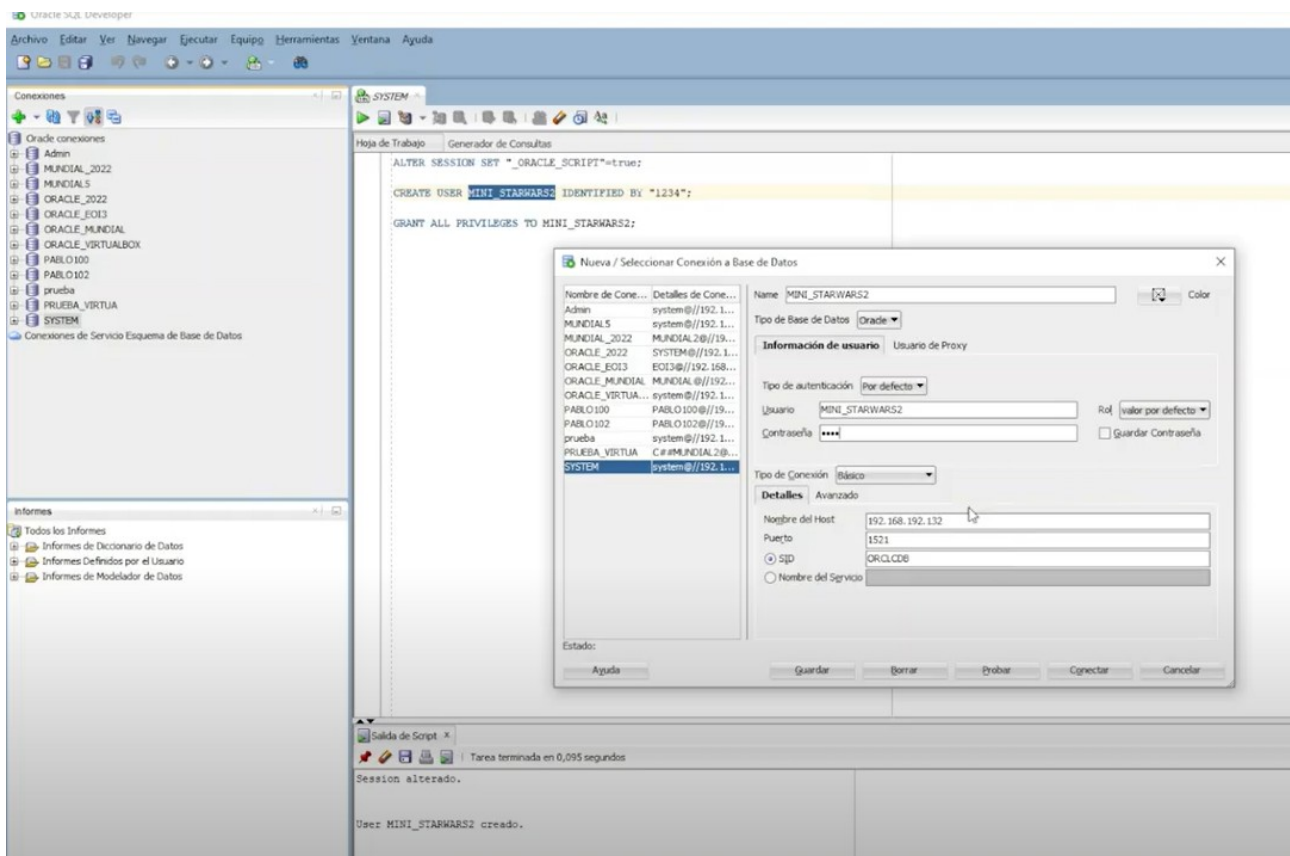
Ada es un lenguaje que se enfoca en la abstracción, ocultamiento de información y otras estrategias de diseño. Gracias a este diseño PL/SQL es un lenguaje muy poderoso que incluye la mayoría de los elementos de los lenguajes procedurales

PL/SQL también permite definiciones de clases e instanciarlas como objetos en el código PL/SQL, muy similares a la programación orientada a objetos de otros lenguajes como Object Pascal, C++ o Java.

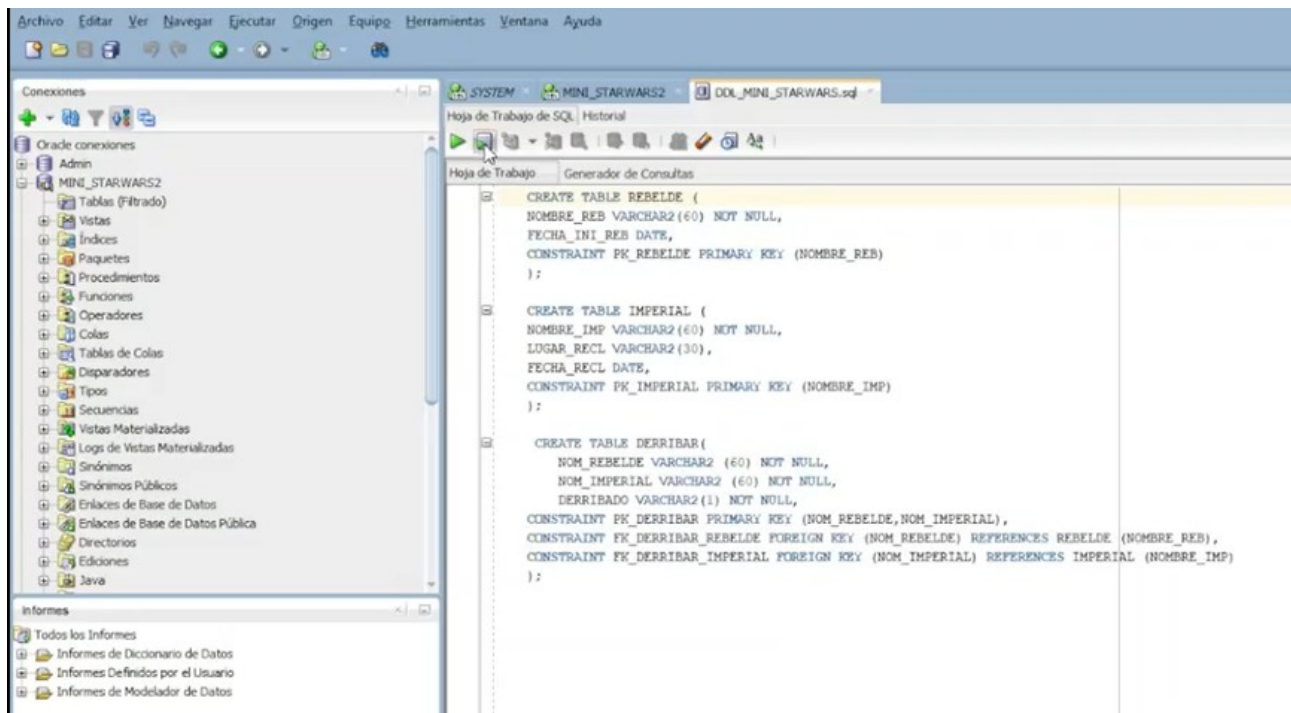
Primeros pasos:



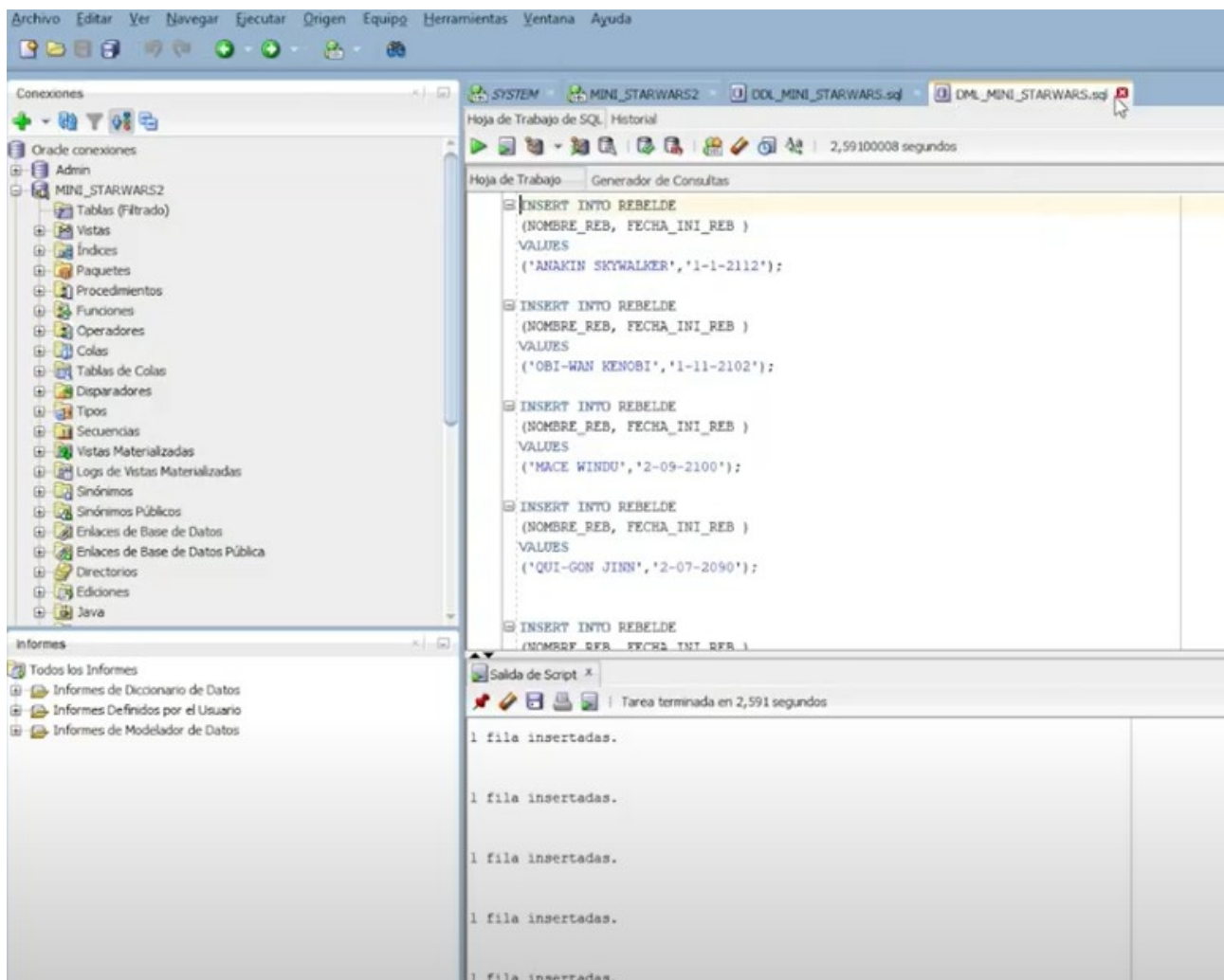
Una vez hacemos esto, ya podemos volver a repetir el proceso dándole a la cruz verde para crear otra conexión pero esta vez nos metemos en el usuario MUNDIAL que es donde trabajaremos.



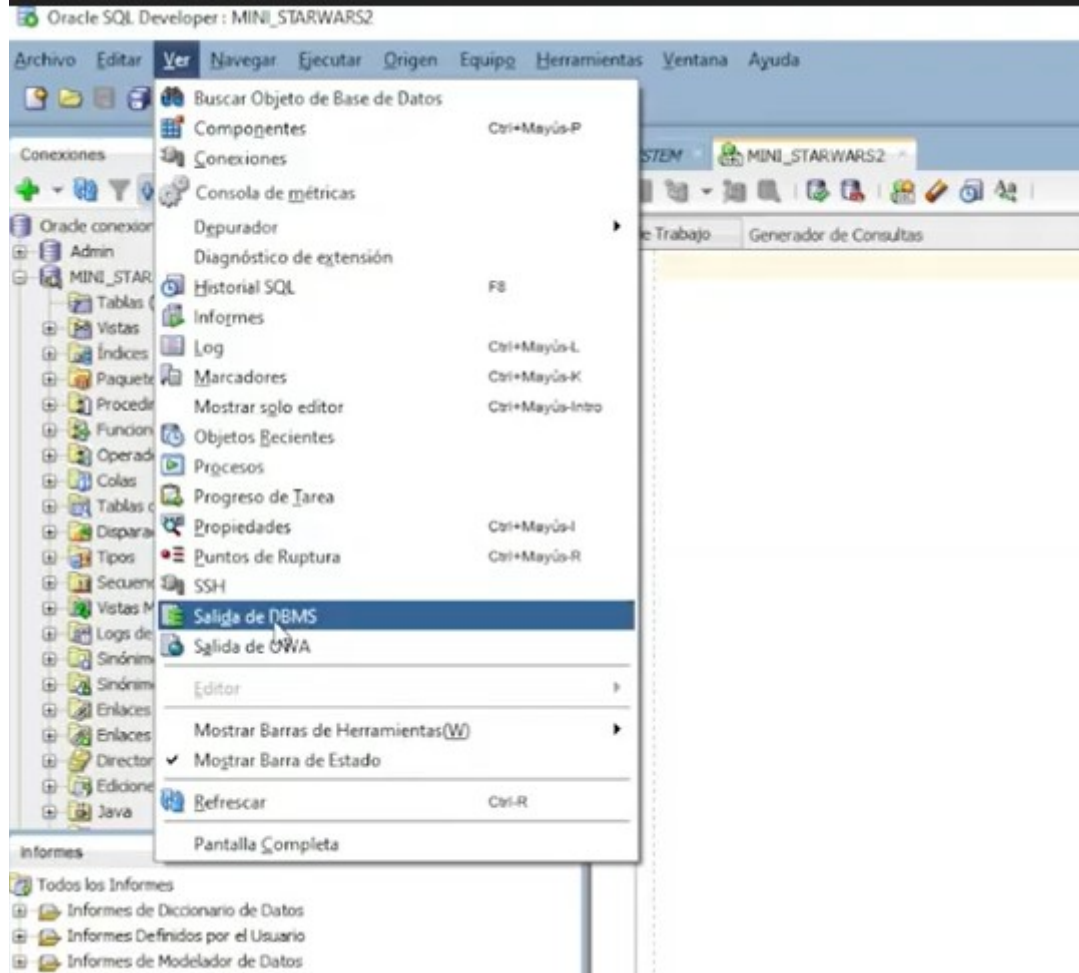
DDL

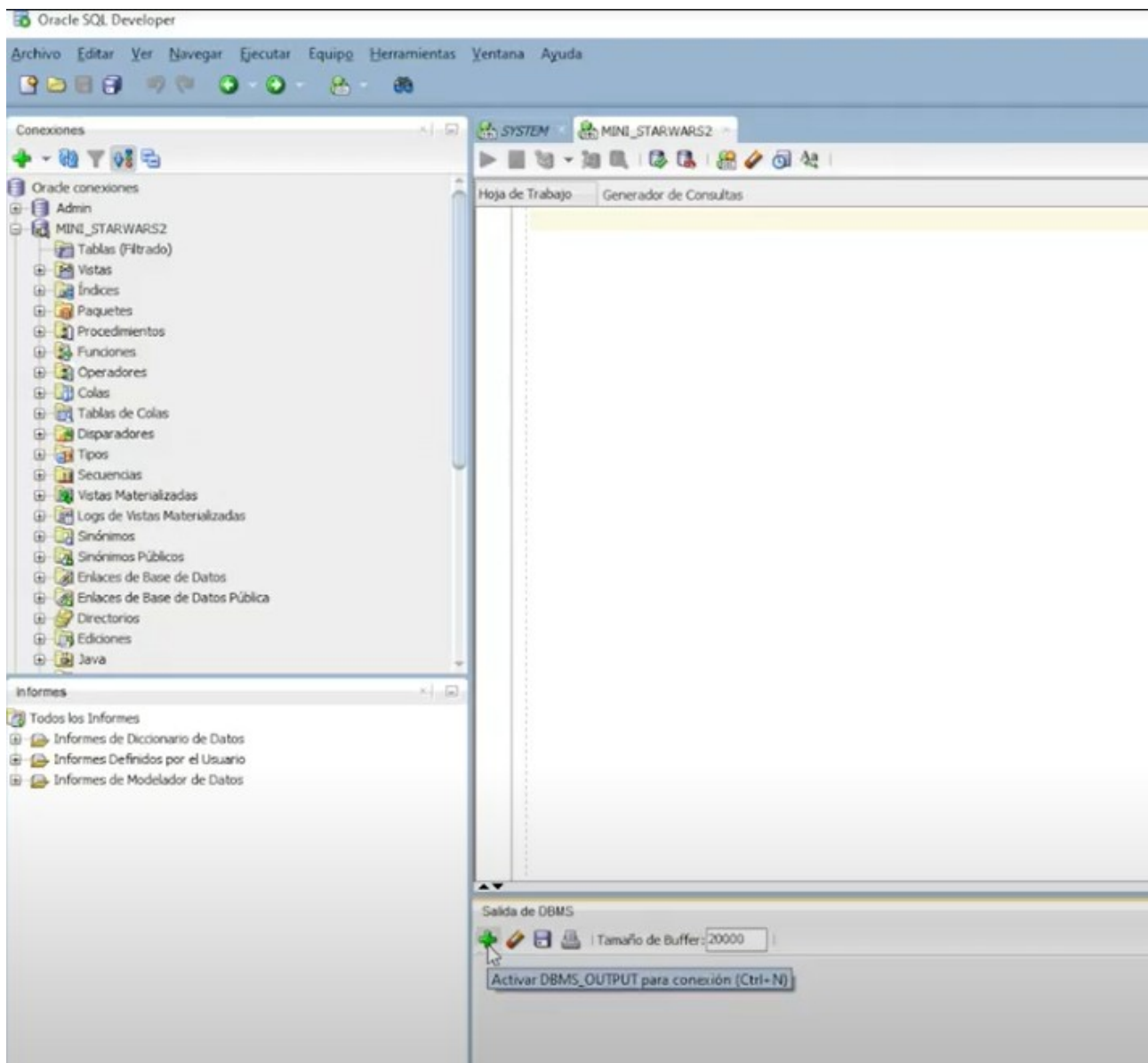


DML



Bloques anónimos





Activar conexión

Escribir bloque anónimo y ejecutar:

Conexiones

Oracle conexiones

- Admin
- MINI_STARWARS2
 - Tablas (Filtrado)
 - Vistas
 - Indices
 - Paquetes
 - Procedimientos
 - Funciones
 - Operadores
 - Colas
 - Tablas de Colas
 - Disparadores
 - Tipos
 - Secuencias
 - Vistas Materializadas
 - Logs de Vistas Materializadas
 - Sinónimos
 - Sinónimos Públicos
 - Enlaces de Base de Datos
 - Enlaces de Base de Datos Pública
 - Directorios
 - Ediciones
 - Java

Informes

- Todos los Informes
- Informes de Diccionario de Datos
- Informes Definidos por el Usuario
- Informes de Modelador de Datos

SYSTEM MINI_STARWARS2

Hoja de Trabajo Generador de Consultas

```

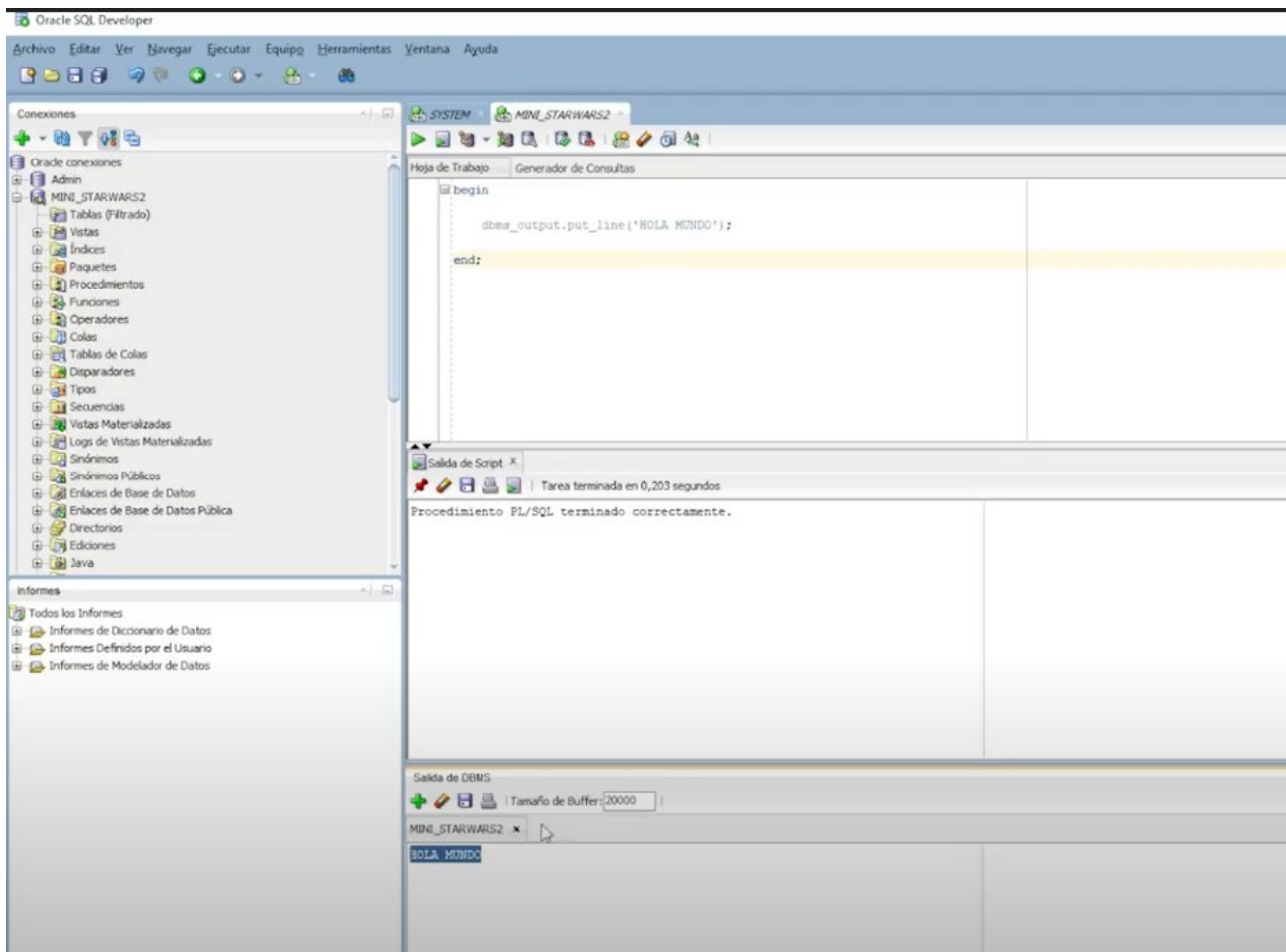
begin
    dbms_output.put_line('HOLA MUNDO');
end;

```

Salida de DBMS

Tamaño de Buffer: 20000

MINI_STARWARS2 x



La estructura básica de PL/SQL es **el bloque**, que fundamentalmente nos los vamos a poder encontrar en:

⑩•**Bloques anónimos**: Como su propio nombre indica, no tiene nombre, por lo que no podrá ser llamado de ningún modo, sirve para realizar scripts puntuales o para hacer pruebas, será con lo que empezamos para realizar nuestros primeros pinitos con el lenguaje. Y lo primero que voy a empezar explicando a continuación.

⑩•**Funciones o Procedimientos**: Es básicamente lo mismo que el anterior, pero en este caso sí tienen nombre, por lo que son los que se usarán para realizar una aplicación y los famosos procedimientos almacenados. Serán ampliamente explicados en este mismo tema, en su último punto.

⑩•**Triggers**: Son disparadores que se ejecutan en el momento que se realiza cualquier acción DML sobre una tabla concreta, son desarrollados en PL/SQL.

BLOQUE ANÓNIMO

En la estructura básica del lenguaje, y por la que siempre se empieza a tomar contacto con el mismo. Vamos a distinguir tres partes muy diferentes:

⑩•**DECLARE (No obligatoria)**: Zona de declaración de variables, en esta parte se declaran todas las variables que va a utilizar el bloque anónimo, las variables SOLO se pueden declarar en esta zona, en ninguna otra zona o nos dará un error, obviamente tampoco podemos utilizar en ningún sitio variables que no se hayan declarado previamente en esta zona. Es optativa, porque en el caso que no haya ninguna variable que declarar no se pondría esta zona.

⑩•**CUERPO. BEGIN... END (Obligatoria)**: Es la zona donde se pondrá todo el código PL/SQL que queremos que se ejecute, no se puede poner NADA de código antes del BEGIN ni después del END. No hay ningún problema en poner nuevos bloques anónimos en su interior infinitamente uno dentro de otro como una muñeca rusa mamushka. En principio no parecer muy útil anidar bloques pero pronto veremos que también hay casos que pueden ser de gran utilidad.

⑩•**EXCEPTION (No obligatoria)**: Es una zona especial, no obligatoria (si no queremos capturar ninguna excepción), que tiene la peculiaridad que solo se accede a la misma, cuando se haya producido alguna excepción (ya veremos lo que es en temas posteriores, ahora para entendernos podemos llamarlo errores, aunque no se ajusta exactamente a la realidad) en una zona de BEGIN...END. Admite volver a poner más bloques anónimos en su interior.

```

1
2 • DECLARE
3
4   -- ZONA DE DECLARACIÓN DE VARIABLES
5
6 • BEGIN
7
8   -- ZONA DEL CUERPO, DONDE SE PONE EL CÓDIGO
9
10 • EXCEPTION -- (OPTATIVA)
11
12   -- ZONA DE CAPTURA DE EXCEPCIONES
13   -- LAS EXCEPCIONES LAS VEREMOS EN EL TEMA 3
14
15 END;
```

ZONA DECLARE

Variables

Declaración de Variables

La declaración de variables solo se puede efectuar en la zona del DECLARE, y tiene la siguiente sintaxis:

Identificador [CONSTANT] tipo_dato [NOT NULL] [:= | DEFAULT expresion];

Vamos comentando cada parte:

Identificador: Obviamente obligatorio. Es el nombre que le vamos a poner a la variable y debe cumplir las siguientes características:

- ⑩•Debe comenzar un carácter alfabético, es decir, debe empezar por letra.
- ⑩•Luego puede contener letras, números, guión bajo (_) y el signo dólar (\$) (se usa para variables del sistema).
- ⑩•No puede tener ningún carácter más ni espacios.
- ⑩•Puede tener un máximo de 30 caracteres.
- ⑩•Debemos ser cuidadosos y no debe tener el nombre de ningún campo de tablas de la base de datos, por ello se suele seguir una guía de nomenclatura para evitarlo.
- ⑩•Obviamente no se pueden poner nombres de palabras reservadas de ORACLE.
- ⑩•Oracle no distingue mayúsculas de minúsculas, luego miVariable, MIVARIABLE o MIvariable es exactamente lo mismo para Oracle.

CONSTANT: Es una opción en la declaración que indica que el valor que se le adjunte a la variable no puede ser modificada en ningún momento, si se intenta dará error de compilación. La típica constante de otros lenguajes. No podemos dejarla con valor NULL, estaremos obligados darle un valor.

PI CONSTANT NUMBER:=3.1416;

tipo_dato: El otro elemento obligatorio en la declaración, debemos elegir un tipo de dato para la variable, en el siguiente apartado indicaremos los más importantes que existen.

NOT NULL: Otra opción que podemos añadir a la variable, donde indicamos que la variable que estamos declarando no puede estar nunca vacía, en breve también veremos las peculiaridades del valor NULL. Obviamente si indicamos esta opción estamos obligados a darle un valor inicial a la variable.

:= | DEFAULT expresión: Son dos formas de decir lo mismo, esto sirve para darle un valor inicial a la variable que estamos declarando, podemos decidirlo con el operador asignación o con la palabra reservada default, podemos poner un valor directamente o una expresión que su resultado sea un valor, por ejemplo:

```
A NUMBER := 5;
A NUMBER DEFAULT 5;
B NUMBER := A*A;
B NUMBER DEFAULT A*A;
```

```
declare
    soy la primera number; -- DA ERROR PORQUE NO PODEMOS PONER ESPACIOS EN BLANCO
    soyLaSegunda number;
    soy_la_tercera number;
    soyConstante constant number; -- DA ERROR PORQUE UNA VARIABLE CONSTANTE HAY QUE ASIGNARLE VALOR
    soyConstanteBuena CONSTANT number := 5;
    soyNoNulo number not null; -- DA ERROR PORQUE UNA VARIABLE NOT NULL HAY QUE ASIGNARLE VALOR
    soyNoNulo2 number not null := 5;
    lvariable number; -- DA ERROR PORQUE NO PUEDE EMPEZAR POR NÚMERO
    variable1 number:=5;
    variable2 number :=variable1*2;
    variable3 number := variable4+variable2; -- DA ERROR PORQUE VARIABLE4 AÚN NO ESTA DECLARADA
    variable4 number := 10;

begin
    NULL;

end;
```

Como cualquier lenguaje tiene muchos y diversos tipos de datos, pero entre los más importantes los podemos resumir en esta tabla:

Tipos de datos Oracle				
Alfanuméricos	Numéricos	Fecha	Binarios	Otros
CHAR VARCHAR2 VARCHAR NCHAR NVARCHAR2 LONG	NUMBER FLOAT	DATE	RAW LONG RAW BLOB CLOB NLOB BFILE	ROWID

NOTA:

Con los tipos NUMBER para números, VARCHAR2 para alfanuméricos y DATE para fechas, suficiente tendrás para la mayoría de las tareas

CHAR[(n)]: Almacena cadenas de caracteres alfanuméricos. n será la longitud máxima de caracteres que deseamos que la variable almacene, si no indicamos este número por defecto será 1. La particularidad de este tipo de datos es que siempre almacena exactamente los caracteres que se indica, es decir, que si ponemos que quedemos almacenar 50 caracteres, la variable almacenará 50, aunque en su interior insertemos una cadena de menos caracteres

oracle lo rellenará con espacios en blanco. Por lo que hay que tener especial cuidado en su uso, sobretodo con las comparaciones para evitar sorpresas, como por ejemplo:
Si comparamos dos cadenas de tipo char:

```
cadena1 char(10):='hola';  
cadena2 char(10):='hola ';
```

**cadena1 es igual a
cadena2**

Sin embargo:

```
cadena1 char(10):='hola';  
cadena2 VARCHAR2(10):='hola';  
cadena3 VARCHAR2(10):='hola ';
```

**cadena1 es distinta a cadena2
cadena1 es igual a cadena3**

Máximo número de caracteres en Tablas de la BD: 2000

Máximo número de caracteres en PL/SQL: 32767

(n): Almacena cadenas de caracteres de longitud variable. n será la longitud máxima de caracteres que tendrá la variable y es obligatorio indicar este número. Lo bueno de este tipo de dato que solo almacena la cantidad de caracteres que uses a diferencia del anterior. Este tipo de dato es el más usado en PL/SQL y en ORACLE en general, y prácticamente podemos realizar cualquier cosa usando solo este tipo en todas las variables.

Máximo número de caracteres en Tablas de la BD: 4000

Máximo número de caracteres en PL/SQL: 32767

VARCHAR (n) (obsoleto): Tipo de datos obsoleto, en principio es exactamente igual que el anterior pero se lo reserva ORACLE para nuevas mejoras en sus tipos de datos y no es recomendable su uso.

NCHAR (n) y NVARCHAR2(n): Son lo mismo que CHAR y VARCHAR2 solo que permiten almacenar caracteres unicode. Por esta razón en BD como usan 2 bytes por carácter solo podrán llegar a 1000 caracteres en el caso del NCHAR y a 2000 en el caso del NVARCHAR2

LONG (obsoleto): Almacena caracteres de longitud variable hasta 2 Gb. Este tipo de dato se soporta para compatibilidad con versiones anteriores. Actualmente se debe usar los tipos de datos CLOB y NLOB para almacenar grandes cantidades de datos alfanuméricos.

NUMÉRICOS

NUMBER [(p,[s])]: Es el tipo numérico por antonomasia, junto con el tipo VARCHAR2 y luego el DATE son los tres tipos de datos básicos para trabajar con PL/SQL, con estos tres tenemos más que suficiente para los programas normales.

Almacena números enteros o en coma flotante, prácticamente soportan cualquier longitud. p es precisión y s escala, podemos declarar el tipo number sin especificar ni precisión ni escala, y entonces permite la cantidad máxima, que son una precisión de 38 dígitos, con una escala de -84 a 127. Es decir que prácticamente podemos almacenar cualquier número con cualquier número de decimales.

En el caso que queramos una precisión muy concreta es cuando deberemos indicar precisión y escala:

⑩•p: Es el número total de números que queremos almacenar, incluyendo los decimales.

⑩•s: Es la escala, es decir cuantos números serán decimales de los indicados en p (podrías superar el número de precisión, pero en ese caso todos los números serán decimales). Puede ser un número negativo, en ese caso redondea tantas cifras como el valor negativo que se indique.

Cabe la opción de especificar uno solo de los dos, si solo especificamos p, entonces no habrá decimales es como si pusiéramos 0 en la s. Y si solo especificamos s, es como si pusiéramos 38 en precisión, veamos algunos ejemplos:

⑩•num NUMBER;

En la variable num se permiten cualquier cantidad de números y decimales.

⑩•num NUMBER (10,2);

En la variable num contendrá un máximo de 10 números de los cuales 2 serán decimales, es decir que en num el número más grande posible será 99999999.99. Es decir que podemos poner un máximo de 8 número enteros y 2 decimales, si pones más de 8 enteros dará error y si ponemos más de 2 decimales no dará error pero ORACLE redondeará el número a 2 decimales.

⑩num NUMBER(10);

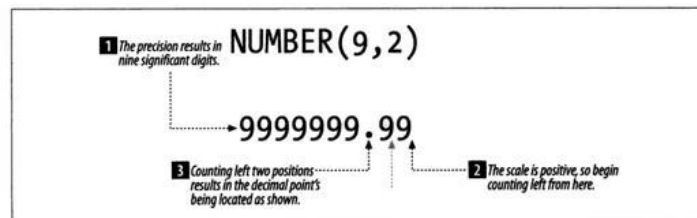
Es exactamente lo mismo que poner num NUMBER (10,0), lo que significa que num almacenará hasta 10 números enteros y no podrá tener decimales, si nos empeñamos en ponerlos, oracle redondeará.

⑩•num NUMBER (*,5);

Esto es lo mismo que poner num NUMBER (38,5), tiene sentido cuando queremos no limitar la cantidad de números enteros permitidos pero sí queremos obligar a que solo hayan 5 decimales.

⑩•Precisión > Escala

Este es el caso más usual, y es rarísimo que debamos usarlo de otro modo, de poner la precisión mayor que la escala:

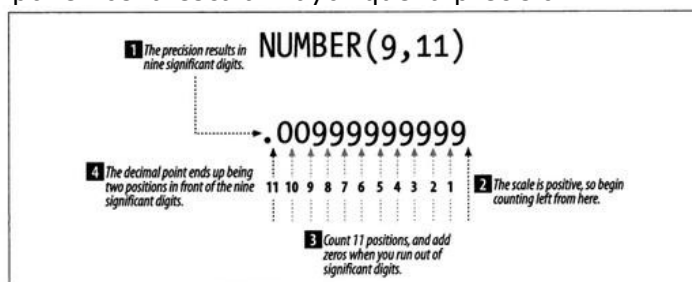


Como resultado de esto, tendremos que podemos insertar 7 enteros y 2 decimales, por lo que tendremos los siguientes casos:

Valor dado	Valor que almacena
1000,45	1000,45
1234567,1223	1234567,12
1234567,1283	1234567,13
1234567,9993	1234568,00
12345678,00	Da error porque excede los 7 enteros

⑩•Precisión < Escala

Pero como he comentado antes, la precisión son un máximo de 38, y la escala puede ser de - 84 a 127, qué pasa si ponemos la escala mayor que la precisión:

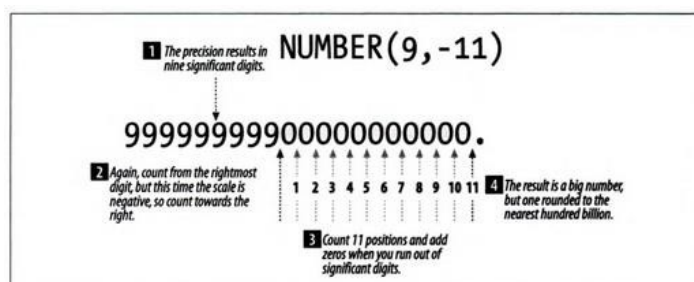


Como corremos el punto mucho más a la izquierda que el número máximo de la precisión, en este caso 2 posiciones a la izquierda, entonces solo podemos almacenar números menores que 0,01. Tendremos que tener mínimo dos ceros en los decimales y luego podremos tener cualquier número pero el máximo que almacenará serán 11 decimales, redondeando si ponemos más.

Valor dado	Valor que almacena
0,00123456789	0,00123456789 (máximo número de decimales)
5	0,000000000001 (Se pasa y redondea hacia arriba)
1	0,000000000000 (Se pasa y redondea hacia abajo)
0,01	Da error porque no deja los dos decimales de la precisión

ⓈEscala negativa

La última opción es con escala negativa, el sentido que tendría esto es para almacenar números tremendamente grandes, sacrificando ciertos números más pequeños como veremos enseguida:



En este caso tenemos 9 números significativos, y la escala en vez de desplazarse a la izquierda, lo hace a la derecha, consiguiendo que almacenemos números gigantescos, pero sacrificando que el número mínimo a almacenar sea superior a 11 dígitos, es decir el mínimo número que podemos guardar es 100.000.000.000 (con 12 enteros). Debemos tener en cuenta que podrá almacenar 9 dígitos y los otros 11 siempre almacenará 11 ceros, si le indicamos otro número redondeará.

Valor dado	Valor que almacena
50.000.000.000,12345	100.000.000.000 (porque redondea hacia arriba)
49.999.999.999,12345	0 (Redondea hacia abajo)
100.000.000.000.000.000.000	Da error, porque se pasada de los 20 dígitos máximo.

FLOAT [(n)]: Es un subtipo de NUMBER. Almacena un número en coma flotante, solo se le puede indicar la precisión, porque decimales siempre son todos los que se quiera, tampoco es obligatoria indicarla, en ese caso será 126. Sinceramente nunca he utilizado este tipo, y con el NUMBER es totalmente prescindible.

FECHA

DATE: Almacena datos de tipo fecha con la hora. Internamente almacena. **Siglo - Año - Mes - Día - Hora - Minutos - Segundo**

Este formato tiene la ventaja que puede alterado en cualquier momento con funciones y solo obtener lo que queramos. Internamente se almacena como números de día desde cierto punto de inicio, lo que permite que las fechas sean tratadas en operaciones aritméticas normales.

'1-JAN-2010'+10 = '11-JAN-2010'

'1-JAN-2010'-1 = '31-DEC-2009'

Con las funciones de conversión podemos moldear y mostrar la parte de la fecha que queramos como veremos en la sección correspondiente.

BINARIOS

LOB: Estos tipos los desarrollaremos en el apartado de tipos de datos complejos, decir ahora que, Oracle desde la versión 8i nos provee un tipo de dato llamado LOB, el cual nos permite almacenar largas estructuras de información estructurada y no estructurada como texto ,

gráficos, audio y video. Asimismo la información multimedia puede residir tanto en la misma base de datos como en el sistema operativo. Este tipo de dato se crea en reemplazo a los tipos de datos antiguos que existían como: LONG, RAW y LONG RAW debido a todas las restricciones y problemas de mantenimiento que presentaban.

La capacidad máxima que un LOB puede albergar es de 4 GB.

Los LOBS se categorizan en CLOB (almacenan texto que contiene grandes cantidades de bytes), NCLOB (es similar al CLOB solo que almacena texto cuyo juego de caracteres está definido por el National Character Set de la base de datos), BLOB (almacena información multimedia dentro de la base de datos), BFILE (similar al BLOB solo que la información multimedia está almacenada en el sistema operativo).

La versión Oracle Database 11g ha hecho varias mejoras sobre los LOB presentando una nueva propuesta llamada Secure Files, el cual entrega mejores tiempos de respuesta en el acceso a los datos, ahorro en espacio y seguridad.

OTROS

BOOLEAN: El operador booleano, solo admite los valores TRUE, FALSE o NULL. Y aunque siempre lo podemos imitar con un VARCHAR2(1) o NUMBER, es muy útil para comparaciones porque podemos poner la variable directamente

DECLARE

a boolean:=TRUE;

BEGIN

IF a then

dbms_output.put_line('CIERTO');

ELSE

dbms_output.put_line('FALSO');

end if;

END;

ROWID: Una de las virguerías de Oracle. Es una pseudocolumna que identifica de manera única a cada fila de cada tabla, es una columna invisible que toda tabla de Oracle tiene, la crea automáticamente Oracle y solo se puede consultar, no aparece en los SELECT ni se puede modificar en los UPDATE, ni en los INSERT. Tampoco se puede utilizar en los CREATE. Es un tipo de dato utilizado exclusivamente por Oracle. Sólo se puede ver su valor utilizando la palabra reservada ROWID en una select.

Tiene una enorme utilidad en PL/SQL, como ya iremos viendo, sus usos más interesantes son:

⑩•Es el medio más rápido de acceso a una fila en la tabla.

⑩•Muestra información de como se almacena una fila en la tabla. Ya que sus caracteres indican: El segmento de la base de datos donde está la fila, el número de fichero de tablespaces relativo que contiene la fila, el bloque de datos que contiene a la fila y el número de fila en el bloque

⑩•Es un identificador único de la fila en la tabla.

Ejemplo: select rowid, nombre, apellidos from ALUMNOS

```

• declare

    identificador rowid;

• begin

    -- PODEMO OBTENER EL ROWID CON UNA SELECT

    select rowid
       into identificador
      from JUGADOR
     where nombre='LIONEL MESSI';

    -- PODEMOS USAR EL ROWID COMO IDENTIFICADOR
    -- PARA ACCEDER A UNA FILA CONCRETA

    select *
       into .....
      from jugador
     where rowid = identificador;

• end;

```

%TYPE

Existe otra forma de declarar tipos de variable, que es con la cláusula %TYPE. Concretamente se trata de declarar una variable con el tipo de dato de otra que ya ha sido declarada previamente, con lo que no ahorramos indicar el tipo de dato y su utilidad es que si cambiamos el tipo de dato de la primera se modifican automáticamente el resto.

Sintaxis:

<nombre_variable> <nombre_variable_origen_del_tipo>%TYPE

Ejemplo:

```

declare

    variable1 varchar2(40);
    variable2 variable1%type;
    variable3 variable1%type;

```

Nota: realmente el uso más extendido del %TYPE no es para que el tipo de dato se obtenga de otra variable, si no de un campo de una tabla de la Base de datos que se desea usar la variable en cuestión para meter datos de dicho campo

Conversiones de tipos

ORACLE prácticamente hace todas las conversiones implícitamente, rara vez es la ocasión que he necesitado realizar conversiones explícitas, salvo cuando he de jugar con fechas. Las funciones más utilizadas para la conversión son:

TO_CHAR: Convierte lo que se le ponga a una cadena alfanumérica, se suele utilizar para convertir fechas a cadena. Por ejemplo sacar el año de una fecha:

TO_CHAR(<variable fecha>, 'YYYY')

El segundo parámetros es la máscara y podemos obtener la parte de la fecha que queramos. Por ejemplo si tenemos una fecha en un campo y queremos sacar diferentes datos de la misma:

```
SELECT to_char(fecha, 'DD') DIA, to_char(fecha, 'MM') MES, to_char(fecha, 'YYYY') from ALUMNOS;
```

Los formatos más utilizados de la máscara son:

DD: Día

MM: Mes

YYYY: Año

HH24: Hora con formato de 24 horas

MI: Minuto

SS: Segundo

TO_DATE: Lo contrario de la anterior, convertir un alfanumérico en fecha, muy interesante para comparar fechas en un formato que tu quieras.

```
select * from alumnos where fecha=to_date('01/05/2017','MM/DD/YYYY');
```

TO_NUMBER: Convertir a número, normalmente no me ha hecho falta usarlo, porque ORACLE lo hace implícitamente.

Ejercicio 2.1:

Indica cuales de estas declaraciones darían error y porque.

DECLARE

primera number:=5.0;

siguiente number not null;

fija constant varchar2(20);

cadena varchar2;

valor1 number not null:=4;

varlor2 number:=valor1/2;

valor3 number:=valor4+valor2;

valor4 number:= default 5;

cad varchar2(10);

proximo valor1%TYPE;

numero number(4,2):=150.3;

CAd varchar2(12);

cad2 varchar2(2):='HOLA';

4num number;

BEGIN

.....

ZONA CUERPO

Cuestiones básicas del lenguaje

Dentro de la zona del BEGIN END; va todo el código del programa, comentemos algunas cuestiones básicas del lenguaje, que debemos tener en cuenta:

Solo DML

En el lenguaje PL/SQL solo podemos realizar sentencias DML (Insert, update, delete y select), como veremos en el punto 2, pero no podemos realizar sentencias DDL (Create, alter, drop...) ni DCL (Creación de usuario o permisos).

Existen varias razones para este tema:

⑩• Por un lado cualquier cambio de estructura, eliminar o crear una columna de una tabla por ejemplo descompila todos los elementos que utilicen esa tabla, por lo que estaríamos haciendo que el propio código al ejecutarse estaría descompilando partes de la aplicación en tiempo de ejecución, algo desastroso para cualquier aplicación.

⑩- Por otro lado, y es muy importante que lo tengamos en cuenta a la hora de realizar nuestros scripts, es que ejecutar cualquier sentencia DDL o DCL realiza un commit automático y no tendríamos la opción de hacer rollback y volver a la situación anterior en caso de detectar problemas (más adelante hablaremos de lo que es commit y rollback, pero en pocas palabras, sería confirmar definitivamente los cambios de datos efectuados). Es por ello que NUNCA, debemos mezclar en nuestros scripts DDL con DML, siempre deben ir en archivos separados. Veremos que existe una forma de saltarnos esta regla y poder realizar DDL o DCL con PL/SQL dinámico, pero con el PL/SQL normal es imposible.

Ejemplo:

```
• BEGIN
•   Insert into EQUIPOS
      (EQUIPO, PAIS, SELECCIONADOR)
    Values
      ('ARABIA SAUDI', NULL, NULL);
•   Insert into EQUIPOS
      (EQUIPO, PAIS, SELECCIONADOR)
    Values
      ('ARGELIA', NULL, NULL);
•   CREATE TABLE TABLA (X1 VARCHAR2(10));
•   Insert into EQUIPOS
      (EQUIPO, PAIS, SELECCIONADOR)
    Values
      ('ARGENTINA', NULL, NULL);
• END;
```

Podemos comprobar con el símbolo de la exclamación, como el TOAD nos está indicando que el CREATE es un error, ya que estamos realizando un bloque PL/SQL y poniendo una sentencia DDL en su interior, lo que da error.

Pero, si quitamos el BEGIN y el END, ya no estaremos en un bloque PL/SQL y el TOAD también puede ejecutar sentencias SQL, de manera que si separamos todas por punto y coma, podemos lanzarlas a la vez

```
•   Insert into EQUIPOS
      (EQUIPO, PAIS, SELECCIONADOR)
    Values
      ('ARABIA SAUDI', NULL, NULL);
•   Insert into EQUIPOS
      (EQUIPO, PAIS, SELECCIONADOR)
    Values
      ('ARGELIA', NULL, NULL);
•   CREATE TABLE TABLA (X1 VARCHAR2(10));
•   Insert into EQUIPOS
      (EQUIPO, PAIS, SELECCIONADOR)
    Values
      ('ARGENTINA', NULL, NULL);
```

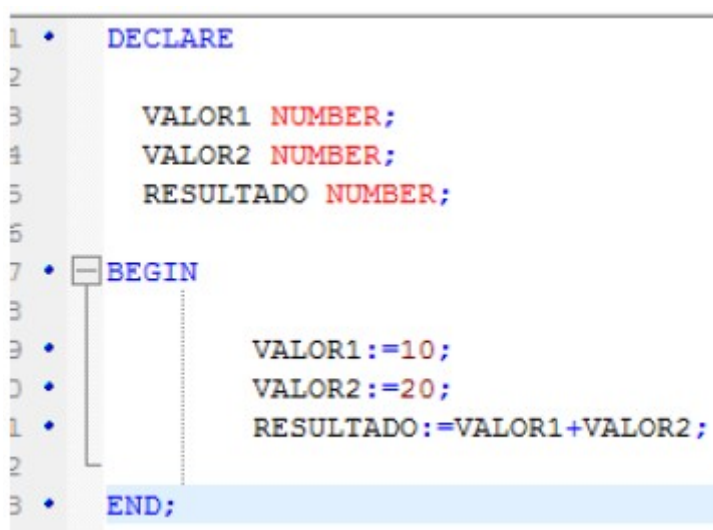
Como vemos en este caso, desaparece la exclamación y nos permitiría ejecutar, pero al llegar a la tercera sentencia (el create table) haría commit, si hubiese cualquier problema después,

por ejemplo en la INSERT de ARGENTINA y fallara el script, ya no podríamos volver a lanzarlo, aunque tiremos todo para atrás con ROLLBACK, porque habría hecho COMMIT la sentencia del create y las insert de ARABIA SAUDI y ARGELIA ya no se podría tirar par atrás ya que habrían quedado confirmadas por el COMMIT implícito del create. Nos tocaría borrarlas del script y empezar a partir de Argentina, podéis imaginar que esto en grandes scritps de miles de líneas podría resultar dramático. Por lo tanto, exactamente como habéis visto que os he pasado los scripts de la base de datos del Mundial, SIEMPRE debemos tener separados los scripts DML y DDL.

Operador punto y coma

Como en mucho lenguajes, en PL/SQL debemos usar el operador punto y coma (;) para indicar que finalizamos cada sentencia o nos dará error de compilación.

Existen algunas excepciones, como puede ser DECLARE, BEGIN o EXCEPTION que no se pone el punto y coma.



```
1 • DECLARE
2
3     VALOR1 NUMBER;
4     VALOR2 NUMBER;
5     RESULTADO NUMBER;
6
7 • BEGIN
8
9     VALOR1:=10;
10    VALOR2:=20;
11    RESULTADO:=VALOR1+VALOR2;
12
13 • END;
```

Uso del Package dbms_output

ORACLE no dispone de una función nativa de entrada y salida, debemos tener en cuenta que los procedimientos almacenados de ORACLE tienen la función de devolver sus resultados a la aplicación que tiene por encima, de modo que para trabajar en la propia base de datos y ver los resultados, para depurar o ir probando nuestros procedimientos, en la propia instalación del sistema gestor, se encuentran una serie de paquetes ya realizados para diversas tareas, durante el curso utilizaremos algunos de ellos. Uno de los más interesantes es el que nos permite imprimir sobre la salida estándar, el tema de los paquetes lo veremos en el último punto 4, pero utilizarlo es sencillo.

El paquete es cuestión es DBMS_OUTPUT, y dentro del paquete usaremos el procedimiento PUT_LINE.

Por lo tanto para imprimir un HOLA MUNDO, deberemos hacer:

```
DBMS_OUTPUT.PUT_LINE('HOLA MUNDO');
```

¿Y donde podemos ver el resultado?, en la pestaña de dbms_output. Previamente deberemos activarla (poniendo el semáforo rojo en verde) y poner el Frequency a 1 segundo para que refresque rápidamente.

```
▶ BEGIN
• dbms_output.put_line('HOLA MUNDO');
• END;
```

Output

Messages | Data Grid | Trace | DBMS Output | Query

Frequency: 1

HOLA MUNDO

Con la función put_line siempre hay un retorno de carro, por lo tanto si hacemos dos DBMS_OUTPUT.PUT_LINE('HOLA MUNDO'); pondrá uno debajo de otro:
Si lo que queremos es que no haga este retorno de carro, debemos usar la función put nada más, pero en este caso debemos tener una cosa en cuenta, debemos poner obligatoriamente un put_line al final para que haga el retorno de carro ya que hasta que no lo pongamos no se imprimirá nada.

```
▶ begin
• DBMS_OUTPUT.PUT('HOLA ');
• DBMS_OUTPUT.PUT('HOLA ');
• DBMS_OUTPUT.PUT('HOLA ');
• DBMS_OUTPUT.PUT_LINE('');
• end;
```

Output

Messages | Data Grid | Trace | DBMS Output | [

Frequency: 1

HOLA HOLA HOLA