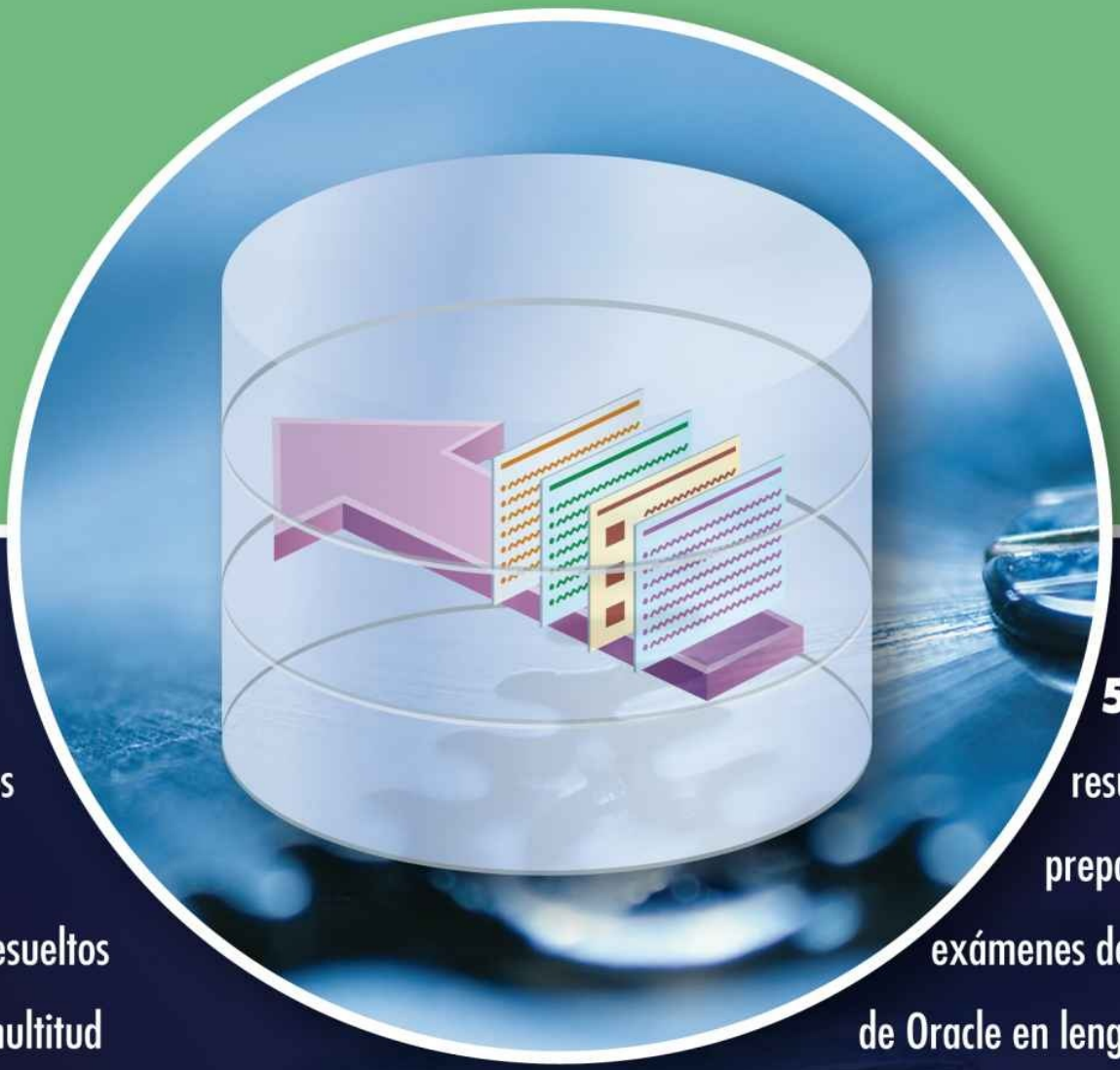
Oracle 11g PL/SQL

Curso práctico de formación



INCLUYE:

15 supuestos prácticos totalmente resueltos además de multitud de ejemplos

55 cuestiones resueltas para la preparación de los exámenes de certificación de Oracle en lenguaje PL/SQL: 1Z0-144; 1Z0-146 y 1Z0-147

ANTOLÍN MUÑOZ CHAPARRO

libros RC

Oracle 11g PL/SQL. Curso práctico de formación
Antolín Muñoz Chaparro

ISBN: 978-84-939450-1-5

EAN: 9788493945015

RC Libros
Calle Mar Mediterráneo, 2. Nave 6
28830 SAN FERNANDO DE HENARES, Madrid
Teléfono: +34 91 677 57 22
Fax: +34 91 677 57 22
Correo electrónico: info@rclibros.es
Internet: www.rclibros.es

Diseño de colección, cubierta y pre-impresión: Grupo RC
Impresión y encuadernación: Service Point
Depósito Legal: M-5313-2012
Impreso en España

16 15 14 13 12 (1 2 3 4 5 6 7 8 9 10 11 12)

Prólogo

Hace más de dos décadas tuve el honor de enseñarle por primera vez a Antolín Muñoz la base de datos Oracle. En ese momento no era consciente de la relación tan fructífera que ambos iban a mantener a lo largo de estos años.

Estas dos décadas prodigiosas para los que hemos tenido la suerte de trabajar en el mundo de la informática, o como se dice ahora: Tecnologías de la Información y de las Comunicaciones, han representado una evolución vertiginosa de todos los aspectos relacionados con esta ciencia. Destacamos en el mundo del hardware y sobre todo en los mini y los mainframe; cómo han ido desapareciendo los fabricantes en unos casos, y en otros cómo han evolucionado hacia la fabricación de dispositivos generalmente periféricos y ordenadores personales. Otro aspecto a destacar es la mejora constante de las comunicaciones en la topología de estrella; desde el ordenador central, a las redes en bus Ethernet y distribuidas mediante Router y Switch, la apertura hacia el mundo exterior; desde las líneas punto a punto hasta las redes que facilitaron el acceso al correo e Internet y la mejora de las velocidades y los anchos de banda. Por último, resulta casi imposible hablar de la microinformática porque pasar de los terminales VT orientados al carácter a los actuales ordenadores personales multimedia, se convertiría en una interminable relación fuera de contexto. Igual nos pasaría con los múltiples desarrollos que han popularizado los PC (Personal Computers), en el campo profesional, de ocio y entretenimiento.

Mientras tanto: ¿Qué hacían Antolín y el gigante Oracle? Oracle lo conocimos en su versión 6 de base de datos, que no solo disponía de un SGBD (Sistema Gestor de Base de Datos), sino que como es lógico, estaba acompañado de algunas herramientas y de un lenguaje

procedimental conocido como PL/SQL.

En 1992 apareció la versión 7 de Oracle donde lo más significativo respecto a la versión anterior era el almacenamiento y ejecución de programas escritos en PL/SQL dentro del SGBD, así como el soporte de la integridad referencial.

Internet empezaba a dar los primeros indicios de su existencia y a convertirse en la excelente realidad actual y en 1999 sale a la luz la versión 8i de Oracle. La “i” es un claro indicativo de que cumple los requerimientos de Internet, permitiendo el almacenamiento y ejecución de contenidos multimedia, y el SGBD incorpora la ejecución y almacenamiento de código Java, al incorporar la máquina virtual de dicho lenguaje. No hace falta decir que se habían acabado los desarrollos orientados a carácter y comenzaban los desarrollos orientados a objeto.

A partir de esta versión pasamos a utilizar una herramienta que nos permitió trabajar en cliente/servidor: Oracle Developer, que es un entorno gráfico para el diseño de aplicaciones, y que nos facilita mucho la creación de formularios, su compilación y ejecución. Es una herramienta bastante intuitiva, aunque presenta como principal desventaja el que el código fuente compilado hay que tenerlo en una carpeta compartida a los demás usuarios, y esto puede provocar la pérdida de las distintas versiones si no se es muy cuidadoso. Pero no debemos olvidarnos de que hablamos de una herramienta de diseño de formularios, y el desarrollo siempre tiene que venir acompañado de un lenguaje, que desde el comienzo siempre ha sido PL/SQL, aunque actualmente existe una herramienta paralela: Oracle JDeveloper que admite el desarrollo en lenguaje Java.

Todas estas versiones de la B.D. Oracle implicaron una actualización de conocimientos para llevar a la práctica con gran éxito: migraciones de las distintas versiones, migraciones de hardware, migraciones de aplicaciones orientadas a carácter a aplicaciones orientadas a objeto, el “efecto 2000”, la llegada del euro,

etc. Todas ellas realizadas por un grupo reducido de gente en las que Antolín fue en todas y cada una de ellas el verdadero gestor ejecutivo, que junto con una cuidada planificación permitió que tan complicados objetivos se realizaran en tiempo récord.

A la vez, Antolín encontraba tiempo para hacer incursiones en el campo de la docencia relacionada con los distintos entornos de estas tecnologías, y adquirir unos conocimientos pedagógicos que le han permitido escribir estos manuales formativos.

El lector tiene ante sí un libro en el que se aúnan 20 años de experiencia con el lenguaje PL/SQL con las últimas adaptaciones a la versión 11g de Oracle, la experiencia pedagógica en la formación de productos de esta empresa, la experiencia en importantes desarrollos de aplicaciones, y sobre todo la seguridad de que esta obra está planificada, desarrollada y ejecutada con toda la precisión que Antolín pone en todos sus trabajos.

Eduardo Solans

Los archivos para la resolución de los supuestos se encuentran disponibles en la página individual del libro, accediendo a la web: www.rclibros.es

Capítulo 1. FUNDAMENTOS DEL LENGUAJE PL/SQL

INTRODUCCIÓN

PL/SQL es un sofisticado lenguaje de programación que se utiliza para acceder a bases de datos Oracle desde distintos entornos. PL/SQL está integrado con el servidor de base de datos, de modo que el código puede ser procesado de forma rápida y eficiente. También se encuentra disponible en varias de las herramientas de cliente que posee Oracle, entre ellas SQL*PLUS, Developer Suite 10g, JDeveloper, etc.

Si nos preguntamos por qué utilizar PL/SQL, la conclusión la encontramos en el propio SQL. Tenemos que recordar que Oracle es una base de datos relacional, que utiliza como lenguaje de datos el propio SQL. Este es un lenguaje flexible y eficiente, con características muy potentes para la manipulación y examen de los datos relacionales, pero que presenta deficiencias a la hora de realizar programaciones procedimentales.

SQL es un lenguaje de cuarta generación (4GL), que como el resto de lenguajes de esta generación, presenta como característica el hecho de que describen lo que debe hacerse, pero no la manera de llevarlo a cabo. Por ejemplo, si analizamos la siguiente instrucción:

```
DELETE FROM estudiantes WHERE nombre like 'Pep%'
```

Esta instrucción determina que queremos borrar de la tabla estudiantes todos aquellos cuyo nombre comience por "Pep", pero no dice cómo va a realizar el gestor de base de datos el proceso para conseguir eliminar dichos registros. Parece presumible que recorrerá los datos de dicha tabla en un cierto orden para determinar qué elementos debe borrar y luego los eliminará; no obstante, es algo que no nos interesa para la instrucción.

En contraposición a los lenguajes 4GL nos encontramos con los lenguajes de tercera generación (3GL), como C y Visual Basic. Son lenguajes más procedimentales donde se implementan algoritmos para resolver unos problemas. Estas estructuras procedimentales y de ejecución paso a paso no se pueden implementar en SQL, así que Oracle necesitaba de un lenguaje que pudiese resolver este tipo de problemas y que estuviera más enfocado no solo al manejo de datos sino a la resolución de problemáticas de todo tipo, así que creó

el lenguaje PL/SQL (Lenguaje Procedimiental / SQL). Este lenguaje no es solo un lenguaje de tipo 3GL, sino que permite utilizar la flexibilidad de SQL como lenguaje de 4GL.

Esta característica que le define, es posible dado que es un lenguaje particular del sistema gestor de bases de datos Oracle y no un lenguaje estándar.

Por tanto, el lenguaje PL/SQL potencia el lenguaje SQL agregando estructuras y objetos del siguiente tipo:

- El bloque.
- Manejo de errores y excepciones.
- Creación de procedimientos y funciones.
- Definición de variables y tipos.
- Estructuras de bucle.
- Cursores.
- Objetos.

El bloque

Es la unidad básica de todo programa en PL/SQL. Todo programa al menos debe poseer un bloque.

Todo bloque consta de una sección declarativa optativa, una sección de ejecución obligatoria y una sección de control de errores optativa.

La sintaxis es la siguiente:

[DECLARE]

<sección declarativa>

BEGIN

<sección de ejecución>

[EXCEPTION]

<sección de control de errores>

END;

SECCIÓN DECLARATIVA

En esta sección se definen las variables, constantes y cursores que se van a utilizar dentro de la sección de ejecución.

SECCIÓN DE EJECUCIÓN

La sección de ejecución presenta las siguientes particularidades:

- Toda instrucción SELECT (no dinámica) que aparezca en esta sección deberá llevar incorporado el parámetro INTO, como se muestra en el siguiente ejemplo:

  SELECT columnas INTO variables FROM tablas WHERE criterios.

- En esta sección solo se admiten instrucciones de SQL del tipo DML (select, insert, update y delete) o instrucciones SQL dinámicas, el resto no están permitidas implementarlas directamente (por ejemplo: alter, create, drop, etc.), salvo que se indique dentro de la instrucción EXECUTE INMEDIATE.

SECCIÓN de control de errores

En esta sección se definen los controles programados para detectar los errores de ejecución del bloque y la solución adoptada para ello.

Tipos de bloque

Se diferencian 3 tipos de bloques:

- Los *bloques anónimos* se construyen de manera dinámica y se ejecutan una sola vez.
- Los *bloques nominados* se construyen identificándolos con un nombre. Al igual que los anteriores, se construyen de forma dinámica y se ejecutan una sola vez.
- Los *subprogramas* son bloques nominados que se almacenan en la base de datos. Nos podemos encontrar con procedimientos, paquetes y funciones de este tipo. Siempre se ejecutan bajo demanda.
- Los *disparadores* son también bloques nominados que se almacenan en la base de datos, pero que no se pueden ejecutar bajo petición de un programa. Se ejecutan cuando tiene efecto el suceso para el que se han programado contra una cierta tabla del sistema.

EJEMPLO DE BLOQUE ANÓNIMO

```
DECLARE
Var1 NUMBER;
BEGIN
Var1 := 1;
END;
```

EJEMPLO DE BLOQUE NOMINADO

```
<<Nombre_Bloque>>
DECLARE
Var1 NUMBER;
BEGIN
Var1 := 1;
END;
```

EJEMPLO DE BLOQUE SUBPROGRAMA

```
CREATE OR REPLACE PROCEDURE MI_PROGRAMA IS
```

```
Var1 NUMBER;
BEGIN
Var1 := 1;
END;
```

EJEMPLO DE disparador

```
CREATE OR REPLACE TRIGGER MI_DISPARADOR IS
BEFORE INSERT OR UPDATE OF numero ON tabla_temporal
FOR EACH ROW
BEGIN
IF :new.numero < 0 THEN
RAISE_APPLICATION_ERROR(-20100,'¡¡¡¡Error!!!');
END;
```

Manejo de errores y excepciones

El lenguaje PL/SQL, como muchos otros procedimentales, permite un control sobre los errores que se produzcan en la ejecución del código. Esta gestión de errores presenta como ventaja la claridad para su manipulación, dado que utiliza una sección independiente a la del código ejecutable.

Creación de procedimientos y funciones

El lenguaje PL/SQL permite la creación de procedimientos almacenados y de funciones que nos devuelvan un valor como resultado de su ejecución.

Definición de variables y tipos

El lenguaje PL/SQL también permite la definición de variables para utilizar en nuestros programas y crear tipos de usuarios a partir de otros predefinidos.

Estructuras de bucle

Para desarrollar nuestros programas y poder realizar operaciones de bifurcación, el lenguaje PL/SQL posee estructuras de control.

Cursores

Este tipo de estructura que se define en la sección declarativa de un bloque nos permite recuperar en memoria un conjunto de filas de una tabla que se recorren una a una para un tratamiento posterior.

Objetos

Oracle, dada la tendencia actual de los lenguajes de programación que hay en el mercado, incorpora también la figura del objeto, de forma que PL/SQL se puede convertir también en un lenguaje orientado a objetos.

Salida por pantalla de los resultados de una ejecución

PL/SQL a través de las herramientas propias de Oracle: SQL*Plus y SQL*Plus Worksheet únicamente visualiza el resultado satisfactorio o no de la ejecución de las instrucciones. Para poder realizar una visualización en pantalla de la ejecución y resultados internos del código PL/SQL, hay que utilizar la invocación de un paquete incluido en Oracle PL/SQL denominado DBMS_OUTPUT, el cual permite dirigir a pantalla resultados mediante el uso de la función PUT_LINE.

No obstante, para que se haga completamente efectiva dicha salida a pantalla, antes hay que activar el comando de ejecución en pantalla del paquete DBMS_OUTPUT, siempre que se inicie una nueva sesión con la herramienta correspondiente. Este comando es el siguiente:

```
SET SERVEROUTPUT ON;
```

EJEMPLO DE un bloque con salida a pantalla de los resultados

```
SET SERVEROUTPUT ON;
DECLARE
Var1 VARCHAR2(50);
BEGIN
Var1 := 'Antolin';
DBMS_OUTPUT.PUT_LINE('El nombre del autor es:'||
Var1);
END;
/
```

unidades léxicas

Es el conjunto de caracteres y gramática a utilizar en la programación de PL/SQL. Contamos con los siguientes elementos:

- Identificadores.
- Palabras reservadas.
- Delimitadores.
- Literales.
- Comentarios.

Identificadores

Dan nombre a los distintos objetos de nuestro programa. Por ejemplo, los identificadores de variable, cursores, tipos, etc.

Un identificador tiene que comenzar obligatoriamente con una letra seguida por una secuencia de caracteres, entre los que se pueden incluir:

- Letras.
- Números.
- El símbolo $.
- El carácter de subrayado _.
- El símbolo #.

La longitud máxima de un identificador es de 30 caracteres.

También hay que tener en cuenta que el lenguaje PL/SQL no es un lenguaje sensible en cuanto a los caracteres, de manera que no distingue mayúsculas o minúsculas, salvo que el nombre vaya encerrado entre comillas dobles (").

Cada objeto tiene un espacio de nombres.

Los espacios de nombres de objeto aglutinan una serie de nombres de objeto, en algunos casos de forma independiente, y en otros común para distintos nombres de objetos diferentes.

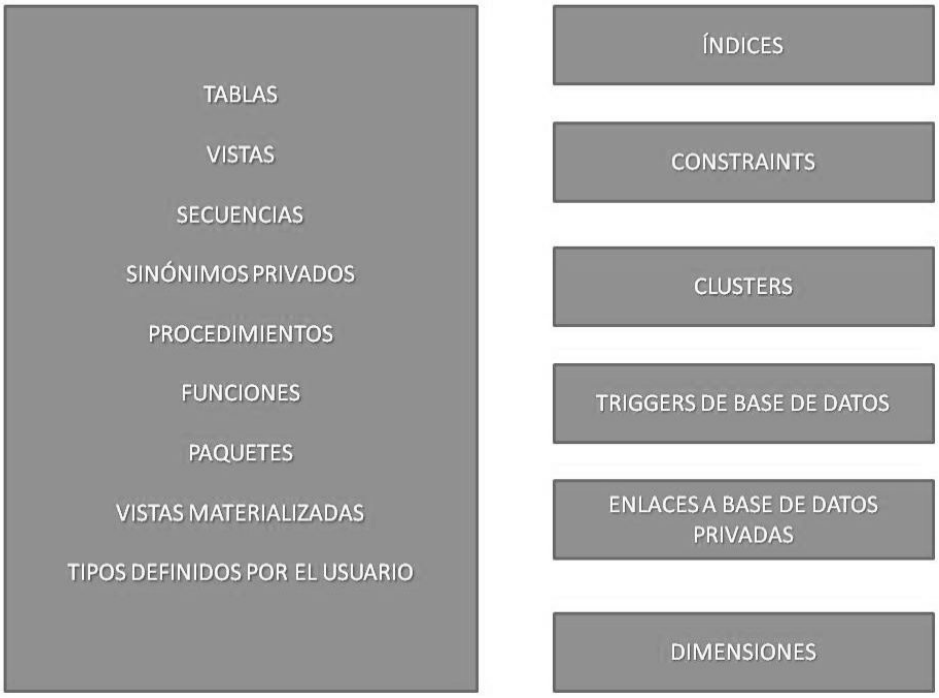


Fig. 1-1 Namespaces (Esquemas de nombre) de objetos de un esquema.

Dentro del mismo espacio de nombres no se puede repetir un nombre de objeto aunque sea de distinto tipo.

Por ejemplo, una tabla y una vista no se pueden denominar con el mismo nombre, porque comparten el mismo esquema de nombres. En cambio, una restricción y un trigger sí se pueden llamar igual, porque cada uno de estos objetos del esquema de base de datos posee esquemas independientes.

Fig. 1-2 Namespaces de objetos que no pertenecen a un esquema.

Ejemplos de nombres de esquema válidos son los siguientes:

EMP
"Emp"
SCOTT.FECHAALTA
"INCLUSO ESTO & VALE!"
UN_NOMBRE_LARGO_Y_VALIDO

Palabras reservadas

Son todas aquellas que define Oracle como restringidas dentro del lenguaje PL/SQL y cuyo nombre no podrá llevar ningún otro objeto de la base de datos.

Delimitadores

Son símbolos con un significado especial dentro de PL/SQL, tal y como se especifica en la siguiente tabla:

Símbolo	Descripción	Símbolo	Descripción
+	Operador de suma	-	Operador de resta
*	Operador de multiplicación	/	Operador de división
=	Operador de igualdad	<	Operador menor que
>	Operador mayor que	(	Delimitador inicial de una expresión
)	Delimitador final de una expresión	;	Fin de una orden
%	Indicador de atributo	,	Separador de elementos
.	Selector de componente	@	Indicador de enlace a base de datos
'	Delimitador de cadena de caracteres	"	Delimitador de cadena entrecomillada
:	Indicador de variable de asignación	**	Operador de exponenciación
<>	Operador distinto de	!=	Operador distinto de
~=	Operador distinto de	^=	Operador distinto de
<=	Operador menor o igual que	>=	Operador mayor o igual que
:=	Operador de asignación	=>	Operador de asociación
..	Operador de rango	||	Operador de concatenación
<<	Delimitador de comienzo de etiqueta	>>	Delimitador fin de etiqueta
--	Indicador de comentario en una línea	/*	Comienzo de comentario multilineal
*/	Fin de comentario multilineal	<space>	Espacio
<tab>	Carácter de tabulación	<cr>	Retorno de carro

Literales

Los literales pueden ser de los siguientes tipos:

- Booleanos: TRUE, FALSE, NULL.
- Numéricos: 123, -7, +12, 0
- Carácter: '123', 'hola'
- De fecha: '1998-12-25' (formato solo fecha)
'1997-10-22 13:01:01' (formato fecha y hora)
'1997-01-31 09:26:56.66 +02:00' (formato fecha y hora)

Comentarios

Si se quieren incluir comentarios en una sola línea, se tiene que escribir la siguiente sintaxis:

```
-- texto
```

Si el texto a incluir ocupa más de una línea, se tiene que escribir la siguiente sintaxis:

```
/* texto
Prueba en varias líneas */
```

Tipos de datos

Los tipos de datos que se pueden utilizar en PL/SQL se dividen en las siguientes categorías:

- Tipos escalares.
- Tipos compuestos.
- Tipos puntero.
- Tipos lob.

Tipos escalares

Los tipos escalares se dividen en estas categorías:

- Numéricos.
- De caracteres.
- Booleanos.
- De fecha y hora.
- Raw.
- Rowid.
- Urowid.

A continuación, se muestra una lista con cada uno de los tipos que se pueden utilizar en cada categoría y una breve descripción de ellos.

Tipos numéricos

Tipo	Descripción
BINARY_DOUBLE	Tipo de dato numérico que almacena números en coma flotante de 64 bits.
BINARY_FLOAT	Tipo de dato numérico que almacena números en coma flotante de 32 bits.
DEC	Tipo de dato ANSI equivalente al tipo NUMBER.
DECIMAL	Tipo de dato ANSI equivalente al tipo NUMBER.
DOUBLE PRECISION	Tipo de dato ANSI con una precisión de 126 en binario.
FLOAT	Es un subtipo del tipo NUMBER con una precisión en el rango de 1 a 126 dígitos binarios.
INT	Tipo de dato ANSI que equivale al tipo NUMBER(38).
INTEGER	Tipo de dato ANSI que equivale al tipo NUMBER(38).
NUMBER	Tipo de dato numérico que admite una precisión de 1 a 38 y una escala de -84 a 127.
NUMERIC	Tipo de dato ANSI equivalente al tipo NUMBER.
PLS_INTEGER	Tipo de dato numérico que almacena enteros con signo en un rango de -2.147.483.648 y 2.147.483.647.
REAL	Tipo de dato ANSI con una precisión de 63 en binario o 18 en decimal.
SMALLINT	Tipo de dato ANSI que equivale al tipo NUMBER(38).

Tipos carácter

Tipo	Descripción
CHAR	Admite un string (conjunto de caracteres) de longitud fija. El tamaño máximo admitido es de 2.000 bytes o caracteres y el mínimo es de 1 byte. En PL/SQL admite strings con un rango de 1 a 32.676 bytes.
CHARACTER	Tipo de dato ANSI equivalente al tipo CHAR.
LONG	Admite un string de longitud variable de hasta 2 gigabytes en SQL. En PL/SQL solo admite strings con un rango de 1 a 32.670 bytes.
LONG VARCHAR	Tipo de dato DB2 equivalente al tipo LONG.
NATIONAL CHARACTER	Tipo de dato ANSI equivalente al tipo NCHAR.
NCHAR	Igual que el tipo CHAR pero codificado en el juego de caracteres nacional que se haya definido.
NATIONAL CHAR	Tipo de dato ANSI equivalente al tipo NCHAR.
NVARCHAR2	Igual que el tipo VARCHAR2 pero codificado en el juego de caracteres nacional que se haya definido.
VARCHAR	Tipo de dato ANSI equivalente al tipo VARCHAR2.
VARCHAR2	Admite un string (conjunto de caracteres) de longitud variable con un tamaño máximo de 4000 bytes o caracteres y un tamaño mínimo de 1 byte en SQL. En PL/SQL admite strings con un rango de 1 a 32.676 bytes.

Tipos booleanos

Tipo	Descripción
BOOLEAN	Tipo de dato condicional que solo admite los valores TRUE, FALSE o NULL.

Tipos de fecha y hora

Tipo	Descripción
DATE	Tipo de dato de fecha y hora que almacena los valores con un tamaño máximo de 7 bytes.
TIMESTAMP	Tipo de datos de fecha y hora con una precisión de 0 a 9 dígitos.
TIMESTAMP WITH TIME ZONE	Igual que tipo TIMESTAMP pero almacenando también los valores de fecha y hora correspondientes a la zona horaria de la base de datos.
TIMESTAMP WITH LOCAL TIME ZONE	Igual que tipo TIMESTAMP pero almacenando también los valores de fecha y hora correspondientes a la zona horaria del servidor.
INTERVAL YEAR TO MONTH	Tipo de dato que almacenará un período de tiempo en años y meses.
INTERVAL DAY TO SECOND	Tipo de dato que almacenará un período en días horas, minutos y segundos.

Tipos Raw

Tipo	Descripción
RAW	Tipo de datos binario con un tamaño máximo de 2.000 bytes.

LONG RAW	Tipo de datos binario con un tamaño máximo de 2 gigabytes.

Tipos Rowid

Tipo	Descripción
ROWID	Tipo de dato que almacena la dirección lógica de ubicación del registro en la tabla. Se representa en formato carácter con base 64.

Tipos URowid

Tipo	Descripción
UROWID	Equivalente al tipo ROWID pero para un tipo de tabla INDEX-ORGANIZED.

Tipos compuestos

Los tipos compuestos se componen a partir de uno o varios de los tipos escalares anteriores, y se distinguen los siguientes tipos:

- RECORD.
- TABLE.
- VARRAY.

En el capítulo 4 se explicarán, de una manera pormenorizada, cada uno de estos tipos.

Tipos punteros

Los tipos punteros se utilizan para referenciar a cursores en memoria o tipos objeto, y se distinguen estos tipos:

- REF CURSOR.
- REF tipo objeto.

Tipos LOB

Permiten referenciar a ficheros de gran tamaño almacenados externamente a la base de datos, pero referenciados desde la misma, y se distinguen los siguientes tipos:

- BFILE: enlaza ficheros binarios de hasta un máximo de 4 GB.
- BLOB: enlaza ficheros binario de hasta (4 GB – 1) * (bloque b.d.).
- CLOB: enlaza ficheros de tipo carácter de hasta (4 GB -1) * (bloque b.d.).
- NLOB: es un subtipo del tipo CLOB, para almacenar ficheros en el juego de caracteres nacional definido.

DECLARACIÓN DE VARIABLES

La declaración de variables se realiza dentro de la sección DECLARE de un bloque y su sintaxis es la siguiente:

<nombre_variable> tipo [CONSTANT tipo | DEFAULT]
[NOT NULL] [:= valor]

A continuación, se muestra una serie de ejemplos con los distintos tipos de definición de variables:

```
DECLARE
    Var1 NUMBER(5);
    Var2 NUMBER := 10;
    Var3 NUMBER(5) NOT NULL := 0;
    Var4 CONSTANT VARCHAR2(10) := ‘Hola’
    Var5 NUMBER DEFAULT 45;
    Intervalo1 INTERVAL YEAR(3) TO MONTH;
    Intervalo2 INTERVAL DAY(3) TO SECOND(3);
    Var6 TIMESTAMP;
    BEGIN
    NULL;
    END;
```

Por defecto, una variable que no se ha inicializado en la sección DECLARE a un valor concreto, tendrá valor NULL al comenzar la sección ejecutable (BEGIN).

Asignación de valores a variables

La sintaxis para la asignación de un valor a las variables es la siguiente:

<variable> := <valor>;

Tipos utilizados con variables

Aparte de los tipos estándar definidos en ORACLE, y que se han definido con anterioridad, podemos utilizar dos más:

- %TYPE.
- %ROWTYPE.

%TYPE.

Permite declarar una variable que asume el tamaño y tipo de la columna de la tabla referenciada.

A continuación, se muestra un ejemplo de uso de este tipo:

```
DECLARE
    Var_emple_nombre empleados.nombre%TYPE;
```

La variable *var_emple_nombre* asume el tamaño y tipo de la columna *nombre* de la tabla *empleados*.

%ROWTYPE.

Permite declarar una variable que asume el conjunto de columnas (en el mismo orden) y los tipos de la tabla referenciada.

A continuación, se muestra un ejemplo de uso de este tipo:

```
DECLARE
    Var_emple_nombre empleados%ROWTYPE;
```

La variable *var_emple_nombre* se convierte en un tipo compuesto (RECORD) que asume el conjunto de columnas y tipos de la tabla *empleados*.

Subtipos

Los subtipos son definiciones de tipos basados en otro tipo predefinido.

A continuación, se muestra un ejemplo de uso de este tipo:

```
DECLARE
    SUBTYPE contador IS NUMBER(4);
    Var_conta contador;
```

Hemos definido en el ejemplo un subtipo *contador* que es de tipo numérico con tamaño de 4.

Petición de valores por pantalla

En PL/SQL puede resultar de utilidad la petición de valores por pantalla, siempre y cuando se utilicen en bloques anónimos que se ejecutan en el momento. No es conveniente en bloques nominados que se almacenan en la base de datos, dado que la ejecución de los mismos quedaría parada a expensas de introducir un valor.

La sintaxis para la petición de un valor por pantalla asociada a una variable numérica es la siguiente:

```
<variable> := &texto_a_mostrar;
```

La sintaxis para la petición de un valor por pantalla asociada a una variable de texto es la siguiente:

```
<variable> := '&texto_a_mostrar';
```

EJEMPLO

```
DECLARE
Var1 NUMBER(3) := &Indique_la_edad;
Var2 VARCHAR2(100) := '&Indique_nombre_y_apellidos';
BEGIN
DBMS_OUTPUT.PUT_LINE('Usted se llama: '||VAR2);
DBMS_OUTPUT.PUT_LINE('Y su edad es: '||
                              TRIM(TO_CHAR(VAR1)));
END;
```

SUPUESTO PRÁCTICO 0: ***Resolución en el Anexo I de este manual.***

Si usted no ha realizado previamente el "Curso de SQL para Oracle 10g" de esta misma editorial, tendrá que cargar la estructura de base de datos de un HOSPITAL que se utilizará como ejemplo durante este curso. Para ello, deberá realizar las siguientes operaciones:

- Ejecutar en SQL*PLUS el script *script_bdhospital.sql* incluido en el Anexo III de este libro. También puede encontrarlo en la web del autor o de la editorial.
- Una vez ejecutado este script se habrá cargado toda la configuración del esquema HOSPITAL (tablas, secuencias, datos…) en el usuario PEPERF con contraseña PEPITO.
.
- Conéctese con el usuario PEPERF y consulte la información de los objetos creados para este usuario dentro de la tabla USER_OBJECTS, cuyo nombre no empiece por 'BIN'.

SUPUESTO PRÁCTICO 1: ***Resolución en el Anexo I de este manual.***

Crear un bloque sin nominar en SQL*PLUS que realice las siguientes operaciones:

- Solicitar el siguiente literal por pantalla: *nombre_de_enfermo,* y almacenarlo en una variable del mismo tipo y tamaño que el campo que almacena el nombre en la tabla enfermo.
- Solicitar el siguiente literal por pantalla: *apellidos_del_mismo,* y almacenarlo en una variable del mismo tipo y tamaño que el campo que almacena el apellido en la tabla enfermo.
- Solicitar el siguiente literal por pantalla: *dirección_donde_reside*, y almacenarlo en una variable del mismo tipo y tamaño que el campo que almacena la dirección en la tabla enfermo.
- Una vez introducidos todos estos datos, se deberá ejecutar el siguiente código:

```
DBMS_OUTPUT.PUT_LINE('DATOS DEL ENFERMO');
DBMS_OUTPUT.PUT_LINE ('-----------------'||
                      CHR(10));
```

- Mostrar en una primera línea el nombre del enfermo introducido por pantalla seguido de una coma (,) y el apellido.
- Mostrar en una segunda línea la dirección del enfermo introducida por pantalla.

```
/* Activación en SQL*PLUS de la variable de
entorno que posibilita ver       resultados por
pantalla mediante el paquete DBMS_OUTPUT de
PL/SQL */

SET SERVEROUTPUT ON
```

Capítulo 2. ESTRUCTURAS DE CONTROL

INTRODUCCIÓN

Las estructuras de control permiten controlar el comportamiento del bloque a medida que este se ejecuta. Estas estructuras incluyen las órdenes condicionales y los bucles.

Estas estructuras, junto con las variables, son las que dotan al lenguaje PL/SQL de su poder y flexibilidad.

Tenemos las siguientes estructuras de control:

- IF…THEN…ELSE…END IF
- CASE…WHEN…END CASE
- LOOP...END LOOP
- WHILE...LOOP...END LOOP
- FOR...LOOP...END LOOP
- GOTO
- NULL

if…then…else…end if

La sintaxis de esta estructura de control es la siguiente:

```
IF <expresión booleana1> THEN
<operaciones1>;
[ELSIF <expresión booleana2> THEN
<operaciones2>;
...
[ELSE
<operaciones3>;]
END IF;
```

Permite evaluar expresiones booleanas y admite anidamiento.

Ejemplo

```
IF var1 < 50 THEN
        <operaciones1>
ELSIF var1 > 70 THEN
        <operaciones2>
ELSE
        <operaciones3>
END IF;
```

En este ejemplo si la variable *var1* tiene un valor inferior a 50, ejecutará el conjunto de operaciones denominado *operaciones1*. Si por el contrario, el valor es superior a 70, entonces ejecutará el conjunto de operaciones denominado *operaciones2*. En caso de que no se cumplan ninguna de las 2 condiciones anteriores, ejecutará el conjunto de operaciones denominado *operaciones3*.

Esta forma de plantear la estructura de control sería equivalente al siguiente formato:

```
IF var1 < 50 THEN
        <operaciones1>
ELSE
        IF var1 > 70 THEN
                <operaciones2>
        ELSE
                <operaciones3>
        END IF;
END IF;
```

CASE

Permite evaluar múltiples opciones booleanas equivalentes en cuanto al tipo de dato. Es una estructura que permite el anidamiento.

La sintaxis de esta estructura es la siguiente:

CASE <variable a comprobar>
WHEN <valor de comparación 1> THEN <operaciones1>;
WHEN <valor de comparación 2> THEN <operaciones2>;
…
[ELSE <operaciones si no se cumplen las otras>];
END CASE;

Ejemplo (Uso de la opción ELSE del CASE)

```
DECLARE
grado CHAR(1);
BEGIN
grado := 'B';
CASE grado
        WHEN 'A' THEN DBMS_OUTPUT.PUT_LINE('Excelente');
        WHEN 'B' THEN DBMS_OUTPUT.PUT_LINE('Bueno');
WHEN 'C' THEN DBMS_OUTPUT.PUT_LINE('Regular');
ELSE DBMS_OUTPUT.PUT_LINE('Malo');
END CASE;
END;
/
```

Aunque la cláusula ELSE es opcional para esta estructura, PL/SQL añade internamente una condición ELSE cuando no se especifica implícitamente en el código. Esta condición interna sería la siguiente:

```
ELSE RAISE CASE_NOT_FOUND;
```

Esto significa que en una sentencia CASE que no se especifique ELSE, el SGBD (Sistema Gestor de Base de Datos) cuando ejecuta el código, si no encuentra ninguna condición WHEN que satisfaga el valor de la variable, ejecutará ELSE RAISE CASE_NOT_FOUND, lo que conlleva que el programa salga del bucle CASE y ejecute la excepción (RAISE) "CASE_NOT_FOUND"; por tanto, cuando se diseñe un código con CASE en el que no se quiera incluir el ELSE hay que tener en cuenta esta excepción en el control de errores del programa.

Ejemplo (Uso del CASE sin ELSE y con control errores)

```
DECLARE
grado CHAR(1);
BEGIN
grado := 'B';
CASE grado
WHEN 'A' THEN DBMS_OUTPUT.PUT_LINE('Excelente');
WHEN 'B' THEN DBMS_OUTPUT.PUT_LINE('Bueno');
END CASE;
EXCEPTION
WHEN CASE_NOT_FOUND THEN
DMBS_OUTPUT.PUT_LINE('Condición fuera del CASE');
END;
/
```

LOOP…END LOOP

Bucle repetitivo de ejecución indefinida con terminación mediante la evaluación de una condición, o mediante salida directa (EXIT).

La sintaxis de esta estructura es la siguiente:

LOOP
<secuencia de órdenes>
[EXIT [WHEN condicion]];
END LOOP;

Ejemplo

```
LOOP
INSERT INTO expediente
...
IF var1 > 50 THEN
EXIT;
END IF;
END LOOP;
```

Este bucle ejecutará una inserción en la tabla *expediente* hasta que el valor de la variable *var1* sea superior a 50, en cuyo caso ejecutará la instrucción EXIT para salir del bucle.

Otra forma similar de haber implementado el código anterior es la siguiente:

```
LOOP
INSERT INTO expediente
...
EXIT WHEN var1 > 50;
END IF;
END LOOP;
```

while…loop…end loop

Bucle repetitivo donde la evaluación de la condición se produce antes de ejecutarse cada iteración del código a ejecutarse.

La sintaxis de esta estructura es la siguiente:

WHILE <condición> LOOP
<secuencia de órdenes>
END LOOP;

Ejemplo

```
WHILE var1 <= 10 LOOP
INSERT INTO expediente
...
END LOOP;
```

Como en cualquier bucle LOOP…END LOOP, podemos forzar la salida del mismo con la instrucción EXIT, aunque no se cumpla la condición de evaluación indicada antes de la ejecución de la secuencia de instrucciones del bucle.

FOR…LOOP…END LOOP

Bucle repetitivo de ejecución con un número determinado de ejecuciones determinado por un contador.

La sintaxis de esta estructura es la siguiente:

FOR <contador> IN [REVERSE] <límite_inferior>..<límite_superior>
LOOP
<secuencia de órdenes>
END LOOP;

Ejemplo

```
FOR v_conta IN 1..50 LOOP
INSERT INTO expediente
...
END LOOP;
```

La variable que se utiliza como índice del bucle (contador) no es necesaria declararla previamente en la sección DECLARE del bloque, se declara implícitamente cuando se asocia a un bucle FOR y Oracle la gestiona con un tipo de dato BINARY_INTEGER. En caso de utilizar la cláusula REVERSE el proceso de recorrido de los límites del bucle se realiza de forma inversa: del límite superior al límite inferior.

GOTO

Realiza un salto a una etiqueta del bloque. La sintaxis de esta estructura es la siguiente:

GOTO etiqueta;

Las etiquetas se definen con la siguiente sintaxis:

<< nombre de la etiqueta >>

Ejemplo

```
BEGIN
...
GOTO eti1;
<<eti1>>
...
END;
```

null

Permite introducirla en un bloque cuando no se quiere ejecutar operación alguna pero es necesario completar la sintaxis de una instrucción de control. Puede ser muy útil cuando se está diseñando un programa en el que se empiezan a escribir las estructuras de control que se van a utilizar pero no se conoce aún el contenido de lo que tendrá que evaluar cada fase de la misma.

Ejemplo

```
IF valor > 1 THEN
...
ELSE
NULL;
END IF;
```

Capítulo 3. CONTROL DE TRANSACCIONES

INTRODUCCIÓN

Una transacción se considera aquella operación de manipulación de datos para su alteración (insert, update o delete), que permanece a la espera de ser validada (confirmada) o rechazada.

Para el control de las transacciones se dispone de las siguientes instrucciones:

- COMMIT
- ROLLBACK
- SAVEPOINT <nombre_marcador>
- ROLLBACK TO <nombre_marcador_savepoint>

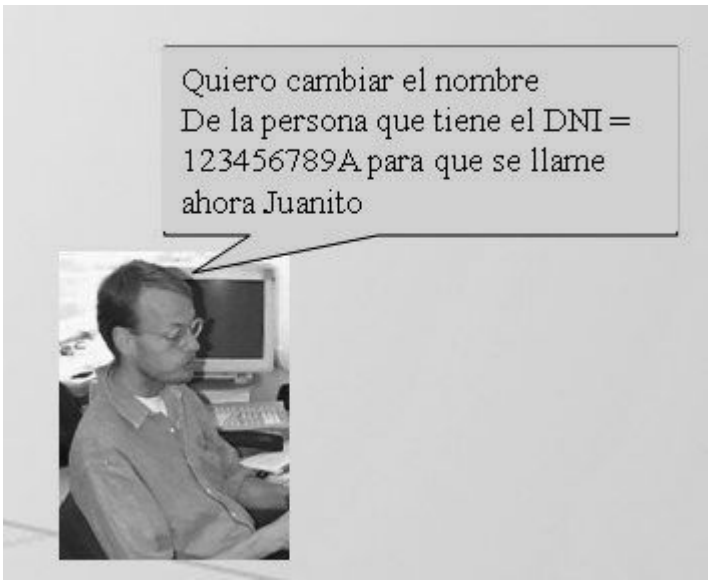
commit

La instrucción COMMIT permite validar toda transacción ejecutada que se encuentre pendiente de confirmar o rechazar.

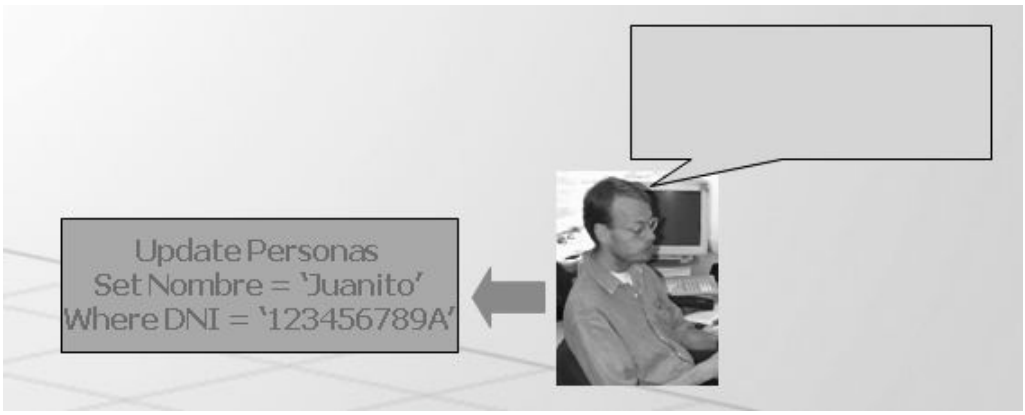
¿Cómo se realiza el proceso de validación (COMMIT)?

A continuación, se muestran de forma esquemática las fases del proceso de validación de una transacción.

1. El proceso de validación de una instrucción comienza con el deseo por parte del usuario de realizar una operación contra la base de datos:

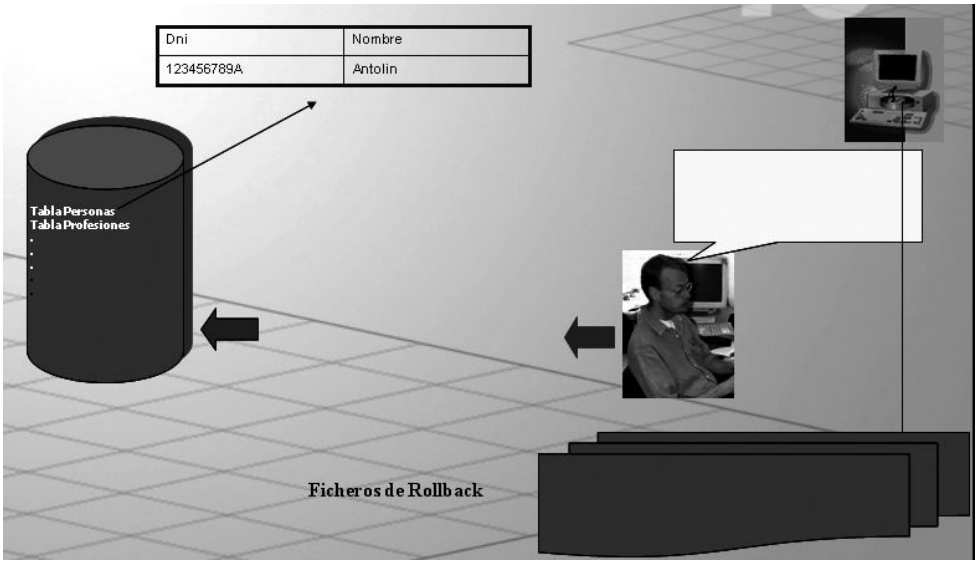


2. El siguiente paso supone la traducción de ese deseo en una instrucción de base de datos. En nuestro ejemplo, una instrucción SQL.

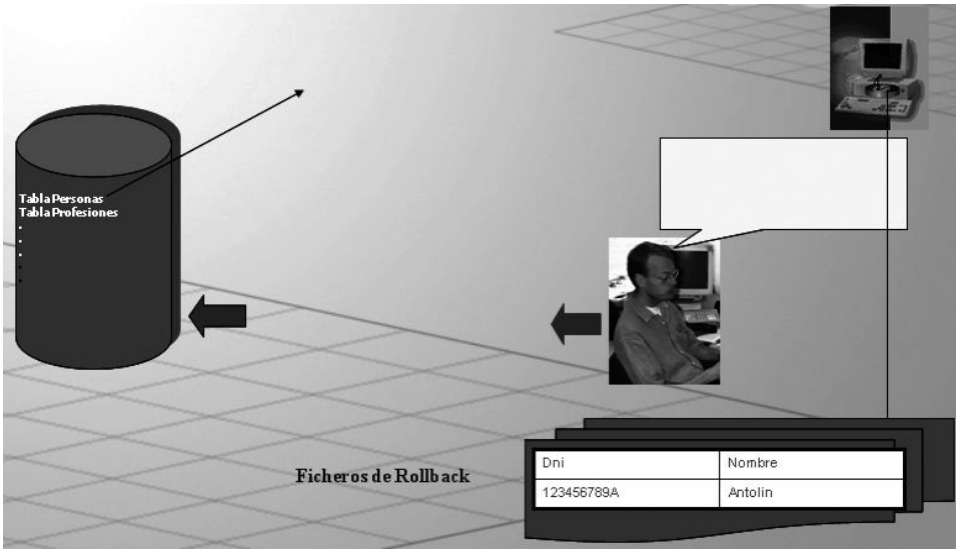


3. A continuación, se procede a la recuperación física de los datos afectados por

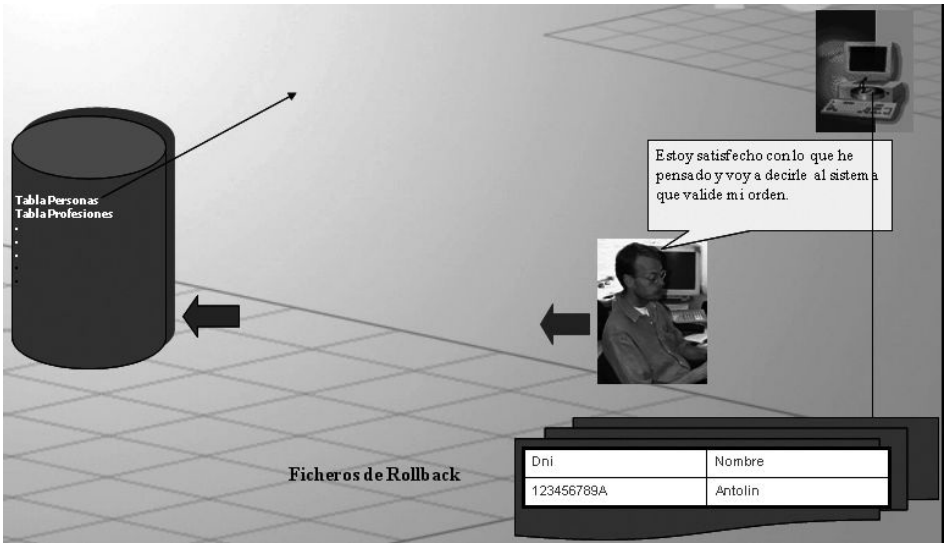
la instrucción. Para ello, el SGBD de Oracle accede hasta la ubicación donde se encuentran los datos y recupera aquellas filas que cumplen las condiciones impuestas en la instrucción enviada por el usuario. Además, prepara los ficheros de rollback, que son aquellos ficheros que almacenan las transacciones efectuadas pendientes de confirmación, así como los datos originales antes del cambio.



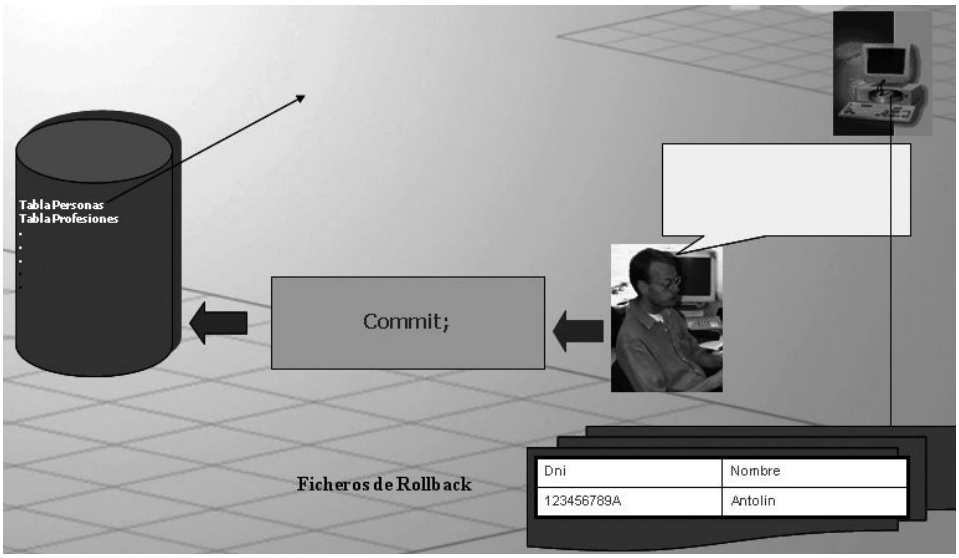
4. Seguidamente el SGBD traslada la información original recuperada de los ficheros físicos de información de la base de datos a los ficheros rollback.



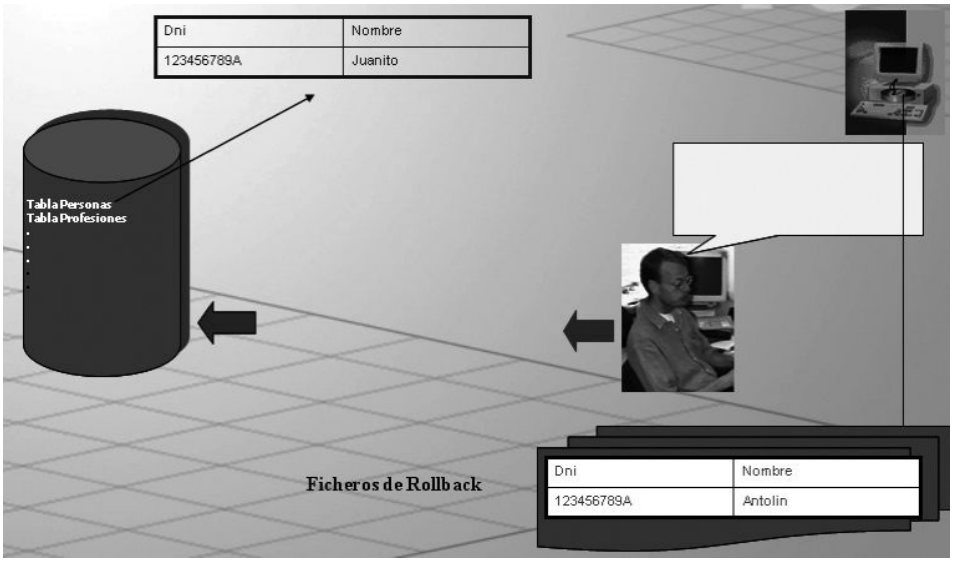
5. A continuación, el usuario se plantea la intencionalidad de validar la información que se ha modificado (en nuestro ejemplo), como consecuencia de la instrucción ejecutada.



6. Seguidamente el usuario ejecuta la instrucción, que implica una ordenación al SGBD de Oracle para que valide la información modificada con su primera instrucción y que se encuentra pendiente en los ficheros rollback, para lo que envía un comando COMMIT.



7. Como consecuencia de ejecutar la instrucción COMMIT, el SGBD modifica físicamente la información contenida en la base de datos, de acuerdo a la instrucción primitiva que envió el usuario.



8. Por último, el SGBD libera los ficheros rollback. Solo libera aquella información original recuperada con la instrucción primitiva del usuario antes de ejecutar el COMMIT.

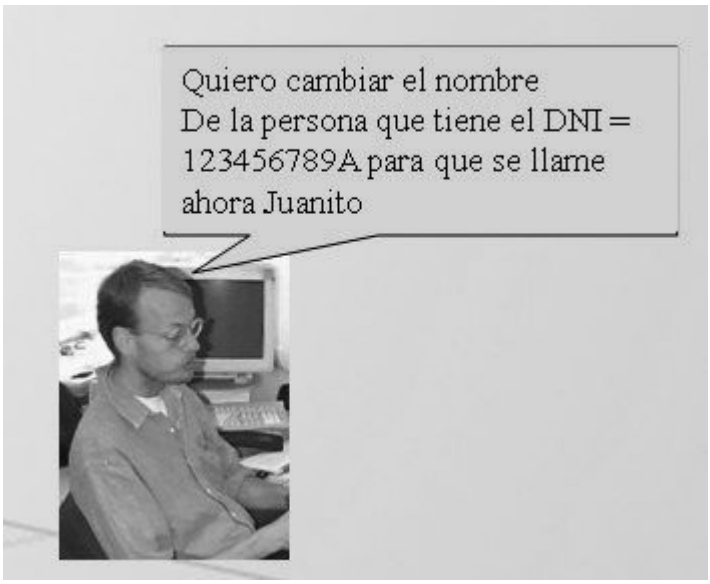
ROLLBACK

La instrucción ROLLBACK permite rechazar (deshacer y dejar a su estado original) toda transacción ejecutada que se encuentre pendiente de confirmar o rechazar.

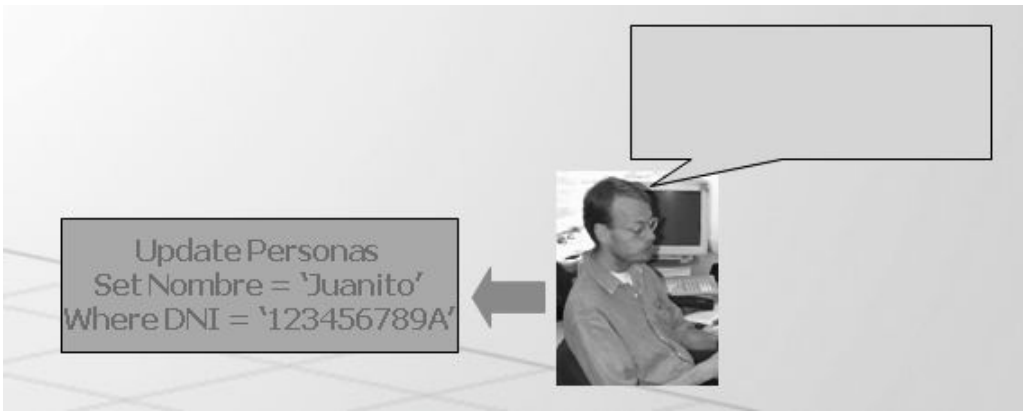
¿Cómo se realiza el proceso de rechazo (ROLLBACK)?

A continuación, se muestran de forma esquemática las fases del proceso de rechazo de una transacción.

1. El proceso de rechazo de una instrucción comienza con el deseo por parte del usuario de realizar una operación contra la base de datos:

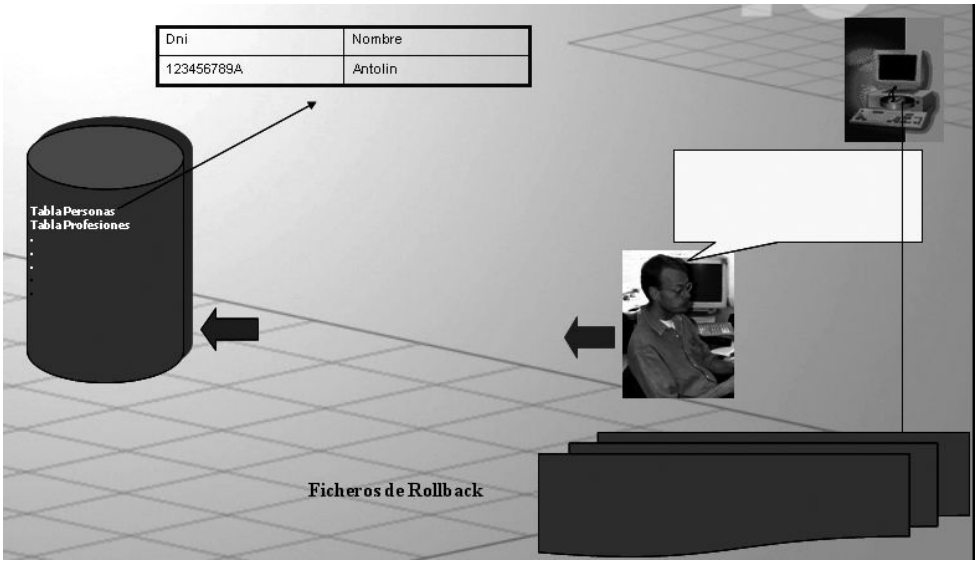


2. El siguiente paso supone la traducción de ese deseo en una instrucción de base de datos. En nuestro ejemplo, una instrucción SQL.

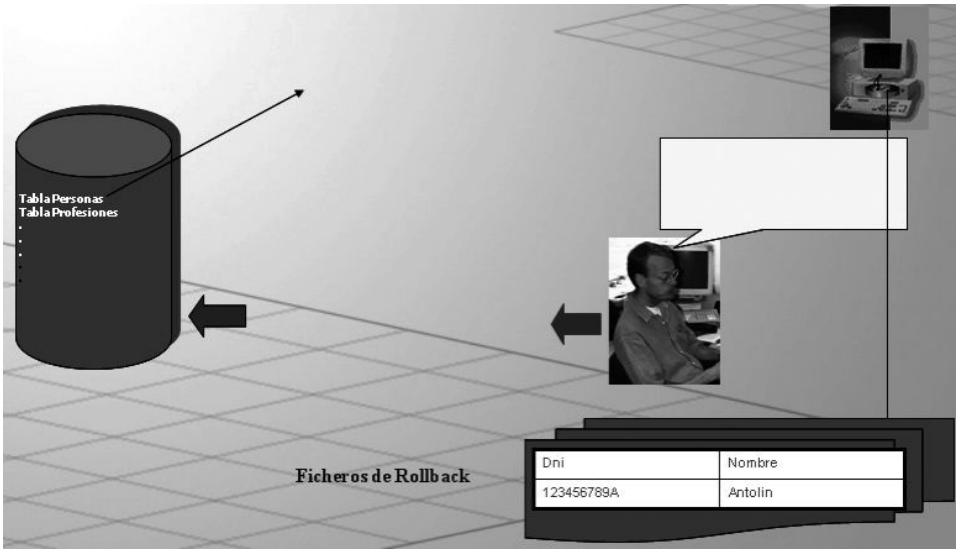


3. A continuación, se procede a la recuperación física de los datos afectados por

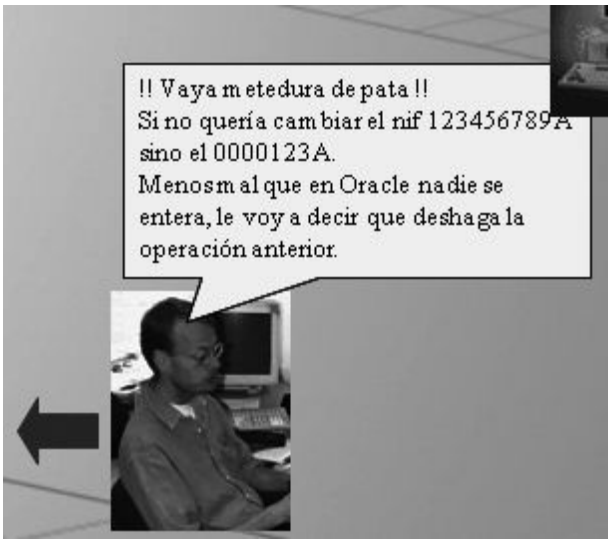
la instrucción. Para ello, el SGBD de Oracle accede hasta la ubicación donde se encuentran los datos y recupera aquellas filas que cumplen las condiciones impuestas en la instrucción enviada por el usuario. Además, prepara los ficheros de rollback, que son aquellos ficheros que almacenan las transacciones efectuadas pendientes de confirmación, así como los datos originales antes del cambio.



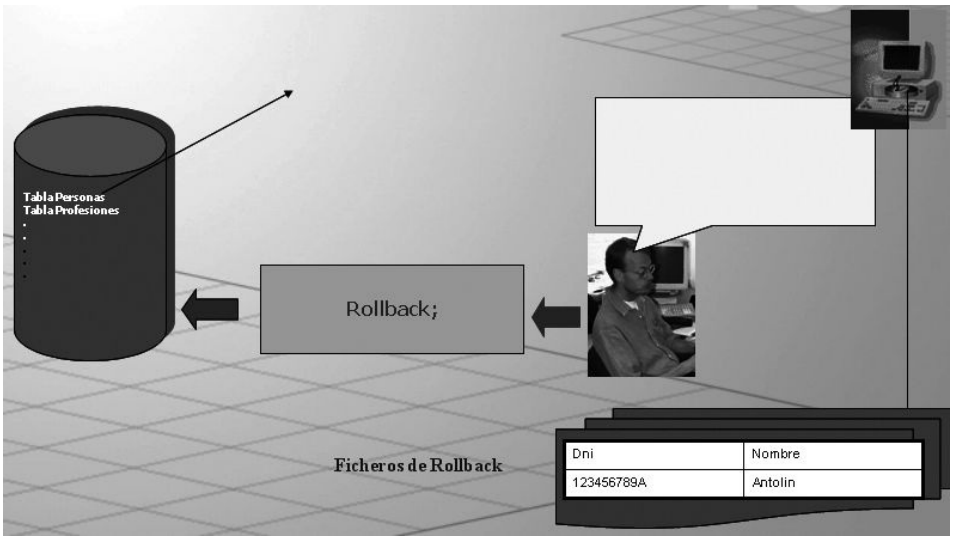
4. Seguidamente el SGBD traslada la información original recuperada de los ficheros físicos de información de la base de datos a los ficheros rollback.



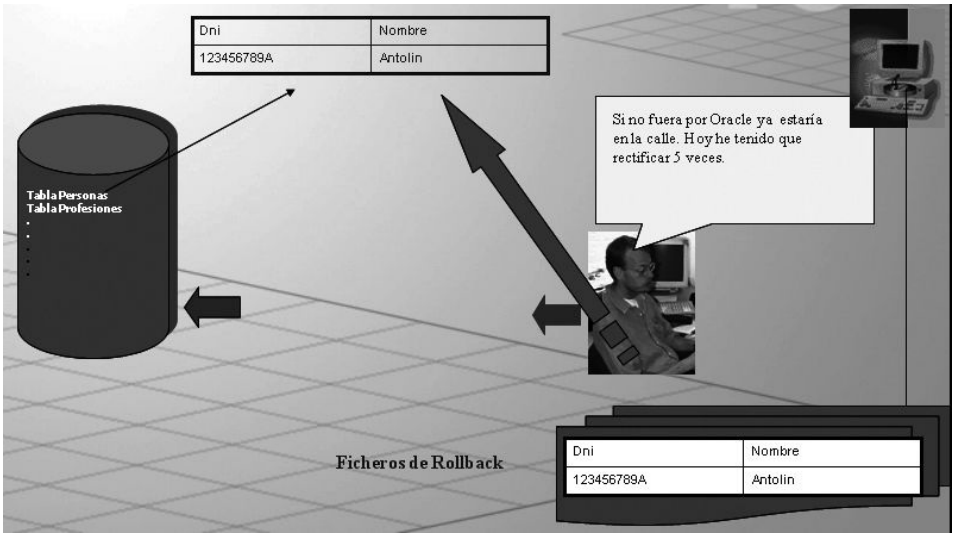
5. A continuación, el usuario se plantea la intencionalidad de rechazar los cambios que se generan tras la ejecución de la instrucción.



6. Seguidamente, el usuario ejecuta la instrucción que le ordena al SGBD de Oracle que rechace la información modificada con su primera instrucción y que se encuentra pendiente en los ficheros rollback, para lo que envía un comando ROLLBACK.



7. Como consecuencia de ejecutar la instrucción ROLLBACK, el SGBD recupera la información original contenida en los ficheros rollback, correspondiente a la instrucción primitiva que ejecutó el usuario, y la mueve a su posición original dentro de los ficheros de la base de datos.



8. Por último, el SGBD dentro del mismo punto anterior libera los ficheros rollback al realizar el movimiento de la información original recuperada con la instrucción primitiva del usuario.

SAVEPOINT

La instrucción SAVEPOINT permite introducir un marcador de posición dentro de una transacción del código PL/SQL, a fin de ser utilizado como referenciación para la instrucción ROLLBACK TO.

La sintaxis es la siguiente:

SAVEPOINT <nombre_marcador>

Por tanto, permite delimitar las transacciones a ejecutar en el programa con objeto de rechazar unas u otras dependiendo de cada situación de ejecución del programa.

ROLLBACK TO

La instrucción ROLLBACK TO permite rechazar las transacciones que existan desde el punto de ruptura (savepoint) indicado, hasta la siguiente instrucción ROLLBAK, COMMIT o END que se encuentre siempre en dirección secuencial hacia abajo en el código PL/SQL.

Ejemplo de uso del control de transacciones

```
DECLARE
V_temp temporal%ROWTYPE;
cuenta number;
BEGIN
INSERT INTO temporal(col1) VALUES ('Inserto 1');
SAVEPOINT A
INSERT INTO temporal(col1)
        VALUES ('Realizo consulta');
SELECT COL_NUMERICA INTO CUENTA FROM tab_misteriosa;
SAVEPOINT B;
INSERT INTO temporal(col1) VALUES (TO_CHAR(cuenta));
COMMIT;
EXCEPCION
-- Si no encuentra datos realiza rollback
        -- desde punto B
WHEN NO_DATA_FOUND THEN
ROLLBACK TO B;
COMMIT;
/* Si se devuelven múltiples filas realiza rollback
        desde el punto B */
WHEN TOO_MANY_ROWS THEN
ROLLBACK TO A;
COMMIT;
/* Si se produce cualquier otro error realiza
rollback Total */
WHEN OTHERS THEN
ROLLBACK;
END;
```

SUPUESTO PRÁCTICO 2: ***Resolución en el Anexo I de este manual.***

Crear un bloque sin nominar en SQL*PLUS que realice las siguientes operaciones:

- Solicitar el siguiente literal por pantalla: *nombre_de_enfermo*, y almacenarlo en una variable del mismo tipo y tamaño que el campo que almacena el nombre en la tabla enfermo.

- Solicitar el siguiente literal por pantalla: *apellidos_del_mismo,* y almacenarlo en una variable del mismo tipo y tamaño que el campo que almacena el apellido en la tabla enfermo.
- Solicitar el siguiente literal por pantalla: *dirección_donde_reside*, y almacenarlo en una variable del mismo tipo y tamaño que el campo que almacena la dirección en la tabla enfermo.
- Mostrar un texto con la siguiente información: 'Nº de filas en tabla enfermo antes de insertar =', concatenado con el resultado de contar el número de filas que hay en la tabla enfermo.
- Insertar en la tabla ENFERMO un registro con la siguiente información:

NUMSEGSOCIAL = 280862486
NOMBRE= el nombre que se ha insertado por pantalla
APELLIDOS= los apellidos introducidos por pantalla
DIRECCION= la que se ha introducido por pantalla
SEXO=M

- Insertar, mediante un bucle repetitivo, diez líneas en la tabla HOSPITAL_ENFERMO con la siguiente información para el mismo:

HOSP_CODIGO = 6
INSCRIPCION = seq_inscripcion.nextval
ENF_NUMSEGSOCIAL = 280862486
FINSCRIPCION = TO_DATE('01012000','DDMMYYYY') + seq_inscripcion.nextval

- Rechazar las filas insertadas.
- Mostrar un texto con la siguiente información: 'Nº de filas en la tabla enfermo después de insertar y rechazar filas=' concatenado con el resultado de contar el número de filas que hay en la tabla enfermo.

Capítulo 4. CREACIÓN DE TIPOS

INTRODUCCIÓN

En PL/SQL aparte de los tipos estándar que ofrece Oracle tanto en SQL como en PL/SQL, también es posible definir otros tipos de usuario, de acuerdo con las siguientes estructuras:

- RECORD (Tipos registro)
- TABLE (Tipos table)
- VARRAY (Tipos array de elementos)

creación de un tipo record (registro)

Es un tipo que define el usuario con la combinación de los tipos ya definidos anteriormente o que pertenecen al lenguaje PL/SQL estandar.

Un registro de Oracle es similar a las estructuras del lenguaje C. Proporciona un mecanismo para tratar con variables diferentes, pero relacionadas como si fueran una unidad.

La sintaxis para definir y crear un registro en un bloque PL/SQL es la siguiente:

```
TYPE tipo_registro IS RECORD (
Campo1 tipo1 [NOT NULL] [:= valor],
...
);
```

Ejemplo

```
TYPE estudiante IS RECORD (
Cod_matricula NUMBER(5),
Nombre VARCHAR2(50),
Apellidos VARCHAR2(40),
Media NUMBER(6,2) NOT NULL := 0);

V_estudiate estudiante;
```

Asignación de valores a un registro de forma directa

La sintaxis es la siguiente:

<nombre_var_registro>.<campo> := <valor>;

Ejemplo

v_estudiante.media := 10;

Asignación de valores a un registro desde una consulta

La consulta se deberá diseñar con tantas columnas como elementos tenga el registro, debiendo existir equivalencia de tipos entre las columnas y los elementos del registro.

Ejemplo

```
DECLARE
TYPE estudiante IS RECORD (
Cod_matricula NUMBER(5),
Nombre VARCHAR2(50),
Apellidos VARCHAR2(40),
Media NUMBER(6,2) NOT NULL := 0);
V_estudiate estudiante;

BEGIN
SELECT cod_matrícula, nom, ape, media
INTO v_estudiante
FROM tab_estudiantes
WHERE cod_matrícula = 50;
END;
/
```

%Rowtype

Permite definir un tipo registro con el total de columnas de una tabla, manteniendo el nombre, tamaño, tipo y orden de las mismas dentro de la tabla.

La sintaxis es la siguiente:

<variable> <tabla>%ROWTYPE;
Ejemplo

```
DECLARE
V_estudiante tab_estudiantes%ROWTYPE;
BEGIN
SELECT *
INTO v_estudiante
FROM tab_estudiantes
WHERE cod_matrícula = 50;
END;
```

También con esta definición de tipo, podemos realizar actualizaciones de tablas de la base de datos.

Actualización de una tabla a partir de un tipo RECORD

Es posible actualizar una/varias/todas las columnas de una fila de una tabla a partir de un tipo RECORD.

Cuando no se actualizan todas las columnas de la tabla, se utiliza la sintaxis vista anteriormente: <nombre_var_registro>.<campo> := <valor>;

Pero para actualizar el contenido completo de una fila de una tabla se utiliza la pseudocolumna **ROW**, que nos permite indicarle al SGBD que lo que se va a actualizar es toda la fila de la tabla.

Ejemplo de actualización de columnas concretas

```
DECLARE
V_estudiante tab_estudiantes%ROWTYPE;
BEGIN
V_estudiante.notafinal := 9.95;
V_estudiante.nombre := 'Antolin';

UPDATE tab_estudiantes
SET
nota = v_estudiante.notafinal
,nombre = v_estudiante.nombre
WHERE matricula = 1234567;
END;
/
```

Ejemplo de actualización del contenido de una fila entera

```
DECLARE
V_estudiante tab_estudiantes%ROWTYPE;
BEGIN
V_estudiante.notafinal := 9.95;
V_estudiante.nombre := 'Antolin';

UPDATE tab_estudiantes
```

```
SET ROW = v_estudiante
WHERE matricula = 1234567;
END;
/
```

creación de un tipo table (pila de elementos)

Este tipo de dato nos permite crear una matriz o secuencia de elementos en memoria. Todos ellos comparten el mismo tipo de dato y se aglutinan en un único objeto.

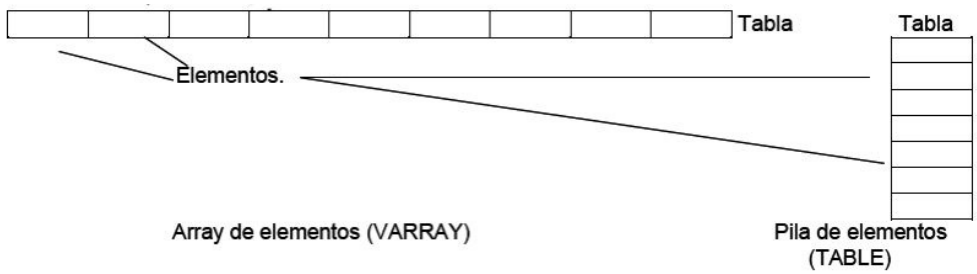


Fig. 4-1 Comparativa de un Varray y un tipo Table.

Aunque sintácticamente un tipo TABLE (o tipo tabla) es una matriz de elementos, lo que la convertiría en un VARRAY, en cuanto al almacenamiento interno de Oracle, un tipo TABLE es una pila de elementos, por lo que se diferencia bastante de un array de elementos.

La sintaxis de la definición de un tipo table es la siguiente:

```
TYPE nombre_tipotabla IS TABLE OF tipo_de_dato
INDEX BY BINARY_INTEGER;
```

Ejemplo

```
DECLARE
    TYPE t_nombres IS TABLE OF estudiante.nombre%TYPE
INDEX BY BINARY_INTEGER;
    TYPE t_estudiantes IS TABLA OF estudiantes%ROWTYPE
INDEX BY BINARY_INTEGER;
    TYPE t_matrículas IS TABLE OF NUMBER(5)
INDEX BY BINARY_INTEGER;

    V_nombres t_nombres;
    V_estudiantes t_estudiantes;
    V_matrículas t_matrículas;
```

Asignación de valores a un tipo TABLE

La sintaxis para la asignación de valores a un tipo tabla (TABLE) es la siguiente:

<variable_tipo_tabla>(numero) := <valor>

Ejemplo

```
DECLARE
        TYPE t_nombres IS TABLE OF estudiante.nombre%TYPE
INDEX BY BINARY_INTEGER;
        TYPE t_estudiantes IS TABLA OF estudiantes%ROWTYPE
INDEX BY BINARY_INTEGER;
        TYPE t_matrículas IS TABLE OF NUMBER(5)
INDEX BY BINARY_INTEGER;

        V_nombres t_nombres;
        V_estudiantes t_estudiantes;
        V_matrículas t_matrículas;
BEGIN
V_matriculas(1) := 10;

SELECT * INTO v_estudiantes(1) FROM estudiantes
WHERE matricula = 1000;

V_nombre(1) := ‘ANTOLIN’;
V_estudiantes(1).nombre := 'PEPE';
END;
```

Atributos de un tipo TABLE

Los atributos que se pueden asociar a un tipo tabla (TABLE) son los siguientes:

- COUNT
- DELETE
- EXISTS (valor)
- FIRST
- LAST
- NEXT (valor)
- PRIOR (valor)

ATRIBUTO COUNT

Permite contar el número de elementos o filas de un tipo tabla.

ATRIBUTO DELETE

Permite borrar filas o elementos de un tipo tabla. Se puede utilizar de 3 formas diferentes:

- DELETE: borra todas las filas del tipo.
- DELETE(valor): borra el elemento indicado en *valor.*
- DELETE(valor1,valor2): borra los elementos entre el *valor1* y el *valor2* incluyendo a ambos.

ATRIBUTO exists (VALOR)

Permite evaluar si existe el elemento cuyo índice sea equivalente a *valor* dentro del tipo.

ATRIBUTO first

Nos devuelve el valor del índice del primer elemento del tipo.

ATRIBUTO last

Nos devuelve el valor del índice del último elemento del tipo.

ATRIBUTO next (valor)

Nos devuelve el valor del índice del siguiente elemento al indicado.

ATRIBUTO PRIOR (valor)

Nos devuelve el valor del índice del elemento anterior al indicado.

Ejemplo

```
DECLARE
TYPE t_matriculas IS TABLE OF NUMBER(5)
INDEX BY BINARY_INTEGER;
V_matricula t_matriculas;
V_total NUMBER;
V_indice NUMBER;
Contador NUMBER;

BEGIN
V_matricula(1) := 10;
V_matricula(2) := 20;
V_matricula(3) := 30;

        -- Devuelve 3 filas.
V_total := v_matricula.COUNT;

        -- Borra todas las filas.
        V_matricula.DELETE;

FOR contador IN 1..5 LOOP
V_matricula(contador) := 10;
END LOOP;

-- Borrar elementos 1 a 3 inclusive, sin rehacer
        -- la numeración.
        V_matricula.DELETE(1,3);

        -- Borra el elemento 4, ya solo existe la Fila 5.
        V_matricula.DELETE(4);

        -- Comprueba si existe la fila 1
        IF v_matricula.EXISTS(1) THEN
NULL;
```

```
        END IF;

        -- Crea las filas 1 a 4 y a la 5 que existe, le
        -- reasigna al valor 10.
        FOR contador IN 1..5 LOOP
V_matricula(contador) := 10;
END LOOP;

-- Ahora preguntamos si existe un valor anterior al
-- primer elemento de la tabla
IF V_matricula.PRIOR(V_matricula.FIRST) IS NULL THEN
DBMS_OUTPUT.PUT_LINE('No existen valores anteriores al 1');
END IF;

        -- Ahora preguntamos si existe un valor posterior al
        -- último elemento de la tabla


IF V_matricula.NEXT(V_matricula.LAST) IS NULL THEN
DBMS_OUTPUT.PUT_LINE('No existen valores posteriores al
'||V_matricula.COUNT);
END IF;

        -- El valor devuelto es 1
        V_indice := v_matricula.FIRST;

        --El valor devuelto es 5
        V_indice := v_matricula.LAST;

        V_indice := v_matricula.FIRST;

        --El valor devuelto es 2
        V_indice := v_matricula.NEXT(1);

        --El valor devuelto es 1
        V_indice := v_matricula.PRIOR(2);
END;
```

varrays

Un VARRAY se manipula de forma muy similar a las tablas de PL pero se implementa de forma diferente.

Los elementos en el VARRAY se almacenan comenzando en el índice 1 hasta la longitud máxima declarada en el tipo VARRAY.

La sintaxis es la siguiente:

TYPE <nombre_tipo> IS VARRAY (tamaño_maximo>) OF <tipo_dato>;

Una consideración a tener en cuenta es que en la declaración de un VARRAY el <tipo_dato> no puede ser uno de los siguientes tipos de datos:

- BOOLEAN
- NCHAR
- NCLOB
- NVARCHAR(n)
- REF CURSOR
- TABLE
- VARRAY

Sin embargo, se puede especificar el tipo utilizando los atributos %TYPE y %ROWTYPE.

Inicialización de un VARRAY

Los VARRAY deben estar inicializados antes de poderse utilizar. Para inicializarlos se utiliza un constructor (podemos inicializar el VARRAY en la sección DECLARE o bien dentro del cuerpo del bloque):

Ejemplo

```
DECLARE
/* Declaramos el tipo VARRAY de cinco elementos
y de tipo VARCHAR2*/
TYPE t_cadena IS VARRAY(5) OF VARCHAR2(50);

/* Asignamos los valores con un constructor */
v_lista t_cadena:= t_cadena('Aitor', 'Alicia',
'Pedro',null,null);
BEGIN
v_lista(4) := 'Tita';
v_lista(5) := 'Ainhoa';
dbms_output.put_line(v_lista(4));
END;
```

El tamaño de un VARRAY se establece mediante el número de parámetros utilizados en el constructor, si declaramos un VARRAY de cinco elementos pero al inicializarlo pasamos solo tres parámetros al constructor, el tamaño del VARRAY será tres. Si se hacen asignaciones a elementos que queden fuera del rango, se producirá un error.

El tamaño de un VARRAY podrá aumentarse utilizando la función EXTEND, pero nunca con mayor dimensión que la definida en la declaración del tipo. Por ejemplo, la variable *v_lista* que solo tiene 3 valores definidos, por lo que se podría ampliar hasta cinco elementos, pero no más allá.

Un VARRAY comparte con las tablas de PL todas las funciones válidas para ellas, pero además añade las siguientes:

- LIMIT
- EXTEND

- EXTEND(n,i): Añade (n) copias del elemento (i) al final del VARRAY.

ATRIBUTO LIMIT

Devuelve el número máximo de elementos que admite el VARRAY.

ATRIBUTO extend

Añade uno o varios elementos NULL al final del VARRAY.

Si utilizamos EXTEND a secas, se añade un único elemento al final del VARRAY.

Si utilizamos EXTEND(n), añadimos *n* elementos al final del VARRAY

ATRIBUTO extend(n,i)

Añade *N* copias del elemento *I* al final del VARRAY.

Ejemplo

```
DECLARE
-- Declaramos el tipo VARRAY de cinco elementos VARCHAR2
TYPE t_cadena IS VARRAY(5) OF VARCHAR2(50);

-- Asignamos los valores con un constructor
v_lista t_cadena:= t_cadena('Aitor', 'Alicia', 'Pedro');

v_lista2 t_cadena;



BEGIN
-- Devuelve: 1
dbms_output.put_line('El elemento más pequeño está en la posición: '||v_lista.first);

-- Devuelve: 3
	dbms_output.put_line('El varray tiene: '||v_lista.count||' elementos');

	-- Devuelve: 5
	dbms_output.put_line('El nº máximo de elementos admitidos en el varray es: '||v_lista.limit);

	-- Se añade 1 elemento
	v_lista.extend;

	-- Devuelve vacio
	dbms_output.put_line(nvl(v_lista(4),'vacio'));

v_lista(4) := 'Tita';
	-- Devuelve: Tita
dbms_output.put_line('Ahora el elemento 4 tiene el valor:
```

```
'||nvl(v_lista(4),'vacio'));

        v_lista2 := t_cadena();
        -- Devuelve: 0
dbms_output.put_line('El varray2 tiene: '||v_lista2.count||' elementos');

        -- Añade 3 elementos
        v_lista2.extend(3);

        -- Devuelve: 3
dbms_output.put_line('El varray2 tiene ahora: '||v_lista2.count||' elementos');

v_lista2(1) := 'Antolin';
        -- Añade 2 elementos copia del elem. 1
        v_lista2.extend(2,1);

-- Devuelve: antolin.
        dbms_output.put_line('Ahora el elemento 5 del varray 2 tiene el valor:
'||nvl(v_lista2(5),'vacio'));
END;
```

SUPUESTO PRÁCTICO 3: ***Resolución en el Anexo I de este manual.***

1) **Realizar una consulta de la tabla DEPARTAMENTO que saque el NOMBRE de todos los departamentos distintos que haya en la misma.**
2) **Crear un bloque sin nominar en SQL*PLUS que realice las siguientes operaciones:**

- Declarar un tipo registro que se componga de los siguientes campos:
  - Un campo que almacene el nombre del departamento y que sea del mismo tipo y tamaño que su homólogo en la tabla departamentos.
  - Un campo que almacene el número de empleados que trabajan en el departamento de tipo numérico de 5 dígitos.
  - Un campo que almacene el tipo de ocupación con formato VARCHAR2(20).
- Declarar un tipo tabla que almacene registros como el que se ha creado en el punto anterior.
- Realizar una consulta que solicite por pantalla el nombre de un departamento y que para el mismo, extraiga el nombre y el número de empleados que trabajan en él. El nombre del departamento se almacenará en el primer campo del registro declarado y el número de empleados en el segundo campo de registro.
- En el tercer campo del registro declarado se almacenarán algunos de los siguientes literales (*BAJA, MEDIA, ALTA*) dependiendo de las siguientes consideraciones:
  - Si el número de empleados de un departamento es menor que 10, entonces se almacenará la palabra *BAJA.*
  - Si el número de empleados de un departamento se encuentra entre 10 y 19, entonces se

almacenará la palabra *MEDIA*.
    - En el resto de casos se almacenará la palabra *ALTA*.
- Después de esto se deberá mostrar la siguiente línea:
  (CHR(10)||CHR(10)||'Datos de ocupación del departamento' concatenado con el nombre del departamento que se ha almacenado en el primer elemento del tipo tabla declarado.
- A continuación, mostrará la siguiente línea:
  ('Ocupación actual (Tipo/nº empleados):' concatenado con el símbolo '/' y por último concatenado con el segundo campo del tipo tabla declarado.

Capítulo 5. SQL vs PL/SQL

INTRODUCCIÓN

SQL es un lenguaje estructurado de consulta que define cómo manipular los datos en Oracle. Esa potencia, unida a los componentes procedimentales que se han empezado a ver anteriormente, y que se proporcionan a través de PL/SQL, hace que la gestión de un sistema gestor de base de datos como Oracle sea completa.

ÓRDENES SQL

Las órdenes SQL se pueden dividir en 5 categorías que se muestran a continuación:

- Lenguaje de manipulación de datos (DML – Data Manipulation Language).
- Lenguaje de definición de datos (DDL – Data Definition Language).
- Las órdenes de control de transacciones.
- Las órdenes de control de la sesión.
- Las órdenes de control del sistema.

Lenguaje DML

El lenguaje DML (Data Manipulation Language) permite cambiar o consultar los datos contenidos en una tabla, pero no permite cambiar la estructura de la misma u otro objeto de la base de datos.

Las órdenes básicas que se utilizan en el lenguaje DML se indican en la siguiente tabla:

Comando	Descripción
SELECT	Se utiliza para consultar registros de las tablas de la base de datos que satisfagan un criterio determinado.
INSERT	Se utiliza para insertar información en las tablas de la base de datos en una única operación.
UPDATE	Se utiliza para modificar los valores de las columnas y registros de las tablas de la base de datos.
DELETE	Se utiliza para eliminar registros de una tabla de una base de datos.

Lenguaje DDL

El lenguaje DDL (Data Definition Language) permite crear, borrar o modificar la estructura de un objeto del esquema de la base de datos. Las órdenes para asignar y revocar permisos sobre estos objetos también pertenecen a este lenguaje.

Las órdenes básicas que se utilizan en el lenguaje DDL se indican en la siguiente tabla:

Comando	Descripción
CREATE	Se utiliza para crear nuevas estructuras: tablas, procedimientos, paquetes, índices, secuencias, vistas, etc.
DROP	Se utiliza para eliminar estructuras creadas.
ALTER	Se utiliza para modificar las estructuras creadas.
GRANT	Se utiliza para asignar derechos.
REVOKE	Se utiliza para revocar derechos asignados.

Órdenes para el control de transacciones

Este conjunto de instrucciones que se indican en la siguiente tabla, garantizan la consistencia de los datos.

Comando	Descripción
COMMIT	Esta orden finaliza la tarea procesada en una transacción haciendo efectivo los cambios sobre la base de datos.
ROLLBACK	Esta orden finaliza la tarea procesada en una transacción, pero deshaciendo los cambios que se hubiesen invocado, permaneciendo la base de datos en el estado original antes de comenzar la transacción.
SAVEPOINT	Esta orden marca un punto en la transacción desde el cual se puede invocar una operación de COMMIT o ROLLBACK.

Órdenes para el control de la sesión

Este conjunto de instrucciones cambian las opciones de una conexión determinada a la base de datos, por ejemplo habilitar la traza SQL.

Las órdenes básicas para el control de la sesión se indican en la siguiente tabla:

Comando	Descripción
ALTER SESSION	Permite alterar parámetros de la sesión actual.
SET ROLE	Permite asignar un role de permisos a un usuario.

Órdenes para el control del sistema

Este conjunto de instrucciones cambian las opciones que afectan a la base de datos completa; por ejemplo, la activación o desactivación del archivado.

Las órdenes básicas para el control del sistema se indican en la siguiente tabla:

Comando	Descripción
ALTER SYSTEM	Permite alterar los parámetros del sistema.

UTILIZACIÓN DE SQL EN PL/SQL

El lenguaje PL/SQL admite la interpretación de sentencias SQL dentro de un bloque BEGIN…END, pero con una serie de restricciones que se indican a continuación:

- No se permite la utilización de sentencias DDL directas dentro del código incluido en el bloque PL/SQL.
- Se pueden utilizar las sentencias DML: insert, update y delete, directamente en el código incluido en el bloque PL/SQL.
- Solo puede utilizar la sentencia SELECT del lenguaje DML, directamente en el código incluido en el bloque PL/SQL, si va acompañada de la cláusula INTO.
- El uso de la sentencia SELECT directamente en el código incluido en el bloque PL/SQL está limitado a que el resultado de la consulta devuelve un solo registro. Si la consulta devuelve más de un registro o ninguno, generará un error de ejecución del bloque PL/SQL.
- Está permitido el uso de sentencias para el control de transacciones dentro del código incluido en el bloque PL/SQL.
- No se permite el uso de sentencias de control del sistema o de control de sesión, directamente en el código incluido en el bloque PL/SQL.

Para poder utilizar el resto de instrucciones del lenguaje SQL o aquellas que no están permitidas directamente en el código, se utiliza SQL dinámico.

La sintaxis de la instrucción SELECT para interpretarla directamente en el código del bloque PL/SQL es la siguiente:

```
    SELECT elemento_lista1[,elemento_lista2,…]
INTO registro_PLSQL/variable1[,variable2,…]
FROM tabla
[WHERE condicionantes]
```

Como se ha indicado anteriormente y como se puede apreciar en la sintaxis, la única diferencia con la sintaxis de un SELECT ejecutado en el lenguaje SQL, es que se incluye la cláusula INTO para devolver los valores que retorna la instrucción en una variable de tipo registro o en variables individuales del mismo tamaño y tipo que las columnas que se están consultado.

A continuación, puede observar una serie de ejemplos del uso correcto e incorrecto de la instrucción SELECT dentro de un bloque PL/SQL, tomando como referencia la tabla EMPLEADOS con el siguiente contenido:

DNI	Nombre	Apellidos	Sexo	Edad
11111111A	JOSE ANTONIO	BLAZQUEZ SANCHEZ	M	31
22222222B	JACINTO	BERMUDEZ TAPIA	M	32
33333333C	ELISA	RIAÑO TORAL	F	33
44444444D	CLARA	YAÑEZ SANCHEZ	F	33

EJEMPLO DE USO CORRECTO DE LA INSTRUCCIÓN SELECT EN PL/SQL

```
DECLARE
V_Edad NUMBER(2);
V_Conteo NUMBER;
BEGIN
SELECT COUNT(*) INTO V_Conteo FROM empleados
WHERE edad = 31;

   IF V_Conteo = 1 THEN

   SELECT edad INTO V_Edad FROM empleados
   WHERE edad = 31;
   ELSIF V_Conteo = 0 THEN

   DBMS_OUTPUT.PUT_LINE('No hay personas con '||
                        '31 años');
ELSE
DBMS_OUTPUT.PUT_LINE('Hay más de 1 persona con'||
' 31 años.');
END IF;
END;
```

En este ejemplo se definen 2 variables (V_Edad y V_Conteo) para la recogida de la información que se va a recuperar posteriormente en las consultas del bloque BEGIN…END.

Dentro del bloque BEGIN…END, en primer lugar se cuenta el número de registros que hay para la condición de búsqueda del SELECT: que la edad sea igual a 31 años, llevando el resultado a la variable V_Conteo.

A continuación, se evalúa mediante un IF, el contenido de la variable V_Conteo, de forma que solo si tiene valor 1 (es decir, la consulta va a devolver únicamente un registro), entonces se ejecuta otra sentencia SELECT que recupera el valor de la edad y la lleva a la variable V_Edad.

En caso de que la variable V_Conteo tenga valor 0 (no se recupera ningún empleado con 31 años) o mayor que 1 (recupera más de un empleado con 31 años), se devuelve un mensaje de error al usuario, sin que el bloque aborte por errores de ejecución.

EJEMPLO DE USO inCORRECTO DE LA INSTRUCCIÓN SELECT EN PL/SQL

```
DECLARE
V_Nombre EMPLEADOS.NOMBRE%TYPE;
BEGIN
SELECT Nombre INTO V_Nombre FROM empleados
WHERE edad = 33;
END;
```

Este ejemplo provocaría un error en tiempo de ejecución del bloque (no en compilación), porque al consultar la tabla empleados para aquellos que tengan 33 años, la consulta devuelve dos registros, lo que impide que dichos valores se puedan almacenar a la vez en la variable V_Nombre.

Una posible solución a este problema habría sido insertar un conteo previo de registros para la misma condición (empleados con 33 años), para determinar cuántos devuelve la consulta, de forma que si solo devuelve 1, entonces puede llevarla a cabo, como se muestra a continuación:

```
DECLARE
V_Nombre EMPLEADOS.NOMBRE%TYPE;
V_Conteo NUMBER;
BEGIN
SELECT COUNT(*) INTO V_Conteo FROM empleados
WHERE edad = 33;
```

```
IF V_Conteo = 1 THEN
SELECT Nombre INTO V_Nombre FROM empleados
WHERE edad = 33;
END IF;
END;
```

Otra alternativa sería la introducción de una sección para el control de errores que se denomina EXCEPTION para que se pudiese capturar el error en tiempo de ejecución sin que aborte el bloque y tratarlo adecuadamente en la sección EXCEPTION, como se muestra a continuación:

```
DECLARE
V_Nombre EMPLEADOS.NOMBRE%TYPE;
BEGIN
SELECT Nombre INTO V_Nombre FROM empleados
WHERE edad = 33;

EXCEPTION
WHEN TOO_MANY_ROWS THEN
DBMS_OUTPUT.PUT_LINE('Hay más de 1 persona con'||
' 33 años.');
WHEN NO_DATA_FOUND THEN
DBMS_OUTPUT.PUT_LINE('No hay personas con'||
' 33 años.');
WHEN OTHERS THEN
DBMS_OUTPUT.PUT_LINE('Se ha producido un error'||
                                    ' al ejecutar el bloque.');

END;
```

EJEMPLO DE USO inCORRECTO DE LA INSTRUCCIÓN SELECT EN PL/SQL

```
DECLARE
V_Nombre EMPLEADOS.NOMBRE%TYPE;
BEGIN
SELECT Nombre FROM empleados
WHERE edad = 33;
END;
```

Este ejemplo provocaría un error en tiempo de compilación, porque no se permite el uso de una instrucción directa SELECT dentro de un bloque PL/SQL si no incluye la cláusula INTO.

SQL DINÁMICO

El lenguaje PL/SQL ofrece la posibilidad de ejecutar sentencias SQL a partir de cadenas de caracteres (uso embebido del lenguaje SQL).

Para ello, debemos emplear la instrucción EXECUTE IMMEDIATE. De esta forma podemos llegar a ejecutar no solo sentencias DML, sino que también podemos llegar a ejecutar cualquier otra sentencia SQL, como las instrucciones DDL.

Para controlar el número de filas afectadas por la ejecución de instrucciones DML del tipo insert, update o delete, ejecutadas mediante EXECUTE IMMEDIATE, se utiliza la instrucción SQL%ROWCOUNT.

El SQL DINÁMICO también permite ejecutar sentencias SELECT, pero no retorna filas salvo que se utilice un cursor.

EJEMPLO DE sql dinámico con una instrucción update

```
SET SERVEROUTPUT ON;

DECLARE
filas_afectadas NUMBER;
Cadena_sql VARCHAR2(1000);

BEGIN

Cadena_sql := 'UPDATE HOSPITAL SET TELEFONO = 910001122';
EXECUTE IMMEDIATE Cadena_sql;
filas_afectadas := SQL%ROWCOUNT;
dbms_output.put_line(TO_CHAR(filas_afectadas));

END;
```

Este ejemplo define una variable de tipo texto (varchar2) denominada Cadena_sql, que es sobre la que se asigna la instrucción de actualización (UPDATE). A continuación, se ejecuta dicha cadena con la instrucción EXECUTE IMMEDIATE y por último, se obtiene el conteo de filas actualizadas con SQL%ROWCOUNT.

EJEMPLO DE sql dinámico con una instrucción select

```
SET SERVEROUTPUT ON;
DECLARE
filas_afectadas NUMBER;
v_cursor REF CURSOR;
v_registro EMPLEADOS%ROWTYPE;
Cadena_sql VARCHAR2(1000);

BEGIN
Cadena_sql := 'SELECT COUNT(*) FROM EMPLEADOS';
EXECUTE IMMEDIATE Cadena_sql INTO filas_afectadas;
dbms_output.put_line(TO_CHAR(filas_afectadas));
Cadena_sql := 'SELECT * FROM EMPLEADOS';
OPEN v_cursor FOR Cadena_sql;
LOOP
FETCH v_cursor INTO v_registro;
EXIT WHEN v_cursor%NOTFOUND;
        dbms_output.put_line('Empleado: '||
                                        v_registro.NOMBRE);

END LOOP;
END;
```

Este ejemplo define una variable de tipo REF CURSOR, que es la que se emplea para recuperar la información de una instrucción SELECT que se ejecuta en SQL dinámico (EXECUTE IMMEDIATE). Asimismo, se utiliza la variable Cadena_sql para almacenar el texto correspondiente a la instrucción SELECT.

Dentro del bloque en primer lugar se define una consulta para recuperar el número de filas que tiene la tabla empleado (SELECT COUNT…). Tras asignar el resultado del conteo a la variable filas_afectadas se visualiza.

A continuación, se define otra consulta (SELECT *…), que recupera todos los registros (filas) de la tabla empleados. Para poder manejar esta información se realiza un bucle (LOOP … END LOOP) que recorre todos los resultados de la consulta, utilizando la variable v_cursor para recorrer los registros como un cursor.

Uso de parámetros en el SQL dinámico

El SQL dinámico permite la parametrización a través de variables host.

Una variable host es una variable que pertenece al programa que está ejecutando la sentencia SQL dinámica, y que podemos asignar en el interior de la sentencia SQL con la palabra clave USING.

Las variables host van precedidas de dos puntos ":".

El siguiente ejemplo muestra el uso de variables host para parametrizar una sentencia SQL dinámica.

```
DECLARE
nuevo_telefono hospital.telefono%type;
sentencia VARCHAR2(1000);

BEGIN

nuevo_telefono := 912001010;
sentencia := 'UPDATE HOSPITAL SET TELEFONO = :nuevo_telefono';
EXECUTE IMMEDIATE sentencia USING nuevo_telefono;
dbms_output.put_line('Este bloque ha actualizado '||SQL%ROWCOUNT||' filas.');

END;
```

Capítulo 6. CURSORES

¿QUÉ ES UN CURSOR?

Para poder procesar una instrucción del lenguaje SQL, Oracle asigna un área de memoria que recibe el nombre de área de contexto.

Esta área contiene información necesaria para completar el procesamiento, incluyendo el número de filas procesadas para la instrucción, un puntero a la versión analizada de la instrucción, y en el caso de las consultas, el conjunto activo, que es el conjunto de filas resultado de la consulta.

Un cursor es un puntero al área de contexto. Mediante los cursores un programa PL/SQL puede controlar el conjunto de valores que ha devuelto una instrucción SELECT.

Existen 3 tipos de cursores:

- Cursores explícitos (CURSOR).
- Cursores implícitos (SQL).
- Cursores dinámicos.

CURSORES EXPLÍCITOS

Un cursor explícito es aquel al que se le asigna explícitamente a una instrucción SELECT con un nombre, mediante la orden CURSOR…IS.

Todo cursor explícito necesita cuatro pasos para su procesamiento correcto:

1. Declaración del cursor (CURSOR ... IS).
2. Apertura del cursor (OPEN ...)
3. Recogida de los resultados en variables PL/SQL: (FETCH...)
4. Cierre del cursor (CLOSE...).

Declaración de un cursor

La sintaxis es la siguiente:

CURSOR <nombre_cursor> IS <operación de SELECT>;

Ejemplo

```
DECLARE
    CURSOR c_estudiantes IS
    SELECT cod_matricula, nombre FROM estudiantes;
```

Apertura de un cursor

La sintaxis es la siguiente:

OPEN <nombre_cursor>;

La sintaxis apertura de un cursor implica las siguientes acciones:

- Se examinan los valores de las variables acopladas, en caso de haberlas.
- Se determina el conjunto activo de valores.
- Se hace apuntar el puntero del conjunto a la primera fila.

Las variables acopladas se eximen en el momento de abrir el cursor y solo en ese momento.

Ejemplo

```
DECLARE
        V_salaID clase.sala_id%TYPE;
V_edificio salas.edificio%TYPE;
V_departamento clase.departamento%TYPE;
V_curso clase.curso%TYPE;
CURSOR c_edificios IS
SELECT edificio
FROM salas, clase
WHERE salas.sala_id = clase.sala_id
and departamento = v_departamento
and curso = v_curso;
BEGIN
-- Asignar valores a las variables de acoplamiento.
V_departamento := 'HIS';
V_curso := 101;

-- Abrimos el cursor.
OPEN c_edificios;
```

```
/* Reasignamos los valores de las variables de acoplamiento, lo que no tiene ningún efecto, ya que el cursor ya está abierto. */
        v_departamento := 'XXX';
        v_curso := -1;
END;
```

Extracción de los datos del cursor

Como se comentó en el capítulo anterior, la instrucción SELECT no permite devolver los valores a pantalla una vez extraídos, así que el resultado de ejecutar el cursor, tendrá que ser devuelto en algún sitio. Para eso se utiliza la instrucción FETCH que permite el envío de los datos de una fila del cursor a variables o a un registro.

Sintaxis:

FETCH <nombre del cursor> INTO <lista_variables | tipo registro>;

Por supuesto, para poder utilizar un FETCH previamente ha de estar abierto el cursor.

Ejemplo

```
DECLARE
V_matricula estudiantes.cod_matricula%TYPE;
V_nombre estudiantes.nombre%TYPE;
CURSOR c_estudiante IS
SELECT matricula, nombre FROM estudiantes;
BEGIN
OPEN c_estudiante;
FETCH c_estudiante INTO v_matricula, v_nombre;
END;
```

Ó

```
DECLARE
TYPE estudiante IS RECORD (
matricula estudiantes.cod_matricula%TYPE,
nombre estudiantes.nombre%TYPE
);
v_estudiante estudiante;
CURSOR c_estudiante IS
SELECT matricula, nombre FROM estudiantes;
```

```
BEGIN
OPEN c_estudiante;
FETCH c_estudiante INTO v_estudiante;
END;
```

Cierre de un cursor

Su sintaxis es:

CLOSE <nombre_cursor>;

Atributos de los cursores

A un cursor se le pueden aplicar los siguientes atributos:

- %FOUND
- %NOTFOUND
- %ISOPEN
- %ROWCOUNT

%FOUND

Devuelve TRUE si el último FETCH ejecutado sobre un cursor recuperó una fila.

%NOTFOUND

Devuelve TRUE si el último FETCH ejecutado no recuperó ninguna fila, como, por ejemplo, cuando se llega al final del cursor.

%isopen

Devuelve TRUE cuando el cursor asociado está abierto.

%ROWCOUNT

Devuelve el número de filas extraídas por el cursor hasta el momento.

Ejemplo

```
DECLARE
CURSOR c_estudiante IS
SELECT * FROM estudiantes
WHERE cod_matricula > 500;
V_matricula estudiantes%ROWTYPE;
BEGIN
OPEN c_estudiante;
LOOP
FETCH c_estudiante INTO v_estudiante;
EXIT WHEN c_estudiante%NOTFOUND;
……
END LOOP;
```

```
CLOSE c_estudiante;
END;
```

Cursores parametrizados

Se utilizan cuando la instrucción SELECT hace referencia en el WHERE a valores que se asignan mediante parámetros en el propio cursor.

Ejemplo

```
DECLARE
Num_matricula estudiantes.cod_matricula%TYPE;
CURSOR c_estudiante IS
SELECT * FROM estudiantes
WHERE cod_matricula > num_matricula;
v_estudiante c_estudiante%ROWTYPE;
BEGIN
OPEN c_estudiante;
END;
```

Esta definición se puede sustituir por:

```
DECLARE
        CURSOR c_estudiante(num_matricula
estudiantes.cod_matricula%TYPE)
        IS
SELECT * FROM estudiantes
WHERE cod_matricula > num_matricula;
v_estudiante c_estudiante%ROWTYPE;
BEGIN
OPEN c_estudiante(1000);
END;
```

CURSORES IMPLÍCITOS

Todo cursor explícito se realiza sobre un contexto de selección de registros previos. Un cursor implícito se puede utilizar sobre otras operaciones como, por ejemplo, UPDATE, DELETE o INSERT.

En estos cursores ya no se aplica el concepto de apertura, recorrido y cierre del cursor, ni el de nominación del mismo. Todos los cursores de este tipo se invocan con la palabra clave SQL.

Ejemplo

```
BEGIN
UPDATE estudiantes
SET nota = 9
        WHERE
NOMBRE = ‘Juan Rodriguez’;

IF SQL%NOTFOUND THEN
INSERT INTO estudiantes
                (cod_matricula, nombre, nota)
VALUES (1000,’Juan Rodriguez’,9);
END IF;
        -- Si no existe en el update, una fila para
        -- actualizar ejecuta el INSERT.
END;
```

CURSORES SQL DINÁMICO

Con SQL dinámico también podemos utilizar cursores.

Para utilizar un cursor dinámico solo debemos construir nuestra sentencia SELECT en una variable de tipo carácter y ejecutarla con EXECUTE IMMEDIATE, utilizando la palabra clave INTO, como se ha explicado en el capítulo 5.

EJEMPLO DE sql dinámico con una instrucción select

```
DECLARE
v_cursor REF CURSOR;
v_registro PAISES%ROWTYPE;
Cadena_sql VARCHAR2(1000);
V_Conteo NUMBER;

BEGIN
Cadena_sql := 'SELECT COUNT(*) FROM PAISES';
EXECUTE IMMEDIATE Cadena_sql INTO V_conteo;
Cadena_sql := 'SELECT * FROM PAISES';
OPEN v_cursor FOR Cadena_sql;
LOOP
FETCH v_cursor INTO v_registro;
EXIT WHEN v_cursor%NOTFOUND;
END LOOP;
END;
```

SUPUESTO PRÁCTICO 4: ***Resolución en el Anexo I de este manual.***

Crear un bloque sin nominar en SQL*PLUS que realice las siguientes operaciones:

- Se necesita incluir a todos los doctores dentro de la tabla de plantilla sanitaria para conseguir una estructura de información más lógica. Para realizar dicha operación, se deberá definir un cursor explícito que recupere, mediante una consulta sobre la tabla DOCTOR_HOSPITAL, todas las filas y columnas que tenga.
- Para insertar los doctores en la tabla PLANTILLA_SALA, se necesita un dato fundamental que no posee la tabla DOCTOR: el código de sala. Para solucionar este problema se van a insertar todos los doctores en la tabla PLANTILLA_SALA con el mismo número de sala: 99.
- Para poder insertar información en la tabla PLANTILLA_SALA, aparte del código del hospital y de la sala, se necesita el nif del doctor en cuestión que previamente deberá haberse insertado en la tabla PLANTILLA. Antes de insertarlo en esta última comprobaremos la existencia de dicho NIF en la tabla. En caso de existir no se realizará la inserción. Si es necesario hacer la inserción, los datos a

introducir serán los siguientes:
NIF = el nif del doctor
SALARIO = 0
NOMBRE = el nombre del doctor
TURNO = M
APELLIDOS = el apellido del doctor
FUNCION = la especialidad del doctor

- También hay que indicar que el código de sala 99 deberá previamente existir en la tabla SALA para el hospital concreto del doctor, por lo que antes de insertar el doctor en la tabla PLANTILLA_SALA, se deberá comprobar la existencia de una sala 99 para el hospital al que pertenece el doctor en la tabla SALA. En caso de que no exista dicha información habrá de insertarse una nueva fila en la tabla sala con la siguiente información:
HOSP_CODIGO = código del hospital al que pertenece el doctor
CODIGO = 99
NUMCAMAS = 0
NOMBRE = 'SALA DOCTORES'
- Cuando se necesite insertar en la tabla SALA, después de la inserción, se deberá mostrar por pantalla un mensaje con el siguiente formato ('Se ha introducido una sala 99 para el hospital'|| código del hospital al que pertenece el doctor que se ha insertado).
- Una vez que se hayan insertado todos los doctores en la tabla plantilla se deberá mostrar por pantalla un mensaje con el siguiente formato: (CHR(10)||'Se han insertado '||número efectivo de doctores insertados||' doctores en la tabla de empleados sanitarios de '||número posible de doctores a insertar||' posibles.').
- Se confirmarán las inserciones realizadas.
- Tras estas operaciones queremos visualizar en pantalla un listado con el nombre, apellidos y función de cada uno de los empleados sanitarios. Este listado se va a visualizar dentro del propio bloque mediante una consulta sobre la tabla correspondiente, pero para poderlo conseguir antes habrá de crearse un tipo tabla que almacene registros de 3 campos (nombre, apellidos y funcion) con el mismo tamaño y tipo que sus homólogos en la tabla de plantilla sanitaria. Y además un cursor que seleccione la información de la tabla de plantilla sanitaria y que se incluya dentro de cada celda del tipo tabla.
- Previo al propio listado se deberá mostrar una línea con el siguiente formato: (CHR(10)||'Se van a visualizar '||número de filas que posee la tabla de plantilla sanitaria||' empleados sanitarios.').
- Como línea de encabezado del listado se mostrará la siguiente:
('NOMBRE'||CHR(9)||'APELLIDOS'||CHR(9)||'FUNCIÓN').
- Para el correcto visionado del listado se deberá hacer por cada fila a visualizar una comprobación previa: comprobar la longitud del apellido de la persona. Si la longitud es = 4, se almacenará en una variable de tipo texto dos tabulaciones CHR(9). En caso contrario 1 sola.
- Cada línea mostrada seguirá el siguiente formato: Los 6 primeros caracteres del nombre||CHR(9)||los apellidos||las tabulaciones correspondientes del punto j||la función).

SUPUESTO PRÁCTICO 5: ***Resolución en el Anexo I de este manual.***

Crear un bloque sin nominar en SQL*PLUS que realice las siguientes operaciones:

- Realizar, mediante un bucle repetitivo, una inserción de 6 registros sobre la tabla DEPARTAMENTO_EMPLEADO, con la siguiente información:
EMP_NIF = 10000000A
DEPT_CODIGO = el valor de la variable que controla el bucle.

  En este bucle se controlará que cuando el código de departamento sea el 4, no se realice inserción en la tabla.
- Confirmar las inserciones realizadas.

- Diseñar un cursor que, para un código de departamento que se habrá de introducir por pantalla, seleccione los nombres de los empleados que trabajan en ellos. Será necesario acceder a las tablas DEPARTAMENTO_EMPLEADO y EMPLEADO.
- Diseñar un tipo tabla para almacenar el nombre de todos aquellos empleados que se recuperan mediante el cursor indicado en el punto anterior.
- Si no se encuentra ningún empleado en el departamento que se indique por pantalla, habrá de mostrarse la siguiente línea: (CHR(10)||'No se recuperó ningún empleado en el departamento con código '||código del departamento introducido); y terminará el programa.
- En caso contrario se mostrará una línea con la siguiente información (CHR(10)||'Se han encontrado '||número de empleados de la tabla empleados que pertenecen al departamento con código introducido||' empleados pertenecientes al departamento '||nombre del departamento cuyo código se ha introducido);.
- Además, habrá de mostrarse debajo todo los nombres que se han almacenado en el tipo tabla.
- Por último, habrán de realizarse 2 actualizaciones:
    - Sobre la tabla EMPLEADO (empleados no sanitarios) se actualizarán las columnas salario y comisión a valor 0 para cada uno de los nombres almacenados en el tipo tabla.
    - Sobre la tabla PLANTILLA (empleados sanitarios), se actualizará la columna salario a valor 0 para cada uno de los nombres coincidentes en el tipo tabla. En este último caso, se deberá controlar mediante un cursor implícito la posibilidad de que no existan empleados que coincidan en el nombre almacenado en el tipo tabla, en este último caso se mostrará la siguiente línea: (CHR(10)||'No se encontraron empleados sanitarios con el nombre de '||nombre almacenado en la celda correspondiente del tipo tabla). Para el caso de que si exista, se mostrará el siguiente mensaje: (CHR(10)||'Se han actualizado '||número de filas que se actualizan||' empleados sanitarios con el nombre de '||nombre almacenado en la celda correspondiente al tipo tabla.
- En el caso de que se ejecute el punto anterior, rechazar las actualizaciones que se hayan producido.

Capítulo 7. SUBPROGRAMAS

INTRODUCCIÓN

Los subprogramas, como se explicó en el primer capítulo, son bloques nominados que se almacenan en la base de datos.

Los bloques que se habían visto hasta ahora eran bloques anónimos que se compilaban cada vez que se ejecutaban y que no se almacenaban en la base de datos.

Los bloques nominados pueden ser de 4 tipos:

- Procedimientos.
- Funciones.
- Paquetes.
- Disparadores.

A los dos primeros, procedimientos y funciones, son los que se les conoce como subprogramas.

PROCEDIMIENTOS

Un procedimiento se compila una vez y se almacena en la base de datos.

Pueden ser invocados desde otro bloque PL/SQL tanto nominado como sin nominar.

Admite en su definición, el paso de parámetros tanto de entrada, salida, como entrada/salida.

Creación de un procedimiento

La sintaxis es la siguiente:

```
CREATE [OR REPLACE] PROCEDURE nombre_proced
[(argumento1 [{IN | OUT | IN OUT}] tipo,
...
argumentoX [{IN | OUT | IN OUT}] tipo)] {IS | AS}
cuerpo del procedimiento;
```

Tipos de parámetros en subprogramas

Los parámetros pueden ser de tres tipos:

- IN
- OUT
- IN OUT

La sintaxis para la definición de un parámetro dentro de un subprograma es la siguiente:

parametro [{IN | OUT | IN OUT}] tipo

Como se puede observar en la sintaxis, la indicación del tipo de parámetro (IN, OUT o IN OUT) es opcional. Si no se indica nada, se asume que el parámetro es de tipo IN (de entrada).

IN

Indica que es un parámetro de entrada.

Cuando se invoque al subprograma que lleve definido un parámetro de este tipo, habrá que pasarle el valor correspondiente. Al terminar la ejecución del subprograma, el parámetro definido con este tipo mantendrá el mismo valor que antes de su ejecución.

Por tanto, un parámetro de tipo IN, es de solo lectura.

out

Indica que es un parámetro de salida.

Cuando se invoque al subprograma que lleve definido un parámetro de este tipo, habrá que pasarle una variable definida en el objeto llamante. Al terminar la ejecución del subprograma, el parámetro definido con este tipo devolverá sobre la variable de llamada, el valor actualizado.

Por tanto, un parámetro de tipo OUT, es de solo escritura.

in out

Indica que es un parámetro de entrada y salida.

Cuando se invoque al subprograma que lleve definido un parámetro de este tipo, habrá que pasarle una variable definida en el objeto llamante. Al comenzar la ejecución del subprograma, utilizará como valor de comienzo el que se le ha pasado al subprograma a través de la variable de llamada, y al terminar la ejecución del subprograma, el parámetro definido con este tipo devolverá, sobre la variable de llamada, el valor actualizado.

Ejemplo

```
CREATE OR REPLACE PROCEDURE TEST
(V_lectura IN NUMBER,
V_escritura OUT NUMBER,
V_lectu_escri IN OUT NUMBER) IS

V_local NUMBER;

    BEGIN
    V_local := v_lectura; -- Correcto
    V_Lectura := 7; -- Error de compilación
    V_escritura := 7; -- Correcto
    V_local := v_escritura; -- Error de compilación
    V_local := v_lectu_escri; -- Correcto
    V_lectu_escri := 7; -- Correcto
    END;
```

Al compilar este procedimiento se producirían dos errores en las líneas . V_Lectura := 7 y V_local := v_escritura porque no se admite ni siquiera en compilación que una variable de tipo IN se le asigne un nuevo valor, ni que una variable de tipo OUT se utilice para asociarle su valor a otra.

En resumen cuando se definen parámetros en un subprograma hay que tener en cuenta las siguientes reglas:

- Un parámetro de tipo IN solo aparecerá en la parte derecha de una asignación.
- Un parámetro de tipo OUT solo aparecerá en la parte izquierda de una asignación.

- Un parámetro de tipo IN OUT aparecerá en la parte izquierda o derecha de una asignación.

Diferencia entre IS y AS

Como se indicó en la sintaxis de creación de un procedimiento, se puede especificar IS o AS antes de comenzar las líneas de código del bloque. Pero esta posibilidad presenta diferencias en su uso:

- Cuando se indica AS en la sintaxis de creación de un procedimiento no se pueden definir variables locales para utilizar en el procedimiento.

- Cuando se indica IS en la sintaxis de creación de un procedimiento sí que se pueden definir variables locales para utilizar en el procedimiento.

En la práctica cuando se está comenzando la definición de un procedimiento del que todavía no se tiene una visión global y concreta, se crea con la cláusula IS para no ponerse impedimentos en la creación o no de variables locales.

Ejemplo

```
CREATE OR REPLACE PROCEDURE Prueba(V_para1 IN NUMBER) IS
V_local NUMBER;
BEGIN
...
END;

CREATE OR REPLACE PROCEDURE Prueba(V_para1 IN NUMBER) AS
BEGIN
...
END;
```

Cuerpo del procedimiento

La sintaxis del cuerpo o bloque interno al procedimiento es la que se indica a continuación:

```
CREATE [OR REPLACE] PROCEDURE nombre_proced
[(parámetros)] {IS | AS}
[/* Sección declarativa */]

BEGIN
/* Sección ejecutable */
[EXCEPTION]
/* Control de errores */
END [nombre_proced];
```

La sección declarativa que aparece entre el IS|AS y el BEGIN de comienzo del bloque es opcional y en ella se definen todas las variables y cursores locales al procedimiento.

La indicación de parámetros en la creación del bloque es opcional.

La sección de control de errores es opcional.

Por último, la indicación del nombre del procedimiento al final del mismo (después del END) es también opcional.

Llamada a un procedimiento dentro de un bloque

Para invocar la ejecución de un procedimiento desde un bloque PL/SQL, se utiliza la siguiente sintaxis:

nombre_proced [(parámetro1, …, parámetroX)]

Ejemplo

```
CREATE OR REPLACE PROCEDURE Prueba(Var IN number) IS
BEGIN
...
END;
```

Tomando este ejemplo de procedimiento, una posible llamada que se podría hacer al mismo desde un bloque PL/SQL sería la siguiente:

```
DECLARE
Var NUMBER(5) := 10;
BEGIN
Prueba(var);
END;
```

Llamada a un procedimiento desde SQL*PLUS

Para invocar la ejecución de un procedimiento directamente desde la línea de comandos de SQL*PLUS, se utiliza la siguiente sintaxis:

{EXECUTE | EXEC} nombre_procedimiento;

Ejemplo

SQL> VARIABLE variable1;

SQL> EXECUTE simple(:variable1);

En este ejemplo se define la variable denominada VARIABLE1 y luego se invoca la ejecución del procedimiento SIMPLE, utilizando la instrucción EXECUTE y pasándole como parámetro al procedimiento la variable VARIABLE1 creada anteriormente.

Restricciones sobre los parámetros

La declaración de parámetros de un procedimiento se realiza obligatoriamente sin indicar el tamaño de los tipos. Es decir, se indica el nombre del parámetro, el tipo de parámetro (entrada, salida o entrada/salida) de forma opcional, y el tipo sin el tamaño.

El tamaño del parámetro se asume por la variable de llamada al subprograma.
Ejemplo

```
CREATE OR REPLACE PROCEDURE Prueba3
(Var1 IN OUT number,
var2 IN OUT VARCHAR2) AS
BEGIN
        Var1 := 5;
        Select to_number(var2) into var1 from dual;
END;

DECLARE
Local1 NUMBER(5);
Local2 VARCHAR2(10);
BEGIN
Local1 := 10918;
Local2 := 'JA';
Prueba3(local1,local2);
Local2 := 'PP';
END;
```

El parámetro Var1 asume como tamaño el de la variable Local1 que tiene NUMBER(5). De la misma forma Var2 asume como tamaño el de la variable Local2 que tiene VARCHAR2(10).

Todas las reglas que se han estudiado en la definición de los tipos de datos son de aplicación a los tipos que se definen en los parámetros de los subprogramas a excepción del tamaño, según lo que se ha explicado anteriormente.
Ejemplo

```
CREATE OR REPLACE PROCEDURE Prueba
(Para1 estudiantes.nombre%TYPE,
Para2 estudiantes.edad%TYPE DEFAULT 50,
Para3 VARCHAR2 DEFAULT ‘holA’) AS
BEGIN
...
END;

DECLARE
...
BEGIN
Prueba ('JOSE');
END;
```

Esta llamada es válida porque los otros dos parámetros: Para2 y Para3 tienen valores por defecto y se asumen estos valores en la llamada al procedimiento.

Alteración y borrado de procedimientos

Una vez creado y almacenado el procedimiento, se puede borrar utilizando la siguiente sintaxis:

```
DROP PROCEDURE nombre_procedimiento;
```

Para alterar un procedimiento creado y almacenado previamente, hay que borrarlo y volverlo a crear, salvo que indiquemos CREATE OR REPLACE PROCEDURE que realiza esta operación automáticamente, reemplazando el código del procedimiento almacenado por el nuevo código.

Notación posicional y nominal de los parámetros

Lo normal es que la llamada a un procedimiento con parámetros se realice en el orden natural en el que se han definido dentro de la cabecera del procedimiento. A este método se le conoce como notación posicional y es la forma que hemos visto hasta el momento en los ejemplos anteriores.

Pero existe una segunda forma de invocar los parámetros de un subprograma que consiste en indicar el nombre del parámetro y su variable asignada. A esta metodología se le denomina notación nominal.

Ejemplo de notación nominal

```
CREATE OR REPLACE PROCEDURE Prueba
(Para1 VARCHAR2,
Para2 NUMBER,
Para3 DATE) AS
BEGIN
...
END;

Sql>BEGIN
Prueba(Para2 => var2,
Para3 => var3,
Para1 => var1);
END;
```

SUPUESTO PRÁCTICO 6: ***Resolución en el Anexo I de este manual.***

1) **Diseñar un procedimiento que convierta el salario anual que existe en la tabla plantilla en un salario mensual, teniendo en cuenta que los miembros de la tabla plantilla solo cobran 12 pagas. Para el diseño de este procedimiento, se seguirán estas pautas:**

- Solo se podrá actualizar el importe a aquellos miembros de la tabla plantilla cuyo salario anual sea igual o superior al máximo que haya en la tabla plantilla.
- La actualización no se hará a todos los miembros de la tabla plantilla cada vez, sino a aquellos que comiencen por una letra determinada o en caso contrario pertenezcan a un turno concreto. Estos datos serán pasados al procedimiento mediante parámetros.
- El procedimiento devolverá el total de filas actualizadas o 0 en caso de que no se actualice ninguna.
- Se mostrarán los siguientes mensajes dependiendo de que se realice una u otra acción:
    - **Si se actualizan filas por la letra de comienzo del nombre**: 'Se han actualizado '||nº de filas

actualizadas||' filas, correspondientes a la plantilla sanitaria cuya letra comienza por la letra '||la letra que se le ha pasado al procedimiento.

- **Si se actualizan filas por el turno**: 'Se han actualizado '||nº de filas actualizadas||' filas, correspondientes a la plantilla sanitaria cuyo turno es '||el turno que se le pasó al procedimiento.
- **Si no se actualizan filas porque no se cumple ninguna de las condiciones anteriores de acuerdo a los datos de la tabla plantilla**: 'No se ha podido actualizar ningún empleado de la plantilla sanitaria cuya letra comience por '||la letra que se le pasó al procedimiento y el mensaje
- **En caso contrario**: 'No se ha podido actualizar ningún empleado de la plantilla sanitaria cuya turno sea '||el turno que se le pasó al procedimiento.

2) **Diseñar un bloque sin nominar en SQL*PLUS que realice la llamada al procedimiento creado anteriormente pasándole como parámetros los siguientes valores:**

LETRA = 'B'
TURNO = 'T'

FUNCIONES

Una función se estructura de la misma manera que un procedimiento, utilizando un bloque, así como parámetros en la cabecera.

La diferencia principal con los procedimientos es que una función siempre devuelve un valor por sí misma; por tanto, la llamada siempre tiene que estar asignada a una variable que recogerá el valor que devuelva la función.

La sintaxis es la siguiente:

```
CREATE [OR REPLACE] FUNCTION nombre_funcion
[(argumento1 [{IN | OUT | IN OUT}] tipo,
...
Argumento [{IN | OUT | IN OUT}] tipo)]
RETURN tipo {IS | AS}
Cuerpo_función.
```

La orden RETURN

Se utiliza dentro del cuerpo de la función y se aplica para devolver el control y opcionalmente un valor a la llamada de la función.

La sintaxis es la siguiente:

RETURN [expresion];

Si no se indica un valor para la expresión que sigue a RETURN, devolverá el control a la llamada de la función sin más.

Ejemplo

```
CREATE OR REPLACE FUNCTION Prueba
(Para1 NUMBER,
Para2 VARCHAR2)
RETURN VARCHAR2 AS
BEGIN
        ...
RETURN ‘Hola’;
END;


DECLARE
Saludo VARCHAR2(10);
BEGIN
Saludo := Prueba(10,'ADIOS');
END;
```

Eliminación y alteración de funciones

Una vez creada y almacenada una función se puede borrar utilizando la siguiente sintaxis:

DROP FUNCTION nombre_funcion;

Para alterar una función creada y almacenada previamente, hay que borrarla y volverla a crear salvo que indiquemos CREATE OR REPLACE FUNTION que realiza esta operación automáticamente reemplazando el código anterior por el nuevo.

Llamada a una función dentro de un bloque PL/SQL

Como toda función devuelve un valor, la llamada a una función dentro de un bloque PL/SQL se realiza mediante una asignación a una variable con la siguiente sintaxis:

Variable := nombre_funcion;

Llamada a una función en una instrucción SELECT

También es posible realizar una llamada a una función dentro de una instrucción SELECT, dentro del SELECT…FROM como si fuese otra columna más a visualizar o incluso en la sección WHERE como otro condicionante.

Ejemplo de llamada a una funciÓn en una instrucción select combinada con columnas

```
SELECT nombre, apellidos, SYSDATE FROM empleados;
```

En este ejemplo se seleccionan todas las filas de la tabla empleados, mostrándose únicamente el nombre y apellidos de los empleados más la fecha actual del sistema invocando la llamada a la función SYSDATE. Esta función es genérica del lenguaje PL/SQL y retorna la fecha y hora del sistema donde se encuentra instalada la base de datos.

Ejemplo de llamada a una funciÓn dentro del where del select

```
SELECT nombre, apellidos FROM empleados
WHERE f_baja = SYSDATE
```

En este ejemplo se seleccionan aquellas filas de la tabla empleados que cumplan que la fecha de baja del empleado sea igual a la fecha actual (SYSDATE), mostrándose únicamente el nombre y apellidos de los empleados.

Ejemplo de llamada única a una funcion dentro del select

```
SELECT SYSDATE FROM DUAL;
```

En este ejemplo únicamente se muestra la fecha actual del sistema. Como no hay acceso a información de ninguna tabla, se utiliza la pseudotabla DUAL para completar la sentencia SELECT y mostrar el valor de la función SYSDATE.

Tablas del sistema asociadas

Los procedimientos y funciones como las tablas y demás objetos se almacenan en la base de datos y como tal en el diccionario de datos.

Las vistas del diccionario que almacenan datos de los subprogramas son las siguientes:

- USER_OBJECTS
- USER_SOURCE
- USER_ERRORS

USER_OBJECTS

Esta tabla del sistema almacena el nombre y tipo de todos los objetos del usuario conectado a la base de datos.

USER_SOURCE

Esta tabla del sistema almacena el código fuente de los subprogramas del usuario conectado a la base de datos.

USER_ERRORS

Esta tabla del sistema almacena los errores (línea y error) de los subprogramas que han producido algún error en compilación.

Ejemplo

```
SQL> CREATE OR REPLACE PROCEDURE SIMPLE IS
2 V_COUNTER NUMBER;
3 BEGIN
4 V_COUNTER := 7;
5 END;
6 /
```

Creamos un procedimiento denominado SIMPLE.

```
SQL> DESC USER_OBJECTS;
Name Null? Type
------------------------------ -------- -------------
```

```
OBJECT_NAME VARCHAR2(128)
SUBOBJECT_NAME VARCHAR2(30)
OBJECT_ID NUMBER
DATA_OBJECT_ID NUMBER
OBJECT_TYPE VARCHAR2(19)
CREATED DATE
LAST_DDL_TIME DATE
TIMESTAMP VARCHAR2(19)
STATUS VARCHAR2(7)
TEMPORARY VARCHAR2(1)
GENERATED VARCHAR2(1)
SECONDARY VARCHAR2(1)
NAMESPACE NUMBER
EDITION_NAME VARCHAR2(30)
```

Invocamos la instrucción DESC (equivalente a DESCRIBE) para visualizar la lista de columnas de la tabla USER_OBJECTS.

```
SQL> SELECT OBJECT_NAME, OBJECT_TYPE, STATUS
2 FROM USER_OBJECTS WHERE OBJECT_NAME = 'SIMPLE';

OBJECT_NAME OBJECT_TYPE STATUS
------------------- ------------------ -----------
SIMPLE PROCEDURE VALID
```

Se seleccionan las columnas OBJECT_NAME, OBJECT_TYPE y STATUS de la tabla USER_OBJECTS para el objeto con nombre (OBJECT_NAME) 'SIMPLE'. El resultado de la consulta devuelve otra vez el nombre del objeto, el tipo de objeto (en este caso PROCEDURE) y el estado de compilación del objeto: VALID.

Un STATUS VALID equivale a que el objeto no tiene errores de compilación, mientras que un STATUS INVALID indica que el objeto en cuestión tiene errores de compilación y, por tanto, no podrá ser ejecutado.

```
SQL> SELECT TEXT FROM USER_SOURCE
2 WHERE NAME = 'SIMPLE' ORDER BY LINE;

TEXT
----------------------------------------------------
```

```
PROCEDURE SIMPLE IS
V_COUNTER NUMBER;
BEGIN
V_COUNTER := 7;
END;
```

En este caso se consulta la columna TEXT de la tabla USER_SOURCE para obtener el código fuente del objeto 'SIMPLE'.

```
SQL> CREATE OR REPLACE PROCEDURE SIMPLE2 IS
2 V_COUNTER NUMBER
3 BEGIN
4 V_COUNTER = 7;
5 END;
6 /

Warning: Procedure created with compilation errors.
```

Ahora se crea un procedimiento SIMPLE2 con errores de compilación. Esto nos lo informa el sistema a través del mensaje Warning: Procedure created with compilation errors.

```
SQL> SELECT LINE, POSITION, TEXT
2 FROM USER_ERRORS
3 WHERE NAME = 'SIMPLE2'
4 ORDER BY SEQUENCE;

LINE POSITION
---------- ----------
TEXT
------------------------------------------------------------
3 1
PLS-00103: Encountered the symbol "BEGIN" when expecting one of the
following:

:= . ( @ % ; not null range default character
The symbol ":= was inserted before "BEGIN" to continue.

5 1
```

PLS-00103: Encountered the symbol "END" when expecting one of the following:

LINE POSITION
---------- ----------
TEXT
----
begin function pragma procedure subtype type <an identifier>
<a double-quoted delimited-identifier> current cursor delete
exists prior

En este caso consultamos las columnas LINE, POSITION y TEXT de la tabla USER_ERRORS para mostrar dónde se encuentran los errores de compilación del procedimiento SIMPLE2 y cuál es el motivo del error de compilación.

Situaciones que provocan el estado INVALID

Cuando se modifica un objeto de tipo tabla para añadir o borrar columnas, e incluso para modificar el tamaño o tipo de las mismas, todos los subprogramas que hagan referencia a este objeto se ponen a estado INVALID en la tabla USER_OBJECTS.

Cuando se recrea un procedimiento o función y se generan errores de compilación en el mismo, todos los subprogramas que invoquen a este automáticamente se ponen a estado INVALID en la tabla USER_OBJECTS.

Llamada a una función desde SQL*PLUS

Para invocar la ejecución de una función dentro del SQL*PLUS sin utilizar una instrucción SELECT, se utiliza la siguiente sintaxis:

SQL> VARIABLE nombre;

SQL> :nombre := función;

SUPUESTO PRÁCTICO 7: ***Resolución en el Anexo I de este manual.***

1) **Diseñar una función recursiva para el cálculo del factorial de un número:**

- Solo admitirá valores enteros.
- En caso de introducir un valor negativo, deberá mostrar el siguiente mensaje 'No se consideran valores negativos para el cálculo del factorial'. Además, la función devolverá como valor -1.
- En el resto de los casos devolverá el resultado del cálculo del factorial.
- El máximo número que se admitirá para el cálculo del factorial será el 33. Números superiores a este se rechazarán con el siguiente mensaje: 'No se consideran valores SUPERIORES a 33 para el cálculo del factorial' y la función devolverá valor 0.

2) **Para probar la función, se pueden utilizar los 3 métodos posibles:**
   a) **Mediante un consulta SELECT.**
   b) **Ejecutándose mediante un bloque sin nominar con declaración de variables privadas.**
   c) Ejecutándose mediante un bloque sin nominar con declaración de variables globales.

Capítulo 8. PAQUETES

INTRODUCCIÓN

Los paquetes son el tercer tipo de bloque PL/SQL nominado después de procedimientos y funciones que se han visto en el capítulo anterior.

El concepto de paquete ha sido introducido por Oracle de acuerdo a las características que tienen los paquetes de lenguaje ADA. De hecho, muchas de las estructuras del PL/SQL provienen de este lenguaje.

Un paquete es una estructura que permite almacenar juntos una serie de objetos relacionados.

Un paquete tiene 2 partes diferenciadas: la especificación y el cuerpo del paquete.

A diferencia de los procedimientos y funciones, los paquetes no pueden estar contenidos dentro de un bloque local, solo se pueden almacenar en la base de datos.

Un paquete es, en esencia, una sección declarativa nominada. Cualquier cosa que pueda incluirse en la sección declarativa de un bloque puede incluirse también en un paquete. Esto abarca procedimientos, funciones, cursores, tipos y variables. Una ventaja de incluir estos objetos en un paquete es la posibilidad de referenciarlos desde otros bloques PL/SQL, con lo que los paquetes permiten disponer de variables globales de PL/SQL.

ESPECIFICACIÓN O CABECERA DEL PAQUETE

Contiene la información acerca del contenido del paquete, pero no el código de ejecución de los procedimientos y funciones que se definen en él.

La sintaxis es la siguiente:

```
CREATE [OR REPLACE] PACKAGE nombre_paquete {IS | AS}
especificación_procedimiento |
especificación_función |
declaración_variable |
definición_tipo |
declaración_excepción |
declaración_cursor
END;
```

Los elementos de la cabecera de un paquete pueden aparecer en cualquier orden salvo porque el elemento referenciado tiene que ser definido antes de utilizarse.

Las declaraciones de procedimientos y funciones solo incluirán la cabecera de los mismos, donde se indica el tipo de objeto (procedimiento o función), el nombre y los parámetros que utilizan (nombre, tipo y utilización del parámetro).

Las variables y tipos definidos en la cabecera del paquete se podrán utilizar en cualquiera de los procedimientos y funciones que se utilicen en el mismo.

Asimismo, esos tipos podrán utilizarse por procedimientos y funciones ajenos al paquete, siempre y cuando se referencie el nombre del paquete y el tipo o variable a utilizar del mismo.

Ejemplo

```
CREATE OR REPLACE PACKAGE paqueteA IS
    FUNCTION calculaA (param1 NUMBER) RETURN VARCHAR2;
    PROCEDURE calculaB(param1 IN OUT NUMBER);
    v_estudiante students%ROWTYPE;
    TYPE t_nombres IS TABLE OF v_estudiante
    INDEX BY BINARY_INTEGER;
```

```
        TYPE t_textos IS TABLE OF VARCHAR2
        INDEX BY BINARY_INTEGER;
END;
```

cuerpo DEL PAQUETE

Contiene el código ejecutable de todos los procedimientos y funciones que se hayan definido en el cabecera del paquete.

No se puede compilar el cuerpo sin haber compilado antes la cabecera.

No puede dejarse sin definir el código ejecutable de ninguno de los procedimientos o funciones que se hayan definido en la cabecera porque generaría errores de compilación en el cuerpo.

Sí es posible definir procedimientos y funciones privadas al cuerpo del paquete que no se haya definido en la cabecera, lo que permite que los procedimientos del cuerpo llamen a otros que solo se hayan definido en este espacio. Los procedimientos y funciones exclusivos del cuerpo (privados) no podrán ser invocados por objetos externos al paquete.

La sintaxis es la siguiente:

```
    CREATE [OR REPLACE] PACKAGE BODY nombre_paquete
    {IS | AS}
especificación_y_codigo_procedimiento |
especifacion_y_codigo_funcion
[declaración_de_var_locales]
[BEGIN
Código_ejecutable_local_al_paquete]
END;
```

Ejemplo

```
CREATE OR REPLACE PACKAGE BODY paqueteA IS

    FUNCTION calculaA (param1 NUMBER) RETURN VARCHAR2
    IS BEGIN
    RETURN to_char(param1);
    END;

    PROCEDURE calculaB(param1 IN OUT NUMBER) IS
```

```
        BEGIN
        param1 := param1 * 10;
        END;
END paqueteA;
```

REFERENCIANDO A LOS PAQUETES

Cuando se quiere nombrar a un elemento cualquiera de los que aparece en la cabecera del mismo hay que hacerlo utilizando la siguiente sintaxis:

Nombre_del_paquete.nombre_del_elemento [(param1…paramn)]

Ejemplo

```
SQL> DECLARE
        var1 NUMBER(5);
        var2 VARCHAR2(6);
        var3 paqueteA.t_textos;
    BEGIN
        paqueteA.calculaB(var1);
        var2 := paqueteA.calculaA(var1);
        var3(1) := var3;
    END;
```

En este ejemplo en primer lugar se define una variable VAR3 del mismo tipo que la variable T_TEXTOS del PAQUETEA. A continuación, en la parte ejecutable del bloque se invoca la ejecución del procedimiento CALCULAB perteneciente al PAQUETEA, y también se asigna el resultado de ejecutar la función CALCULAA del PAQUETEA a la variable VAR2.

Ejemplo

```
SQL> SELECT paquetea.calculaA(10) FROM DUAL;
```

En este otro ejemplo se invoca la ejecución de la función CALCULAA del PAQUETEA, utilizando una instrucción SELECT… FROM DUAL.

INICIALIZACIÓN DE UN PAQUETE

Cuando se referencia a cualquiera de los elementos de un paquete por primera vez en una sesión, es posible ejecutar de forma predeterminada un código que se arrancará antes de ejecutar el objeto referenciado. Para conseguirlo hay que insertar dicho código ejecutable en la sección BEGIN…END propia del cuerpo del paquete.

Ejemplo

```
CREATE OR REPLACE PACKAGE BODY paqueteA IS
   var_local NUMBER(5);

   FUNCTION calculaA(param1 NUMBER DEFAULT var_local)
   RETURN VARCHAR2 IS
   BEGIN
   RETURN to_char(param1);
   END;

   PROCEDURE calculaB(param1 IN OUT NUMBER) IS
   BEGIN
   Param1 := param1 * 10;
   END;
BEGIN
Var_local := 10;
END;
```

Cuando se invoque por primera vez a cualquiera de los objetos del paquete dentro de una sesión, antes se inicializará la variable VAR_LOCAL a valor 10.

SOBRECARGA DE PAQUETES

Dentro de un paquete pueden sobrecargarse los procedimientos y funciones; es decir, puede haber más de un procedimiento o función con el mismo nombre pero con distintos parámetros.

Ejemplo

```
CREATE OR REPLACE PACKAGE PRUEBAS AS
   PROCEDURE MAS_ESTUDIANTES(CODIGO IN NUMBER);
   PROCEDURE MAS_ESTUDIANTES(NOMBRE IN VARCHAR2,
   APELLIDOS IN VARCHAR2);
END PRUEBAS;
```

Este es un ejemplo de sobrecarga de un paquete en el que se han definido dos procedimientos con el mismo nombre pero con distinto número de parámetros, por lo que PL/SQL los considera diferentes.

Restricciones de la sobrecarga

A continuación, se muestran una serie de reglas que restringen la sobrecarga de objetos dentro de un paquete:

1. No se pueden sobrecargar dos subprogramas si sus parámetros solo difieren en el tipo de parámetro: entrada, salida o entrada/salida. A continuación, se muestra un ejemplo erróneo de sobrecarga:

```
CREATE OR REPLACE PACKAGE ERROR_SOBRECARGA AS
        PROCEDURE prueba (p_param IN NUMBER);
        PROCEDURE prueba (p_param OUT NUMBER);
END;
```

2. No se pueden sobrecargar dos funciones basándose solo en su tipo de dato de retorno. A continuación, se muestra un ejemplo erróneo de sobrecarga:

```
CREATE OR REPLACE PACKAGE ERROR_SOBRECARGA AS
        FUNCTION prueba RETURN DATE;
        FUNCTION prueba RETURN NUMBER;
END;
```

3. No se pueden sobrecargar subprogramas en los que exista igualdad del número, nombre de los parámetros y familia del tipo de parámetro, y únicamente difiera el tipo del parámetro dentro de la misma familia. A continuación, se muestra un ejemplo erróneo de sobrecarga:

```
CREATE OR REPLACE PACKAGE ERROR_SOBRECARGA AS
        PROCEDURE prueba (p_param1 CHAR);
        PROCEDURE prueba (p_param1 VARCHAR2);
END;
```

4. No se pueden sobrecargar subprogramas en los que solo difiera el nombre del parámetro, sin cambiar el modo o el tipo del mismo. A continuación, se muestra un ejemplo erróneo de sobrecarga:

```
CREATE OR REPLACE PACKAGE ERROR_SOBRECARGA AS
```

```
        PROCEDURE prueba (p_param1 IN NUMBER);
        PROCEDURE prueba (p_var1 IN NUMBER);
END;
```

DEPENDENCIAS

Las dependencias nos permiten cambiar el cuerpo del paquete sin tener que cambiar la cabecera del mismo, de forma que no será necesario recompilar otros objetos que dependan de la cabecera del paquete.

Si lo que se recompila o modifica es la cabecera del paquete, el cuerpo del mismo queda automáticamente en estado INVALID puesto que depende ella.

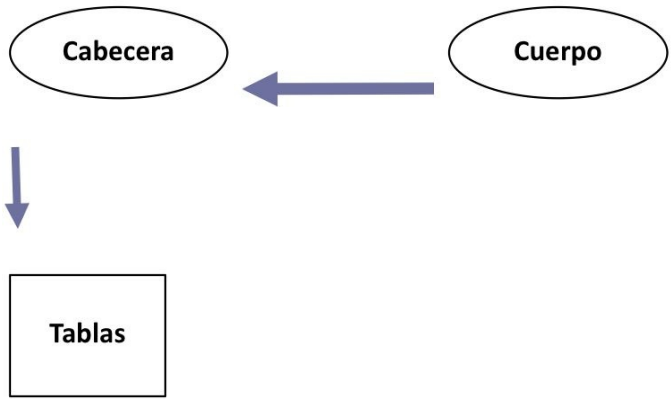


Fig. 8-1 Dependencia entre cabecera y cuerpo de un paquete con las tablas.

Esta figura no muestra la dependencia directa que existe entre la cabecera y el cuerpo de un paquete, así como entre el cuerpo y las tablas de la base de datos.

Repercusión del estado de los objetos con un paquete

Partiendo de la base de las dependencias que existen entre los elementos de un paquete y entre este y las tablas, se pueden presentar las siguientes casuísticas que pueden derivar en un cambio de estado en los objetos dependientes como se indica a continuación:

1. Si se cambia el cuerpo de un paquete sin cambiar la cabecera de procedimientos y funciones incluidos en el mismo, la cabecera del paquete permanecerá como VALID (válida).

2. Si se cambia el cuerpo del paquete cambiando la cabecera de algún procedimiento o función, la cabecera del paquete pasará a estar INVALID (inválida).

3. Si se alteran tablas de la base de datos que afectan al cuerpo del paquete, este quedará automáticamente en estado INVALID, mientras que la cabecera del paquete continuará como VALID.

4. Si se borra el cuerpo del paquete, la cabecera permanecerá como VALID.

Ejemplo de la repercusión del estado de objetos

```
SQL> CREATE TABLE SIMPLE
        (campo1 NUMBER(10));

SQL> CREATE OR REPLACE PACKAGE prueba IS
PROCEDURE ejemplo (p_val IN NUMBER);
END prueba;

SQL> CREATE OR REPLACE PACKAGE BODY prueba IS
PROCEDURE ejemplo(p_val IN NUMBER) IS
BEGIN
INSERT INTO simple VALUES (p_val);
COMMIT;
END ejemplo;
END prueba;
```

```
SQL> CREATE OR REPLACE PROCEDURE llamada
        (p_val IN NUMBER) IS
BEGIN
Prueba.ejemplo(p_val + 1);
END;

SQL> -- Alteramos el cuerpo del paquete cambiando VALUES
-- (p_val) por VALUES (p_val - 1);

CREATE OR REPLACE PACKAGE BODY prueba IS
PROCEDURE ejemplo(p_val IN NUMBER) IS
BEGIN
INSERT INTO simple VALUES (p_val - 1);
COMMIT;
END ejemplo;
END prueba;
```

Los resultados que obtenemos al consultar la tabla USER_OBJECTS para los objetos de este ejemplo son los siguientes:

```
OBJECT_NAME OBJECT_TYPE STATUS
----------- ----------- -------
PRUEBA PACKAGE BODY VALID
PRUEBA PACKAGE VALID
SIMPLE TABLE VALID

OBJECT_NAME OBJECT_TYPE STATUS
----------- ----------- -------
        LLAMADA PROCEDURE VALID

SQL> DROP TABLE SIMPLE;
```

Los resultados que obtenemos al consultar la tabla USER_OBJECTS para los objetos de este ejemplo son los siguientes:

```
        OBJECT_NAME OBJECT_TYPE STATUS
----------- ----------- -------
PRUEBA PACKAGE BODY INVALID
PRUEBA PACKAGE VALID
LLAMADA PROCEDURE VALID
```

```
SQL> DROP PACKAGE BODY PRUEBA;
```

Los resultados que obtenemos al consultar la tabla USER_OBJECTS para los objetos de este ejemplo son los siguientes:

```
        OBJECT_NAME OBJECT_TYPE STATUS
----------- ----------- -------
PRUEBA PACKAGE VALID
LLAMADA PROCEDURE VALID
```

```
SQL> -- Alteramos la cabecera del procedimiento cambiando
-- (p_val IN NUMBER) por (p_val IN CHAR).

CREATE OR REPLACE PACKAGE prueba IS
PROCEDURE ejemplo (p_val IN CHAR);
END prueba;
```

Los resultados que obtenemos al consultar la tabla USER_OBJECTS para los objetos de este ejemplo son los siguientes:

```
        OBJECT_NAME OBJECT_TYPE STATUS
----------- ----------- -------
PRUEBA PACKAGE VALID
LLAMADA PROCEDURE INVALID
```

SUPUESTO PRÁCTICO 8: ***Resolución en el Anexo I de este manual.***

Crear un paquete (cabecera y cuerpo) denominado SUPUESTO8 que incluya en su interior la funcionalidad capaz de realizar las operaciones siguientes en funciones o procedimientos separados:

- Calcular el factorial de un número.
- Dado un número en pesetas convertirlo en euros.
- Dado un número en euros convertirlo a pesetas.
- Dado un número en dígitos (de un máximo de 7 dígitos y 3 decimales) convertirlo a carácter con los separadores de miles y decimales correspondientes, teniendo en cuenta lo siguiente:
    - **9**: Indica que se sustituirá por un dígito.
    - **G**: Indica que se sustituirá por el separador de miles.
    - **D**: Indica que se sustituirá por el separador decimal.

      Ejemplo: Dado 5325.33 el paso a carácter sería TO_CHAR(5325.33, '9G999D99') con resultado 5.325,33.

- Indicar una variable constante con el valor de conversión de pesetas a euros, y viceversa: 166.386.

Nota: En las conversiones se utilizará el redondeo y para ello se utilizará la función ROUND que opera de la siguiente manera: ROUND (numero_a_redondear, dígitos_decimales_del_resultado).

Ej: ROUND (166.386,2) = 166.39

SUPUESTO PRÁCTICO 9: ***Resolución en el Anexo I de este manual.***

Crear un paquete (cabecera y cuerpo) denominado DEVUELVEREGISTRO, que incluya en su interior, función y un procedimiento para realizar las siguientes operaciones:

- La función devolverá como resultado un elemento con formato tipo tabla. Para ello, consultará todos los miembros de plantilla sanitaria cuyo código de hospital coincida con el que se le haya pasado como parámetro de entrada a la función. Cada miembro se irá almacenando con toda su información (el registro o fila entera) dentro de una celda distinta del tipo tabla. Y una vez recorrido todo el cursor se devolverá el tipo tabla entero.
- Un procedimiento que admita como parámetro de entrada un código de hospital y que con el mismo ejecute la función anterior. El resultado de la función (un tipo tabla entero) lo manejará este procedimiento con objeto de listar el nombre y apellidos de todos los miembros de la plantilla que pertenecían al hospital indicado.
- Para ejecutar este paquete, se ejecutará el procedimiento anterior indicando el código del hospital que se quiere consultar para obtener la lista de su plantilla sanitaria.

Capítulo 9. PAQUETES PREDETERMINADOS

INTRODUCCIÓN

En este capítulo se relacionan los paquetes que se instalan de forma predeterminada con una base de datos Oracle 11g, y también se desarrollan algunos de los paquetes que más habitualmente se utilizan en la programación contra bases de datos Oracle.

Para consultar en más detalle el resto de paquetes, se puede acceder a la guía oficial de Oracle *PL/SQL Package and Types Reference 11g Rel2.*

lista de paquetes predeterminados

Todos los paquetes que se relacionan a continuación pueden ser invocados por cualquier usuario con permisos para ejecutar procedimientos (GRANT RESOURCE), porque son creados con sinónimos públicos.

Nombre paquete	Descripción
APEX_CUSTOM_AUTH	Proporciona una interfaz para la autenticación y gestión de la sesión.
APEX_APPLICATION	Habilita a los usuarios para obtener las ventajas de uso de las variables globales.
APEX_ITEM	Habilita a los usuarios a crear elementos dinámicos de formulario basados en consultas SQL en vez de crear elementos individuales página a página.
APEX_UTIL	Proporciona utilidades para obtener y configurar el estado de la sesión, obteniendo ficheros, chequeando autorizaciones para usuario, reseteando diferentes estados para usuarios, y también obteniendo y configurando preferencias para usuarios.
CTX_ADM	Permite administrar servidores y diccionarios de datos.
CTX_CLS	Permite generar reglas CTXRULE para un conjunto de documentos.
CTX_DDL	Permite crear y gestionar las preferencias, listas de secciones y grupos requeridos para los índices Text.
CTX_DOC	Permite solicitar servicios de documentos.
CTX_OUTPUT	Permite gestionar el log indexado.
CTX_QUERY	Permite generar consultas, contadores y crear expresiones de consultas almacenadas.
CTX_REPORT	Permite crear varios informes indexados.
CTX_THES	Permite gestionar el buscador.
CTX_ULEXER	Se utiliza con el USER-LEXER.
DBMS_ADDM	Facilita el uso de Advisor.
DBMS_ADVANCED_REWRITE	Permite interceptar sentencias SQL y reemplazarlas por sentencias alternativas.
DBMS_ADVISOR	Parte del SQLAccess Advisor, un sistema experto que identifica y te ayuda a resolver los problemas relacionados con la ejecución de sentencias SQL.
DBMS_ALERT	Proporciona soporte para la notificación asíncrona de los eventos de base de datos.
DBMS_APPLICATION_INFO	Permite registrar un nombre de aplicación con la base de datos para auditarla.

DBMS_APPLY_ADM	Proporciona procedimientos administrativos para arrancar, parar y configurar un proceso de aplicación.
DBMS_AQ	Permite añadir un mensaje (de un tipo de objeto predefinido) en una consulta.
DBMS_AQADM	Permite administrar funciones en una consulta para mensajes de tipos de objetos predefinidos.
DBMS_AQELM	Proporciona procedimientos para gestionar la configuración de la notificación asíncrona de Advanced Queuing por e-mail y HTTP.
DBMS_AQIN	Juega un parte fundamental en el acceso de seguridad de la interfaz de Oracle JMS.
DBMS_ASSERT	Proporciona una interfaz para validar las propiedades de los valores de entrada.
DBMS_AUTO_TASK_ADMIN	Usado por el Administrador de base de datos, junto con el Enterprise Manager permite acceder a los controles AUTOTASK.
DBMS_AUTO_TASK_IMMEDIATE	Consta de un subprograma cuya función es iniciar el optimizador de estadísticas que están retrasadas.
DBMS_AW_STATS	Contiene un subprograma que genera y almacena las estadísticas del optimizador para cubos y dimensiones.
DBMS_CAPTURE_ADM	Describe procedimientos administrativos para arrancar, parar y configurar un proceso de captura. Se usa en STREAMS.
DBMS_CDC_PUBLISH	Identifica nuevos datos que han sido añadidos a, modificados o borrados de, tablas relacionales y publicados los datos cambiados en un formulario que es utilizado por una aplicación.
DBMS_COMPARISON	Proporciona interfaces para comparar objetos de base de datos entre diferentes bases de datos.
DBMS_COMPRESSION	Proporciona una interfaz para facilitar la elección del nivel correcto de comprensión para una aplicación.
DBMS_CONNECTION_POOL	Proporciona una interfaz para gestionar Database Resident Connection Pool.
DBMS_CQ_NOTIFICATION	Es parte de un conjunto de características que los clientes usan para recibir notificaciones cuando el conjunto de resultados de una consulta ha cambiado.
DBMS_CRYPTO	Permite encriptar y descifrar datos almacenados.
DBMS_CSX_ADMIN	Proporciona una interfaz para personalizar la configuración cuando se transporta un tablespace que contiene datos binarios XML.
DBMS_CUBE	Contiene subprogramas para crear cubos OLAP y dimensiones.
DBMS_CUBE_ADVISE	Contiene subprogramas para evaluar las

	vistas materializadas de cubos.
DBMS_DATA_MINING	Implementa la interfaz de Oracle Data Mining para su gestión.
DBMS_DATA_MINING_TRANSFORM	Proporciona subrutinas que pueden ser usadas para preparar los datos para Oracle Data Mining.
DBMS_DATAPUMP	Permite mover todo, o parte de una base de datos entre bases de datos, incluyendo tanto datos como metadatos.
DBMS_DBFS_CONTENT	Proporciona una interfaz para uno o más Store Providers.
DBMS_DBFS_CONTENT_SPI	Proporciona la especificación API para los proveedores de servicio del paquete DBMS_DBFS_CONTENT.
DBMS_DBFS_HS	Proporciona a los usuarios la habilidad para usar los servicios de Amazon S3 Web.
DBMS_DBFS_SFS	Proporciona una interfaz para operar con SFS (SecureFile-based) almacenados descritos en el paquete DBMS_DBFS_CONTENT.
DBMS_DB_VERSION	Especifica el número de versión de Oracle y otra información útil para la compilación basada en la selección de la versión de Oracle.
DBMS_DDL	Proporciona acceso a algunas sentencias SQL DDL desde procedimientos almacenados.
DBMS_DEBUG	Implementa herramientas de debug para las rutinas implementadas en PL/SQL.
DBMS_DEFER	Proporciona una interfaz de usuario para la replicación transaccional. Requiere tener instalado Distributed Option.
DBMS_DEFER_QUERY	Permite consultar la cola de procesos RPC (Remote Procedure Calls). Requiere tener instalado Distributed Option.
DBMS_DEFER_SYS	Proporciona al administrador del sistema una interfaz para la replicación transaccional. Requiere tener instalado Distributed Option.
DBMS_DESCRIBE	Describe los argumentos de un procedimiento almacenado.
DBMS_DG	Permite a las aplicaciones la notificación de la base de datos primaria en un entorno de Oracle Data Guard.
DBMS_DIMENSION	Habilita para verificar las relaciones entre dimensiones y proporcionar una alternativa a Enterprise Manager Dimension Wizard para visualizar la definición de las dimensiones.
DBMS_DISTRIBUTED_TRUST_ADMIN	Mantiene Trusted Database List, que es usado para determinar si un enlace de base de datos de un servidor particular puede ser aceptado.
DBMS_EPG	Implementa la pasarela de PL/SQL embebido que habilita a un explorador web para invocar procedimientos PL/SQL almacenados a través de un listener HTTP.

DBMS_ERRLOG	Proporciona un procedimiento que te habilita para crear una tabla de errores capturados de modo que las operaciones DML pueden continuar después de ser encontrados en vez de abortarse la ejecución y realizar rollback.
DBMS_EXPFIL	Contiene todos los procedimientos usados para gestionar el conjunto de atributos, conjunto de expresiones, índices, estadísticas optimizadas y privilegios para Expression Filter.
DBMS_FGA	Proporciona funciones de seguridad.
DBMS_FILE_GROUP	Es uno de los paquetes que permite gestionar los Streams.
DBMS_FILE_TRANSFER	Permite copiar un fichero binario dentro de una base de datos o transferir un fichero binario entre bases de datos.
DBMS_FLASHBACK	Permite crear un FDA (Flachback Data Archive).
DBMS_FLASHBACK_ARCHIVE	Contiene procedimientos para disociar y reasociar un archivo FDA.
DBMS_FREQUENT_ITEMSET	Habilita el conteo de conjuntos de ítems frecuentes.
DBMS_HM	Contiene constantes y declaración de procedimiento para Health Check Management.
DBMS_HPROF	Proporciona una interfaz para la ejecución de aplicaciones PL/SQL.
DBMS_HS_PARALLEL	Habilita el procesamiento paralelo.
DBMS_HS_PASSTHROUGH	Permite usar Heterogeneous Services.
DBMS_IOT	Crea una tabla con referencias a las filas de una Index Organized Table.
DBMS_JAVA	Proporciona una interfaz PL/SQL para acceder a bases de datos con funcionalidad desde Java.
DBMS_JOB	Gestiona los trabajos en cola.
DBMS_LDAP	Proporciona funciones y procedimientos para acceder a los datos de los servidores LDAP.
DBMS_LDAP_UTL	Proporciona la utilidad Oracle Extension para LDAP.
DBMS_LIBCACHE	Prepara la librería caché de una instancia Oracle.
DBMS_LOB	Proporciona rutinas de propósito general para operaciones con tipos de datos LOB: BLOP, CLOB y BFILES.
DBMS_LOCK	Permite gestionar los servicios de Oracle Lock Management.
DBMS_LOGMNR	Proporciona funciones para inicializar y ejecutar el lector Log.
DBMS_LOGMNR_D	Consulta las tablas del diccionario de una base de datos.
DBMS_LOGSTDBY	Describe procedimiento para configurar y gestionar el Standby lógico de un entorno de

	base de datos.
DBMS_METADATA	Permite a los objetos llamadores recuperar definiciones completas de objetos metadato del diccionario.
DBMS_METADA_DIFF	Contiene una interfaz para comparar dos documentos de metadatos en formato SXML.
DBMS_MGD_ID_UTL	Proporciona un conjunto de utilidades para subprogramas.
DBMS_MGWADM	Describe los tipos de objeto, métodos, constantes y subprogramas que trabajan con Messaging Gateway. Se usa en Advanced Queuing.
DBMS_MONITOR	Permite usar PL/SQL para monitorizar.
DBMS_MVIEW	Permite refrescar Snapshots.
DBMS_NETWORK_ACL_ADMIN	Proporciona la interfaz para administrar ACL (Access Control List).
DBMS_NETWORK_UTL	Proporciona la interfaz para administrar ACL (Access Control List).
DBMS_OBFUSCATION_TOOLKIT	Proporciona procedimientos para Data Encryption Standards.
DBMS_ODCI	Devuelve el coste de CPU consumida por una función de usuario.
DBMS_OFFLINE_OG	Proporciona una interfaz pública para referenciación offline.
DBMS_OLAP	Proporciona procedimientos para gestión de OLAP.
DBMS_OUTLN	Proporciona la interfaz para procedimientos y funciones asociadas con OUTLN_PKG.
DBMS_OUTPUT	Acumula información en un buffer de modo que pueda ser recuperada posteriormente.
DBMS_PARALLEL_EXECUTE	Habilita al usuario para incrementar la actualización de datos de tablas en paralelo.
DBMS_PCLXUTIL	Se usa con índices particionados.
DBMS_PIPE	Proporciona un servicio de canal DBMS que habilita mensajes para ser enviados entre sesiones.
DBMS_PREDICTIVE_ANALYTICS	Proporciona subrutinas que implementan operaciones sobre Data Mining.
DBMS_PREPROCESSOR	Proporciona una interfaz para imprimir o recuperar el código fuente de una unidad PL/SQL.
DBMS_PROFILER	Proporciona una API Probe Profiler.
DBMS_PROPAGATION_ADM	Proporciona procedimientos administrativos para configurar la propagación de una consulta.
DBMS_RANDOM	Proporciona un generador de número aleatorios.
DBMS_RECTIFIER_DIFF	Proporciona una interfaz para detectar y resolver inconsistencias entre dos sitios replicados.

DBMS_REDEFINITION	Permite realizar una reorganización de tablas online.
DBMS_REFRESH	Permite crear grupos de Snapshots que pueden ser refrescados juntos.
DBMS_REPAIR	Proporciona un procedimiento para reparar datos corruptos.
DBMS_REPCAT	Proporciona rutinas para administrar y actualizar el catálogo de replicación.
DBMS_REPCAT_ADMIN	Permite crear usuarios con los privilegios necesarios para facilitar la replicación simétrica.
DBMS_REPCAT_INSTANTIATE	Son plantillas de desarrollo. Requiere Replication Option.
DBMS_REPCAT_RGT	Son plantillas para refrescar grupos. Requiere Replication Option.
DBMS_REPUTIL	Proporciona rutinas para generar tablas, triggers y paquetes para la replicación de tablas.
DBMS_RESCONFIG	Proporciona una interfaz para operar con Resource Configuration List.
DBMS_RESOURCE_MANAGER	Mantiene planes, grupos de consumo y directivas sobre planes de ejecución.
DBMS_RESOURCE_MANAGER_PRIVS	Mantiene privilegios asociados con los grupos de consumidores de recursos.
DBMS_RESULT_CACHE	Proporciona una interfaz para operar en Result Cache.
DBMS_RESUMABLE	Permite suspender operaciones que llevan largo tiempo en ejecución.
DBMS_RLMGR	Contiene varios procedimientos para crear y gestionar reglas para Rules Manager.
DBMS_RLS	Proporciona una interfaz para administrar niveles de seguridad.
DBMS_ROWID	Proporciona procedimientos para crear rowids y para interpretar su contenido.
DBMS_RULE	Describe el procedimiento EVALUATE usado en los Streams.
DBMS_RULE_ADM	Describe la interfaz administrativa para crear y gestionar reglas.
DBMS_SCHEDULER	Proporciona una colección de funciones del gestor de cola de trabajos.
DBMS_SERVER_ALERT	Permite gestionar alertas.
DBMS_SERVICE	Permite crear, borrar, activar y desactivar servicios para una instancia individual.
DBMS_SESSION	Proporciona acceso a la sentencia SQL ALTER SESSION.
DBMS_SHARED_POOL	Permite mantener objetos en la memoria compartida.
DBMS_SPACE	Proporciona información sobre los segmentos de espacio.
	Proporciona administración sobre tablespaces

DBMS_SPACE_ADMIN	y segmentos.
DBMS_SPM	Soporta la gestión de planes SQL.
DBMS_SQL	Permite usar SQL dinámico.
DBMS_SQLDIAG	Proporciona una interfaz para SQL Diagnosability.
DBMS_SQLPA	Proporciona una interfaz para implementar el SQL Perfomance Analyzer.
DBMS_SQLTUNE	Proporciona la interfaz para sentencias SQL con mejora de rendimiento (tune).
DBMS_STAT_FUNCS	Proporciona funciones estadísticas.
DBMS_STATS	Proporciona un mecanismo a los usuarios para visualizar y modificar las estadísticas.
DBMS_STORAGE_MAP	Comunica con FMON para invocar operaciones de mapas.
DBMS_STREAMS	Describe la interfaz para convertir SYS.AnyData objectos en LCR objetos.
DBMS_STREAMS_ADMIN	Describe procedimientos administrativos para añadir y borrar reglas simples.
DBMS_STREAMS_ADVISOR_ADM	Proporciona una interfaz para obtener información de Oracle Streams.
DBMS_STREAMS_AUTH	Proporciona una interfaz para la gestión de privilegios sobre Streams.
DBMS_STREAMS_HANDLER_ADM	Proporciona una interfaz para encolar mensajes.
DBMS_STREAMS_MESSAGING	Proporciona una interfaz para encolar mensajes.
DBMS_STREAMS_TABLESPACE_ADM	Proporciona procedimientos administrativos para copiar tablespaces entre bases de datos.
DBMS_TDB	Informa si una base de datos puede ser transportada entre plataformas usando RMAN.
DBMS_TRACE	Proporciona rutinas para arrancar y parar el trazador PL/SQL.
DBMS_TRANSACTION	Proporciona acceso a sentencias de transacción SQL.
DBMS_TRANSFORM	Proporciona una interfaz para formatear los mensajes en Oracle Advanced Queuing.
DBMS_TTS	Chequeo sobre conjuntos transportables.
DBMS_TYPES	Consta de constantes que representan las built-in de Oracle y tipos definidos por el usuario.
DBMS_UTILITY	Proporciona varias rutinas.
DBMS_WARNING	Proporciona una interfaz para consultar, modificar y borrar las configuraciones del sistema o la sesión.
DBMS_WM	Describe cómo usar la interfaz para Oracle Database Workspace Manager.
DBMS_WORKLOAD_CAPTURE	Configura el sistema Workload Capture.
DBMS_WORLOAD_REPLAY	Proporciona una interfaz para reproducir e

	imprimir un registro del Workload.
DBMS_WORKLOAD_REPOSITORY	Permite gestionar el Workload Repository.
DBMS_XA	Contiene la interfaz de aplicaciones para llamar a la interfaz XA para PL/SQL.
DBMS_XDB	Describe Resource Managenent.
DBMS_XDB_ADMIN	Proporciona una interfaz para implementar operaciones de administración de XMLIndex.
DBMS_XDBRESOURCE	Proporciona una interfaz para operar con recursos de XDB.
DBMS_XDB_VERSION	Describe la versión de interfaz.
DBMS_XDBT	Describe cómo un administrador puede crear un índice ConText.
DBMS_XDBZ	Controla el repositorio de Oracle XML DB.
DBMS_XEVENT	Proporciona tipos relacionados con eventos y soporta subprogramas.
DBMS_XMLDOM	Explica el acceso a los objetos XMLType.
DBMS_XMLGEN	Convierte los resultados de una consulta SQL a formato XML.
DBMS_XMLINDEX	Proporciona una interfaz para implementar índices asíncronos.
DBMS_XMLPARSER	Explica el acceso a los contenidos y estructuras de documentos XML.
DBMS_XMLQUERY	Proporciona funcionalidad en el sentido base de datos a XMLType.
DBMS_XMLSAVE	Proporciona funcionalidad en el sentido XML a tipos de base de datos.
DBMS_XMLSHEMA	Explica procedimientos para registrar y borrar esquemas XML.
DBMS_XMLSTORE	Proporciona las habilidades para almacenar datos XML en tablas relacionales.
DBMS_XMLTRANSLATIONS	Proporciona una interfaz para realizar traducciones de modo que las cadenas de caracteres puedan ser buscadas o visualizadas en varios lenguajes.
DBMS_XPLAN	Describe cómo formatear la salida del comando EXPLAIN PLAN.
DBMS_XSLPROCESSOR	Explica el acceso a los contenidos y estructura de documento XML.
DEBUG_EXTPROC	Permite revisar (debug) procedimientos externos.
HTF	Es una función Hypertext para generar tags HTML.
HTTP	Son procedimientos Hypertext para generar tags HTML.
ORD_DICOM	Soporta la gestión y manipulación de imágenes digitales.
ORD_DICOM_ADMIN	Es utilizado por Oracle Multimedia Digital.
OWA_CACHE	Proporciona una interfaz que habilita la caché de PL/SQL.
	Proporciona una interfaz para enviar y recibir

OWA_COOKIE	cookies http desde exploradores del cliente.
OWA_CUSTOM	Proporciona funciones de Global PLSQL Agent Authorization.
OWA_IMAGE	Proporciona una interfaz para acceder a las coordenadas en las que el usuario pulsa una imagen.
OWA_OPT_LOCK	Contiene subprogramas con estrategias para prevenir actualizaciones perdidas.
OWA_PATTERN	Proporciona una interfaz para localizar patrones textuales dentro de cadenas de caracteres.
OWA_SEC	Proporciona una interfaz para autenticación del cliente.
OWA_TEXT	Contiene subprogramas usados por OWA_PATTERN para manipular cadenas de caracteres.
OWA_UTIL	Contiene subprogramas de utilidad.
SDO_CS	Proporciona funciones para coordinar transformaciones del sistema.
SDO_CSW_PROCESS	Contiene subprogramas para soportar CSW (Catalog Services for the Web).
SDO_GCDR	Contiene subprogramas de Oracle Spatial.
SDO_GEOM	Proporciona funciones para implementar operaciones relacionadas con GeoRaster.
SDO_GEOR	Contiene funciones y procedimientos de GeoRaster.
SDO_GEOR_ADMIN	Contiene subprogramas para administrar operaciones de GeoRaster.
SDO_GEOR_UTIL	Contiene funciones y procedimientos de utilidad para Spatial GeoRaster.
SDO_LRS	Proporciona funciones para soportar sistemas de referenciación lineal.
SDO_MIGRATE	Proporciona funciones para migrar datos espaciales de versiones anteriores.
SDO_NET	Proporciona funciones y procedimientos para trabajar con datos modelados y enlaces en una red.
SDO_NET_MEM	Contiene funciones y procedimientos para editar y analizar operaciones en redes.
SDO_OLS	Contiene funciones y procedimientos para editar y analizar operaciones en redes.
SDO_PC_PKG	Contiene subprogramas que soportan el uso de nubes de punto en Oracle Spatial.
SDO_SAM	Contiene funciones y procedimientos para analizar datos espaciales.
SDO_TIN_PKG	Contiene subprogramas que soportan el uso de redes trianguladas irregulares (TINs) en Spatial.
SDO_TOPO	Proporciona procedimientos para crear y manejar topologías de Spatial.
	Contiene programas para editar topología de

SDO_TOPO_MAP	Spatial.
SDO_TUNE	Proporciona funciones para el manejo de esquemas de índices espaciales.
SDO_UTIL	Proporciona funciones de utilidad y procedimientos para Oracle Spatial.
SDO_WFS_LOCK	Contiene subprogramas para soportar WFS.
SDO_WFS_PROC	Proporciona funciones de utilidad y procedimientos para Oracle Spatial.
SEM_APIS	Contiene subprogramas para trabajar con el Resource Description Framework (RDF) en OWL (Ontology Language).
SEM_PERF	Contiene subprogramas para examinar y mejorar el RDF.
SEM_RDFCTX	Contiene subprogramas para manejar políticas de extracción y semántica de índices creados para los documentos.
SEM_RDFSA	Contiene subprogramas para proporcionar control de acceso a los datos RDF.
UTL_COLL	Habilita programas PL/SQL que usan localizadores de colecciones de datos para consulta y actualización.
UTL_COMPRESS	Proporciona un conjunto de utilidades de compresión de datos.
UTL_DBWS	Proporciona servicios web de base de datos.
UTL_ENCODE	Proporciona funciones para codificar datos RAW.
UTL_FILE	Habilita tus programas PL/SQL para leer y escribir ficheros de texto.
UTL_HTTP	Habilita llamadas http desde PL/SQL y SQL.
UTL_I18N	Proporciona un conjunto de servicios que ayudan a los desarrolladores a construir aplicaciones multilenguaje.
UTL_INADDR	Proporciona un procedimiento para soportar direccionamiento de Internet.
UTL_IDENT	Especifica lo que la base de datos o un cliente PL/SQL está ejecutando.
UTL_LMS	Recupera y formatea mensajes de error en diferentes lenguajes.
UTL_MAIL	Es una utilidad para manejar email.
UTL_NLA	Expone un subconjunto de BLAS y LAPACK.
UTL_RAW	Proporciona funciones SQL para manipular tipos de datos RAW.
UTL_RECOMP	Recompila módulos inválidos de PL/SQL.
UTL_REF	Habilita a un programa PL/SQL para acceder a un objeto proporcionando una referencia al mismo.
UTL_SMTP	Proporcionar funcionalidad PL/SQL para enviar emails.
UTL_SPADV	Proporciona subprogramas para recoger y

	analizar estadísticas para Oracle Streams.
UTL_TCP	Proporciona funcionalidad PL/SQL para soportar comunicaciones simples basadas en TCP/IP.
UTL_URL	Proporciona mecanismos para caracteres URL.
WPG_DOCLOAD	Proporciona una interfaz para descargar ficheros de tipo BLOB y BFILE.

DBMS_DB_VERSION

El paquete DBMS_DB_VERSION especifica el número de versión de Oracle y otra información útil para la compilación condicional basadas en la versión de Oracle.

Constantes

El paquete DBMS_DB_VERSION contiene diferentes constantes para las distintas versiones de las bases de datos Oracle, que se muestran en la siguiente tabla:

Nombre	Tipo dato	Valor	Descripción
VERSION	PLS_INTEGER	10	Versión actual.
RELEASE	PLS_INTEGER	2	Revisión actual.
VER_LE_9	BOOLEAN	FALSE	Versión <= 9
VER_LE_9_1	BOOLEAN	FALSE	Versión <= 9 y revisión <= 1
VER_LE_9_2	BOOLEAN	FALSE	Versión <= 9 y revisión <= 2
VER_LE_10	BOOLEAN	TRUE	Versión <= 10
VER_LE_10_1	BOOLEAN	FALSE	Versión <= 10 y revisión <= 1
VER_LE_10_2	BOOLEAN	TRUE	Versión <= 10 y revisión <= 2
VER_LE_11	BOOLEAN	FALSE	Versión <= 11
VER_LE_11_1	BOOLEAN	TRUE	Versión <= 11 y revisión <= 1

Ejemplo

```
CREATE OR REPLACE PROCEDURE compila_condicional IS
SUBTYPE t_tipo IS
$IF DBMS_DB_VERSION.VER_LE_9 $THEN NUMBER
$ELSE VARCHAR(10)
$END;

Var1 CONSTANT t_tipo := $IF DBMS_DB_VERSION.VER_LE_9
$THEN 123456
$ELSE 'Hola'
$END;

PROCEDURE prueba(p1 IN t_tipo) IS
BEGIN
DBMS_OUTPUT.PUT_LINE ('p1 = '||p1);
END prueba;


BEGIN
```

```
Prueba(var1);
END compila_condicional;

SQL> SET SERVEROUTPUT ON;

SQL> EXECUTE COMPILA_CONDICIONAL;
p1 = Hola

PL/SQL procedure successfully completed.
```

DBMS_FILE_TRANSFER

El paquete DBMS_FILE_TRANSFER proporciona procedimientos para copiar archivos binarios dentro de una base de datos o a transferirlos entre bases de datos diferentes.

Este paquete consta de los siguientes subprogramas:

Nombre Subprograma	Tipo	Descripción
COPY_FILE	Procedimiento	Lee un fichero del directorio origen y crea una copia en el directorio destino.
GET_FILE	Procedimiento	Contacta con una base de datos remota para leer un fichero remoto y crear una copia del mismo en el file system local o ASM.
PUT_FILE	Procedimiento	Lee un fichero local o ASM y contacta con una base de datos remota para crear una copia del mismo en el file system remoto.

Procedimiento COPY_FILE

Este procedimiento lee un fichero de un directorio origen y crea una copia del mismo en un directorio destino. Tanto el directorio origen como destino están en un FILE SYSTEM local o en un grupo de discos ASM (Automatic Storage Management).

Se puede copiar cualquier tipo de fichero a y desde un FILE SYSTEM local. Sin embargo, solo se pueden copiar ficheros de base de datos (tales como DATAFILES, TEMPFILES, CONTROLFILES, etc.), a y desde un grupo de discos ASM.

El fichero destino no se cerrará hasta que el procedimiento se termine correctamente.

SINTAXIS

La sintaxis de este procedimiento es la siguiente:

```
DBMS_FILE_TRANSFER.COPY_FILE(
        Objeto_directorio_origen IN VARCHAR2,
        Nombre_fichero_origen IN VARCHAR2,
        Objeto_directorio_destino IN VARCHAR2,
        Nombre_fichero_destino IN VARCHAR2);
```

DESCRIPCIÓN DE LOS parámetros

A continuación, se describe cada uno de los parámetros que se usan en el procedimiento COPY_FILE.

objeto_directorio_origen

Es el objeto directorio que se designa como origen. El objeto directorio debe existir ya.

Para crear un objeto directorio, hay que utilizar el comando CREATE DIRECTORY.

nombre_fichero_origen

Es el nombre del fichero que se quiere copiar. Este fichero debe existir en el

directorio origen.

objeto_directorio_destino

Es el objeto directorio que se designa como destino. El objeto directorio debe existir ya. Si el destino es ASM, el objeto directorio deberá designar también un nombre de grupo (por ejemplo, +diskgroup1) o un directorio creado a través de un alias. En el caso de un directorio, se debe especificar la ruta completa (por ejemplo: +diskgroup1/dbs/control).

nombre_fichero_destino

El nombre que se asigna al fichero en el directorio destino. No puede existir un fichero con el mismo nombre en el directorio destino.

RESTRICCIONES DE USO DEL PROCEDIMIENTO

Para poder ejecutar correctamente este procedimiento, el usuario que lo invoca debe tener los siguientes privilegios:

- Privilegio de lectura (READ) en el objeto directorio especificado como origen.
- Privilegio de escritura (WRITE) en el objeto directorio especificado como destino.

Este procedimiento convierte los nombres de objeto directorio indicado en los parámetros a mayúsculas a menos que se especifiquen entre dobles comillas. Sin embargo, el procedimiento no hace esto mismo con los nombres de fichero.

Los ficheros a copiar deben cumplir los siguientes requerimientos:

- El tamaño de los ficheros a copiar debe ser un múltiplo de 512 bytes.
- El tamaño de los ficheros a copiar debe ser menor o igual a 2 terabytes.

La transferencia del fichero no es transaccional. El fichero copiado es tratado como un fichero binario, y no se realiza ninguna conversión de juegos de caracteres. Para monitorizar el progreso de fichero de gran tamaño que se está copiando, se puede consultar la vista dinámica V$SESSION_LONGOPS.

EJEMPLO

```
SQL> CREATE DIRECTORY DGROUP AS '+diskgroup1/dbs/backup';
```

Directory create.

```
SQL> BEGIN
DBMS_FILE_TRANSFER.COPY_FILE('SOURCEDIR',
                                        't_xdbtmp.f',
                                        'DGROUP',
                                        't_xdbtmp.f');
END;
/

PL/SQL procedure successfully completed.

SQL> EXIT

$ASMCMD
ASMCMD> ls
DISKGROUP1/
ASMCMD> cd diskgroup1/dbs/backup
ASMCMD> ls
t_xdbtmp.f => +DISKGROUP1/ORCL/TEMPFILE/COPY_FILE.267.546
```

Procedimiento GET_FILE

Este procedimiento contacta con una base de datos remota para leer un fichero remoto y crear una copia del mismo en el FILE SYSTEM local o ASM. El fichero que es copiado es el fichero origen, y el nuevo fichero que resulta de la copia es el fichero destino.

El fichero destino no se cerrará hasta que el procedimiento se termine correctamente.

SINTAXIS

La sintaxis de este procedimiento es la siguiente:

```
DBMS_FILE_TRANSFER.GET_FILE(
        Objeto_directorio_origen IN VARCHAR2,
        Nombre_fichero_origen IN VARCHAR2,
        Base_datos_origen IN VARCHAR2,
        Objeto_directorio_destino IN VARCHAR2,
        Nombre_fichero_destino IN VARCHAR2);
```

DESCRIPCIÓN DE LOS parámetros

A continuación, se describe cada uno de los parámetros que se usan en el procedimiento GET_FILE.

objeto_directorio_origen

Es el objeto directorio que se designa como origen. El objeto directorio debe existir ya.

nombre_fichero_origen

Es el nombre del fichero que se quiere copiar. Este fichero debe existir en el directorio origen.

base_datos_origen

Es el nombre de una base de datos para enlazar a una base de datos remota donde se localiza el fichero.

objeto_directorio_destino

Es el objeto directorio que se designa como destino.

nombre_fichero_destino

Es el nombre que se asigna al fichero en el directorio destino. No puede existir un fichero con el mismo nombre en el directorio destino.

RESTRICCIONES DE USO DEL PROCEDIMIENTO

Las mismas que se indicaron en el procedimiento anterior COPY_FILE.

Procedimiento PUT_FILE

Este procedimiento lee un fichero local o ASM y contacta con una base de datos remota para crear una copia del mismo en un FILE SYSTEM remoto. El fichero que es copiado es el fichero origen, y el nuevo fichero que resulta de la copia es el fichero destino. El fichero destino no se cerrará hasta que el procedimiento se termine correctamente.

SINTAXIS

La sintaxis de este procedimiento es la siguiente:

```
DBMS_FILE_TRANSFER.PUT_FILE(
        Objeto_directorio_origen IN VARCHAR2,
        Nombre_fichero_origen IN VARCHAR2,
        Objeto_directorio_destino IN VARCHAR2,
        Nombre_fichero_destino IN VARCHAR2,
        Base_datos_destino IN VARCHAR2);
```

DESCRIPCIÓN DE LOS parámetros

A continuación, se describe cada uno de los parámetros que se usan en el procedimiento PUT_FILE.

objeto_directorio_origen

Es el objeto directorio que se designa como origen. El objeto directorio debe existir ya.

nombre_fichero_origen

Es el nombre del fichero que se quiere copiar. Este fichero debe existir en el directorio origen.

objeto_directorio_destino

Es el objeto directorio que se designa como destino.

nombre_fichero_destino

Es el nombre que se asigna al fichero en el directorio destino. No puede existir un fichero con el mismo nombre en el directorio destino.

base_datos_destino

Es el nombre de una base de datos para enlazar a una base de datos remota en la que se copiará el fichero.

RESTRICCIONES DE USO DEL PROCEDIMIENTO

Las mismas que se indicaron en el procedimiento COPY_FILE.

DBMS_OUTPUT

El paquete DBMS_OUTPUT se utiliza normalmente por el usuario para realizar procesos de revisión del código (debug) o para visualizar mensajes.

Para visualizar los mensajes que se lanzan con el procedimiento PUT_LINE de este paquete dentro de SQL*PLUS, antes hay que ejecutar el comando SET SERVEROUPUT ON que tiene el mismo efecto que invocar a DBMS_OUTPUT.ENABLE (buffer_size => NULL). En ambos casos no hay límite en la salida del buffer.

Este paquete consta de los siguientes subprogramas:

Nombre Subprograma	Tipo	Descripción
DISABLE	Procedimiento	Deshabilita la salida de mensajes.
ENABLE	Procedimiento	Habilita la salida de mensajes.
GET_LINE	Procedimiento	Recupera una línea del buffer.
GET_LINES	Procedimiento	Recupera un array de líneas del buffer.
NEW_LINE	Procedimiento	Termina una línea creada con PUT.
PUT	Procedimiento	Coloca una línea parcial en el buffer.
PUT_LINE	Procedimiento	Coloca una línea en el buffer.

Errores de la sección EXCEPTION

Los subprogramas de DBMS_OUTPUT pueden generar errores de aplicación del tipo ORA-20000, y los mensajes que se definen con los procedimientos pueden devolver los siguientes errores:

Error	Descripción
ORU-10027	*Buffer overflow*. Sobrepasado el tamaño del buffer de salida de mensajes.
ORU-10028	*Line length overflow.* Sobrepasado la longitud de salida de una línea.

Procedimiento DISABLE

Este procedimiento deshabilita las llamadas a PUT, PUT_LINE, NEW_LINE, GET_LINE y GET_LINES y limpia el buffer de cualquier información que aún permaneciese en el mismo.

SINTAXIS

La sintaxis de este procedimiento es la siguiente:

```
DBMS_OUTPUT.DISABLE;
```

Procedimiento ENABLE

Este procedimiento habilita las llamadas a PUT, PUT_LINE, NEW_LINE, GET_LINE y GET_LINES. Las llamadas a estos procedimientos son ignoradas si el paquete DBMS_OUTPUT no está activado.

SINTAXIS

La sintaxis de este procedimiento es la siguiente:

```
DBMS_OUTPUT.ENABLE(Tamaño_buffer IN INTEGER DEFAULT
20000);
```

DESCRIPCIÓN DE LOS parámetros

A continuación, se describe cada uno de los parámetros que se usan en el procedimiento ENABLE.

tamaño_buffer

Es el límite, en bytes, de la información que puede contener el buffer. Si se indica NULL para este parámetro indica que no debería haber límite.

RESTRICCIONES DE USO DEL PROCEDIMIENTO

No es necesario llamar a este procedimiento cuando se usa la instrucción SET SERVEROUTPUT ON en SQL*Plus.

Si se producen varias llamadas a este procedimiento, el tamaño del buffer quedará fijado al valor de la última llamada. El tamaño máximo es de 1.000.000 y el mínimo es de 2.000.

Normalmente se espera un NULL para el parámetro del tamaño del buffer de forma que por defecto asuma 20.000, por compatibilidad con versiones anteriores de las bases de datos Oracle que no soportaban un buffer ilimitado.

Procedimiento GET_LINE

Este procedimiento recupera una única línea de información del buffer.

SINTAXIS

La sintaxis de este procedimiento es la siguiente:

```
DBMS_OUTPUT.GET_LINE(line OUT VARCHAR2,
Status OUT INTEGER);
```

DESCRIPCIÓN DE LOS parámetros

A continuación, se describe cada uno de los parámetros que se usan en el procedimiento GET_LINE.

line

Devuelve una única línea de información del buffer, excluyendo el carácter final de salto de línea. La variable que recoja el valor de este parámetro se debería definir con un tamaño y tipo de VARCHAR2(32767) para evitar el riesgo de un error del tipo ORA-06502: PL/SQL numeric or value error: character string buffer too small.

status

Si la llamada se completa correctamente, entonces este parámetro devolverá 0. Si no hay más líneas en el buffer, entonces devolverá 1.

Procedimiento GET_LINES

Este procedimiento recupera un array de líneas de información del buffer.

SINTAXIS

La sintaxis de este procedimiento es la siguiente:

```
DBMS_OUTPUT.GET_LINES (lineas OUT CHARARR,
      numlineas IN OUT INTEGER);

DBMS_OUTPUT.GET_LINES (lineas OUT DBMSOUTPUT_LINESARRAY,
numlineas IN OUT INTEGER);
```

El tipo CHARARR es un tipo table con la siguiente sintaxis:

```
TYPE CHARARR IS TABLE OF VARCHAR2(32767) INDEX BY
BINARY_INTEGER;
```

DESCRIPCIÓN DE LOS parámetros

A continuación, se describe cada uno de los parámetros que se usan en el procedimiento GET_LINES.

lineas

Devuelve un array de líneas de información del buffer. La máxima longitud de cada línea en el array es de 32.767 bytes.

numlineas

Número de líneas que se quieren recuperar del buffer. Después de recuperar el número especificado de líneas, el procedimiento devuelve el número de líneas actualmente recuperadas. Si este número es menor que el de las líneas solicitadas, entonces no hay más líneas en el buffer.

Procedimiento NEW_LINE

Este procedimiento coloca un carácter de fin de línea.

SINTAXIS

La sintaxis de este procedimiento es la siguiente:

```
DBMS_OUTPUT.NEW_LINE;
```

Procedimiento PUT

Este procedimiento coloca una línea parcial en el buffer.

SINTAXIS

La sintaxis de este procedimiento es la siguiente:

```
DBMS_OUTPUT.PUT (cadena IN VARCHAR2);
```

DESCRIPCIÓN DE LOS parámetros

A continuación, se describe cada uno de los parámetros que se usan en el procedimiento PUT.

cadena

Cadena a incluir en el buffer.

Procedimiento PUT_LINE

Este procedimiento coloca una línea en el buffer.

SINTAXIS

La sintaxis de este procedimiento es la siguiente:

```
DBMS_OUTPUT.PUT_LINE (cadena IN VARCHAR2);
```

DESCRIPCIÓN DE LOS parámetros

A continuación, se describe cada uno de los parámetros que se usan en el procedimiento PUT_LINE.

cadena

Cadena a incluir en el buffer.

Reglas y límites

El tamaño máximo de una línea de salida de mensajes es de 32.767 bytes.

El tamaño por defecto del buffer es de 20.000 bytes. El tamaño mínimo es de 2.000 bytes y el máximo es ilimitado.

DBMS_random

El paquete DBMS_RANDOM proporciona mecanismos para generar números aleatorios.

Este paquete consta de los siguientes subprogramas:

Nombre Subprograma	Tipo	Descripción
INITIALIZE	Procedimiento	Inicializa el paquete con un valor a partir de otro de referencia.
NORMAL	Función	Devuelve números aleatorios en una distribución normal.
RANDOM	Procedimiento	Genera un número aleatorio.
SEED	Procedimiento	Resetea los valores de referencia utilizados para la generación de números aleatorios.
STRING	Función	Obtiene una cadena aleatoria.
TERMINATE	Procedimiento	Termina el paquete.
VALUE	Función	Obtiene un número aleatorio, mayor o igual a 0 y menor que 1, con 28 dígitos a la derecha del punto decimal.

Los subprogramas INITIALIZE, RANDOM y TERMINATE se encuentran descatalogados para la versión de 11g de la base de datos de Oracle. La compañía recomienda no utilizarlos en esta versión. Se mantienen por compatibilidad con las versiones anteriores de las bases de datos Oracle.

DBMS_RANDOM.RANDOM genera números enteros en el rango -2^^31 y 2^^31.

El paquete DBMS_RANDOM puede ser explícitamente inicializado, pero no necesita ser inicializado antes de la llamada al generador de números aleatorios. Este se inicializará automáticamente con la fecha, identificador de usuario e identificador de proceso.

Procedimiento INITIALIZE

Este procedimiento inicializa el generador de números aleatorios.

Este procedimiento está descatalogado para la versión 11g y aunque está soportado en el paquete y no genera errores de compilación si se usa, Oracle recomienda que no sea utilizado.

SINTAXIS

La sintaxis de este procedimiento es la siguiente:

```
DBMS_RANDOM.INITIALIZE (valor IN BINARY_INTEGER);
```

DESCRIPCIÓN DE LOS parámetros

A continuación, se describe cada uno de los parámetros que se usan en el procedimiento INITIALIZE.

valor

Número de referencia usado para la generación del número aleatorio.

Función NORMAL

Esta función devuelve un número aleatorio en una distribución estándar.

SINTAXIS

La sintaxis de esta función es la siguiente:

```
DBMS_RANDOM.NORMAL RETURN NUMBER;
```

DESCRIPCIÓN DEl valor de retorno de la función

Esta función devuelve un número aleatorio de tipo NUMBER.

Procedimiento RANDOM

Este procedimiento genera un número aleatorio.

Este procedimiento está descatalogado para la versión 11g y aunque está soportado en el paquete y no genera errores de compilación si se usa, Oracle recomienda que no sea utilizado.

SINTAXIS

La sintaxis de este procedimiento es la siguiente:

```
DBMS_RANDOM.RANDOM RETURN binary_integer;
```

DESCRIPCIÓN DE LOS parámetros

Este procedimiento devuelve un valor de tipo BINARY_INTEGER correspondiente al número aleatorio, siendo este mayor o igual a -2^{31} y menor que 2^{31}.

Procedimiento SEED

Este procedimiento resetea los valores de referencia utilizados para la generación de números aleatorios.

SINTAXIS

La sintaxis de este procedimiento es la siguiente:

```
DBMS_RANDOM.SEED(p_referencia IN binary_integer);
```

DESCRIPCIÓN DE LOS parámetros

A continuación, se describe cada uno de los parámetros que se usan en el procedimiento SEED.

p_referencia

Número o cadena de referencia usada para la generación del número aleatorio.

La cadena de texto usada como referencia puede superar la longitud de 2.000 caracteres.

Función STRING

Esta función obtiene una cadena de texto aleatoria.

SINTAXIS

La sintaxis de esta función es la siguiente:

```
DBMS_RANDOM.STRING (patron IN CHAR,
longitud IN NUMBER)
RETURN VARCHAR2;
```

DESCRIPCIÓN DE LOS parámetros y valores de retorno

A continuación, se describe cada uno de los parámetros que se usan en la función STRING, así como el valor de retorno de la misma.

patrón

Especifica que la cadena devuelta se parece a los siguientes patrones porque los incluye dentro de la misma:

- 'u', 'U'
- 'l', 'L'
- 'a', 'A'
- 'x', 'X'
- 'p', 'P'

longitud

Longitud de la cadena devuelta.

RETURN VARCHAR2

La función devuelve una cadena de tipo VARCHAR2.

Procedimiento TERMINATE

Cuando se quiere finalizar la ejecución del paquete se llama a este procedimiento.

Este procedimiento está descatalogado para la versión 11g y aunque está soportado en el paquete y no genera errores de compilación si se usa, Oracle recomienda que no sea utilizado.

SINTAXIS

La sintaxis de este procedimiento es la siguiente:

```
DBMS_RANDOM.TERMINATE;
```

Función VALUE

Esta función obtiene un número aleatorio mayor o igual a 0 y menor que 1 con 38 dígitos de precisión a la derecha decimal. Como alternativa a esta funcionalidad, también se permite obtener un número aleatorio mayor o igual al límite_inferior y menor que el límite_superior, cuando se le pasen estos valores a la llamada de la función.

SINTAXIS

La sintaxis de esta función es la siguiente:

```
DBMS_RANDOM.VALUE(límite_inferior IN NUMBER,
límite_superior IN NUMBER)
                    RETURN NUMBER;
```

DESCRIPCIÓN DE LOS parámetros y valores de retorno

A continuación, se describe cada uno de los parámetros que se usan en la función VALUE, así como el valor de retorno de la misma.

límite_inferior

Especifica el número más pequeño que se puede obtener como aleatorio. El número aleatorio puede ser igual al límite_inferior.

límite_superior

Especifica el número más grande que se puede obtener como aleatorio. El número aleatorio tiene que ser menor al límite_superior.

RETURN NUMBER

La función devuelve un número de tipo NUMBER.

UTL_FILE

El paquete UTL_FILE proporciona mecanismos para leer y escribir sobre ficheros de textos del sistema operativo.

Este paquete consta de los siguientes subprogramas:

Nombre Subprograma	Tipo	Descripción
FCLOSE	Procedimiento	Cierra un fichero.
FLOSE_ALL	Procedimiento	Cierra los manejadores de todos los ficheros abiertos.
FCOPY	Procedimiento	Copia una porción contigua de un fichero a un nuevo fichero.
FFLUSH	Procedimiento	Escribe físicamente todas las salidas pendientes a un fichero.
FGETATTR	Procedimiento	Lee y devuelve los atributos de un fichero de disco.
FGETPOS	Función	Devuelve en bytes la posición relativa (offset) dentro de un fichero.
FOPEN	Función	Abre un fichero para entrada o salida.
FOPEN_NCHAR	Función	Abre un fichero en Unicode para entrada o salida.
FREMOVE	Procedimiento	Borra un fichero de disco asumiendo que se tienen los suficientes privilegios para hacerlo.
FRENAME	Procedimiento	Renombra un fichero existente a un nuevo nombre. Es similar a la función mv de UNIX.
FSEEK	Procedimiento	Ajusta el puntero del fichero hacia delante o hacia atrás dentro del mismo, a partir del número de bytes especificados.
GET_LINE	Procedimiento	Lee un texto desde un fichero abierto.
GET_LINE_NCHAR	Procedimiento	Lee un texto en formato Unicode desde un fichero abierto.
GET_RAW	Procedimiento	Lee un string RAW de un fichero y ajusta el puntero del fichero por el número de bytes leídos.
IS_OPEN	Función	Determina si el manejador de fichero referencia a un fichero abierto.
NEW_LINE	Procedimiento	Escribe uno o más finalizadores de línea en el fichero.
PUT	Procedimiento	Escribe una cadena de caracteres en el fichero.
PUT_LINE	Procedimiento	Escribe una línea entera (cadena de caracteres + finalizador de línea) en el fichero.

PUT_LINE_NCHAR	Procedimiento	Escribe una línea Unicode en el fichero.
PUT_NCHAR	Procedimiento	Escribe una cadena de caracteres Unicode en el fichero.
PUTF	Procedimiento	Un procedimiento PUT formateado.
PUTF_NCHAR	Procedimiento	Un procedimiento PUT_NCHAR formateado y escribe una cadena de caracteres Unicode en el fichero también formateada.
PUT_RAW	Procedimiento	Acepta una entrada de datos tipo RAW y escribe el valor en el buffer de salida.

Para realizar operaciones de lectura y escritura sobre ficheros del sistema operativo, el usuario tiene que tener estos permisos sobre el objeto directorio que apunta a la ruta de red donde se encuentran físicamente los ficheros.

Para crear este tipo de objetos directorio, se utiliza la instrucción CREATE DIRECTORY.

Para manejar un fichero se tiene que definir una variable de tipo UTL_FILE.FILE_TYPE.

Errores de la sección EXCEPTION

Los subprogramas de UTL_FILE pueden generar alguno de los errores que se especifican en la siguiente tabla:

Error	Descripción
INVALID_PATH	La ruta de localización del fichero es incorrecta.
INVALID_MODE	El parámetro que indica el modo de apertura del fichero en FOPEN es incorrecto.
INVALID_FILEHANDLE	El manejador del fichero es incorrecto.
INVALID_OPERATION	El fichero podría no estar abierto o no se puede operar contra él, como se solicita.
READ_ERROR	El buffer de destino es demasiado pequeño u ocurrió un error durante la operación de lectura del fichero.
WRITE_ERROR	Un error ocurrió durante la operación de escritura en el fichero.
INTERNAL_ERROR	Error PL/SQL inespecífico.
CHARSETMISMATCH	Se abrió un fichero con la función FOPEN_NCHAR pero se están usando operaciones incompatibles como PUTF o GET_LINE.
FILE_OPEN	La operación solicitada falló porque el fichero ya está abierto.
INVALID_MAXLINESIZE	El valor MAX_LINESIZE de la función FOPEN es incorrecto. Debería estar en un rango de 1 a 32767.
INVALID_FILENAME	El parámetro correspondiente al nombre del fichero es incorrecto.
ACCESS_DENIED	Se han denegado los permisos para acceder al fichero en la ruta especificada.
INVALID_OFFSET	Las causas por las que se puede producir este error son: • ABSOLUTE_OFFSET = NULL y RELATIVE_OFFSET = NULL. • ABSOLUTE_OFFSET < 0 • El valor de offset causa la búsqueda más allá del final del fichero.
DELETE_FAILED	La operación de borrado solicitada falló.
RENAME_FAILED	La operación de renombrado solicitada falló.

El procedimiento UTL_FILE también puede devolver errores predefinidos de PL/SQL como pueden ser NO_DATA_FOUND cuando se intente leer un fichero que está vacío o VALUE_ERROR cuando se intente asignar el contenido de una línea leída del fichero a un tipo de variable incompatible.

ejemplo

```
SQL> CREATE DIRECTORY user_dir AS '/app1/g1/user';
SQL> GRANT READ ON DIRECTORY user_dir TO PUBLIC;
SQL> GRANT WRITE ON DIRECTORY user_dir TO PUBLIC;
```

En esta parte del ejemplo se crea un objeto directorio denominado USER_DIR que apunta a una ruta física de red Unix denomina 'app1/g1/user'. A continuación, se le otorgan permisos de lectura (READ) y escritura (WRITE) a todos los usuarios existentes en la base de datos (PUBLIC).

```
DECLARE
V1 VARCHAR2(32767);
F1 UTL_FILE.FILE_TYPE;
BEGIN
F1 := UTL_FILE.FOPEN('USER_DIR','f_prueba.tmp','R',256);
UTL_FILE.GET_LINE(F1,V1,32767);
UTL_FILE.FCLOSE(F1);

F1 := UTL_FILE.FOPEN('USER_DIR','f_prueba.tmp','R');
UTL_FILE.GET_LINE(F1,V1,32767);
UTL_FILE.FCLOSE(F1);

F1 := UTL_FILE.FOPEN('USER_DIR','f_prueba.tmp','A');
UTL_FILE.PUT_LINE(F1,'Esta es la nueva linea insertada');
UTL_FILE.FCLOSE(F1);

END;
```

En esta parte del ejemplo, se define un bloque sin nomina (DECLARE… BEGIN…END) en el que en primer lugar se define una variable V1 para almacenar la ruta íntegra del fichero (incluido también el propio nombre del mismo), y la variable necesaria para manejar el fichero: F1 de tipo UTL_FILE.FILE_TYPE. Dentro del bloque ejecutable (BEGIN…END), en el primer grupo de sentencias se abre el fichero "f_prueba.tmp" únicamente para lectura con un tamaño máximo de buffer de lectura de 256 caracteres, por lo que aunque se define la lectura de una línea del fichero con tamaño de 32.767 en el comando GET_LINE, prevalece el ancho del buffer y solo se leerán los primeros 256 caracteres.

En el segundo grupo de sentencias del bloque ejecutable, se vuelve a abrir el mismo fichero para lectura pero sin indicación del ancho del buffer. Como el valor por defecto del ancho del buffer es 1.024 si no se especifica ningún valor, pues pese a que otra vez se define la lectura de una línea de 32.767, solo se leerán los primeros 1.024 caracteres.

Por último, en el tercer grupo de sentencias del bloque ejecutable, se abre el mismo fichero de los dos casos anteriores pero para escritura añadiendo las líneas al final del mismo (APPEND) y se inserta una línea con el texto 'Esta es la nueva linea insertada' añadiendo un retorno de carro (final de línea) tras el mismo.

Procedimiento FCLOSE

Este procedimiento cierra un fichero abierto, identificado por un manejador de fichero.

SINTAXIS

La sintaxis de este procedimiento es la siguiente:

```
UTL_FILE.FCLOSE(file IN OUT FILE_TYPE);
```

DESCRIPCIÓN DE LOS parámetros

A continuación, se describe cada uno de los parámetros que se usan en el procedimiento FCLOSE.

file

Manejador de fichero activo devuelto por las llamadas a las funciones FOPEN o FOPEN_NCHAR.

Procedimiento FCLOSE_ALL

Este procedimiento cierra todos los manejadores de fichero de la sesión en curso. Este procedimiento se debería usar como un procedimiento de limpieza de emergencia; por ejemplo, cuando un programa PL/SQL aborta por un error/excepción.

SINTAXIS

La sintaxis de este procedimiento es la siguiente:

```
UTL_FILE.FCLOSE_ALL;
```

Procedimiento FCOPY

Este procedimiento copia una porción contigua de un fichero a un nuevo fichero creado. Por defecto, el fichero completo es copiado si los parámetros start_line y end_line son omitidos. El fichero origen se abre en modo lectura. El fichero destino es abierto en modo escritura. Se puede especificar opcionalmente una línea de comienzo y finalización para seleccionar una porción desde el centro del fichero origen a copiar.

SINTAXIS

La sintaxis de este procedimiento es la siguiente:

```
UTL_FILE.FCOPY(src_location IN VARCHAR2,
               src_filename IN VARCHAR2,
               dest_location IN VARCHAR2,
               dest_filename IN VARCHAR2,
               start_line IN BINARY_INTEGER
               DEFAULT 1,
               end_line IN BINARY_INTEGER
               DEFAULT NULL);
```

DESCRIPCIÓN DE LOS parámetros

A continuación, se describe cada uno de los parámetros que se usan en el procedimiento FCOPY.

src_location

Directorio de localización del fichero origen. Es uno de los objetos directorio definidos en la base de datos.

src_filename

Fichero origen que será copiado.

dest_location

Directorio destino donde el fichero destino será creado.

dest_filename

El fichero destino creado a partir del fichero origen.

start_line

Número de línea de comienzo de la copia. El valor por defecto es 1 para la primera línea si se omite este parámetro.

end_line

Número de línea de finalización de la copia. El valor por defecto es NULL que equivale a indicar el final del fichero, cuando se omite este parámetro.

Procedimiento FFLUSH

Este procedimiento físicamente escribe los datos pendientes al fichero identificado por el manejador. Normalmente, los datos que son escritos al fichero son los que se encuentran en el buffer. Los datos deben de terminarse con un retorno de carro (nueva línea).

Este procedimiento es útil cuando el fichero debe ser leído mientras está todavía abierto. Por ejemplo cuando se obtienen mensajes de un proceso de debug, estos pueden ser copiados a un fichero de modo que puedan ser leídos inmediatamente.

SINTAXIS

La sintaxis de este procedimiento es la siguiente:

```
UTL_FILE.FFLUSH(file IN FILE_TYPE);
```

DESCRIPCIÓN DE LOS parámetros

A continuación, se describe cada uno de los parámetros que se usan en el procedimiento FFLUSH.

file

Manejador de fichero activo devuelto por las llamadas a las funciones FOPEN o FOPEN_NCHAR.

Procedimiento FGETATTR

Este procedimiento lee y devuelve los atributos de un fichero de disco.

SINTAXIS

La sintaxis de este procedimiento es la siguiente:

```
UTL_FILE.FGETATTR(location IN VARCHAR2,
                  filename IN VARCHAR2,
                  fexists OUT BOOLEAN,
                  file_length OUT NUMBER,
                  block_size OUT BINARY_INTEGER);
```

DESCRIPCIÓN DE LOS parámetros

A continuación, se describe cada uno de los parámetros que se usan en el procedimiento FGETATTR.

location

Directorio de localización del fichero origen. Es uno de los objetos directorio definidos en la base de datos.

filename

Nombre del fichero que será examinado.

fexists

Un valor booleano que indica si existe o no el fichero.

file_length

Longitud del fichero en bytes. Es NULL si el fichero no existe.

block_size

Tamaño en bytes del bloque del fichero del sistema. Es NULL si el fichero no existe.

Función FGETPOS

Esta función devuelve la posición relativa del offset dentro del fichero, en bytes.

SINTAXIS

La sintaxis de esta función es la siguiente:

```
UTL_FILE.FGETPOS(file IN FILE_TYPE)
                 RETURN PLS_INTEGER;
```

DESCRIPCIÓN DE LOS parámetros

A continuación, se describe cada uno de los parámetros que se usan en la función FGETPOS.

file

Directorio de localización del fichero origen. Es uno de los objetos directorio definidos en la base de datos.

RETURN PLS_INTEGER

La función devuelve un número de tipo PLS_INTEGER.

Función FOPEN

Esta función abre un fichero. Se puede especificar un máximo de tamaño en líneas y tener un máximo de 50 ficheros abiertos simultáneamente.

SINTAXIS

La sintaxis de esta función es la siguiente:

```
UTL_FILE.FOPEN(location IN VARCHAR2,
               filename IN VARCHAR2,
               open_mode IN VARCHAR2,
               max_linesize IN BINARY_INTEGER DEFAULT 1024)
               RETURN FILE_TYPE;
```

DESCRIPCIÓN DE LOS parámetros

A continuación, se describe cada uno de los parámetros que se usan en la función FOPEN.

location

Directorio de localización del fichero origen. Es uno de los objetos directorio definidos en la base de datos.

filename

Nombre del fichero que será examinado.

open_mode

Especifica cómo se abre el fichero. Los modos permitidos son:

- r: solo lectura.
- w: solo escritura (sobreescribe todo lo existente).
- a: append (escritura al final del fichero sin borrar lo existente).
- rb: solo lectura por bytes.
- wb: solo escritura por bytes.

- ab: append por bytes.

Si se intenta abrir un fichero especificando el modo 'a' o 'ab' pero este fichero no existe ya, entonces el fichero será creado en modo escritura.

max_linesize

Máximo número de caracteres para cada línea, incluyendo el carácter de nueva línea, para este fichero (mínimo valor 1, máximo valor 32.767). Si no se especifica, Oracle asumirá el valor por defecto de 1.024.

RETURN FILE_TYPE

Devuelve el manejador del fichero abierto.

Función FOPEN_NCHAR

Esta función abre un fichero en el juego de caracteres nacional instalado en la base de datos, para operaciones de entrada o salida con el máximo de tamaño en número de líneas que se haya especificado. Con esta función se puede leer o escribir un fichero de texto en formato Unicode en vez de en el juego de caracteres de la base de datos

Aunque el contenido de un buffer de tipo NVARCHAR2 puede tener información en formato AL16UTF16 o UTF8 (dependiendo del juego de caracteres nacional instalado en la base de datos), el fichero siempre será leído o escrito en formato UTF8. UTL_FILE convierte entre UTF8 y AL16UTF16 si es necesario.

SINTAXIS

La sintaxis de esta función es la siguiente:

```
UTL_FILE.FOPEN_NCHAR(location IN VARCHAR2,
                     filename IN VARCHAR2,
                     open_mode IN VARCHAR2,
                     max_linesize IN BINARY_INTEGER DEFAULT 1024)
                     RETURN FILE_TYPE;
```

DESCRIPCIÓN DE LOS parámetros

A continuación, se describe cada uno de los parámetros que se usan en la función FOPEN_NCHAR.

location

Directorio de localización del fichero origen. Uno de los objetos directorio definidos en la base de datos.

filename

Nombre del fichero que será examinado.

open_mode

Especifica cómo se abre el fichero. Los modos permitidos son:

- r: solo lectura.
- w: solo escritura (sobreescribe todo lo que exista).
- a: append (escritura al final del fichero sin borrar lo existente).
- rb: solo lectura por bytes.
- wb: solo escritura por bytes.
- ab: append por bytes.

Si se intenta abrir un fichero especificando el modo 'a' o 'ab' pero este fichero no existe ya, entonces el fichero será creado en modo escritura.

max_linesize

Máximo número de caracteres para cada línea, incluyendo el carácter de nueva línea, para este fichero (mínimo valor 1, máximo valor 32.767). Si no se especifica, Oracle asume el valor por defecto de 1.024.

RETURN FILE_TYPE

Devuelve el manejador del fichero abierto.

Procedimiento FREMOVE

Este procedimiento borra un fichero del disco asumiendo que se tienen los permisos (del sistema operativo) necesarios para realizar esta operación.

SINTAXIS

La sintaxis de este procedimiento es la siguiente:

```
UTL_FILE.FREMOVE( location IN VARCHAR2,
                  filename IN VARCHAR2);
```

DESCRIPCIÓN DE LOS parámetros

A continuación, se describe cada uno de los parámetros que se usan en el procedimiento FREMOVE.

location

Directorio de localización del fichero origen. Es uno de los objetos directorio definidos en la base de datos.

filename

Nombre del fichero que será eliminado.

Procedimiento FRENAME

Este procedimiento renombra un fichero existente a un nuevo nombre. Esta operación es similar a la función mv de Unix, en la que el fichero en realidad se elimina del directorio origen y se copia con en otro directorio distinto con el mismo nombre u otro diferente.

SINTAXIS

La sintaxis de este procedimiento es la siguiente:

```
UTL_FILE.FRENAME(src_location IN VARCHAR2,
                 src_filename IN VARCHAR2,
                 dest_location IN VARCHAR2,
                 dest_filename IN VARCHAR2,
                 overwrite IN BOOLEAN DEFAULT
                 FALSE);
```

DESCRIPCIÓN DE LOS parámetros

A continuación, se describe cada uno de los parámetros que se usan en el procedimiento FRENAME

src_location

Directorio de localización del fichero origen. Es uno de los objetos directorio definidos en la base de datos.

filename

Nombre del fichero origen que será renombrado.

dest_location

Directorio destino del fichero. Es uno de los objetos directorio definidos en la base de datos.

dest_filename

Nuevo nombre del fichero.

overwrite

Por defecto es FALSE. Se debe disponer de los permisos de base de datos necesarios para realizar esta operación en ambos objetos directorio. Se usa este parámetro para especificar si se sobreescribe o no un fichero que ya exista con el mismo nombre en el directorio destino. El valor por defecto FALSE indica que no se sobreescribe.

Procedimiento FSEEK

Este procedimiento ajusta el puntero del fichero hacia delante o hacia atrás dentro del número de bytes que se especifiquen.

SINTAXIS

La sintaxis de este procedimiento es la siguiente:

```
UTL_FILE.FSEEK( file IN OUT UTL_FILE.FILE_TYPE,
absolute_offset IN PLS_INTEGER
                                    DEFAULT NULL,
                relative_offset IN PLS_INTEGER
                                    DEFAULT NULL);
```

DESCRIPCIÓN DE LOS parámetros

A continuación, se describe cada uno de los parámetros que se usan en el procedimiento FSEEK.

file

Manejador del fichero.

absolute_offset

Localización absoluta del fichero que se busca. Por defecto, el valor es NULL.

relative_offset

Número de bytes a buscar hacia delante o hacia atrás. Un valor positivo indica búsqueda hacia delante y un valor negativo búsqueda hacia atrás. Cero indica buscar en la posición actual. Por defecto este parámetro tiene valor NULL.

Procedimiento GET_LINE

Este procedimiento lee un texto del fichero abierto identificado por un manejador de fichero y lo lleva al parámetro del buffer de salida. El texto leído no incluye la marca de final de línea, o final de fichero o retorno de carro. La lectura no puede exceder del tamaño indicado en el parámetro max_linesize de la función FOPEN.

SINTAXIS

La sintaxis de este procedimiento es la siguiente:

UTL_FILE**¡Error! Marcador no definido.**.GET_LINE (file IN FILE_TYPE,
buffer OUT VARCHAR2
len IN PLS_INTEGER DEFAULT
NULL);

DESCRIPCIÓN DE LOS parámetros

A continuación, se describe cada uno de los parámetros que se usan en el procedimiento GET_LINE.

file

Manejador del fichero. El fichero debe estar abierto en modo R (lectura) porque si no, devolverá un error/excepción del tipo INVALID_OPERATION.

buffer

Buffer de información que recibe el contenido de la línea leída del fichero.

len

El número de bytes leídos del fichero. Por defecto, el valor es NULL. Si no se especifica otro valor, se asumirá el valor del parámetro max_linesize.

Procedimiento GET_LINE_NCHAR

Este procedimiento lee un texto del fichero abierto identificado por un manejador de fichero y lo lleva al parámetro del buffer de salida. Con esta función se puede leer o escribir un fichero de texto en formato Unicode en vez de en el juego de caracteres de la base de datos.

El fichero debe estar abierto en el juego de caracteres nacional y debe estar codificado en UTF8. El tipo de dato que se espera para el buffer será NVARCHAR2. Si la variable es de otro tipo, tal como NCHAR, NCLOB o VARCHAR2, PL/SQL realizará un conversión implícita desde NVARCHAR2 después de haber leído el texto.

SINTAXIS

La sintaxis de este procedimiento es la siguiente:

```
UTL_FILE.GET_LINE_NCHAR(file IN FILE_TYPE,
buffer OUT VARCHAR2
len IN PLS_INTEGER
                                DEFAULT NULL);
```

DESCRIPCIÓN DE LOS parámetros

A continuación, se describe cada uno de los parámetros que se usan en el procedimiento GET_LINE_NCHAR.

file

Manejador del fichero. El fichero debe estar abierto en modo R (lectura) porque si no, devolverá un error/excepción del tipo INVALID_OPERATION.

buffer

Buffer de información que recibe el contenido de la línea leída del fichero.

len

El número de bytes leídos del fichero. Por defecto, el valor es NULL. Si no se especifica otro valor, se asumirá el valor del parámetro max_linesize.

Procedimiento GET_RAW

Este procedimiento lee una cadena de tipo RAW desde un fichero y ajusta la posición del puntero al número de bytes leído. UTL_FILE.GET_RAW ignora los finalizadores de línea (retorno de carro, fin de línea o fin de fichero).

SINTAXIS

La sintaxis de este procedimiento es la siguiente:

```
UTL_FILE.GET_RAW(file IN FILE_TYPE,
buffer OUT NOCOPY RAW
len IN PLS_INTEGER DEFAULT
NULL);
```

DESCRIPCIÓN DE LOS parámetros

A continuación, se describe cada uno de los parámetros que se usan en el procedimiento GET_RAW

file

Manejador del fichero.

buffer

Información leída en formato RAW.

len

El número de bytes leídos del fichero. Por defecto el valor es NULL. Si no se especifica otro valor, se asumirá la máxima longitud del tipo RAW.

Función IS_OPEN

Esta función comprueba el manejador del fichero para ver si identifica a un fichero abierto. IS_OPEN informa si el manejador apunta a un fichero que ha sido abierto y todavía no se ha cerrado.

SINTAXIS

La sintaxis de esta función es la siguiente:

```
UTL_FILE.IS_OPEN(file IN FILE_TYPE)
                 RETURN BOOLEAN;
```

DESCRIPCIÓN DE LOS parámetros

A continuación, se describe cada uno de los parámetros que se usan en la función IS_OPEN.

file

Manejador del fichero.

RETURN BOOLEAN

Devuelve TRUE si el fichero está abierto o FALSE en caso contrario.

Procedimiento NEW_LINE

Este procedimiento escribe uno o más finalizadores de línea (carácter fin de línea) en el fichero especificado por el manejador.

SINTAXIS

La sintaxis de este procedimiento es la siguiente:

```
UTL_FILE.NEW_LINE(file IN FILE_TYPE,
lines IN BINARY_INTEGER := 1);
```

DESCRIPCIÓN DE LOS parámetros

A continuación, se describe cada uno de los parámetros que se usan en el procedimiento NEW_LINE

file

Manejador del fichero.

lines

Número de finalizadores de línea que serán escritos en el fichero.

Procedimiento PUT

Este procedimiento escribe una cadena de caracteres en el buffer que se llevará al fichero abierto. El fichero debe estar abierto en modo escritura. Con este procedimiento no se añaden símbolos de fin de línea (para ello hay que usar el procedimiento NEW_LINE o PUT_LINE).

SINTAXIS

La sintaxis de este procedimiento es la siguiente:

```
UTL_FILE.PUT( file IN FILE_TYPE,
buffer IN VARCHAR2);
```

DESCRIPCIÓN DE LOS parámetros

A continuación, se describe cada uno de los parámetros que se usan en el procedimiento PUT.

file

Manejador del fichero.

buffer

Buffer que contiene el texto a escribir en el fichero.

Procedimiento PUT_LINE

Este procedimiento escribe una cadena de caracteres en el buffer que se llevará al fichero abierto. El fichero debe estar abierto en modo escritura. Con este procedimiento sí que se añade un símbolo de fin de línea después de la cadena de caracteres a escribir.

SINTAXIS

La sintaxis de este procedimiento es la siguiente:

```
UTL_FILE.PUT_LINE(file IN FILE_TYPE,
buffer IN VARCHAR2,
                  autoflush IN BOOLEAN DEFAULT
                  FALSE);
```

DESCRIPCIÓN DE LOS parámetros

A continuación, se describe cada uno de los parámetros que se usan en el procedimiento PUT_LINE.

file

Manejador del fichero.

buffer

Buffer que contiene el texto a escribir en el fichero.

autoflush

Limpia el buffer del disco después de la escritura.

Procedimiento PUT_LINE_NCHAR

Este procedimiento escribe una cadena de caracteres en el buffer que se llevará al fichero abierto. Con esta función se puede escribir un fichero de texto en formato Unicode en vez de en el juego de caracteres de la base de datos. El fichero debe estar abierto en modo escritura. Con este procedimiento sí que se añade un símbolo de fin de línea después de la cadena de caracteres a escribir.

SINTAXIS

La sintaxis de este procedimiento es la siguiente:

```
UTL_FILE.PUT_LINE_NCHAR(file IN FILE_TYPE,
buffer IN VARCHAR2);
```

DESCRIPCIÓN DE LOS parámetros

A continuación, se describe cada uno de los parámetros que se usan en el procedimiento PUT_LINE_NCHAR.

file

Manejador del fichero.

buffer

Buffer que contiene el texto a escribir en el fichero.

Procedimiento PUT_NCHAR

Este procedimiento escribe una cadena de caracteres en el buffer que se llevará al fichero abierto. Con esta función se puede escribir un fichero de texto en formato Unicode en vez de en el juego de caracteres de la base de datos. El fichero debe estar abierto en modo escritura.

El texto a escribir debe estar en el juego de caracteres UTF8. El buffer de salida espera un tipo de dato NVARCHAR2. Si la variable es de otro tipo, se realizará una conversión implícita a NVARCHAR2 antes de escribir el texto.

SINTAXIS

La sintaxis de este procedimiento es la siguiente:

```
UTL_FILE.PUT_NCHAR(file IN FILE_TYPE,
buffer IN VARCHAR2);
```

DESCRIPCIÓN DE LOS parámetros

A continuación, se describe cada uno de los parámetros que se usan en el procedimiento PUT_NCHAR.

file

Manejador del fichero.

buffer

Buffer que contiene el texto a escribir en el fichero. El usuario debe abrir el fichero en modo "w" o en modo "a". En caso contrario devolverá el error/excepción INVALID_OPERATION.

Procedimiento PUTF

Este procedimiento es un formateo de su homólogo PUT. Trabaja como un printf() limitado.

SINTAXIS

La sintaxis de este procedimiento es la siguiente:

```
UTL_FILE.PUTF(file IN FILE_TYPE,
format IN VARCHAR2,
        [arg1 IN VARCHAR2 DEFAULT NULL,
            …
        arg5 IN VARCHAR2 DEFAULT NULL]);
```

DESCRIPCIÓN DE LOS parámetros

A continuación, se describe cada uno de los parámetros que se usan en el procedimiento PUTF.

file

Manejador del fichero.

format

Cadena de caracteres formateada que puede contener un texto en el que se hayan formateado los caracteres \n y %s.

arg1...arg5

Desde uno hasta 5 cadenas de argumentos para operaciones de formateo.

%s indica que se sustituya la secuencia con la cadena del próximo argumento en la lista de argumentos.

\n indica que se sustituya con el carácter de fin de línea apropiado a la plataforma en la que se esté trabajando.

Ejemplo

Hola mundo!

Yo vengo de Zork con saludos para todos los terrícolas.

```
DECLARE
My_world VARCHAR2(4) := 'Zork';
Manejador UTL_FILE.FILE_TYPE;

BEGIN
….
PUTF (manejador,'Hola mundo!\nYo vengo de %s con %s. \n',My_world,'saludos para todos los terrícolas');
…
END;
```

En este ejemplo en primer lugar se definen 2 líneas de un fichero cualquiera y, a continuación, parte de un bloque sin nominar en la que en el código ejecutable se introduce una sentencia PUTF.

Procedimiento PUTF_NCHAR

Este procedimiento es un formateo de su homólogo PUT_NCHAR.

SINTAXIS

La sintaxis de este procedimiento es la siguiente:

```
UTL_FILE.PUTF_NCHAR(
        file IN FILE_TYPE,
format IN VARCHAR2,
        [arg1 IN VARCHAR2 DEFAULT NULL,
                        …
        arg5 IN VARCHAR2 DEFAULT NULL]);
```

DESCRIPCIÓN DE LOS parámetros

A continuación, se describe cada uno de los parámetros que se usan en el procedimiento PUTF_NCHAR.

file

Manejador del fichero.

format

Cadena de caracteres formateada que puede contener un texto en el que se hayan formateado los caracteres \n y %s.

arg1...arg5

Desde uno hasta 5 cadenas de argumentos para operaciones de formateo.

%s indica que se sustituya la secuencia con la cadena del próximo argumento en la lista de argumentos.

\n indica que se sustituya con el carácter de fin de línea apropiado a la plataforma en la que se esté trabajando.

Procedimiento PUT_RAW

Este procedimiento acepta como entrada un valor de tipo RAW y lo escribe en el buffer de salida.

SINTAXIS

La sintaxis de este procedimiento es la siguiente:

```
UTL_FILE.PUT_RAW(file IN UTL_FILE.FILE_TYPE,
buffer IN RAW,
                 autoflush IN BOOLEAN DEFAULT
                                  FALSE);
```

DESCRIPCIÓN DE LOS parámetros

A continuación, se describe cada uno de los parámetros que se usan en el procedimiento PUT_RAW.

file

Manejador del fichero.

buffer

Buffer que contiene la información RAW a escribir.

autoflush

Si es TRUE, entonces realizará una operación de limpieza después de escribir el valor en el buffer de salida. El valor por defecto es FALSE.

UTL_MAIL

El paquete UTL_MAIL no está instalado por debido a los requerimientos de configuración de SMTP_OUT_SERVER y la seguridad que implica. En la instalación de UTL_MAIL se deberían tomar precauciones y asignar las operaciones adecuadas para que el puerto que se defina en SMTP_OUT_SERVER no sea utilizado por otras transmisiones.

Para instalar UTL_MAIL, hay que ejecutar el siguiente código:

sqlplus sys/<password>

SQL> @$ORACLE_HOME/rdbms/admin/utlmail.sql
SQL> @$ORACLE_HOME/rdbms/admin/prvtmail.plb

Para definir los parámetros de SMTP_OUT_SERVER, hay que abrir el fichero de inicialización de la base de datos: INIT.ORA.

Este paquete consta de los siguientes subprogramas:

Nombre Subprograma	Tipo	Descripción
SEND	Procedimiento	Empaqueta un mensaje de e-mail en el formato apropiado, localiza la información SMTP y envía el mensaje al servidor SMTP para que lo remita a los destinatarios.
SEND_ATTACH_RAW	Procedimiento	Representa al procedimiento SEND sobrecargado para adjuntar contenido RAW.
SEND_ATTACH_VARCHAR2	Procedimiento	Representa al procedimiento SEND sobrecargado para adjuntar contenido VARCHAR2.

Procedimiento SEND

Este procedimiento empaqueta un mensaje de e-mail en el formato apropiado, localiza la información SMTP y remite el mensaje al servidor SMTP para que a su vez le dé traslado a los receptores del mensaje.

SINTAXIS

La sintaxis de este procedimiento es la siguiente:

```
UTL_MAIL.SEND (
sender IN VARCHAR2 CHARACTER SET ANY_CS,
recipients IN VARCHART2 CHARACTER SET ANY_CS,
cc IN VARCHART2 CHARACTER SET ANY_CS
DEFAULT NULL,
bcc IN VARCHART2 CHARACTER SET ANY_CS
                    DEFAULT NULL,
subject IN VARCHART2 CHARACTER SET ANY_CS
DEFAULT NULL,
message IN VARCHART2 CHARACTER SET ANY_CS
DEFAULT NULL,
mime_type IN VARCHART2 DEFAULT CHARACTER SET
ANY_CS DEFAULT 'text/plain;
character=us-ascii',
priority IN PLS_INTEGER DEFAULT 3,
replyto IN VARCHART2 CHARACTER SET ANY_CS
DEFAULT NULL);
```

DESCRIPCIÓN DE LOS parámetros

A continuación, se describe cada uno de los parámetros que se usan en el procedimiento SEND.

sender

Dirección e-mail del que envía el mensaje.

recipients

Dirección e-mail del receptor o receptores del mensaje (en caso de varios, han

de ir separados por comas).

cc

Dirección e-mail del receptor o receptores a los que se les envía una copia del mensaje. Al igual que antes, si son varios se separarán con comas. El valor por defecto es NULL.

bcc

Dirección e-mail del receptor o receptores a los que se les envía una copia oculta del mensaje. Al igual que antes, si son varios se separarán con comas. El valor por defecto es NULL.

subject

Cadena de texto para identificar el asunto que se incluye en el mensaje.

message

Texto correspondiente al cuerpo del mensaje.

mime_type

Mensaje de tipo MIME. Por defecto, el valor es 'text/plain;charset=us-ascii'.

priority

Prioridad del mensaje. 1 es la prioridad más alta y 5 la más baja. El valor por defecto es 3.

replyto

Define a quién se le reenviará el mensaje.

Procedimiento SEND_ATTACH_RAW

Este procedimiento es similar al anterior (SEND) pero sobrecargado para que admita adjuntar elementos RAW.

SINTAXIS

La sintaxis de este procedimiento es la siguiente:

```
UTL_MAIL.SEND_ATTACH_RAW(
sender IN VARCHAR2 CHARACTER SET ANY_CS,
recipients IN VARCHART2 CHARACTER SET ANY_CS,
cc IN VARCHART2 CHARACTER SET ANY_CS
DEFAULT NULL,
bcc IN VARCHART2 CHARACTER SET ANY_CS
                    DEFAULT NULL,
subject IN VARCHART2 CHARACTER SET ANY_CS
DEFAULT NULL,
message IN VARCHART2 CHARACTER SET ANY_CS
DEFAULT NULL,
mime_type IN VARCHART2 DEFAULT CHARACTER SET
ANY_CS DEFAULT 'text/plain;
character=us-ascii',
priority IN PLS_INTEGER DEFAULT 3,
attachment IN RAW,
att_inline IN BOOLEAN DEFAULT TRUE,
att_mime_type IN VARCHAR2 CHARACTER SET ANY_CS
DEFAULT 'text/plain; character=us-
ascii',
att_filename IN VARCHAR2 SET ANY_CS DEFAULT NULL,
replyto IN VARCHART2 CHARACTER SET ANY_CS
DEFAULT NULL);
```

DESCRIPCIÓN DE LOS parámetros

A continuación, se describe cada uno de los parámetros que se usan en el procedimiento SEND_ATTACH_RAW.

sender

Dirección e-mail del que envía el mensaje.

recipients

Dirección e-mail del receptor o receptores del mensaje (en caso de varios, han de ir separados por comas).

cc

Dirección e-mail del receptor o receptores a los que se les envía una copia del mensaje. Al igual que antes, si son varios se separarán con comas. El valor por defecto es NULL.

bcc

Dirección e-mail del receptor o receptores a los que se les envía una copia oculta del mensaje. Al igual que antes, si son varios se separarán con comas. El valor por defecto es NULL.

subject

Cadena de texto para identificar el asunto que se incluye en el mensaje.

message

Texto correspondiente al cuerpo del mensaje.

mime_type

Mensaje de tipo MIME. Por defecto, el valor es 'text/plain;charset=us-ascii'.

priority

Prioridad del mensaje. 1 es la prioridad más alta y 5 la más baja. El valor por defecto es 3.

attachment

Fichero de tipo RAW adjunto.

att_inline

Especifica si la información adjunta es visible dentro del propio cuerpo del mensaje. Por defecto, el valor es TRUE.

att_mime_type

Tipo MIME del la información adjunta. Por defecto, es 'application/octet'.

att_filename

Cadena de texto que especifica el nombre del fichero contenido en la información adjunta al mensaje. Por defecto, el valor es NULL.

replyto

Define a quién se le reenviará el mensaje.

Procedimiento SEND_ATTACH_VARCHAR2

Este procedimiento es similar al anterior (SEND) pero sobrecargado para que admita adjuntar elementos VARCHAR2.

SINTAXIS

La sintaxis de este procedimiento es la siguiente:

```
UTL_MAIL.SEND_ATTCH_VARCHAR2(
sender IN VARCHAR2 CHARACTER SET ANY_CS,
recipients IN VARCHART2 CHARACTER SET ANY_CS,
cc IN VARCHART2 CHARACTER SET ANY_CS
DEFAULT NULL,
bcc IN VARCHART2 CHARACTER SET ANY_CS
                    DEFAULT NULL,
subject IN VARCHART2 CHARACTER SET ANY_CS
DEFAULT NULL,
message IN VARCHART2 CHARACTER SET ANY_CS
DEFAULT NULL,
mime_type IN VARCHART2 DEFAULT CHARACTER SET
                ANY_CS DEFAULT 'text/plain;
                character=us-ascii',
priority IN PLS_INTEGER DEFAULT 3
attachment IN VARCHAR2 CHARACTER SET ANY_CS,
att_inline IN BOOLEAN DEFAULT TRUE,
att_mime_type IN VARCHAR2 CHARACTER SET ANY_CS
DEFAULT 'text/plain; character=us-
ascii',
att_filename IN VARCHAR2 SET ANY_CS DEFAULT NULL,
replyto IN VARCHAR2 CHARACTER SET ANY_CS DEFAULT
NULL);
```

DESCRIPCIÓN DE LOS parámetros

A continuación, se describe cada uno de los parámetros que se usan en el procedimiento SEND_ATTACH_VARCHAR2.

sender

Dirección e-mail del que envía el mensaje.

recipients

Dirección e-mail del receptor o receptores del mensaje (en caso de varios, han de ir separados por comas).

cc

Dirección e-mail del receptor o receptores a los que se les envía una copia del mensaje. Al igual que antes, si son varios se separarán con comas. El valor por defecto es NULL.

bcc

Dirección e-mail del receptor o receptores a los que se les envía una copia oculta del mensaje. Al igual que antes, si son varios se separarán con comas. El valor por defecto es NULL.

subject

Cadena de texto para identificar el asunto que se incluye en el mensaje.

message

Texto correspondiente al cuerpo del mensaje.

mime_type

Mensaje de tipo MIME. Por defecto, el valor es 'text/plain;charset=us-ascii'.

priority

Prioridad del mensaje. 1 es la prioridad más alta y 5 la más baja. El valor por defecto es 3.

attachment

Fichero de tipo texto adjunto.

att_inline

Especifica si la información adjunta es visible dentro del propio cuerpo del mensaje. Por defecto, el valor es TRUE.

att_mime_type

Tipo MIME del la información adjunta. Por defecto, es 'text/plain; charset=us-ascii'.

att_filename

Cadena de texto que especifica el nombre del fichero contenido en la información adjunta al mensaje. Por defecto, el valor es NULL.

replyto

Define a quién se le reenviará el mensaje.

UTL_http

El paquete UTL_HTTP hace llamadas al protocolo HTTP (Hypertext Transfer Protocol) desde SQL y PL/SQL.

Cuando el paquete actúa contra datos de un sitio web usando HTTPS, requiere Oracle Wallet Manager que puede ser creado por Oracle Wallet Manager o por la utilidad orapki.

Este paquete reúne una serie de constantes agrupadas en las siguientes categorías:

- Constantes sobre versiones HTTP.
- Constantes sobre puertos por defecto.
- Constantes sobre códigos de estado HTTP 1.1.

Las constantes sobre versiones HTTP son:

Nombre Constante	Tipo	Valor	Descripción
HTTP_VERSION_1_0	VARCHAR2(10)	'HTTP/1.0'	Implica una versión 1.0 HTTP que puede ser usada en la función BEGIN_REQUEST.
HTTP_VERSION_1_1	VARCHAR2(10)	'HTTP/1.1'	Implica una versión 1.1 HTTP que puede ser usada en la función BEGIN_REQUEST.

Las constantes sobre puertos por defecto son:

Nombre Constante	Tipo	Valor	Descripción
DEFAULT_HTTP_PORT	PLS_INTEGER	80	El puerto por defecto TCP/IP 80 en el que escucha un servidor web o un servidor proxy.
DEFAULT_HTTPS_PORT	PLS_INTEGER	443	El puerto por defecto TCP/IP 443 en el que escucha un servidor HTTPS.

Las constantes sobre códigos de estado HTTP 1.1 son numerosas y quedan fuera de la redacción de este curso. Puede consultarlas en la documentación oficial de Oracle: *PL_SQL Package and Types Reference 11g Rel2.*

Este paquete también presenta la definición de tipos de datos personalizados y

nominados de la siguiente forma:

- Tipo REQ.
- Tipo RESP.
- Tipos COOKIE y COOKIE_TABLE.
- Tipo CONNECTION.
- Tipo REQUEST_CONTEXT_KEY.

Por último, consta de una serie de subprogramas agrupados en las siguientes categorías:

- Subprogramas de llamadas HTTP simples.
- Subprogramas de configuración de sesión.
- Subprogramas de petición HTTP.
- Subprogramas de peticiones contextuales HTTP.
- Subprogramas de respuesta HTTP.
- Subprogramas de cookies HTTP.
- Subprogramas de conexiones HTTP persistentes.
- Subprogramas para el tratamiento de las condiciones de error.
- Constantes sobre códigos de estado HTTP 1.1.

Los subprogramas de llamadas HTTP simples son los siguientes:

Nombre Subprograma	Tipo	Descripción
REQUEST	Función	Devuelve por encima de los primeros 2.000 bytes de los datos devueltos de la URL indicada. Esta función se puede utilizar directamente en consultas SQL.
REQUEST_PIECES	Función	Devuelve una tabla PL/SQL con celdas de 2.000 bytes cargadas con los datos devueltos por la URL indicada.

Los subprogramas de configuración de sesión son los siguientes:

Nombre Subprograma	Tipo	Descripción
GET_BODY_CHARSET	Procedimiento	Devuelve el juego de caracteres por defecto del cuerpo de todas las peticiones HTTP futuras.
GET_COOKIE_SUPPORT	Procedimiento	Devuelve la configuración de la COOKIE actual soportada.
GET_DETAILED_EXCP_ SUPPORT	Procedimiento	Comprueba si el paquete UTL_HTTP abortará por una excepción detallada o no.
GET_FOLLOW_REDIRECT	Procedimiento	Devuelve la configuración del siguiente redireccionamiento en la sesión actual.
GET_PERSISTENT_CONN _SUPPORT	Procedimiento	Comprueba si el soporte de la conexión persistente está habilitado y obtiene el máximo de conexiones persistentes en la sesión actual.
GET_PROXY	Procedimiento	Devuelve la configuración del proxy actual.
GET_RESPONSE_ERROR_ CHECK	Procedimiento	Comprueba si está activado el control de errores de respuesta o no.
GET_TRANSFER_TIMEOUT	Procedimiento	Devuelve el valor actual del TIMEOUT de transferencia de red.
SET_TRANSFER_TIMEOUT	Procedimiento	Establece el conjunto de caracteres por defecto del cuerpo para las peticiones http futuras cuando el MEDIA TYPE es texto y no se ha especificado un conjunto de caracteres para la cabecera CONTENT_TYPE.
SET_COOKIE_SUPPORT	Procedimiento	Establece si una futura petición HTTP soportará o no cookies. Establece el número máximo de cookies mantenidas en la sesión actual de base de datos del usuario.
SET_DETAILED_EXCP_ SUPPORT	Procedimiento	Establece que el paquete UTL_HTTP abortará por una excepción detallada.
SET_FOLLOW_REDIRECT	Procedimiento	Establece el número máximo de veces que UTL_HTTP seguirán las instrucciones de redireccionamiento HTTP en la respuesta a las peticiones futuras en la función GET_RESPONSE
SET_PERSISTENT_CONN_ SUPPORT	Procedimiento	Establece si la petición HTTP soportará o no conexiones persistentes HTTP 1.1. Establece el número máximo de conexiones persistentes mantenidas en la sesión actual de base de datos del usuario.
SET_PROXY	Procedimiento	Establece el proxy que se utilizará para las peticiones de HTTP u otros protocolos.
		Establece si GET_RESPONSE abortará o no por una excepción

SET_RESPONSE_ERROR_ CHECK	Procedimiento	cuando el servidor web devuelva un código de estado que indique un error (códigos en los rangos 4xx o 5xx).
SET_TRANSFER_TIMEOUT	Procedimiento	Establece el valor del TIMEOUT para las lecturas de las respuestas HTTP desde el servidor web o servidor proxy.
SET_WALLET	Procedimiento	Establece el Oracle Wallet utilizado para todas las peticiones HTTP sobre SSL, que sean HTTPS.

Los subprogramas de peticiones HTTP son los siguientes:

Nombre Subprograma	Tipo	Descripción
BEGIN_REQUEST	Función	Comienza una nueva petición HTTP.
SET_HEADER	Procedimiento	Establece una cabecera de petición HTTP.
SET_AUTHENTICATION	Procedimiento	Establece la información de autenticación en la cabecera de petición HTTP.
SET_AUTHENTICATION_ FROM_WALLET	Procedimiento	Establece la información de autenticación en la cabecera de petición HTTP, necesaria para la petición de autorización al servidor web, usando las credenciales (usuario y password) almacenadas en Oracle Wallet.
SET_BODY_CHARSET	Procedimiento	Establece el conjunto de caracteres del cuerpo de la petición cuando el MEDIA TYPE es texto y no se ha especificado un conjunto de caracteres en la cabecera CONTENT-TYPE.
SET_COOKIE_SUPPORT	Procedimiento	Establece si una futura petición HTTP soportará o no cookies. Establece el número máximo de cookies mantenidas en la sesión actual de base de datos del usuario.
SET_FOLLOW_REDIRECT	Procedimiento	Establece el número máximo de veces que UTL_HTTP seguirán las instrucciones de redireccionamiento HTTP en la respuesta a las peticiones futuras en la función GET_RESPONSE
SET_PERSISTENT_CONN_ SUPPORT	Procedimiento	Establece si la petición HTTP soportará o no conexiones persistentes HTTP 1.1.
SET_PROXY	Procedimiento	Escribe una línea de texto en el cuerpo de la petición http y finaliza la línea con un carácter de nueva línea (CRLF – retorno de carro y salto de línea).
WRITE_RAW	Procedimiento	Escribe algunos datos binarios en el cuerpo de la petición HTTP.
WRITE_TEXT	Procedimiento	Escribe algunos datos de texto en el cuerpo de la petición HTTP.

Los subprogramas de peticiones contextuales HTTP son los siguientes:

Nombre Subprograma	Tipo	Descripción
CREATE_REQUEST_ CONTEXT	Función	Crea una petición contextual en UTL_HTTP para una cartera (WALLET) y una tabla de cookies.
DESTROY_REQUEST_ CONTEXT	Procedimiento	Destruye la petición contextual en UTL_HTTP.

Los subprogramas de COOKIES HTTP son los siguientes:

Nombre Subprograma	Tipo	Descripción
END_RESPONSE	Procedimiento	Finaliza la respuesta HTTP.
GET_AUTHENTICATION	Procedimiento	Recupera la información de autenticación http necesaria para que la petición sea aceptada por el servidor web como se indica en la cabecera de respuesta HTTP.
GET_HEADER	Procedimiento	Devuelve los "n" nombres de la cabecera de respuesta HTTP y el valor devuelto en la respuesta.
GET_HEADER_BY_NAME	Procedimiento	Devuelve el valor de la cabecera de respuesta devuelto al acceder por el nombre de la cabecera.
GET_HEADER_COUNT	Función	Devuelve el número de cabeceras de respuesta HTTP devueltas en la respuesta.
GET_RESPONSE	Función	Lee la respuesta HTTP.
READ_LINE	Procedimiento	Lee el cuerpo de la respuesta HTTP en formato de texto hasta que se alcance el final de línea y devuelve la salida en un buffer.
READ_RAW	Procedimiento	Lee el cuerpo de la respuesta HTTP en formato binario y devuelve la salida en un buffer.
READ_TEXT	Procedimiento	Lee el cuerpo de la respuesta HTTP en formato texto y devuelve la salida en un buffer.
SET_BODY_CHARSET	Procedimiento	Establece el conjunto de caracteres del cuerpo de la petición cuando el MEDIA TYPE es texto y no se ha especificado un conjunto de caracteres en la cabecera CONTENT-TYPE.

Los subprogramas de respuestas HTTP son los siguientes:

Nombre Subprograma	Tipo	Descripción
ADD_COOKIES	Procedimiento	Añade la cookie a la petición contextual o al estado de la sesión del paquete UTL_HTTP.
		Limpia todas las cookies que se

CLEAR_COOKIES	Procedimiento	mantienen en la petición contextual o en el estado de la sesión del paquete UTL_HTTP.
GET_COOKIE_COUNT	Función	Devuelve el número de cookies que se mantienen en la petición contextual o en el estado de la sesión del paquete UTL_HTTP.
GET_COOKIES	Función	Devuelve todas las cookies que se mantienen en la petición contextual o en el estado de la sesión del paquete UTL_HTTP.

Los subprogramas de las conexiones persistentes HTTP son los siguientes:

Nombre Subprograma	Tipo	Descripción
CLOSE_PERSISTENT_ CONN	Procedimiento	Cierra una conexión persistente HTTP que se mantiene por el paquete UTL_HTTP en la sesión de base de datos actual.
CLOSE_PERSISTENT_ CONNS	Procedimiento	Cierra un grupo de conexiones persistentes http que se mantienen por el paquete UTL_HTTP en la sesión de base de datos actual.
GET_PERSISTENT_CONN _COUNT	Función	Devuelve el número de conexiones de red que actualmente se mantienen persistentes por el paquete UTL_HTTP a los servidores web.
GET_PERSISTENT_CONNS	Procedimiento	Devuelve todas las conexiones de red que actualmente se mantienen persistentes por el paquete UTL_HTTP a los servidores web.

Los subprogramas para el control de las condiciones de error son los siguientes:

Nombre Subprograma	Tipo	Descripción
GET_DETAILED_SQLCODE	Función	Devuelve el valor de SQLCODE del último error que se ha producido.
GET_DETAILED_SQLERRM	Función	Devuelve el valor de SQLERRM del último error que se ha producido.

EJEMPLO

A continuación, se muestra un ejemplo de uso del paquete UTL_HTTP de manera general:

```
SET SERVEROUTPUT ON SIZE 40000

DECLARE
req UTL_HTTP.REQ
resp UTL_HTTP.RESP;
```

```
value VARCHAR2(1024);

BEGIN
UTL_HTTP.SET_PROXY('proxy.my-company.com',
'corp.my-company.com');
req := UTL_HTTP.BEGIN_REQUEST('http://www-hr.corp.my-company.com');
UTL_HTTP.SET_HEADER(req, 'User-Agent',
'Mozilla/4.0');
resp := UTL_HTTP.GET_RESPONSE(req);
LOOP
UTL_HTTP.READ_LINE(resp, value, TRUE);
DBMS_OUTPUT.PUT_LINE(value);
END LOOP;
UTL_HTTP.END_RESPONSE(resp);
EXCEPTION
WHEN UTL_HTTP.END_OF_BODY THEN
UTL_HTTP.END_RESPONSE(resp);
END;
```

Errores de la sección EXCEPTION

El paquete UTL_HTTP puede devolver los siguientes errores que se describen a continuación:

Excepción	Cód.Error	Descripción
BAD_ARGUMENT	29261	Son incorrectos los argumentos que se han pasado a la interfaz.
BAD_URL	29262	La petición URL está incorrectamente formulada.
END_OF_BODY	29266	Se alcanzó el final del cuerpo de respuesta HTTP.
HEADER_NOT_FOUND	29265	No se encuentra la cabecera.
HTTP_CLIENT_ERROR	29268	Desde GET_RESPONSE, el parámetro STATUS_CODE indica que ha ocurrido un error en el cliente (código de error en el rango 4xx). Desde BEGIN_REQUEST, indica que el proxy ha devuelto un código de estado en el rango 4xx cuando hizo una petición HTTPS a través del proxy.
HTTP_SERVER_ERROR	29269	Desde GET_RESPONSE, el código de estado de respuesta indica que ha ocurrido un error en el cliente (código de error en el rango 5xx). O desde BEGIN_REQUEST, el proxy HTTP devuelve un código de estado en el rango 4xx cuando una petición HTTPS a través del proxy.
NETWORK_ACCESS_ DENIED	24247	El acceso al equipo de red remoto o las credenciales de Oracle se han denegado.
ILLEGAL_CALL	29267	La llamada a UTL_HTTP es ilegal en el estado actual de la petición HTTP.
PARTIAL_MULTIBYTE_ EXCEPTION	29275	No se ha leído ningún carácter completo y se encontró un carácter parcial multibyte al final del cuerpo de respuesta.
PROTOCOL_ERROR	29263	Un error de protocolo HTTP ocurrió cuando se comunicaba con el servidor web.
REQUEST_FAILED	29273	La petición falló al ejecutarse.
TOO_MANY_REQUESTS	29270	Demasiadas peticiones o respuestas están abiertas.
TRANSFER_TIMEOUT	29276	No se ha leído ningún dato y ocurrió un timeout (finalización por exceso de tiempo) en la lectura.
UNKNOWN_SCHEME	29264	El esquema de la petición URL es desconocido.

Tipo REQ

Se usa este tipo de registro PL/SQL para representar una petición HTTP.

La información que se devuelve sobre este tipo desde la interfaz BEGIN_REQUEST es solo para lectura. Los cambios que se hagan en los valores del registro no tienen efecto sobre la petición.

SINTAXIS

La sintaxis de este tipo de dato es la siguiente:

```
TYPE req IS RECORD( url VARCHAR2(32767),
                    method VARCHAR2(64),
                    http_version VARCHAR2(64));
```

DESCRIPCIÓN DE LOS parámetros DEL TIPO DE DATO

A continuación, se describe cada uno de los parámetros que se usan en la definición del tipo de dato REQ.

url

La URL de la petición HTTP.

method

Es el método a llevar a cabo en el recurso identificado por la URL.

http_version

La versión del protocolo HTTP que se usa para enviar la petición.

Tipo REQUEST_CONTEXT_TYPE

Este tipo se usa para representar la clave del contexto de una petición. El contexto de una petición engloba una parte privada y una tabla de cookies para hacer la petición HTTP.

SINTAXIS

La sintaxis de este tipo de dato es la siguiente:

```
TYPE request_context_key IS PLS_INTEGER;
```

Tipo RESP

Se usa este tipo de registro PL/SQL para representar una respuesta HTTP.

La información que se devuelve sobre este tipo desde la interfaz GET_RESPONSE es solo para lectura.

SINTAXIS

La sintaxis de este tipo de dato es la siguiente:

```
TYPE resp IS RECORD( status_code VARCHAR2(32767),
                     reason_phrase VARCHAR2(64),
                     http_version VARCHAR2(64));
```

DESCRIPCIÓN DE LOS parámetros DEL TIPO DE DATO

A continuación, se describe cada uno de los parámetros que se usan en la definición del tipo de dato RESP.

status_code

El código de estado devuelto por el servidor web. Es un entero de 3 dígitos que indica el resultado de la petición http como manejador del servidor web.

reason_phrase

El mensaje corto de texto devuelto por el servidor web que describe el código de estado. Ofrece una breve descripción del resultado de la petición HTTP.

http_version

La versión del protocolo HTTP que se usa para enviar la respuesta.

Tipos COOKIE y COOKIE_TABLE

El tipo COOKIE es un tipo registro PL/SQL que representa una cookie HTTP. El tipo COOKIE_TABLE es un tipo tabla PL/SQL que almacena elementos COOKIE en su colección de datos.

SINTAXIS

La sintaxis del tipo COOKIE es la siguiente:

```
TYPE cookie IS RECORD(
        name VARCHAR2(256),
        value VARCHAR2(1024),
        domain VARCHAR2(256),
        expire TIMESTAMP WITH TIME ZONE,
        path VARCHAR2(1024),
        secure BOOLEAN,
        version PLS_INTEGER,
        comment VARCHAR2(1024));
```

La sintaxis del tipo COOKIE_TABLE es la siguiente:

```
TYPE cookie_table IS TABLE OF cookie INDEX BY
                BINARY_INTEGER;
```

DESCRIPCIÓN DE LOS parámetros DEL TIPO DE DATO cookie

A continuación, se describe cada uno de los parámetros que se usan en la definición del tipo de dato COOKIE.

name

El nombre de la cookie HTTP.

value

El valor de la cookie.

domain

El dominio por el que la cookie se valida.

expire

El tiempo máximo en el que la cookie expirará.

path

El subconjunto de URLs que aplica la cookie.

secure

Indica que la cookie debería ser devuelta al servidor web usando solo mecanismos de seguridad.

version

La versión de la especificación del HTTP de la cookie. Este campo estará a NULL para cookies de Netscape.

comment

El comentario que describe la intención de uso de la cookie. Este campo estará a NULL para cookies de Netscape.

Tipo CONNECTION

Se usa este tipo de registro PL/SQL para representar el equipo remoto y puertos TCP/IP de una conexión de red que se mantiene persistente después de que se complete una petición HTTP de acuerdo a la especificación del protocolo HTTP 1.1.

Una conexión de red persistente puede ser reutilizada por una subsecuencia de petición HTTP en el mismo equipo y puerto.

Para una conexión HTTP persistente a un servidor web, los parámetros HOST y PORT contienen en el nombre del equipo y el puerto TCP/IP (respectivamente), del servidor web.

El parámetro SSL indica si se está utilizando SSL (Secured Socked Layer) en una conexión HTTP persistente. Una petición HTTPS es una petición HTTP realizada sobre SSL.

SINTAXIS

La sintaxis de este tipo de dato es la siguiente:

```
TYPE connection IS RECORD(
                        host VARCHAR2(256),
                        port PLS_INTEGER,
                        proxy_host VARCH2(256),
                        proxy_port PLS_INTEGER,
                        ssl BOOLEAN));
```

DESCRIPCIÓN DE LOS parámetros DEL TIPO DE DATO

A continuación, se describe cada uno de los parámetros que se usan en la definición del tipo de dato CONNECTION.

host

El nombre del equipo al que se quiere conectar.

port

El puerto por el que se va a conectar con el equipo.

proxy_host

El nombre del proxy que se utiliza en la conexión.

proxy_port

El puerto del proxy que se utiliza en la conexión.

ssl

TRUE para indicar que se va a utilizar HTTPS.

Procedimiento ADD_COOKIES

Añade la cookie a la petición contextual o al estado de la sesión del paquete UTL_HTTP.

SINTAXIS

La sintaxis de este procedimiento es la siguiente:

```
UTL_HTTP.ADD_COOKIES(
        cookies IN cookie_table,
        request_context IN resquest_context_key
                                DEFAULT NULL);
```

DESCRIPCIÓN DE LOS parámetros DEL PROCEDIMIENTO

A continuación, se describe cada uno de los parámetros que se usan en la definición del procedimiento ADD_COOKIES.

cookies

Las cookies que se añadirán.

request_context

La petición contextual para añadir las cookies. Si es NULL, las cookies serán añadidas al estado de la sesión del paquete UTL_HTTP en vez de a la petición contextual.

Función BEGIN_REQUEST

Comienza una nueva petición HTTP. UTL_HTTP establece la conexión de red al servidor web de origen o al servidor proxy y envía la petición HTTP. El programa PL/SQL continúa la petición utilizando llamadas a otras interfaces para poder completarla. La URL puede contener el nombre del usuario y el password necesarios para autenticar la petición en el servidor. El formato es:

scheme://[user[:password]@host[:port]/[…]

SINTAXIS

La sintaxis de esta función es la siguiente:

```
UTL_HTTP.BEGIN_REQUEST (url IN VARCHAR2,
method IN VARCHAR2 DEFAULT 'GET',
http_version IN VARCHAR2 DEFAULT NULL,
request_context IN request_context_key DEFAULT NULL)
RETURN req;
```

DESCRIPCIÓN DE LOS parámetros DE La función

A continuación, se describe cada uno de los parámetros que se usan en la definición de la función BEGIN_REQUEST.

url

La URL de la petición HTTP.

method

El método implementado en el recurso identificado por la URL. Por defecto, si no se especifica nada, se asumirá el método GET.

http_version

La versión del protocolo HTTP que se envía en la petición. El formato de la versión del protocolo es: HTTP/major-version.minor-version, donde major-version y minor-version son números positivos. Si el parámetro tiene un valor NULL, UTL_HTTP utilizará la versión del último protocolo HTTP que se soporta para el envío de la petición. La última versión que el paquete soporta es

1.1 y puede ser actualizada a una versión posterior. El valor por defecto del parámetro es NULL.

request_context

La petición contextual que soporta tanto la cartera (WALLET) privada como la tabla de cookies que se usa en la petición HTTP. Si el parámetro es NULL, se usará la cartera (WALLET) y la tabla de cookies compartidas en la sesión de base de datos actual.

RETURN REQ

Esta función devuelve un tipo REQ en la respuesta a la llamada que se haga sobre la misma.

Procedimiento CLEAR_COOKIES

Limpia todas las cookies que se mantienen en la petición contextual o en el estado de la sesión del paquete UTL_HTTP.

SINTAXIS

La sintaxis de este procedimiento es la siguiente:

```
UTL_HTTP.CLEAR_COOKIES(
        request_context IN resquest_context_key
                                DEFAULT NULL);
```

DESCRIPCIÓN DE LOS parámetros DEL PROCEDIMIENTO

A continuación, se describe cada uno de los parámetros que se usan en la definición del procedimiento CLEAR_COOKIES.

request_context

La petición contextual para eliminar las cookies. Si es NULL, se limpiarán las cookies que se mantienen en el estado de la sesión del paquete UTL_HTTP.

Procedimiento CLOSE_PERSISTENT_CONN

Cierra una conexión persistente HTTP que se mantiene por el paquete UTL_HTTP en la sesión de base de datos actual.

SINTAXIS

La sintaxis de este procedimiento es la siguiente:

```
UTL_HTTP.CLOSE_PERSISTENT_CONN (conn IN connection);
```

DESCRIPCIÓN DE LOS parámetros DEL PROCEDIMIENTO

A continuación, se describe cada uno de los parámetros que se usan en la definición del procedimiento CLOSE_PERSISTENT_CONN.

conn

La conexión HTTP persistente que se va a cerrar.

Procedimiento CLOSE_PERSISTENT_CONNS

Cierra un grupo de conexiones persistentes http que se mantienen por el paquete UTL_HTTP en la sesión de base de datos actual.

SINTAXIS

La sintaxis de este procedimiento es la siguiente:

```
UTL_HTTP.CLOSE_PERSISTENT_CONNS (
    host IN VARCHAR2 DEFAULT NULL,
    port IN PLS_INTEGER DEFAULT NULL,
    proxy_host IN VARCHAR2 DEFAULT NULL,
    proxy_port IN PLS_INTEGER DEFAULT NULL,
    ssl IN BOOLEAN DEFAULT NULL);
```

DESCRIPCIÓN DE LOS parámetros DEL PROCEDIMIENTO

A continuación, se describe cada uno de los parámetros que se usan en la definición del procedimiento CLOSE_PERSISTENT_CONNS.

host

El equipo para el que se cerrarán las conexiones persistentes.

port

El número del puerto para el que se cerrarán las conexiones persistentes.

proxy_host

El equipo proxy para el que se cerrarán las conexiones persistentes.

proxy_port

El número del puerto del proxy para el que se cerrarán las conexiones persistentes.

ssl

Cierra las conexiones persistentes SSL.

EJEMPLOS

En el siguiente ejemplo se cierran todas las conexiones permanentes a un equipo denominado SERVIDOR1:

```
UTL_HTTP.CLOSE_PERSISTENT_CONNS(host => 'servidor1');
```

En el siguiente ejemplo se cierran todas las conexiones a través del proxy PROXY1, por el puerto 80:

```
UTL_HTTP.CLOSE_PERSISTENT_CONNS(proxy_host => 'proxy1',
                                proxy_port => 80);
```

En este último ejemplo se cierran todas las conexiones permanentes:

```
UTL_HTTP.CLOSE_PERSISTENT_CONNS;
```

Función CREATE_REQUEST_CONTEXT

Crea una petición contextual. Una petición contextual es un contexto que soporta una cartera (WALLET) y una cookie para uso privado en la petición HTTP. Esto permite a la petición HTTP usar una cartera (WALLET) y una tabla de cookies que no será compartida con otras aplicaciones que hacen peticiones HTTP en la misma sesión de base de datos.

SINTAXIS

La sintaxis de esta función es la siguiente:

```
UTL_HTTP.CREATE_REQUEST_CONTEXT(
wallet_path IN VARCHAR2 DEFAULT NULL,
wallet_password IN VARCHAR2 DEFAULT NULL,
enable_cookies IN BOOLEAN DEFAULT TRUE,
max_cookies IN PLS_INTEGER DEFAULT 300,
max_cookies_per_site IN PLS_INTEGER DEFAULT 20)
RETURN request_context_key;
```

DESCRIPCIÓN DE LOS parámetros DE La función

A continuación, se describe cada uno de los parámetros que se usan en la definición de la función CREATE_REQUEST_CONTEXT.

wallet_path

La ruta del directorio que contiene la cartera (WALLET) Oracle. El formato es file:directory-path.

wallet_password

El password necesario para abrir la cartera (WALLET). Si la cartera tiene habilitado el sistema de auto-conexión, el password se podrá omitir y debería establecerse este parámetro a NULL.

enable_cookies

Establece si la petición HTTP debería soportar cookies HTTP o no. TRUE para habilitar que las soporte y FALSE para deshabilitarlo.

max_cookies

Establece el número máximo de cookies que se mantendrán en la petición contextual.

max_cookies_per_site

Establece el número máximo de cookies por cada sitio web que se mantendrán en la petición contextual.

RETURN REQUEST_CONTEXT_KEY

Esta función devuelve la petición contextual creada, mediante un tipo REQUEST_CONTEXT_KEY.
EJEMPLO

```
DECLARE
request_context UTL_HTTP.REQUEST_CONTEXT_KEY;
req UTL_HTTP.req;

BEGIN
request_context := UTL_HTTP.CREATE_REQUEST_CONTEXT(
wallet_path => 'file:/oracle/wallets/test_wallets',
wallet_password => NULL,
enable_cookies => TRUE,
max_cookies => 300,
max_cookies_per_site => 20);
req := UTL_HTTP.BEGIN_REQUEST(
url => 'http://www.example.com/',
request_context => request_context);
END;
```

Procedimiento DESTROY_REQUEST_CONTEXT

Destruye una petición contextual en UTL_HTTP. Una petición contextual no puede ser destruida cuando está siendo usada por una petición o respuesta HTTP.

SINTAXIS

La sintaxis de este procedimiento es la siguiente:

```
UTL_HTTP.DESTROY_REQUEST_CONTEXT(
    request_context IN request_context_key);
```

DESCRIPCIÓN DE LOS parámetros DEL PROCEDIMIENTO

A continuación, se describe cada uno de los parámetros que se usan en la definición del procedimiento DESTROY_REQUEST_CONTEXT.

request_context

Petición contextual a destruir.

EJEMPLO

```
DECLARE
request_context UTL_HTTP.REQUEST_CONTEXT_KEY;

BEGIN
request_context := UTL_HTTP.CREATE_REQUEST_CONTEXT(…);
UTL_HTTP.DESTROY_REQUEST_CONTEXT(request_context);
END;
```

Procedimiento END_REQUEST

Finaliza una petición HTTP. Para terminarla sin completar la petición y esperar una respuesta, el programa puede llamar a este procedimiento. La otra manera de operar debería seguir la secuencia normal que consiste en comenzar una petición, obtener la respuesta y cerrar la respuesta. La conexión de red siempre será cerrada y no será reusada.

SINTAXIS

La sintaxis de este procedimiento es la siguiente:

```
UTL_HTTP.END_REQUEST (r IN OUT NOCOPY req);
```

DESCRIPCIÓN DE LOS parámetros DEL PROCEDIMIENTO

A continuación, se describe cada uno de los parámetros que se usan en la definición del procedimiento END_REQUEST.

r

La petición HTTP.

Procedimiento END_RESPONSE

Finaliza una respuesta HTTP. Completa la petición y la respuesta HTTP.

SINTAXIS

La sintaxis de este procedimiento es la siguiente:

```
UTL_HTTP.END_RESPONSE(r IN OUT NOCOPY resp);
```

DESCRIPCIÓN DE LOS parámetros DEL PROCEDIMIENTO

A continuación, se describe cada uno de los parámetros que se usan en la definición del procedimiento END_RESPONSE.

r

La respuesta HTTP.

Procedimiento GET_AUTHENTICATION

Devuelve la información de autenticación HTTP necesaria para que la petición sea aceptada por el servidor web como se indicó en la cabecera de respuesta HTTP.

SINTAXIS

La sintaxis de este procedimiento es la siguiente:

```
UTL_HTTP.GET_AUTHENTICATION(
        r IN OUT NOCOPY resp,
        scheme OUT VARCHAR2,
        realm OUT VARCHAR2,
        for_proxy IN BOOLEAN DEFAULT FALSE);
```

DESCRIPCIÓN DE LOS parámetros DEL PROCEDIMIENTO

A continuación, se describe cada uno de los parámetros que se usan en la definición del procedimiento GET_AUTHENTICATION.

r

La respuesta HTTP.

scheme

El esquema para la autenticación HTTP requerida.

realm

El realm para la autenticación HTTP requerida.

for_proxy

Devuelve la información de autenticación HTTP requerida para el acceso al servidor proxy HTTP en vez de la del servidor web. El valor por defecto es FALSE.

Procedimiento GET_BODY_CHARSET

Devuelve el juego de caracteres por defecto para el cuerpo de todas las próximas peticiones HTTP.

SINTAXIS

La sintaxis de este procedimiento es la siguiente:

```
UTL_HTTP.GET_BODY_CHARSET(charset OUT VARCHAR2);
```

DESCRIPCIÓN DE LOS parámetros DEL PROCEDIMIENTO

A continuación, se describe cada uno de los parámetros que se usan en la definición del procedimiento GET_BODY_CHARSET.

charset

El juego de caracteres por defecto.

Función GET_COOKIE_COUNT

Devuelve el número de cookies que se mantienen tanto en la petición contextual como en la sesión de estado del paquete UTL_HTTP.

SINTAXIS

La sintaxis de esta función es la siguiente:

```
UTL_HTTP.GET_COOKIE_COUNT (
request_context IN request_context_key DEFAULT NULL)
RETURN PLS_INTEGER;
```

DESCRIPCIÓN DE LOS parámetros DE La función

A continuación, se describe cada uno de los parámetros que se usan en la definición de la función GET_COOKIE_COUNT.

request_context

La petición contextual para la que devuelve el conteo de cookies. Si es NULL, el conteo que se devuelve corresponde a la sesión de estado del paquete UTL_HTTP.

RETURN PLS_INTEGER

Esta función devuelve en un tipo PLS_INTEGER el conteo de cookies.

Procedimiento GET_COOKIE_SUPPORT

Devuelve la configuración de soporte de la cookie actual.

SINTAXIS

La sintaxis de este procedimiento es la siguiente:

```
UTL_HTTP.GET_COOKIE_SUPPORT(
    enable OUT BOOLEAN,
    max_cookies OUT PLS_INTEGER,
    max_cookies_per_site OUT PLS_INTEGER);
```

DESCRIPCIÓN DE LOS parámetros DEL PROCEDIMIENTO

A continuación, se describe cada uno de los parámetros que se usan en la definición del procedimiento GET_COOKIE_SUPPORT.

enable

Indica si la futura petición HTTP debería soportar cookies HTTP (en caso de que el valor sea TRUE) o no las debería soportar (en caso de que el valor sea FALSE).

max_cookies

Indica el número total máximo de cookies que se mantienen en la sesión actual.

max_cookies_per_site

Indica el número total máximo de cookies que se mantienen en la sesión actual de cada sitio web.

Función GET_COOKIES

Devuelve todas las cookies que se mantienen tanto en la petición contextual como en el estado de la sesión del paquete UTL_HTTP.

SINTAXIS

La sintaxis de esta función es la siguiente:

```
UTL_HTTP.GET_COOKIES (
cookies IN OUT NOCOPY cookie_table,
request_context IN request_context_key DEFAULT NULL);
```

DESCRIPCIÓN DE LOS parámetros DE La función

A continuación, se describe cada uno de los parámetros que se usan en la definición de la función GET_COOKIES.

cookies

Las cookies devueltas.

request_context

La petición contextual para la que devuelve las cookies. Si el valor es NULL, se devolverán las cookies del estado de la sesión del paquete UTL_HTTP.

Procedimiento GET_DETAILED_EXCP_SUPPORT

Chequea si el paquete UTL_HTTP alcanzará una excepción detallada o no.

SINTAXIS

La sintaxis de este procedimiento es la siguiente:

```
UTL_HTTP.GET_DETAILED_EXCP_SUPPORT(
      enable OUT BOOLEAN);
```

DESCRIPCIÓN DE LOS parámetros DEL PROCEDIMIENTO

A continuación, se describe cada uno de los parámetros que se usan en la definición del procedimiento GET_DETAILED_EXCP_SUPPORT.

enable

TRUE si UTL_HTTP alcanza una excepción detallada; en caso contrario, FALSE.

Función GET_DETAILED_SQLCODE

Devuelve el parámetro SQLCODE detallado de una excepción alcanzada.

SINTAXIS

La sintaxis de esta función es la siguiente:

```
UTL_HTTP.GET_DETAILED_SQLCODE
RETURN PLS_INTEGER;
```

Función GET_DETAILED_SQLERRM

Devuelve el parámetro SQLERRM de la última excepción alcanzada.

SINTAXIS

La sintaxis de esta función es la siguiente:

```
UTL_HTTP.GET_DETAILED_SQLERRM
RETURN VARCHAR2;
```

Procedimiento GET_FOLLOW_REDIRECT

Devuelve la configuración del siguiente redireccionamiento en la sesión actual.

SINTAXIS

La sintaxis de este procedimiento es la siguiente:

```
UTL_HTTP.GET_FOLLOW_REDIRECT(
      Max_redirects OUT PLS_INTEGER);
```

DESCRIPCIÓN DE LOS parámetros DEL PROCEDIMIENTO

A continuación, se describe cada uno de los parámetros que se usan en la definición del procedimiento GET_FOLLOW_REDIRECT.

max_redirects

El número máximo de redireccionamientos para todas las peticiones http futuras.

Procedimiento GET_HEADER

Devuelve el enésimo nombre de cabecera de respuesta HTTP y el valor devuelto en la respuesta.

SINTAXIS

La sintaxis de este procedimiento es la siguiente:

```
UTL_HTTP.GET_HEADER(
r IN OUT NOCOPY resp,
n IN PLS_INTEGER,
name OUT NOCOPY VARCHAR2,
value OUT NOCOPY VARCHAR2);
```

DESCRIPCIÓN DE LOS parámetros DEL PROCEDIMIENTO

A continuación, se describe cada uno de los parámetros que se usan en la definición del procedimiento GET_HEADER.

r

La respuesta HTTP.

n

La enésima cabecera a devolver.

name

El nombre de la cabecera de respuesta HTTP.

value

El valor de la cabecera de respuesta HTTP.

Procedimiento GET_HEADER_BY_NAME

Devuelve el valor de la cabecera de respuesta, devuelto en la respuesta dada por el nombre de la cabecera.

SINTAXIS

La sintaxis de este procedimiento es la siguiente:

```
UTL_HTTP.GET_HEADER_BY_NAME(
r IN OUT NOCOPY resp,
name IN VARCHAR2,
value OUT NOCOPY VARCHAR2,
n IN PLS_INTEGER DEFAULT 1);
```

DESCRIPCIÓN DE LOS parámetros DEL PROCEDIMIENTO

A continuación, se describe cada uno de los parámetros que se usan en la definición del procedimiento GET_HEADER_BY_NAME.

r

La respuesta HTTP.

name

El nombre de la cabecera de respuesta HTTP para la que el valor es devuelto.

value

El valor de la cabecera de respuesta HTTP.

n

La enésima ocurrencia de una cabecera de respuesta HTTP por el nombre especificado para devolver. El valor por defecto es 1.

Función GET_HEADER_COUNT

Devuelve el número de cabeceras de respuesta HTTP devuelta en la respuesta.

SINTAXIS

La sintaxis de esta función es la siguiente:

```
UTL_HTTP.GET_HEADER_COUNT (
      r IN OUT NOCOPY resp)
RETURN PLS_INTEGER;
```

DESCRIPCIÓN DE LOS parámetros DE La función

A continuación, se describe cada uno de los parámetros que se usan en la definición de la función GET_HEADER_COUNT.

r

La respuesta HTTP.

RETURN PLS_INTEGER

Esta función devuelve en un tipo PLS_INTEGER el número de cabeceras.

Procedimiento GET_PERSISTENT_CONN_SUPPORT

Este procedimiento chequea:

- Si está habilitado el soporte para las conexiones persistentes.
- Obtención del número máximo de conexiones persistentes en la sesión actual.

SINTAXIS

La sintaxis de este procedimiento es la siguiente:

```
UTL_HTTP.GET_PERSISTENT_CONN_SUPPORT (
    enable OUT BOOLEAN,
    max_conns OUT PLS_INTEGER);
```

DESCRIPCIÓN DE LOS parámetros del procedimiento

A continuación, se describe cada uno de los parámetros que se usan en la definición del procedimiento GET_PERSISTENT_CONN_SUPPORT.

enable

TRUE si se ha habilitado el soporte para conexiones persistentes; en caso contrario FALSE.

max_conns

El número máximo de conexiones persistentes que se mantienen en la sesión actual.

Procedimiento GET_PERSISTENT_CONNS

Devuelve todas las conexiones de red que se mantienen persistentes por el paquete UTL_HTTP en los servidores web.

SINTAXIS

La sintaxis de este procedimiento es la siguiente:

```
UTL_HTTP.GET_PERSISTENT_CONNS (
connections IN OUT NOCOPY connection_table);
```

DESCRIPCIÓN DE LOS parámetros DEL PROCEDIMIENTO

A continuación, se describe cada uno de los parámetros que se usan en la definición del procedimiento GET_PERSISTENT_CONNS.

connections

Las conexiones de red que se mantienen persistentes.

Función GET_PERSISTENT_CONN_COUNT

Devuelve el número de conexiones de red que actualmente se mantienen persistentes por el paquete UTL_HTTP del servidor web.

SINTAXIS

La sintaxis de esta función es la siguiente:

```
UTL_HTTP.GET_PERSISTENT_CONN_COUNT
RETURN PLS_INTEGER;
```

Procedimiento GET_PROXY

Devuelve la configuración actual del proxy.

SINTAXIS

La sintaxis de este procedimiento es la siguiente:

```
UTL_HTTP.GET_PROXY (
proxy OUT NOCOPY VARCHAR2,
no_proxy_domains OUT NOCOPY VARCHAR2);
```

DESCRIPCIÓN DE LOS parámetros DEL PROCEDIMIENTO

A continuación, se describe cada uno de los parámetros que se usan en la definición del procedimiento GET_PROXY.

proxy

El proxy (equipo y número de puerto opcional) que está siendo usado actualmente por el paquete UTL_HTTP.

no_proxy_domains

La lista de los equipos y dominios en los que no se utiliza ningún proxy para todas las peticiones.

Función GET_RESPONSE

Esta función devuelve la respuesta HTTP. Cuando la función devuelve la información, entonces la línea de estado y las cabeceras de respuesta HTTP han sido leídas y procesadas. Esta función completa la sección de cabeceras HTTP.

SINTAXIS

La sintaxis de esta función es la siguiente:

```
UTL_HTTP.GET_RESPONSE (
      r IN OUT NOCOPY req)
RETURN resp;
```

DESCRIPCIÓN DE LOS parámetros DE La función

A continuación, se describe cada uno de los parámetros que se usan en la definición de la función GET_RESPONSE.

r

La respuesta HTTP.

RETURN resp

Esta función devuelve en un tipo RESP la respuesta HTTP.

Procedimiento GET_RESPONSE_ERROR_CHECK

Chequea si se ha habilitado o no la comprobación de errores en la respuesta.

SINTAXIS

La sintaxis de este procedimiento es la siguiente:

```
UTL_HTTP.GET_RESPONSE_ERROR_CHECK(
enable OUT BOOLEAN);
```

DESCRIPCIÓN DE LOS parámetros DEL PROCEDIMIENTO

A continuación, se describe cada uno de los parámetros que se usan en la definición del procedimiento GET_RESPONSE_ERROR_CHECK.

enable

TRUE si se ha habilitado la comprobación de errores en la respuesta. En otro caso FALSE.

Procedimiento GET_TRANSFER_TIMEOUT

Retorna el valor por defecto del TIMEOUT (tiempo máximo de espera) para todas las futuras respuestas HTTP.

SINTAXIS

La sintaxis de este procedimiento es la siguiente:

```
UTL_HTTP.GET_TRANSFER_TIMEOUT(
timeout OUT PLS_INTEGER);
```

DESCRIPCIÓN DE LOS parámetros DEL PROCEDIMIENTO

A continuación, se describe cada uno de los parámetros que se usan en la definición del procedimiento GET_TRANSFER_TIMEOUT.

timeout

El valor del TIMEOUT de transferencia de la red, medido en segundos.

Procedimiento READ_LINE

Lee el cuerpo de la respuesta HTTP en formato de texto hasta que se alcance el final de línea y devuelve la salida al buffer. El final de línea es el que se haya definido en la función READ_LINE de UTL_TCP. La excepción END_OF_BODY se alcanza cuando se llega al final del cuerpo de respuesta HTTP. La información que se obtiene del cuerpo de la respuesta es convertida automáticamente desde el juego de caracteres del propio cuerpo al juego de caracteres de la base de datos.

SINTAXIS

La sintaxis de este procedimiento es la siguiente:

```
UTL_HTTP.READ_LINE (
r IN OUT NOCOPY resp,
data OUT NOCOPY VARCHAR2 CHARACTER SET ANY_CS,
remove_crlf IN BOOLEAN DEFAULT FALSE);
```

DESCRIPCIÓN DE LOS parámetros DEL PROCEDIMIENTO

A continuación, se describe cada uno de los parámetros que se usan en la definición del procedimiento READ_LINE.

r

La respuesta HTTP.

data

El cuerpo de respuesta HTTP en formato de texto.

remove_crlf

Borra los caracteres de línea nueva si se le indica el valor TRUE.

Procedimiento READ_RAW

Lee el cuerpo de la respuesta HTTP en formato binario y devuelve la salida al buffer. La excepción END_OF_BODY se alcanza cuando se llega al final del cuerpo de respuesta HTTP.

SINTAXIS

La sintaxis de este procedimiento es la siguiente:

```
UTL_HTTP.READ_RAW(
r IN OUT NOCOPY resp,
data OUT NOCOPY RAW,
len IN PLS_INTEGER DEFAULT NULL);
```

DESCRIPCIÓN DE LOS parámetros DEL PROCEDIMIENTO

A continuación, se describe cada uno de los parámetros que se usan en la definición del procedimiento READ_RAW.

r

La respuesta HTTP.

data

El cuerpo de respuesta HTTP en formato binario.

len

El número de bytes de los datos a leer. Si este parámetro es NULL, leerá tantas entradas como sean posibles hasta rellenar el buffer reservado para el parámetro DATA. La cantidad actual de datos devueltos puede ser menor que la especificada si no hay suficientes datos disponibles antes de que se alcance el final del cuerpo de la respuesta o bien se ha superado el tiempo de espera especificado en TRANSFER_TIMEOUT. El valor por defecto del parámetro es NULL.

Procedimiento READ_TEXT

Lee el cuerpo de la respuesta HTTP en formato texto y devuelve la salida al buffer. La excepción END_OF_BODY se alcanza cuando se llega al final del cuerpo de respuesta HTTP. La información que se obtiene del cuerpo de la respuesta es convertida automáticamente desde el juego de caracteres del propio cuerpo al juego de caracteres de la base de datos.

SINTAXIS

La sintaxis de este procedimiento es la siguiente:

```
UTL_HTTP.READ_TEXT(
r IN OUT NOCOPY resp,
data OUT NOCOPY VARCHAR2 CHARACTER SET ANY_CS,
len IN PLS_INTEGER DEFAULT NULL);
```

DESCRIPCIÓN DE LOS parámetros DEL PROCEDIMIENTO

A continuación, se describe cada uno de los parámetros que se usan en la definición del procedimiento READ_TEXT.

r

La respuesta HTTP.

data

El cuerpo de respuesta HTTP en formato texto.

len

El número de bytes de los datos a leer. Si este parámetro es NULL, lee tantas entradas como sean posibles hasta rellenar el buffer reservado para el parámetro DATA. La cantidad actual de datos devueltos puede ser menor que la especificada si no hay suficientes datos disponibles antes de que se alcance el final del cuerpo de la respuesta o bien se ha superado el tiempo de espera especificado en TRANSFER_TIMEOUT. El valor por defecto del parámetro es NULL.

Función REQUEST

Devuelve los primeros 2.000 bytes de datos recuperados de la URL proporcionada. Esta función puede ser usada directamente en consultas SQL. La URL puede contener el nombre de usuario y el password necesarios para autenticarse contra el servidor. El formato es:

scheme://[user[:password]@]host[:port]/[…]

Se puede especificar un usuario y password para el proxy indicándolo con el siguiente formato:

[http://][user[:password]@]host[:port]/

SINTAXIS

La sintaxis de esta función es la siguiente:

```
UTL_HTTP.REQUEST (
        url IN VARCHAR2,
        proxy IN VARCHAR2 DEFAULT NULL,
        wallet_path IN VARCHAR2 DEFAULT NULL,
        wallet_password IN VARCHAR2 DEFAULT NULL)
    RETURN VARCHAR2;
```

DESCRIPCIÓN DE LOS parámetros DE La función

A continuación, se describe cada uno de los parámetros que se usan en la definición de la función REQUEST.

url

La URL especificada (Uniform Resource Locator).

proxy

Parámetro opcional. Especifica un servidor proxy que se usa cuando se hace la petición http.

wallet_path

Parámetro opcional. Especifica una cartera (wallet) del lado del cliente.

wallet_password

Parámetro opcional. Especifica el password necesario para abrir la cartera (wallet).

RETURN VARCHAR2

El tipo devuelto es una cadena de 2.000 o menos bytes que contiene los primeros 2.000 bytes del resultado HTML devuelto desde la petición http para la URL especificada.

EXCEPCIONES QUE PUEDE DEVOLVER LA FUNCIÓN

Esta función puede llegar a devolver las siguientes excepciones como errores:

- INIT_FAILED
- REQUEST_FAILED

EJEMPLO

A continuación, se muestra un ejemplo de uso de esta función:

```
SQL> SELECT UTL_HTTP.REQUEST ('http://www.company.com/')
       FROM DUAL;

UTL_HTTP.REQUEST ('HTTP://WWW.COMPANY.COM/')
<html>
<head><title>My Company Home Page</title>
<!changed Jan. 16, 19
1 row selected.
```

Si se está utilizando un FIREWALL, se puede incluir el parámetro del proxy de la siguiente forma:

```
SQL> SELECT UTL_HTTP.REQUEST ('http://www.company.com/',)
       'www-proxy.es.company.com') FROM DUAL;
```

Función REQUEST_PIECES

Devuelve una tabla PL/SQL con elementos de 2.000 bytes de datos cada uno, recuperados de la URL proporcionada. Se puede especificar un usuario y el password para el proxy indicándolo con el siguiente formato:

[http://][user[:password]@]host[:port]/

SINTAXIS

La sintaxis de esta función es la siguiente:

```
UTL_HTTP.REQUEST_PIECES (
        url IN VARCHAR2,
        max_pieces IN NATURAL DEFAULT 32767,
        proxy IN VARCHAR2 DEFAULT NULL,
        wallet_path IN VARCHAR2 DEFAULT NULL,
        wallet_password IN VARCHAR2 DEFAULT NULL)
    RETURN html_pieces;
```

DESCRIPCIÓN DE LOS parámetros DE La función

A continuación, se describe cada uno de los parámetros que se usan en la definición de la función REQUEST_PIECES.

url

La URL especificada (Uniform Resource Locator).

max_pieces

Parámetro opcional. El máximo número de elementos de la tabla PL/SQL (cada 2.000 caracteres en longitud, excepto para el último elemento que puede ser más corto), que deberían devolverse. Si se indica, el valor deberá ser un número de tipo INTEGER positivo.

proxy

Parámetro opcional. Especifica un servidor proxy que se usa cuando se hace la petición http.

wallet_path

Parámetro opcional. Especifica una cartera (wallet) del lado del cliente.

wallet_password

Parámetro opcional. Especifica el password necesario para abrir la cartera (wallet).

RETURN html_pieces

Devuelve una tabla PL/SQL de tipo UTL_HTTP.HTML_PIECES. Cada elemento de la tabla es una cadena de un máximo de 2.000 caracteres. Los elementos devueltos son porciones de caracteres (de 2.000 caracteres) sucesivos de la información obtenida de la petición HTTP para la URL indicada.

EXCEPCIONES QUE PUEDE DEVOLVER LA FUNCIÓN

Esta función puede llegar a devolver las siguientes excepciones como errores:

- INIT_FAILED
- REQUEST_FAILED

Procedimiento SET_AUTHENTICATION

Establece la información de autentificación HTTP en la cabecera de la petición HTTP. El servidor web necesita esta información para autorizar la petición.

SINTAXIS

La sintaxis de este procedimiento es la siguiente:

```
UTL_HTTP.SET_AUTHENTICATION (
r IN OUT NOCOPY req,
username IN VARCHAR2,
password IN VARCHAR2,
scheme IN VARCHAR2 DEFAULT 'Basic',
for_proxy IN BOOLEAN DEFAULT FALSE);
```

DESCRIPCIÓN DE LOS parámetros DEL PROCEDIMIENTO

A continuación, se describe cada uno de los parámetros que se usan en la definición del procedimiento SET_AUTHENTICATION.

r

La petición HTTP.

username

El usuario para la autentificación del HTTP.

password

El password para la autentificación del HTTP.

scheme

El esquema de autentificación HTTP. El valor por defecto es 'Basic'.

for_proxy

Identifica si la información de autentificación http es para acceso a un servidor proxy en vez de a un servidor web. El valor por defecto es FALSE.

Procedimiento SET_AUTHENTICATION_FROM_WALLET

Establece la información de autentificación HTTP en la cabecera de la petición http necesaria para que la petición sea autorizada por el servidor web usando las credenciales de usuario y password almacenadas en Oracle wallet.

SINTAXIS

La sintaxis de este procedimiento es la siguiente:

```
UTL_HTTP.SET_AUTHENTICATION_FROM_WALLET(
r IN OUT NOCOPY req,
alias IN VARCHAR2,
scheme IN VARCHAR2 DEFAULT 'Basic',
for_proxy IN BOOLEAN DEFAULT FALSE);
```

DESCRIPCIÓN DE LOS parámetros DEL PROCEDIMIENTO

A continuación, se describe cada uno de los parámetros que se usan en la definición del procedimiento SET_AUTHENTICATION_FROM_WALLET.

r

La petición HTTP.

alias

Alias para identificar y devolver las credenciales del usuario y password almacenados en Oracle wallet.

scheme

El esquema de autentificación HTTP. El valor por defecto es 'Basic'.

for_proxy

Identifica si la información de autentificación http es para acceso a un servidor proxy en vez de a un servidor web. El valor por defecto es FALSE.

Procedimiento SET_BODY_CHARSET

Este procedimiento está sobrecargado, lo que supone que permite realizar diferentes funcionalidades dependiendo de la sintaxis que se utilice.

SINTAXIS

La primera sintaxis del procedimiento establece el juego de caracteres por defecto del cuerpo de todas las futuras peticiones HTTP.

```
UTL_HTTP.SET_BODY_CHARSET (
charset IN VARCHAR2 DEFAULT NULL);
```

La segunda sintaxis del procedimiento establece el juego de caracteres por defecto del cuerpo de una petición concreta HTTP.

```
UTL_HTTP.SET_BODY_CHARSET (
r IN OUT NOCOPY req,
charset IN VARCHAR2 DEFAULT NULL);
```

La última sintaxis del procedimiento establece el juego de caracteres por defecto del cuerpo de una respuesta concreta HTTP.

```
UTL_HTTP.SET_BODY_CHARSET (
r IN OUT NOCOPY resp,
charset IN VARCHAR2 DEFAULT NULL);
```

DESCRIPCIÓN DE LOS parámetros DEL PROCEDIMIENTO

A continuación, se describe cada uno de los parámetros que se usan en la definición del procedimiento SET_BODY_CHARSET.

r

La petición/respuesta HTTP.

charset

El juego de caracteres por defecto del cuerpo de la petición/respuesta. Puede establecerse en notación Oracle o IANA (Internet Assigned Numbers Authority). Si este parámetro tiene un valor NULL, se asumirá el juego de caracteres que esté establecido en la base de datos.

Procedimiento SET_COOKIE_SUPPORT

Este procedimiento está sobrecargado, lo que supone que permite realizar diferentes funcionalidades dependiendo de la sintaxis que se utilice.

SINTAXIS

La primera sintaxis del procedimiento habilita o deshabilita el soporte para las cookies HTTP en las peticiones. Use este procedimiento para cambiar la configuración de soporte de las cookies en una petición inherente desde la sesión que por defecto se haya configurado:

```
UTL_HTTP.SET_COOKIE_SUPPORT (
r IN OUT NOCOPY REQ,
enable IN BOOLEAN DEFAULT TRUE);
```

La segunda sintaxis del procedimiento establece si las peticiones HTTP futuras soportarán cookies, y el máximo número de cookies que serán mantenidas en la sesión actual del usuario establecida contra la base de datos.

```
UTL_HTTP.SET_COOKIE_SUPPORT (
enable IN BOOLEAN,
max_cookies IN PLS_INTEGER DEFAULT 300,
max_cookies_per_site IN PLS_INTEGER DEFAULT 20);
```

DESCRIPCIÓN DE LOS parámetros DEL PROCEDIMIENTO

A continuación, se describe cada uno de los parámetros que se usan en la definición del procedimiento SET_COOKIE_SUPPORT.

r

La petición HTTP.

enable

Para habilitar el soporte de las cookies HTTP se establecerá a valor TRUE. En caso de querer deshabilitarlas se indicará FALSE.

max_cookies

Establece el número total máximo de cookies que se mantendrán en la sesión

actual.

max_cookies_per_site

Establece el número máximo de cookies que se mantendrán en la sesión actual para cada sitio web.

Procedimiento SET_DETAILED_EXCP_SUPPORT

Establece que el paquete UTL_HTTP alcanzará excepciones detalladas. Por defecto, UTL_HTTP alcanza la excepción REQUEST_FAILED cuando falla una petición HTTP. Use GET_DETAILED_SQLCODE y GET_DETAILED_SQLERRM para obtener más información sobre el error (excepción) obtenido.

SINTAXIS

La sintaxis de este procedimiento es la siguiente:

```
UTL_HTTP.SET_DETAILED_EXCP_SUPPORT (
enable IN BOOLEAN DEFAULT FALSE);
```

DESCRIPCIÓN DE LOS parámetros DEL PROCEDIMIENTO

A continuación, se describe cada uno de los parámetros que se usan en la definición del procedimiento SET_DETAILED_EXCP_SUPPORT.

enable

Si el valor es TRUE, pedirá a UTL_HTTP que alcance una excepción detallada, directamente. En caso contrario establezca el valor a FALSE.

Procedimiento SET_FOLLOW_REDIRECT

Establece el número máximo de veces que el paquete UTL_HTTP seguirá redireccionando la instrucción HTTP en la respuesta a esta petición, o a futuras peticiones. Este procedimiento está sobrecargado.

SINTAXIS

La primera sintaxis del procedimiento se utiliza para establecer el número máximo de redireccionamientos.

```
UTL_HTTP.SET_FOLLOW_REDIRECT (
Max_redirects IN PLS_INTEGER DEFAULT 3);
```

La segunda sintaxis del procedimiento cambia el número máximo de redireccionamientos de una petición inherente a la sesión por defecto.

```
UTL_HTTP.SET_FOLLOW_REDIRECT (
r IN OUT NOCOPY req,
max_redirects IN PLS_INTEGER DEFAULT 3);
```

DESCRIPCIÓN DE LOS parámetros DEL PROCEDIMIENTO

A continuación, se describe cada uno de los parámetros que se usan en la definición del procedimiento SET_FOLLOW_REDIRECT.

r

La petición HTTP.

max_redirects

El número máximo de redireccionamientos. Establézcalo a valor cero para deshabilitar los redireccionamientos.

Procedimiento SET_HEADER

Establece la cabecera de una petición HTTP. La cabecera de la petición es enviada al servidor web en cuanto se establece.

SINTAXIS

La sintaxis de este procedimiento es la siguiente:

```
UTL_HTTP.SET_HEADER (
r IN OUT NOCOPY req,
name IN VARCHAR2,
value IN VARCHAR2);
```

DESCRIPCIÓN DE LOS parámetros DEL PROCEDIMIENTO

A continuación, se describe cada uno de los parámetros que se usan en la definición del procedimiento SET_FOLLOW_REDIRECT.

r

La petición HTTP.

name

El nombre de la cabecera de la petición HTTP.

value

El valor de la cabecera de la petición HTTP.

Procedimiento SET_PERSISTENT_CONN_SUPPORT

Habilita o deshabilita el soporte para conexiones persistentes HTTP 1.1 en las peticiones.

SINTAXIS

La sintaxis de este procedimiento es la siguiente:

```
UTL_HTTP.SET_PERSISTENT_CONN_SUPPORT (
r IN OUT NOCOPY req,
enable IN BOOLEAN DEFAULT FALSE);
```

DESCRIPCIÓN DE LOS parámetros DEL PROCEDIMIENTO

A continuación, se describe cada uno de los parámetros que se usan en la definición del procedimiento SET_PERSISTENT_CONN_SUPPORT.

r

La petición HTTP.

enable

Cuando tiene valor TRUE, mantiene las conexiones de red persistentes. En otro caso habrá que indicar valor FALSE.

Procedimiento SET_PROXY

Establece el proxy que se usará en las peticiones del protocolo HTTP o de cualquier otro, excluyendo del uso del proxy, aquellos hosts y dominios que se hayan indicado en el parámetro NO_PROXY_DOMAINS.

SINTAXIS

La sintaxis de este procedimiento es la siguiente:

```
UTL_HTTP.SET_PROXY (
proxy IN VARCHAR2,
no_proxy_domains IN VARCHAR2);
```

DESCRIPCIÓN DE LOS parámetros DEL PROCEDIMIENTO

A continuación, se describe cada uno de los parámetros que se usan en la definición del procedimiento SET_PROXY.

proxy

El proxy (host y número de puerto opcional) que se usa por el paquete UTL_HTTP.

no_proxy_domains

La lista de los hosts y dominios en los que no se usará ningún proxy para todas las peticiones que vayan dirigidas a los mismos.

Procedimiento SET_RESPONSE_ERROR_CHECK

Establece si se habilita o no la posibilidad de que GET_RESPONSE alcance una excepción cuando el servidor web devuelva un código de estado que indique un error. El código de estado se encontrará en el rango de valores de 4xx a 5xx. Por ejemplo, cuando la petición URL no encuentra el servidor web destino, se devuelve un código de estado 404 (documento no encontrado).

SINTAXIS

La sintaxis de este procedimiento es la siguiente:

```
UTL_HTTP.SET_RESPONSE_ERROR_CHECK (
enable IN BOOLEAN DEFAULT FALSE);
```

DESCRIPCIÓN DE LOS parámetros DEL PROCEDIMIENTO

A continuación, se describe cada uno de los parámetros que se usan en la definición del procedimiento SET_RESPONSE_ERROR_CHECK.

enable

TRUE para habilitar el servicio de respuesta de errores; en otro caso se indicará FALSE.

Procedimiento SET_TRANSFER_TIMEOUT

Establece el tiempo límite de espera por defecto antes de terminar todas las peticiones futuras que el paquete UTL_HTTP debería llevar a cabo mientras lee las respuestas HTTP del servidor web o del servidor proxy. El valor de tiempo límite de espera puede ser usado para evitar que los programas PL/SQL se queden bloqueados porque el servidor web esté ocupado o tenga un tráfico de red congestionado mientras recupera las páginas web de los servidores web.

SINTAXIS

La sintaxis de este procedimiento es la siguiente:

```
UTL_HTTP.SET_TRANSFER_TIMEOUT (
timeout IN PLS_INTEGER DEFAULT 60);
```

DESCRIPCIÓN DE LOS parámetros DEL PROCEDIMIENTO

A continuación, se describe cada uno de los parámetros que se usan en la definición del procedimiento SET_TRANSFER_TIMEOUT.

timeout

El tiempo límite de espera en las transferencias efectuadas en la red, medido en segundos.

Procedimiento SET_WALLET

Establece la cartera (wallet) Oracle utilizada para todas las peticiones HTTP sobre SSL (Secured Socket Layer), denominadas HTTPS.

SINTAXIS

La sintaxis de este procedimiento es la siguiente:

```
UTL_HTTP.SET_WALLET (
path IN VARCHAR2,
password IN VARCHAR2 DEFAULT NULL);
```

DESCRIPCIÓN DE LOS parámetros DEL PROCEDIMIENTO

A continuación, se describe cada uno de los parámetros que se usan en la definición del procedimiento SET_WALLET.

path

La ruta que contiene la cartera (wallet) Oracle. El formato es file:directory-path.

El formato de WALLET_PATH en un PC es por ejemplo, file:c:\WINNT\Profiles*username*\WALLETS, y en Unix es por ejemplo, file:/home/*username*/wallets.

Cuando el paquete UTL_HTTP se ejecuta en un servidor de base de datos Oracle, la cartera (wallet) es accesible desde el servidor de base de datos. Por tanto, la ruta de la cartera (wallet) debe ser accesible desde el servidor de base de datos.

password

El password necesario para abrir la cartera (wallet). Si la cartera (wallet) tiene habilitado el sistema auto-login, se deberá omitir este parámetro y pasarle un valor NULL.

Procedimiento WRITE_LINE

Escribe una línea de texto en el cuerpo de la petición HTTP y finaliza la línea con un carácter de nueva línea (CRLF como se definió en UTL_TCP). Los datos del texto son automáticamente convertidos del juego de caracteres de la base de datos al juego de caracteres del cuerpo de la petición.

SINTAXIS

La sintaxis de este procedimiento es la siguiente:

```
UTL_HTTP.WRITE_LINE (
r IN OUT NOCOPY req,
data IN VARCHAR2 CHARACTER SET ANY_CS);
```

DESCRIPCIÓN DE LOS parámetros DEL PROCEDIMIENTO

A continuación, se describe cada uno de los parámetros que se usan en la definición del procedimiento WRITE_LINE.

r

La petición HTTP.

data

La línea de texto para enviar en el cuerpo de petición HTTP.

Procedimiento WRITE_RAW

Escribe datos binarios en el cuerpo de petición HTTP.

SINTAXIS

La sintaxis de este procedimiento es la siguiente:

```
UTL_HTTP.WRITE_RAW (
r IN OUT NOCOPY req,
data IN RAW);
```

DESCRIPCIÓN DE LOS parámetros DEL PROCEDIMIENTO

A continuación, se describe cada uno de los parámetros que se usan en la definición del procedimiento WRITE_RAW.

r

La petición HTTP.

data

Los datos binarios a enviar en el cuerpo de petición HTTP.

Procedimiento WRITE_TEXT

Escribe datos de texto en el cuerpo de petición HTTP. El texto es convertido automáticamente del juego de caracteres de la base de datos al juego de caracteres del cuerpo de la petición.

SINTAXIS

La sintaxis de este procedimiento es la siguiente:

```
UTL_HTTP.WRITE_TEXT (
r IN OUT NOCOPY req,
data IN VARCHAR2 CHARACTER SET ANY_CS);
```

DESCRIPCIÓN DE LOS parámetros DEL PROCEDIMIENTO

A continuación, se describe cada uno de los parámetros que se usan en la definición del procedimiento WRITE_TEXT.

r

La petición HTTP.

data

Los datos de texto a enviar en el cuerpo de petición HTTP.

Capítulo 10. DISPARADORES O TRIGGERS

INTRODUCCIÓN

El cuarto tipo de bloque PL/SQL nominado que se va a estudiar en este libro es el disparador o trigger.

Los disparadores se asemejan a los subprogramas debido a que son bloques PL/SQL nominados con secciones declarativa, ejecutable y de control de errores.

Al igual que los paquetes, los disparadores deben almacenarse en la base de datos y no pueden ser locales a un bloque.

La diferencia fundamental entre los disparadores y los subprogramas vistos hasta el momento es que mientras que un subprograma se ejecuta explícitamente, cuando es invocado desde una sección ejecutable de un código por su nombre, un disparador o trigger no puede ser invocado explícitamente.

Un disparador se ejecuta de manera implícita cada vez que tiene lugar el suceso del disparo para el que fue creado, mediante la asociación a una tabla.

Un disparador no admite argumentos y no devuelve valores.

El acto que provoca la ejecución del disparador (suceso del disparo) es una operación DML que provoque la alteración de una tabla (INSERT, UPDATE o DELETE).

UTILIDAD DE LOS TRIGGERS

Los disparadores pueden emplearse para muchas cosas diferentes, entre las que se incluyen:

- Mantenimiento de restricciones de integridad complejas, que no sean posibles con las restricciones declarativas definidas al crear la tabla.
  Por ejemplo, al crear una tabla y definir una columna de clave ajena (FOREIGN KEY), el propio SQL admite la opción DELETE CASCADE, lo que supone que se eliminen en la tabla dependiente todas las filas que contengan el valor que se borre en la tabla padre, para esa columna.
  Pero el lenguaje SQL de Oracle no soporta UPDATE CASCADE, que pertenece al SQL/92 estándar. Esta opción permite que cuando se cambie el valor de la tabla padre, se actualicen todos los registros en la tabla hija, cuya columna se vea afectada. Para poder implementar esto en Oracle hay que crear un trigger.
- Un trigger permite la auditoría de la información contenida en una tabla, registrando los cambios realizados y la identidad del que los llevó a cabo.
- Permiten el aviso automático a otros programas, de que hay que llevar a cabo una determinada acción cuando se realiza un cambio en una tabla.

Sintaxis

```
CREATE [OR REPLACE] TRIGGER nombre_disparador
{BEFORE | AFTER} suceso_disparo ON tabla
[FOR EACH ROW [WHEN condicion_disparo]]

cuerpo_disparador
```

INTEGRIDAD REFERENCIAL

El concepto de integridad referencial se aplica dentro de la creación de una tabla mediante la cláusula FOREIGN KEY. Es decir, cuando manejamos claves ajenas.

Este concepto nos va a permitir mantener una consistencia entre los datos relacionados en la tabla padre e hijo, de manera que las columnas que forman la clave ajena en la tabla hija no podrán tener valores distintos a sus homólogas referenciadas en la tabla padre.

En el lenguaje estándar SQL se aprobaron en su momento las posibilidades de integridad referencial que se podían dar entre columnas de una tabla padre e hijo, pero no todas estas opciones están implementadas en los SGBD de Oracle. A continuación, vemos un cuadro con las opciones que se aprobaron en el lenguaje SQL y las que contienen el SGBD de Oracle.

Enunciadas en SQL estándar	Implementadas en Oracle
ON DELETE RESTRICT	SÍ
ON DELETE CASCADE	SÍ
ON UPDATE RESTRICT	NO
ON UPDATE CASCADE	NO implementada directamente

Vamos a ver a continuación el significado de cada una, y cómo afectan a los datos. Asimismo, podremos enunciar cómo se indican mediante instrucciones del lenguaje SQL de Oracle.

On delete restrict

Esta cláusula impide que se borren datos en la tabla padre, siempre y cuando existan en la tabla hija datos en las columnas referenciadas, que coincidan con dicho valor.

ESTADOS

Codigo	EstadoCivil
S	ANTOLIN
C	PEPITO
V	JUANITO

PERSONA

Nif	Nombre	CodEstado
1241231	ANTOLIN	C
4553232	JUANITO	S
532342	PEPITO	V
3423423	CARLOS	C

Tomando el ejemplo anterior vemos que existe una tabla padre que es Estados y una tabla hija Persona donde existe una clave ajena que es Persona.CodEstado sobre la columna Estados.Codigo. Para crear dicha clave ajena, mediante código SQL de Oracle, utilizaríamos un constraint al final de la definición de la tabla Persona de la siguiente manera:

```
CONSTRAINT FK_PERSONA_ESTADOS FOREIGN KEY (codestado)
REFERENCES estados(codigo)
```

Si ahora quisiéramos borrar la línea de la tabla Estados que contiene en la columna codigo el valor de 'C' (casado), utilizaríamos la siguiente instrucción:

```
DELETE ESTADOS WHERE CODIGO = 'C';
```

El sistema nos devolverá un error indicándonos que no es posible llevar a cabo la operación, porque existe información de una tabla hija que referencia el valor de esta columna. En concreto, en nuestro ejemplo hay 2 columnas con el contenido 'C' (casado) en la columna CodEstado, que son la primera y la última.

On update restrict

Esta cláusula impide que se modifiquen datos en la tabla padre, siempre y cuando existan en la tabla hija datos en las columnas referenciadas, que coincidan con dicho valor.

La indicación de este tipo de restricción mediante sentencia SQL de Oracle es la misma que en la opción anterior:

```
CONSTRAINT FK_PERSONA_ESTADOS FOREIGN KEY (codestado)
REFERENCES estados(codigo)
```

Si ahora quisiéramos actualizar la línea de la tabla Estados que contiene en la columna codigo el valor de 'V' (viudo), utilizaríamos la siguiente instrucción:

```
UPDATE ESTADOS SET CODIGO = 'X' WHERE CODIGO = 'V';
```

El sistema nos devolverá un error indicándonos que no es posible llevar a cabo la operación, porque existe información de una tabla hija que referencia el valor de esta columna. En concreto, en nuestro ejemplo la 3ª fila tiene el valor 'V' en la columna CodEstado.

On delete cascade

Esta cláusula permite que se borren valores en la tabla padre aunque existan valores iguales al borrado en la hija. Pero para mantener la integridad referencial en los datos, también borra todas las filas de la tabla hija que contenga la clave de la fila borrada en el padre.

La indicación de este tipo de restricción mediante sentencia SQL de Oracle es la siguiente:

```
CONSTRAINT FK_PERSONA_ESTADOS FOREIGN KEY (codestado)
REFERENCES estados(codigo)
ON DELETE CASCADE
```

Si ahora quisiéramos borrar la línea de la tabla Estados que contiene en la columna codigo el valor de 'C' (casado), utilizaríamos la siguiente instrucción:

```
DELETE ESTADOS WHERE CODIGO = 'C';
```

El resultado que obtendremos en cuanto a las tablas de nuestro ejemplo sería el siguiente:

ESTADOS

Codigo	EstadoCivil
C	PEPITO
V	JUANITO

PERSONA

Nif	Nombre	CodEstado
4553232	JUANITO	S
532342	PEPITO	V

On update cascade

Esta cláusula permite que se actualicen valores en la tabla padre aunque existan valores iguales al borrado en la hija. Para mantener la integridad referencial en los datos, también se actualizan todas las filas de la tabla hija que contengan el valor actualizado en el padre.

Esta opción no tiene implementación en Oracle mediante una instrucción directa asociada a un CONSTRAINT, pero se puede llegar a realizar una implementación que posibilite la operación de ON UPDATE CASCADE, creando un TRIGGER asociado a la tabla padre.

ESPACIO DE nombres del disparador

Los procedimientos, funciones, paquetes y tablas entre otros, comparten el mismo espacio de nombres, por lo que solo puede haber un nombre distinto que se asigne a cualquiera de estos objetos.

Pero los triggers tienen su propio espacio de nombre, así pues podrán tener el mismo nombre que cualquier de los anteriores.

Ejemplo

```
CREATE TABLE PRUEBAS
(……

CREATE PROCEDURE PRUEBAS IS
...
BEGIN
        ...
END;
```

Este ejemplo provocaría un error al crear el procedimiento PRUEBAS porque ya existe un objeto denominado PRUEBAS (la tabla) que comparte el mismo espacio de nombres.

```
CREATE TABLE PRUEBAS
( ...

CREATE TRIGGER PRUEBAS
...
BEGIN
        ...
END;
```

En cambio este segundo ejemplo no provocaría ningún error porque al tener espacios de nombres diferentes la tabla y el trigger, ambos objetos pueden denominarse de la misma forma.

MOMENTO DEL DISPARO

Un trigger se diseña para ejecutarse antes (BEFORE) o después (AFTER) de que se produzca una sentencia DML sobre la tabla a la que se ha asociado el trigger.

SUCESO DEL DISPARO

Los sucesos que provocan el disparo del trigger son sentencias DML que producen alteración en los datos contenidos en la tabla sobre la que se definió el trigger, y en concreto pueden ser 3:

- DELETE: cuando se borre una fila.
- INSERT: cuando se inserte una fila.
- UPDATE: cuando se modifique una fila.

nivel de disparo

Nos define el número de veces que se va a ejecutar el trigger una vez que se produzca el suceso de disparo.

En concreto un disparador tiene 2 posibilidades de nivel de disparo:

- Una única vez por sentencia (es el valor por defecto).
- Una vez por fila de la tabla asociada que se vea modificada (FOR EACH ROW).

CONDICIÓN DE DISPARO

Un trigger por fila (FOR EACH ROW) aparte de ejecutarse cuando se produce el suceso de disparo, también se le pueden condicionar por otra circunstancia que se programa en el propio trigger, y que se conoce como condición de disparo.

Ejemplo

```
CREATE OR REPLACE TRIGGER comprueba_salario
BEFORE INSERT OR UPDATE salario, empleo ON emp
FOR EACH ROW WHEN (new.empleo <> 'PRESIDENTE')

DECLARE
Minsalario NUMBER;
BEGIN
        ...
END;

INSERT INTO emp (empleo) VALUES ('PRESIDENTE');
```

Con esta instrucción el trigger COMPRUEBA_SALARIO no se dispararía, dado que la instrucción de inserción no cumple la condición de disparo necesaria (<> 'PRESIDENTE').

```
INSERT INTO emp (empleo) VALUES ('COMERCIAL');
```

Con esta instrucción sí que se dispararía el trigger COMPRUEBA_SALARIO al cumplirse la condición de disparo necesaria. Al ser un trigger del nivel FOR EACH ROW, se ejecutaría 1 sola vez, porque solo se inserta una fila, en caso de que hubiese más inserciones que cumpliesen la condición de disparo, se ejecutaría tantas veces como filas se insertasen.

SENTENCIAS DE BORRADO Y ALTERACIÓN DE TRIGGERS

Para borrar o alterar un trigger hay que ejecutar las sentencias DDL que se describen a continuación.

Borrado de un trigger

La sintaxis de borrado de un trigger es la siguiente:

```
DROP TRIGGER nombre_trigger;
```

Alteración de un trigger

Un trigger únicamente permite alterar mediante una instrucción DDL el estado en el que se encuentra, es decir, habilitado o deshabilitado. Para cambiar cualquier otro detalle del trigger: nivel de disparo, condición de disparo, tabla asociada, bloque de ejecución, etc., hay que volver a crear el trigger con la instrucción de creación vista al comienzo de este capítulo.

La sintaxis que permite deshabilitar o habilitar el funcionamiento de un trigger con el fin de que se pueda o no disparar cuando se produzca el evento del disparo, es la siguiente:

```
ALTER TRIGGER nombre_trigger {DISABLE | ENABLE}
```

La sintaxis que permite deshabilitar o habilitar todos los triggers asociados a una tabla son los siguientes:

```
ALTER TABLE nombre_trigger {DISABLE | ENABLE} ALL TRIGGERS;
```

USO DE LOS PREDICADOS :OLD y :NEW

Cuando se produce el suceso que hace disparar el trigger entran en funcionamiento los conceptos de acceso a la fila que está siendo actualmente procesada mediante dos pseudo-registros que se denominan **:old** y **:new**.

Estos pseudo-registros solo se pueden utilizar con disparadores a nivel de fila y su significado es el siguiente:

- :OLD Maneja los datos originales de la fila en la tabla antes de ser cambiados.
- :NEW Maneja los nuevos datos que van a sustituir a los que hay en :old para esa fila.

Orden de disparo	:OLD	:NEW
INSERT	NULL en todas las columnas.	Los nuevos valores a insertar en las columnas
UPDATE	Los valores previos de las columnas antes de actualizar.	Los nuevos valores que se actualizan en las columnas.
DELETE	Los valores previos de las columnas antes de borrarlos.	NULL en todas las columnas.

USO DE LOS PREDICADOS BOOLEANOS

Para poder evaluar el tipo de suceso que ha provocado la ejecución de un trigger disponemos de una serie de predicados booleanos que interrogan al trigger para conocer el suceso que le ha acontecido.

Estos predicados son los que se indican a continuación con su significado:

- INSERTING: Verdadero si el trigger se ejecuta como consecuencia de un INSERT.
- UPDATING: Verdadero si el trigger se ejecuta como consecuencia de un UPDATE.
- DELETING: Verdadero si el trigger se ejecuta como consecuencia de un DELETE.

Ejemplo

```
CREATE OR REPLACE TRIGGER auditoria
BEFORE INSERT OR DELETE OR UPDATE ON emp
FOR EACH ROW
BEGIN
IF INSERTING THEN
        INSERT INTO TEMPORAL VALURES
        ('Se ha producido una inserción en la tabla');
    ELSIF DELETING THEN
        INSERT INTO TEMPORAL VALURES
        ('Se ha producido un borrado en la tabla');
    ELSIF UPDATING THEN
        INSERT INTO TEMPORAL VALURES
        ('Se ha producido una inserción en la tabla');
    END IF;
END;
```

TABLAS MUTANTES

Se denomina tabla mutante a aquella que se está modificando actualmente por una orden DML.

Las tablas que pueden necesitar ser actualizadas como resultado de restricciones de integridad referencial DELETE_CASCADE definidas en una cláusula FOREING KEY también se consideran tablas mutantes.

Tablas de restricción

Una tabla de restricción es aquella que es necesario leer para resolver una restricción de integridad referencial.

Por ejemplo:

```
    CREATE TABLE estu_registrados ( -- tabla mutante
    Id NUMBER(5) NOT NULL,
Dpto CHAR(3) NOT NULL,
Curso CHAR(3) NOT NULL,
Grado CHAR(1) NOT NULL,
CONSTRAINT fk_estudiante_id FOREIGN KEY (id)
REFERENCES estudiante(id), -- tabla de restricción
    CONSTRAINT fk_dpto_curso FOREIGN KEY (dpto, curso)
REFERENCES clase(dpto, curso)-- tabla de restricción
    CONSTRAINT fk_grado FOREIGN KEY (grado)
REFERENCES grados (grado) -- tabla mutante
DELETE_CASCADE
    CONSTRAINT pk_estu_registrados
    PRIMARY KEY (id, dpto, curso)
    );
```

Restricciones en triggers con nivel de fila

No se puede leer o modificar ninguna tabla mutante dentro del código ejecutable del trigger asociada a la tabla mutante.

No se puede leer o modificar columnas de clave primaria únicas, o externas de una tabla de restricción en el cuerpo del trigger asociado a la tabla mutante.

Por ejemplo, tomemos en consideración el siguiente caso:

Tabla ESTUDIANTE (Clave: ID)
Tabla CLASE (Clave: ID, DPTO, CURSO)

```
CREATE OR REPLACE TRIGGER cascada
BEFORE INSERT ON estu_registrados
FOR EACH ROW
DECLARE
V_creditos clase.num_creditos%TYPE;
BEGIN
/* La siguiente instrucción es correcta dado que no
lee la clave primaria de la tabla de restricción
clase. */

SELECT num_creditos INTO v_creditos
FROM clase
WHERE dpto = :new.dpto
AND curso = :new.curso;

/* Las siguientes instrucciones son también
correctas, dado que no modifican elementos de clave
primaria de las tablas de restricción */

UPDATE estudiantes
SET creditos_actual = creditos_actual + v_creditos
WHERE id = :new.id;
```

```
UPDATE clase
SET num_estudiantes = num_estudiantes + 1
WHERE dpto = :new.dpto
AND curso = :new.curso;
```

Otro ejemplo que podemos considerar es el siguiente:

```
CREATE OR REPLACE TRIGGER incorrecto
BEFORE INSERT OR UPDATE OF dpto
ON estu_registrados FOR EACH ROW
DECLARE
V_maxestu NUMBER(3);
BEGIN
/* La siguiente instrucción es incorrecta, dada que
se intenta leer de una tabla mutante */

SELECT count(*)
INTO v_maxestu
FROM estu_registrados
WHERE dpto = :new.dpto;

/* La siguiente instrucción es incorrecta dado que
actualiza un campo de la clave de la tabla de
restricción. */

UPDATE clase
SET dpto = :new.dpto
WHERE id = :old.id;

END;

SQL> UPDATE estu_registrados
SET grado = 'A'
WHERE dpto = 'HISTORIA';
```

A continuación nos aparecería el siguiente error:

```
ERROR at line 1:
ORA-04091: table ESTU_REGISTRADOS is muting, trigger/fuction may
```

not see it.

SUPUESTO PRÁCTICO 10: ***Resolución en el Anexo I de este manual.***

Comprobar el funcionamiento de un TRIGGER ejecutando un código dado (omitiendo las instrucciones correspondientes a los comentarios):

```
SQL> /* Creamos una tabla denominada ESTUDIANTE */
SQL> CREATE TABLE ESTUDIANTE
2 (NOMBRE VARCHAR2(10));

Tabla creada.

SQL> /* Creamos una tabla denominada SUCESOS */
SQL> CREATE TABLE SUCESOS
2 (NOMBRE VARCHAR2(50),
3 EN_TABLA VARCHAR2(10),
4 DIA DATE);

Tabla creada.

SQL> /* Creamos un trigger asociado a la tabla estudiante
2 que se dispara cuando se produce una operación de INSERT
3 sobre dicha tabla y que tiene como misión insertar una fila */
SQL> CREATE OR REPLACE TRIGGER GRABA_SUCESO
2 BEFORE INSERT ON ESTUDIANTE
3 BEGIN
4 INSERT INTO SUCESOS VALUES ('TRIGGER','ESTUDIANTE',SYSDATE);
5 END GRABA_SUCESO;
6 /

Disparador creado.

SQL> /* Seleccionamos todas las filas de la tabla ESTUDIANTE */
SQL> SELECT * FROM ESTUDIANTE;

ninguna fila seleccionada

SQL> /* Seleccionamos todas las filas de la tabla SUCESOS */
SQL> SELECT * FROM SUCESOS;

ninguna fila seleccionada

SQL> /* Realizamos un select sobre la tabla del sistema USER_OBJECTS
2 que nos devuelve el nombre, tipo y estado del trigger cuyo nombre
3 empieza por GRA para ver en qué estado se encuentran */
SQL> SELECT OBJECT_NAME, OBJECT_TYPE, STATUS FROM USER_OBJECTS
2 WHERE OBJECT_NAME LIKE 'GRA%';

OBJECT_NAME
-------------------------------------------------------------OBJECT_TYPE STATUS
------------- -------
GRABA_SUCESO
TRIGGER VALID


SQL> /* Insertamos una fila en la tabla ESTUDIANTE lo cual provoca que
2 se dispare nuestro trigger GRABA_SUCESO */
SQL> INSERT INTO ESTUDIANTE VALUES ('PEPITO');

1 fila creada.

SQL> /* Seleccionamos todas las filas de la tabla ESTUDIANTE para comprobar
2 que se ha ejecutado el INSERT anterior. */
```

```
SQL> SELECT * FROM ESTUDIANTE;

NOMBRE
----------
PEPITO

SQL> /* Seleccionamos todas las filas de la tabla SUCESOS para comprobar
2 que se ha disparado nuestro trigger GRABA_SUCESO */
SQL> SELECT * FROM SUCESOS;

NOMBRE EN_TABLA
DIA
-------------------------------------------------- ----------
TRIGGER ESTUDIANTE
22/11/00

SQL> /* Nos arrepentimos y deshacemos el INSERT sobre la tabla ESTUDIANTES
2 ¿qué ocurre entonces con la tabla SUCESOS y su información generada
3 por el trigger GRABA_SUCESO? */
SQL> ROLLBACK;

Rollback terminado.

SQL> /* Seleccionamos todas las filas de la tabla ESTUDIANTE y vemos que tras
2 el ROLLBACK volvemos al estado inicial que era una tabla sin filas */
SQL> SELECT * FROM ESTUDIANTE;

ninguna fila seleccionada

SQL> /* El ROLLBACK deshace cualquier operación de actualización realizada o sea
2 que también deja la tabla SUCESOS como estaba al principio, sin filas
3 para ello ejecutamos la select que devuelve todos los datos y vemos que
4 está vacía */
SQL> SELECT * FROM SUCESOS;

ninguna fila seleccionada

SQL> /* Volvemos a ejecutar la operación de inserción sobre tabla ESTUDIANTE */
SQL> INSERT INTO ESTUDIANTE VALUES ('PEPITO');

1 fila creada.

SQL> /* Confirmamos la inserción y la hacemos definitiva */
SQL> COMMIT;

Validación terminada.

SQL> /* Consultamos las filas de la tabla ESTUDIANTE */
SQL> SELECT * FROM ESTUDIANTE;

NOMBRE
----------
PEPITO

SQL> /* Consultamos las filas de la tabla SUCESOS y vemos que se ha ejecutado
2 el trigger */
SQL> SELECT * FROM SUCESOS;

NOMBRE EN_TABLA
DIA
-------------------------------------------------- ----------
TRIGGER ESTUDIANTE
22/11/00

SQL> /* Borramos la tabla ESTUDIANTE */
SQL> DROP TABLE ESTUDIANTE;

Tabla borrada.
```

```
SQL> /* Consultamos la tabla SUCESOS para ver si su estado ha variado al borrar
2 la tabla ESTUDIANTE y vemos que no ha variado */
SQL> SELECT * FROM SUCESOS;

NOMBRE EN_TABLA
DIA
-------------------------------------------------- ----------
TRIGGER ESTUDIANTE
22/11/00

SQL> /* Consultamos la tabla del sistema USER_OBJECTS para ver que ha ocurrido
2 con el trigger al borrar la tabla sobre la que estaba asociado el mismo. El
3 resultado es que se ha borrado también porque un trigger depende de una tabla
4 y si la misma se borra, se eliminan automáticamente todos los elementos asociados
5 a ella: Indices, Vistas y Triggers */
SQL> SELECT * FROM USER_OBJECTS WHERE OBJECT_NAME LIKE 'GR%';

ninguna fila seleccionada

SQL> /* Borramos la tabla sucesos */
SQL> drop table sucesos;

Tabla borrada.

SQL> /* Fin del ejemplo de utilización de TRIGGERS */
.
```

SUPUESTO PRÁCTICO 11: ***Resolución en el Anexo I de este manual.***

Comprobar el funcionamiento de un TRIGGER ejecutando un código dado (omitiendo las instrucciones correspondientes a los comentarios):

```
SQL> /* Creamos una tabla estudiantes */
SQL> CREATE TABLE estudiantes
2 (ID NUMBER PRIMARY KEY,
3 NOMBRE VARCHAR2(25),
4 APELLIDOS VARCHAR2(30));

Table created.

SQL> /* Creamos una secuencia */
SQL> CREATE SEQUENCE contador
2 START WITH 1
3 INCREMENT BY 1
4 NOCACHE;

Sequence created.

SQL> CREATE OR REPLACE TRIGGER GeneraID
2 BEFORE INSERT OR UPDATE ON estudiantes
3 FOR EACH ROW
4 BEGIN
5 SELECT CONTADOR.NEXTVAL INTO :NEW.ID FROM DUAL;
6 END;
7 /

Trigger created.

SQL> /* Hacemos una primera inserción en la tabla estudiantes */
SQL> INSERT INTO estudiantes VALUES(10,'Juan','Gabriel Sanchez');

1 row created.


SQL> /* Vemos que se ha insertado en realidad en tabla ESTUDIANTES */
```

```
SQL> SELECT * FROM estudiantes;

ID NOMBRE APELLIDOS
--------- ------------------------- ----------------------
1 Juan Gabriel Sanchez

SQL> /* Realizamos otro insert en tabla ESTUDIANTES */
SQL> INSERT INTO estudiantes (NOMBRE, APELLIDOS) VALUES ('Pepe','Domingo Castro');

1 row created.

SQL> /* Vemos que se ha insertado */
SQL> SELECT * FROM estudiantes;

ID NOMBRE APELLIDOS
--------- ------------------------- ----------------------
1 Juan Gabriel Sanchez
2 Pepe Domingo Castro


SQL> /* Como se puede ver por las 2 inserciones el trigger lo que hace
2> es dar un valor para el campo ID (el valor del siguiente número de la secuencia)
3>independientemente de que se inserte o no un valor para dicho campo */
SQL> /* Fin de la ejecución del trigger */
.
```

SUPUESTO PRÁCTICO 12: ***Resolución en el Anexo I de este manual.***

Diseñar un trigger que haga las funciones de la restricción de integridad referencial ON UPDATE CASCADE, que no implemente directamente el lenguaje SQL de Oracle. Se creará este trigger sobre la tabla *departamento*, de manera que cuando se actualice algún código de departamento, se actualicen todas las filas de la tabla empleado que contengan dicho código.

- Además, por cada operación de manipulación que se lleve a cabo en la tabla *departamento* y que afecte a la columna *código de departamento* se producirá una inserción en la tabla HISTO_CAMBIOS_DEPARTAMENTO cuya estructura se define más abajo y que habrá de crearse previamente. Los datos a introducir en dicha tabla por cada operación que se produzca, son los siguientes:
  - En caso de producirse una inserción en la tabla *departamento*, habrá de introducirse un registro en la tabla HISTO_CAMBIOS_DEPARTAMENTO con los siguientes valores:

    FECHAYHORA = *la fecha y hora actual*
    TEXTO= 'Se ha insertado una fila con datos: '||*nuevo código de departamento introducido*||'-'|| *nuevo nombre del departamento*

  - En caso de producirse un borrado en la tabla *departamento*, habrá de comprobarse si existen datos de empleados que pertenezcan a ese departamento, en cuyo caso se introducirá un registro en la tabla HISTO_CAMBIOS_DEPARTAMENTO con los siguientes valores:

    FECHAYHORA = *la fecha y hora actual*
    TEXTO= 'Se ha intentado borrar el departamento: '|| *código de departamento que se intenta borrar*

    En caso de que no existan datos de empleados para ese departamento, se introducirá un registro en la tabla HISTO_CAMBIOS_DEPARTAMENTO con los siguientes valores:

    FECHAYHORA = *la fecha y hora actual*
    TEXTO= 'Se ha borrado el departamento: '|| *código de departamento que se intenta borrar*

  - En caso de producirse una actualización en la tabla *departamento*, habrá de comprobarse si existen datos de empleados que pertenezcan a ese departamento, en cuyo caso primero se actualizará el departamento a dichos empleados.
    Tanto si existen datos de empleados pertenecientes a ese departamento como si no, se introducirá un registro en la tabla HISTO_CAMBIOS_DEPARTAMENTO con los siguientes valores:

FECHAYHORA = *la fecha y hora actual*
TEXTO= 'Se ha actualizado el departamento: '|| *código del departamento antes de que se actualizase*||' al valor '|| *nuevo código del departamento.*

HISTO_CAMBIOS_DEPARTAMENTO
DIAYHORA DATE,
TEXTO VARCHAR2(500)
PRIMARY KEY(diayhora)

Capítulo 11. TRATAMIENTO DE ERRORES

INTRODUCCIÓN

PL/SQL implementa mecanismos para el control de errores mediante 2 métodos:

- Excepciones.
- Gestores de excepciones.

Las excepciones se asocian a:

- Errores de Oracle.
- Errores definidos por el usuario.

Los errores tratados en este capítulo son los que se producen en tiempo de ejecución.

Los errores de compilación se muestran durante la fase de compilación pero no se pueden tratar.

Cuando se produce un error, se genera lo que se llama una "EXCEPCIÓN".

Cuando se lanza un error de tipo EXCEPTION se pasa el control al gestor de excepciones. Es la sección de un bloque que se denomina EXCEPTION, en la cual se ejecutarán las operaciones necesarias para controlar ese error.

Sintaxis:

```
BEGIN
...
EXCEPTION
        /* Gestión de excepciones */
END;
```

DECLARACIÓN DE EXCEPCIONES

Toda excepción ha de seguir un proceso de declaración, generación y tratamiento:

- La declaración se realiza dentro de la sección DECLARE de un bloque.
- La generación se produce en la sección BEGIN…END.
- El tratamiento se realiza en la sección EXCEPTION.

excepciones definidas por el usuario

Son errores que se definen dentro del programa en la parte declarativa DECLARE de un bloque.

No tienen por quÉ ser errores de ORACLE.

Sintaxis:

<nombre_excepcion> EXCEPTION;

Ejemplo:

```
DECLARE
Demasiados_estudiantes EXCEPTION;
```

Excepciones predefinidas

En este apartado se definen las excepciones que se han definido internamente junto con el lenguaje PL/SQL y que tienen asignado un nombre concreto con el que se pueden referenciar.

Código Error	Excepción	Descripción
ORA-0001	DUP_VAL_ON_INDEX	Violación de una restricción de unicidad.
ORA-0051	TIMEOUT_ON_RESOURCE	Se produjo un fin de intervalo mientras se esperaba un cierto recurso.
ORA-1001	INVALID_CURSOR	Operación ilegal con un cursor.
ORA-1012	NOT_LOGGED_ON	No existe conexión con Oracle.
ORA-1017	LOGIN_DENIED	Nombre de usuario o contraseña inválidos.
ORA-1403	NO_DATA_FOUND	No se ha encontrado ningún dato.
ORA-1422	TOO_MANY_ROWS	Hay más de una fila que corresponde a una orden SELECT…INTO.
ORA-1476	ZERO_DIVIDE	División por cero.
ORA-1722	INVALID_NUMBER	Falló la conversión a un número; por ejemplo, 'IA' no es válido.
ORA-6500	STORAGE_ERROR	Error interno PL/SQL, generado cuando PL/SQL se queda sin memoria.
ORA-6501	PROGRAM_ERROR	Error interno PL/SQL.
ORA-6502	VALUE_ERROR	Error de truncamiento, aritmético o de conversión.
ORA-6504	ROWTYPE_MISMATCH	Una variable de cursos del host y una variable de cursor PL/SQL tienen tipos de fila incompatibles.
ORA-6511	CURSOR_ALREADY_OPEN	Se ha intentado abrir un cursor que ya estaba abierto.
ORA-6530	ACCESS_INTO_NULL	Se ha intentado asignar valores a los atributos de un objeto que tiene el valor NULL.
ORA-6531	COLLECTION_IS_NULL	Se ha intentado aplicar métodos de colección distintos de EXISTS a una tabla o varray PL/SQL con valor NULL.
ORA-6532	SUBSCRIPT_OUTSIDE_LIMIT	Una referencia a una tabla anidada o índice de varray se encuentra fuera del rango declarado (ej: -1).
ORA-6533	SUBSCRIPT_BEYOND_COUNT	Una referencia a una tabla anidada o índice de varray es mayor que el número de elementos de la colección.
ORA-6592	CASE_NOT_FOUND	Se produce cuando la cláusula CASE… END, recibe un valor que no está contemplado en las condiciones WHEN,

		y además no se ha indicado la cláusula ELSE.
ORA-6548	NO_DATA_NEEDED	Excepción que se provoca cuando se manejan tablas PIPELINED. Si se incluye a una excepción OTHER en un bloque que incluye una sentencia PIPE ROW. Si el código que alimenta una sentencia PIPE ROW no va seguido de un procedimiento de limpieza.
ORA-1410	SYS_INVALID_ROW	Ocurre cuando la conversión de un valor ROWID en formato texto a formato universal ROWID, falla porque el resultado de la conversión no representa una fila ROWID válida.

PROVOCAR EXCEPCIONES

Las excepciones normalmente se generan automáticamente como consecuencia de un error en la ejecución del código, pero también es posible forzar que una excepción definida por el usuario se provoque.

La sintaxis es la siguiente:

```
RAISE nombre_excepcion;
```

Ejemplo

```
DECLARE
        Demasiados_estu EXCEPTION;
        V_registro estudiantes%ROWTYPE;
        Cuenta NUMBER;
BEGIN
        SELECT COUNT(*) INTO cuenta FROM estudiantes
        WHERE depto = 'VENTAS';

        SELECT * INTO V_registro FROM estudiantes
        WHERE depto = 'VENTAS';

        IF cuenta > 1 THEN
        RAISE demasiados_estu;
        END IF;
EXCEPTION
        WHEN demasiados_estu THEN
        INSERT INTO temporal;
        WHEN OTHERS
        INSERT INTO otroerror;
END;
```

sintaxis de la sección exception

Esta sección es la encargada de gestionar los errores que se produce en la sección ejecutable de un bloque.

La sintaxis es la siguiente:

```
EXCEPTION
        WHEN nombre_excepcion THEN
        Ordenes;
        WHEN OTHERS THEN
        Ordenes;
END;
```

La cláusula WHEN nos permite gestionar las excepciones predefinidas y las del usuario, permitiendo realizar operaciones cuando las detecte.

El gestor OTHERS controla todas aquellas excepciones generadas y no tratadas con anterioridad. Este gestor debe aparecer en último lugar y se activa cuando los gestores anteriores no satisfacen la excepción que se ha provocado en cuyo caso se activa el gestor OTHERS para resolverlo.

Ejemplo

```
DECLARE
        V_registro estudiantes%ROWTYPE;
BEGIN
        SELECT * INTO v_registro
        FROM estudiantes
        WHERE depto = 'VENTAS';
EXCEPTION
        WHEN NO_DATA_FOUND OR TOO_MANY_ROWS THEN
        INSERT INTO temporal ……;
        WHEN OTHERS
        INSERT INTO otroerror……;
END;
```

USO DE SQLCODE Y SQLERRM

Son variables predefinidas por el lenguaje PL/SQL que tratan el código y el texto del error que se produce en una excepción y su significado es el siguiente:

- SQLCODE: Almacena el código del error en formato NUMBER.
- SQLERRM: Almacena el texto del error en formato VARCHAR2(512).

Ejemplo

```
DECLARE
        V_error NUMBER;
BEGIN
        ...
EXCEPTION
        WHEN OTHERS
        V_error := SQLCODE;
        V_texto := SQLERRM;
END;
```

UTILIZACIÓN DE RAISE_APPLICATION_ERROR

Permite al usuario crear y almacenar sus propios mensajes de error que luego podrá mostrarlos en pantalla.

La sintaxis es la siguiente:

```
RAISE_APPLICATION_ERROR(numero_error, mensaje
                        [,preserver_errores]);
```

Donde:

- NUMERO_ERROR: Puede ser un valor entre -20.000 y -20.999.
- MENSAJE: Puede ser un texto de un máximo de 512 caracteres.
- PRESERVAR_ERRORES: Si se indica TRUE para este parámetro, el error se añade a la lista. Si se indica FALSE (que es el valor predeterminado), el error sustituye a toda la lista de errores anteriormente almacenados por el usuario.

Ejemplo

```
BEGIN
        ...
        IF valor > 50
        RAISE_APPLICATION_ERROR(-2000, 'Hay un error');
        END IF;
EXCEPTION
        ...
END;
```

UTILIZACIÓN DE EXCEPTION_INIT

Permite realizar una asociación entre los errores definidos por el usuario con un código de error de Oracle.

La sintaxis es la siguiente:

```
PRAGMA EXCEPTION_INIT(nombre_excepcion,
                      numero_error_Oracle);
```

Ejemplo

```
DECLARE
        V_error VARCHAR2(512);
        E_sindicatos EXCEPTION;
        PRAGMA EXCEPTION_INIT (e_sindicatos, -1400);
BEGIN
        INSERT INTO estudiantes (id) VALUES (null);
EXCEPTION
        WHEN e_sindicatos THEN
        BEGIN
        V_error := SQLERRM;
        INSERT INTO log_table(cod, message)
        VALUES (-1400, v_error);
        END;
END;
```

PROPAGACIÓN DE EXCEPCIONES

Hasta el momento se han planteado en este capítulo los errores que ocurrían en la sección ejecutable, pero ¿qué ocurre con errores que se producen en otras secciones? ¿Cómo se tratan?

Un error en el código PL/SQL puede llegar a producirse en cualquiera de las secciones de las que consta un bloque, y dependiendo de dónde se produzca hay que tratarlo de una manera diferente. En las siguientes secciones se explica cómo manejar las excepciones que se puedan producir.

Tratamiento de errores en la sección ejecutable

Si se produce un error en dicha sección, Oracle realiza las siguientes operaciones:

1. Busca la sección EXCEPTION del bloque actual. Si la encuentra, realizará las acciones pertinentes y devolverá el control al bloque superior.

2. Si no existe sección EXCEPTION en el bloque actual, se propagará el error al bloque superior y se volverá a realizar la operación anterior.

EJEMPLO

```
DECLARE
    A EXCEPTION;
BEGIN
    ...
    BEGIN
    RAISE A; -- provocamos el error.
    EXCEPTION
    WHEN A THEN -- tratamos el error.
    ...
    END;
    ... -- se devuelve el control.
```

EJEMPLO

```
DECLARE
    A EXCEPTION;
    B EXCEPTION;
BEGIN
    ...
    BEGIN
    RAISE B; -- provocamos el error.
    EXCEPTION
    WHEN A THEN -- no podemos tratar el error.
    ...
    END;
    ... -- se devuelve el control.
```

```
EXCEPTION
WHEN B THEN
        ... -- se propaga al bloque
        -- superior y se trata.
END;
```

EJEMPLO

```
DECLARE
        A EXCEPTION;
        B EXCEPTION;
        C EXCEPTION;
BEGIN
        ...
        BEGIN
        RAISE C; -- provocamos el error.
        EXCEPTION
        WHEN A THEN -- no podemos tratar el error.
        ...
        END;
        ... -- se devuelve el control.
EXCEPTION
WHEN B THEN
        ... -- No se trata y se propaga al
                                -- bloque superior.
END; -- Se propaga al final del
        -- programa y produce un error.
```

Tratamiento de errores en la sección declarativa

Si se produce un error en dicha sección, Oracle realiza las siguientes operaciones:

1. Propaga el error al bloque superior si es que existe.
2. Trata el error en la sección EXCEPTION del bloque superior al que se ha producido el error, si es que existe.

EJEMPLO

```
BEGIN
        ...
        DECLARE
        V_number NUMBER(3) := 'ABC';
        /* provocamos un error de tipo value_error */
        BEGIN
                ...
        EXCEPTION
        WHEN OTHERS THEN
        ...
        /* En este bloque no se trata el error y
        se propaga al bloque superior */
        END;
EXCEPTION
WHEN OTHERS THEN -- se trata el error.
                ...
END;
```

Tratamiento de errores en la sección de excepciones

Si se produce un error en dicha sección, Oracle realiza las siguientes operaciones:

1. Propaga el error al bloque superior si es que existe.
2. Trata el error en la sección EXCEPTION del bloque superior al que se ha producido el error, si es que existe.

EJEMPLO

```
DECLARE
        A EXCEPTION;
        B EXCEPTION;
BEGIN
RAISE A; -- se produce un primer error.
EXCEPTION
WHEN A THEN -- se trata el primer error
RAISE B -- se produce un segundo error
WHEN B THEN
                ...
                /* El segundo error no se trata aquí y se
                propaga al bloque superior.
                Como no hay bloque superior se aborta la
                ejecución del programa con un error */
END;
```

EJEMPLO

```
BEGIN
        DECLARE
                A EXCEPTION;
                B EXCEPTION;
        BEGIN
RAISE A; -- se produce un primer error.
        EXCEPTION
WHEN A THEN -- se trata el primer error
RAISE B -- se produce un segundo error
```

```
WHEN B THEN
                    ...
                    /* No se trata el error y se propaga al
                    al bloque superior */
        END;

EXCEPTION
WHEN OTHERS THEN -- se trata el segundo error
                ...
END;
```

Ámbito de las excepciones

El ámbito de las excepciones es igual que el de las variables. Si una excepción definida por el usuario se propaga fuera de su ámbito, no podrá ser referenciada por su nombre. Para poderlo remediar se utilizan paquetes.

Por ejemplo:

```
CREATE OR REPLACE PACKAGE global
        B EXCEPTION;
END;

SQL> DECLARE
                ...
                BEGIN
                RAISE global.B;
                END;
EXCEPTION
WHEN global.B THEN
...
        END;
```

SUPUESTO PRÁCTICO 13: ***Resolución en el Anexo I de este manual.***

1) Diseñar un procedimiento que sea capaz de dividir 2 números que se soliciten como argumentos del procedimiento. En dicho bloque queremos realizar las siguientes operaciones:

- Realizar la operación de división.
- Mostrar el resultado de la división de la siguiente manera: 'El resultado de la división es: '|| <resultado>.
- Controlar los supuestos de error que puedan producirse en este bloque utilizando las excepciones por defecto que se han aprendido en el curso (al menos hay que controlar 1 supuesto de error que se puede producir).
- En caso de producirse un error se deberá mostrar lo siguiente:
    - Un texto que diga: 'Número de error ORACLE: '||<el código de error de Oracle>.
    - Un texto que diga: 'El mensaje de error ORACLE es: '||<el mensaje de error de Oracle>
    - El texto que diga: 'El valor del dividendo es: '||<valor>
    - El texto que diga: 'El valor del divisor es: '||<valor>
    - Controlar mediante una excepción creada por el usuario en el bloque, la posibilidad de que se introduzca valores negativos para el dividendo o el divisor, mostrando el siguiente

mensaje: 'Los valores introducidos para el dividendo: '||<dividendo>||' o el divisor: '|| <divisor>||' son negativos.'

2) Ejecutar el procedimiento con la instrucción EXECUTE para los siguientes ejemplos:

- Dividendo = -1, divisor = -5
- Dividendo = 15, divisor = 0
- Dividendo = 32.759, divisor = 28
- Dividendo = 5.879, divisor = -4

SUPUESTO PRÁCTICO 14: ***Resolución en el Anexo I de este manual.***

1) Diseñar un procedimiento que tome como único argumento un código de hospital y que realice las siguientes operaciones:

- Detecte mediante un control de errores la no existencia de enfermos en dicho hospital y que muestre un mensaje de notificación de dicha situación.
- Detecte mediante un control de errores la existencia de más de un enfermo en dicho hospital, y en ese caso muestre el número total de enfermos que hay en el mismo, mediante un mensaje.

2) Probar la ejecución del procedimiento para los siguientes supuestos:

- Código de hospital = 1
- Código de hospital = 6
- Código de hospital = 7

Capítulo 12. CURSORES AVANZADOS

BUCLES DE EXTRACCIÓN

Cuando nos enfrentamos al diseño de un cursor, tenemos que tener en cuenta el tipo de bucle que vamos a utilizar para extraer los datos y consecuentemente, para recorrer las filas que reporte la consulta.

Disponemos de tres tipos de bucles que se puede utilizar con un cursor:

- Bucles simples (LOOP…END LOOP).
- Bucles WHILE.
- Bucles FOR.

BUCLES SIMPLES (loop…end loop)

Implican el proceso completo de un cursor explícito: apertura, recorrido y cierre.

El proceso de recorrido por los valores que devuelve el cursor se realizará implementando un bucle LOOP...END LOOP.

Por ejemplo:

```
DECLARE
        V_nombre estudiantes%TYPE;
        v_apellido estudiantes%TYPE;
        CURSOR c_nombres IS
        SELECT nombre, apellido from estudiantes;
BEGIN
        OPEN c_nombres;
        LOOP
                FETCH c_nombres INTO v_nombre, v_apellido;
        EXIT WHEN c_nombres%NOTFOUND;

        /* Tratamiento del registro */

END LOOP;
        CLOSE c_nombres;
END;
```

BUCLES WHILE

Implican el proceso completo de un cursor explícito: apertura, recorrido y cierre.

El proceso de recorrido por los valores que devuelve el cursor se realizará implementando un bucle LOOP...END LOOP.

Por ejemplo:

```
DECLARE
        v_nombre estudiantes%TYPE;
        v_apellido estudiantes%TYPE;
        CURSOR c_nombres IS
        SELECT nombre, apellido from estudiantes;
BEGIN
        OPEN c_nombres;
        FETCH c_nombres INTO v_nombre, v_apellido;
        WHILE c_nombres%FOUND LOOP

        /* Tratamiento del registro */
                FETCH c_nombres INTO v_nombre, v_apellido;
        END LOOP;
        CLOSE c_nombres;
END;
```

bucles for

Este tipo de bucles no hace uso del proceso de un cursor explícito pero tampoco se puede considerar un cursor implícito, dado que a los que se realizan con bucles FOR se les nomina previamente (característica de los cursores explícitos).

Así pues, es un tipo de cursor que comparte características de ambos. En cuanto a las características que le puedan asimilar a un cursor implícito, se encuentra que la apertura, cierre y recorrido del cursor se realizan de forma implícita por el bucle, sin necesidad de expresar órdenes para realizar estas acciones.

El proceso de recorrido por los valores que devuelve el cursor se realizará implementando un bucle FOR LOOP...END LOOP.

Por ejemplo:

```
DECLARE
        v_nombre estudiantes%TYPE;
        v_apellido estudiantes%TYPE;
        CURSOR c_nombres IS
        SELECT nombre, apellido from estudiantes;
BEGIN
        /*
        Al hacer el FOR se abre el cursor y se recupera la primera fila si se puede asignándolo a la variable implícita que no hay que declarar v_nombres.
        */

        FOR v_nombres IN c_nombres LOOP

                -- Tratamiento que se vaya a realizar sobre los
                -- registros.

        /* Los movimientos entre registros son automáticos, no se necesita el fetch haciéndose aquí una comprobación implícita de c_nombres%NOTFOUND. */
```

```
        END LOOP;

        -- Al hacer el end loop se cierra implícitamente el
        -- cursor abierto.
END;
```

cursores select for update

Hasta ahora habíamos visto cursores que consultaban un conjunto de registros y que posteriormente se operaba sobre los valores de esas filas para realizar otras operaciones.

Los cursores SELECT FOR UPDATE tienen como misión la actualización de las propias filas que retorna el cursor.

Poseen 2 partes diferenciadas:

- FOR UPDATE
- WHERE CURRENT OF

For update

Es la última cláusula de la instrucción SELECT. Aparecerá al final de la instrucción tras la cláusula ORDER BY (si es que se utiliza esta última para hacer el SELECT).

La sintaxis es:

```
SELECT …
FROM …
FOR UPDATE [OF columna1[, …columnaX]] [NOWAIT];
```

Se puede indicar una, varias o ninguna columna para actualizar después del OF. En caso de hacerlo únicamente se podrán actualizar en la tabla las columnas que se hayan incluido después de FOR UPDATE OF.

Si no se indica la cláusula OF, se podrá actualizar cualquier columna que retorne el cursor.

Cuando se realiza un SELECT FOR UPDATE se bloquean todas las filas que reporta el cursor, no siendo accesibles para ningún otro proceso que realice una operación sobre las filas bloqueadas de la tabla.

Todo intento de modificar dichas filas quedará congelado hasta que el proceso que ha abierto el cursor para actualización ejecute un COMMIT.

La cláusula NOWAIT impide que otro SELECT FOR UPDATE se quede esperando al COMMIT, inmediatamente presenta un error que indica que las filas están bloqueadas y se ha decidido no esperar hasta que sean desbloqueadas.

EJEMPLO

```
DECLARE
    CURSOR c_estudiantes IS
    SELECT * FROM estudiantes
    FOR UPDATE OF nombre, apellidos;
```

En este ejemplo hemos definido un cursor para actualizar la tabla ESTUDIANTES. Únicamente permitiremos actualizar las columnas NOMBRE y APELLIDOS de dicha tabla.

EJEMPLO

```
DECLARE
    CURSOR c_estudiantes IS
    SELECT curso FROM estudiantes
    WHERE curso = 10
    FOR UPDATE;
```

En este ejemplo hemos definido un cursor para actualizar la tabla ESTUDIANTES. En este caso sí que permitimos actualizar cualquier columna de la tabla.

Where current of

Esta cláusula solo se utiliza en una sentencia UPDATE para indicar que existe un cursor SELECT…FOR UPDATE desde el que se realizará la operación de modificación y únicamente sobre las filas que ha retornado dicho cursor.

La sintaxis es:

```
UPDATE tabla_del_cursor
SET
Columna1 … columna X
WHERE CURRENT OF nombre_cursor;
```

EJEMPLO

```
DECLARE
    CURSOR c_estudiantes IS
    SELECT * FROM estudiantes
    FOR UPDATE OF nombre, apellidos;
BEGIN
FOR v_estudiante IN c_estudiantes LOOP
UPDATE estudiantes
SET
Nombre = 'X-'||nombre
WHERE CURRENT OF c_estudiantes;
END LOOP;
COMMIT;
END;
```

En este ejemplo hemos definido un cursor para actualizar la tabla ESTUDIANTES. Únicamente permitiremos actualizar las columnas NOMBRE y APELLIDOS de dicha tabla.

Durante el bucle de recorrido del cursor se actualiza cada fila retornada, actualizando la columna NOMBRE, introduciéndole la cadena 'X-' delante del contenido que ya tuviese.

EJEMPLO

```
DECLARE
```

```
        CURSOR c_estudiantes IS
        SELECT curso FROM estudiantes
        WHERE curso = 10
        FOR UPDATE;
BEGIN
FOR v_estudiante IN c_estudiantes LOOP
UPDATE estudiantes
SET
id_matricula = id_matricula||'_2001'
WHERE CURRENT OF c_estudiantes;
END LOOP;
COMMIT;
END;
```

En este ejemplo hemos definido un cursor para actualizar la tabla ESTUDIANTES. En este caso sí que permitimos actualizar cualquier columna de la tabla y durante el recorrido de las filas que retorna el cursor, se actualiza la columna MATRICULA introduciendo al contenido que ya tuviese la cadena '_2001'.

SUPUESTO PRÁCTICO 15: ***Resolución en el Anexo I de este manual.***

1) Diseñar un procedimiento que sea capaz mediante un cursor del tipo FOR UPDATE, de actualizar única y exclusivamente el campo SALARIO de la tabla plantilla sanitaria de todos aquellos doctores (para determinar si es un doctor se deberá comparar las tablas plantilla sanitaria y doctores).

- El nuevo salario que se deberá asignar a todos los doctores de la tabla plantilla sanitaria será el máximo que exista en dicha tabla.

2) Ejecutar una consulta que nos muestre el nombre y salario de todos los doctores de la tabla plantilla (comparándolos con la tabla doctores) y observar el salario de todos ellos antes de actualizarlo.

3) Ejecutar el procedimiento creado en el punto 1.

4) Volver a ejecutar la consulta del punto 2 y comprobar que se han actualizado los salarios.

Capítulo 13. OBJETOS

introducción

Los objetos son una de las principales características que ya introdujo la versión de Oracle 8 con respecto a sus versiones anteriores. De esta forma PL/SQL se convirtió en un lenguaje no solo orientado a la programación, sino también a los objetos, característica esta última de los lenguajes más extendidos, como el lenguaje C entre otros.

bases de la programación orientada a objetos

¿Por qué escribimos aplicaciones informáticas? Una posible respuesta es: para modelar el mundo real.

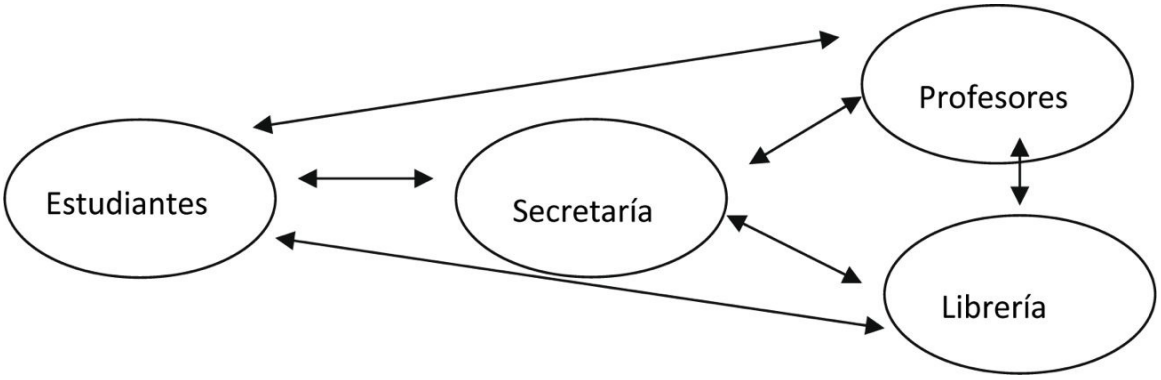
Un sistema software se diseña para simular los objetos que hay en el mundo y las interacciones existentes entre ellos.

Una vez que se han modelado los objetos y sus interacciones, la aplicación se puede utilizar para ver cómo evolucionan y automatizar los procesos implicados.

Para poder entender mejor la programación orientada a objetos, consideremos el siguiente ejemplo:

- Consideremos una universidad. ¿Cuáles son las entidades de este mundo? Tenemos estudiantes que se comunican con la secretaría para contratar los cursos.
- La secretaría informa a los profesores que los estudiantes se han apuntado a los cursos.
- Los profesores se comunican con los estudiantes durante las clases y cuando asignan los niveles.
- Los profesores deben informar a las personas que rigen la librería de la universidad qué libros desean para sus clases, para poder ponerlos a disposición de los estudiantes.

Este modelo quedaría ilustrado de la forma que aparece en la siguiente figura.



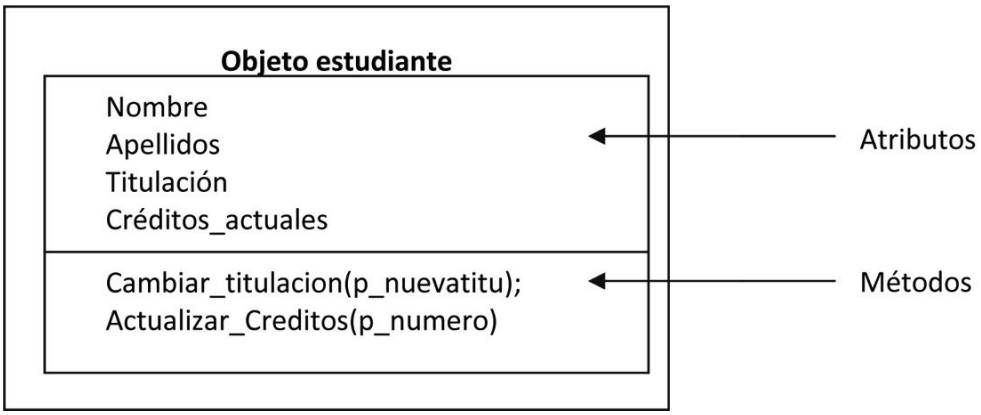
El modelo orientado a objetos implementa directamente este modelo en una

aplicación informática, con las siguientes premisas:

- Cada una de las entidades se representa mediante un objeto del sistema.
- Un objeto representa los atributos de la entidad del mundo real, y las operaciones que actúan sobre dichos atributos.

Por ejemplo y continuando con el ejemplo de la universidad antes descrito:

Consideremos el objeto estudiante, que se representa en la figura que se muestra a continuación.



La interpretación que hacemos de este objeto es que un estudiante tiene atributos tales como el nombre, los apellidos, su titulación y los créditos actuales.

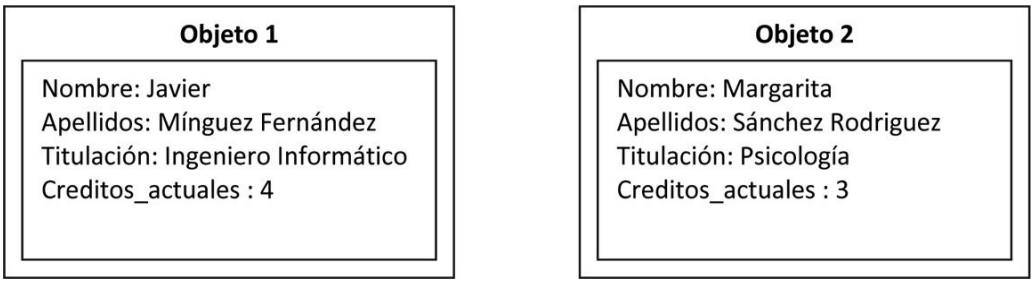
Además, para poder interactuar con este objeto también se incluyen las operaciones que actúan sobre estos atributos tales como el cambio de titulación y la adición de créditos. Esto es lo que se llama los métodos de un objeto.

OBJETOS E INSTANCIAS DE LOS OBJETOS

Es importante destacar la diferencia entre un tipo de objeto y una instancia de dicho tipo. En un entorno dado solo puede haber un tipo de objeto, aunque pueden existir muchas instancias de él.

Una instancia de objeto es similar a una variable: cada instancia tiene su propia memoria y, por tanto, su propia copia de los atributos.

La figura que se muestra a continuación presente 2 instancias del objeto tipo estudiante.



BASES DE DATOS OBJETO-RELACIONALES

Como se ha comentado al comienzo de este capítulo, existen muchos lenguajes de programación orientados a objetos, entre los que se incluyen: C, C++ y Java.

Estos lenguajes permiten definir objetos y manipularlos. Sin embargo, de lo que están faltos estos lenguajes es de características como la persistencia, la capacidad para almacenar y recuperar objetos de forma segura y consistente.

Aquí es donde entran las bases de datos objeto-relacionales como Oracle, la cual está diseñada para almacenar y recuperar datos de objetos igual que sucede con los datos relacionales, utilizando el SQL como método estándar de comunicación con la base de datos.

En una base de datos objeto-relacional se pueden usar los lenguajes SQL y PL/SQL para manipular datos de objetos y datos relacionales.

Oracle10g también proporciona las ventajas de un control de transacciones consistente, de la realización de copias de seguridad y recuperación, de un rendimiento excelente en operaciones de consulta y de las funcionalidades de bloqueo, concurrencia y de escalabilidad.

Combinando los objetos y el modelo relacional tenemos lo mejor de ambos mundos: la potencia y confiabilidad de una base de datos relacional junto con la flexibilidad y capacidades de modelización de los objetos.

definición de los tipos de objetos

La definición de un objeto en Oracle se realiza de una forma muy similar a la definición de un paquete de base de datos, donde aparece una especificación y cuerpo.

En este apartado aprenderemos a realizar las siguientes operaciones:

- Definición de la especificación de un objeto.
- Declaración, inicialización y asignación de valores a un objeto.
- Especificación de métodos.
- Definición del cuerpo de un objeto.

Especificación de un objeto

La sintaxis es:

```
CREATE [OR REPLACE] TYPE [esquema.]nombre_tipo
AS OBJECT
(nombre_atributo tipo
[,nombre2 tipo] …
| [{MAP | ORDER} MEMBER especificación_función]
| [MEMBER {especif_funcion | especif_procedimiento}
| [,MEMBER … ]);
```

Hay varios puntos a destacar sobre los tipos de objetos:

1. La orden CREATE TYPE es una orden DDL. Por tanto, no se puede utilizar directamente en un bloque PL/SQL.

2. Es necesario tener el privilegio del sistema CREATE TYPE que es parte del papel RESOURCE, para poder crear un tipo objeto.

3. Los tipos objeto se crean como objetos del diccionario de datos. En consecuencia se crean en el esquema actual, a menos que se especifique un esquema diferente en la orden CREATE TYPE ... AS OBJECT.

4. Los atributos del nuevo tipo creado se especifican de forma similar a los campos de un registro PL/SQL o las columnas de una tabla en la orden CREATE TABLE.

5. A diferencia de los campos de registro, los atributos de un tipo de objeto no pueden restringirse para que tomen el valor NOT NULL, ni tampoco pueden inicializarse con un valor predeterminado.

6. Al igual que un registro PL/SQL, puede referenciar los atributos de un objeto utilizando la notación de punto.

Existen también varias restricciones sobre el tipo de datos de los atributos del objeto. En concreto los atributos de los objetos pueden ser de cualquier tipo de datos Oracle excepto:

- LONG o LONG RAW. Sin embargo, pueden ser de tipo LOB.
- Cualquier tipo de lenguaje nacional tal como NCHAR, NVARCHAR2 o NCLOB.
- ROWID.
- Los tipos disponible solo en PL/SQL pero no en la base de datos. Entre estos se incluyen BINARY_INTEGER, BOOLEAN, PLS_INTEGER, RECORD y REF CURSOR.
- Un tipo definido con %TYPE o %ROWTYPE.
- Tipos definidos dentro de un paquete PL/SQL.

EJEMPLO

```
CREATE OR REPLACE TYPE estudiante AS OBJECT
(id NUMBER(5),
Nombre VARCHAR2(20)
);
```

Declaración, inicialización y asignación de valores a un objeto

Al igual que cualquier otra variable PL/SQL un objeto se declara simplemente incluyéndolo en la sección declarativa de un bloque.

Por ejemplo, para declarar una variable de tipo objeto podemos utilizar la siguiente estructura:

```
DECLARE
        V_estudiante estudiante;
```

Para declarar y a la vez inicializar una variable de tipo objeto, podemos utilizar la siguiente estructura:

```
DECLARE
V_estudiante estudiante := estudiante(1000,'Pepe');
```

Si por el contrario lo que queremos es hacer una asignación de valores a una variable de tipo objeto previamente declarada, se utilizará la siguiente estructura (por ejemplo, con la variable declarada anteriormente):

```
DECLARE
V_estudiante estudiante := estudiante(1000,'Pepe');
BEGIN
V_estudiante.nombre := 'JUAN';
END;
```

Si un objeto no es inicializado, entonces el objeto tendrá valor nulo (NULL) y sus atributos no estarán disponibles. Si por ejemplo queremos evaluar si un objeto está o no nulo, podremos utilizar la siguiente estructura:

```
DECLARE
V_estudiante estudiante;
BEGIN
IF v_estudiante IS NULL THEN …
END IF;
END;
```

Especificación de métodos

Para definir los métodos asociados a un objeto, se utiliza la siguiente sintaxis:

MEMBER {FUNCTION | PROCEDURE} nombre

EJEMPLO

```
CREATE OR REPLACE TYPE estudiante AS OBJECT
(id NUMBER(5),
nombre VARCHAR2(20),
apellidos VARCHAR2(20),
nota NUMBER(2),
MEMBER FUNCTION nombre_ape RETURN VARCHAR2,
MEMBER PROCEDURE cambio_nota (nueva IN NUMBER)
);
```

Como se puede observar en el ejemplo, la definición de los métodos asociados a un objeto siempre se realiza después de la definición de atributos del mismo.

Cuerpo de un objeto

La definición del cuerpo de un objeto se realiza con la siguiente sintaxis:

```
CREATE [OR REPLACE] TYPE [esquema.]nombre
BODY AS
[{MAP | ORDER} MEMBER declaración_función;]
| [MEMBER {declaración_procedimiento | declaración_función};
…
END;
```

EJEMPLO

```
CREATE OR REPLACE TYPE BODY estudiante AS
    MEMBER FUNCTION nombre_ape RETURN VARCHAR2 IS
    BEGIN
    RETURN nombre||' '||apellidos;
    END;

    MEMBER PROCEDURE cambio_nota (nueva IN NUMBER) IS
    BEGIN
    Nota := nueva;
END;

END;
```

LLAMADA A UN MÉTODO

Para invocar a un método de un objeto hay que utilizar la siguiente sintaxis:

nombre_objeto.metodo

EJEMPLO

```
DECLARE
    V_estudiante estudiante := estudiante (1000, 'PEPE', 'Rodriguez', 3);

BEGIN
    V_estuadiante.cambio_nota(10);
    DBMS_OUTPUT.PUT_LINE(v_estudiante.nombre_ape);
END;
```

borrar un objeto

Sintaxis:

DROP TYPE nombre [FORCE]

Cuando se utiliza la sentencia de borrado de un objeto junto a la cláusula FORCE se está indicando a Oracle que elimine el tipo de objeto cuyo nombre se ha indicado, aunque puedan existir dependencias de él.

En caso de que no se indique la cláusula FORCE y existir dependencias del objeto el SGBD de Oracle no permitirá llevar a cabo el borrado del objeto.

MODIFICAR un objeto

Sintaxis:

ALTER TYPE nombre COMPILE [SPECIFICATIONS | BODY]

Durante la modificación de un objeto solo se permite la recompilación de una de las dos partes de las que consta un objeto: especificación o cuerpo.

No se permite cambiar el contenido de un objeto con esta cláusula, para ello habría que reescribir el comando de creación del objeto.

Como alternativa a la creación de nuevo de la especificación de un objeto, se admite el comando ALTER para reemplazar un objeto por otro, con la siguiente sintaxis:

ALTER TYPE nombre REPLACE AS OBJETCT (espec_tipo_objeto)

CREACIÓN DE TABLAS DE OBJETOS

Al igual que está permitida la creación de tabla relacionales, también está permitida la creación de tablas de objetos, con la sintaxis que se indica a continuación:

```
CREATE TABLE nombre OF objeto;
```

inserción de valores en una tabla de objetos

Cuando necesitemos insertar valores en una tabla de objetos, utilizaremos la siguiente sintaxis:

INSERT INTO tabla VALUES (objeto(valor1,…,valorX));

Por ejemplo:

```
INSERT INTO tab_estudiante VALUES
(estudiante(2000, 'JOSE', 'FRANCISCO', 4));
```

Capítulo 14. ENTORNOS DE EJECUCIÓN PL/SQL

introducción

Los entornos de ejecución son los programas que nos van a permitir lanzar código PL/SQL contra el sistema gestor de base de datos Oracle.

En algunos entornos como Oracle Forms y Procedure Builder, los bloques PL/SQL pueden ejecutarse completamente en el cliente sin interacciones con el servidor de la base de datos.

En otros entornos como SQL*Plus, los precompiladores o SQL-Station, se pueden enviar desde un programa cliente para ejecutarse en el servidor.

SQL*PLUS / Isql*plus / SQL*WORKSHEET

SQL*Plus permite al usuario introducir órdenes SQL y bloques PL/SQL interactivamente. Si se producen errores hay que reescribir el código para volverlo a ejecutar.

iSQL*Plus permite al usuario introducir órdenes SQL y bloque PL/SQL a través de un navegador (Explorer, Firefox, etc.) y permite la corrección de los errores sin necesidad de reescribir todo el código.

SQL*Worksheet permite al usuario introducir órdenes SQL y bloques PL/SQL a través de una consulta en Java dividida en 2 áreas: área de comandos (introducción de sentencias) y área de resultados. Permite la corrección de los errores sin necesidad de reescribir todo el código. Presenta como problema la dificultad para ejecutar bloques PL/SQL donde se soliciten valores por pantalla.

En todos los casos las órdenes se envían directamente a la base de datos y los resultados se visualizan en la pantalla.

EJECUCIÓN DE CÓDIGO SQL

Cuando queremos que se ejecute un código habrá de introducirse la sentencia y terminar con el símbolo punto y coma (;). Esto hace que automáticamente se ejecute la misma dentro de SQL*Plus.

Dentro de iSQL*Plus y SQL*Worksheet habrá de pulsarse sobre el botón ejecutar.

Por ejemplo:

```
SQL> SELECT SYSDATE FROM DUAL;
```

ejecución de código pl/sql

Cuando queremos que se ejecute un código PL/SQL en SQL*Plus habrá de introducirse el mismo y al terminar indicar el símbolo /. Esto permite al intérprete de comandos que evalúe la expresión, la compile para ver si es correcta y la ejecute.

Dentro de iSQL*Plus y SQL*Worksheet habrá de pulsarse sobre el botón ejecutar para que lo ejecute y no es necesario que detrás del “END;” se introduzca el símbolo /.

Por ejemplo:

```
SQL> BEGIN
2 > NULL;
3 > END;
4 > /
```

definición de VARIABLES GLOBALES

La sintaxis para la definición de una variable global es la siguiente:

VARIABLE nombre tipo;

Una variable global es accesible por cualquier código SQL o PL/SQL que se lance en la misma sesión donde se ha declarado. Una vez que se cierra dicha sesión la variable global desaparece.

Por ejemplo:

```
SQL> VARIABLE v_num NUMBER;
```

USO DE VARIABLES GLOBALES

¿Cómo podemos visualizar el valor de una variable global? La sintaxis es:

PRINT nombre_variable_global;

¿Cómo se puede utilizar una variable global dentro de un bloque si nominar o nominado? La sintaxis es:

:nombre

Por ejemplo:

```
SQL> VARIABLE v_pruebas NUMBER;
SQL> SET SERVEROUTPUT ON;

SQL> BEGIN
2 > :pruebas := 20;
3 > dbms_output.put_line(:pruebas);
4 > end;
5 > /

SQL> PRINT v_pruebas;
```

CÓMO SE PUEDE LLAMAR A UN PROCEDIMIENTO ALMACENADO

La sintaxis para invocarlo directamente en la línea de comandos es:

EXEC[UTE] nombre_procedimiento [(parámetro1,…,parámetroX)];

Por ejemplo, para invocar un procedimiento que no tiene parámetros:

```
SQL> EXEC pruebas;
```

Por ejemplo, para invocar un procedimiento que solo tiene parámetros de entrada:

```
SQL> EXECUTE pruebas(1);
```

Por ejemplo, para invocar un procedimiento que tiene parámetros de entrada/salida:

```
SQL> VARIABLE :v_cuenta NUMBER;

SQL> BEGIN
2 > :v_cuenta := 1;
3 > END;
4 > /
SQL> EXEC pruebas(:v_cuenta);

SQL> PRINT :v_cuenta;
```

CÓMO SE PUEDE LLAMAR A UNa función almacenada

La sintaxis para invocarlo dentro de una instrucción SELECT en la línea de comandos es la siguiente:

```
SELECT nombre_funcion [ (parámetros) ] FROM DUAL;
```

La sintaxis para invocarlo dentro de un bloque sin nominar en la línea de comandos es la siguiente:

```
BEGIN
:nombre_variable_global := nombre_funcion [ (parámetros) ];
END;
```

Por ejemplo, para invocarlo desde una consulta:

```
SQL> SELECT FACTORIAL(3) FROM DUAL;
```

Por ejemplo, para invocarlo desde un bloque sin nominar:

```
SQL> VARIABLE v_cuenta NUMBER;
SQL> BEGIN
2 > :v_cuenta := FACTORIAL(3);
3 > END;
4 > /
SQL> PRINT v_cuenta;
```

ENVÍO DE RESULTADOS A UN ARCHIVO

En muchas ocasiones el resultado de la ejecución de los comandos SQL que lanzamos en el intérprete de comandos correspondiente lo queremos almacenar en un fichero para su revisión. Esta opción es posible utilizando la siguiente sintaxis:

SPOOL ruta_y_nombre_archivo;

Después del comando SPOOL, debemos indicar un nombre de archivo donde se enviará el resultado de todo lo que se ejecute desde el momento de lanzar el comando, teniendo en cuenta que el archivo debe contener una ruta completa accesible desde el intérprete de comandos.

Cuando se quiera terminar el envío de los resultados de ejecutar los comandos SQL y PL/SQL al archivo, habrá de indicarse la siguiente sintaxis que cierra el archivo:

SPOOL OFF;

Por ejemplo:

```
SQL> SPOOL C:\TMP\resultado.txt;
SQL> SELECT SYSDATE FROM DUAL;
SQL> SPOOL OFF;
```

Después de ejecutar el ejemplo anterior, si abriésemos el fichero RESULTADO.TXT que se ha almacenado en la ruta C:\TMP, el resultado que obtendríamos sería el siguiente:

```
SQL> SELECT SYSDATE FROM DUAL;

SYSDATE
20/09/11

SQL> SPOOL OFF;
```

EJECUCIÓN DE SCRIPTS (ARCHIVOS) DE COMANDOS

Todos los intérpretes de comandos SQL y PL/SQL admite la ejecución de un script (archivo) de comandos que se haya construido en formato de texto plano.

La sintaxis para la ejecución del script es:

START ruta_y_nombre_archivo;

También se puede utilizar la siguiente sintaxis en sustitución de la anterior:

@ruta_y_nombre_archivo;

Por ejemplo:

```
SQL> START C:\TMP\archivocomandos.sql;
SQL> @C:\TMP\archivocomandos.sql;
```

mostrar errores de compilación

En este libro hemos aprendido a visualizar los errores que se producen en la compilación de bloques tanto nominados (procedimientos, funciones, paquetes y triggers), como sin nominar, utilizando la vista del sistema USER_ERRORS, pero también existe otra opción equivalente, que resulta más rápida para capturarlos:

```
SHOW ERRORS;
```

Por ejemplo:

```
SQL> CREATE PROCEDURE PRUEBAS2 IS
1 > BEGIN
2 > NULL
3 > END;
4 > /

Warning: Procedure created with compilation errors.

SQL> SHOW ERRORS;

LINE/COL ERROR
-------- -----------------------------------------------------------------
3/5 PLS-00103: Encountered the symbol "NULL" when expecting one of
the following:
constant exception <an identifier>
<a double-quoted delimited-identifier> table LONG_ double
```

En este ejemplo concreto hemos forzado que se produzca un error de compilación al haber omitido el símbolo ; después de la instrucción NULL.

HERRAMIENTAS DE DISEÑO

Con SQL*Plus y en general cualquier intérprete de comandos SQL y PL/SQL podemos diseñar estructuras de programa que nos permitan realizar las acciones que necesitamos, pero resulta evidente que no podemos diseñar un programa entero que posea pantallas, botones, campos de texto, listas, etc.

Para ello, necesitamos herramientas de diseño que nos permitan trabajar con código SQL y PL/SQL contra Oracle, y que dispongan de una interfaz visual moldeable por el usuario a través de formularios y pantallas. Entre ellas nos encontramos con:

- Oracle Developer Suite 10g.
- Oracle Developer Tools for Visual Studio.

Oracle Developer Suite 10g

Es una herramienta de diseño de Oracle que permite crear programas con todos los elementos que permiten las herramientas de diseño: ventanas, formularios, elementos de navegación e introducción de datos, botones, generadores de informes, generadores de gráficos estadísticos, conexiones con otros programas Windows, etc.

Oracle Developer Suite 10g es un paquete que está compuesto de los siguientes componentes:

- Oracle JDeveloper.
- Oracle Forms.
- Oracle Reports.
- Oracle Designer.
- Developer Suite Software Configuration Manager.
- Oracle Discover.
- Oracle Business Intelligence Beans.

Oracle JDeveloper

Es un componente de la herramienta Oracle Developer Suite 10g orientado a la programación en arquitecturas SOA y Java. Permite el diseño completo de aplicaciones para dichos entornos programando en Java.

Oracle Forms

Es un componente de la herramienta Oracle Developer Suite 10g que permite la creación de aplicaciones en entornos gráficos programando directamente en SQL y PL/SQL. La filosofía de trabajo de la herramienta es el uso de formularios cuya funcionalidad se obtiene a través de eventos que ocurren sobre los elementos del mismo.

Posee asistentes para la creación de los elementos más importantes de un proyecto: bloques, lista de valores, elementos de texto, pantallas, etc.

Posee un potente depurador para poder controlar y visualizar paso a paso y mediante interrupciones el código que se ejecuta en el proyecto.

Permite la creación de aplicaciones para su publicación en web.

Es un programa abierto dado que interactúa con otras aplicaciones y herramientas gracias a que tiene habilitado recursos de Java Beans, controles Active X, OLE (Object linking and embedding) y DDE (Dynamic Data Exchange).

Permite además el desarrollo de aplicaciones para acceso a otras bases de datos como RDB, SQL SERVER, INFORMIX, SYBASE, DB2 y ACCESS.

Oracle Reports

Es un componente de la herramienta Oracle Developer Suite 10g que permite la creación de informes o listados.

Como en caso anterior utiliza asistentes para la creación de los elementos más importantes del informe. Y también permite la generación de listados/informes para la web.

Es capaz de generar listados/informes en los siguientes formatos: HTML, PDF, XML, RTF y texto.

Oracle Designer

Es un componente de la herramienta Oracle Developer Suite 10g que permite el diseño y modelado de una aplicación, considerándose una herramienta CASE.

Permite el modelado de los procesos de negocio, el análisis de los sistemas, el diseño de aplicaciones software y, por último, la generación del aplicativo: creación de estructuras y elementos de base de datos, así como esqueletos de formularios en Oracle Forms y listados en Oracle Reports.

Dispone de un repositorio donde se almacenan todos los elementos del diseño de la aplicación.

Oracle Business Intelligence Beans

Es un componente de la herramienta Oracle Developer Suite 10g que permite a los desarrolladores de construir la funcionalidad correspondiente a la inteligencia del negocio aprovechando la potencia de OLAP sobre bases de datos Oracle.

Esta herramienta consta de presentaciones gráficas, tablas de información, consultas, constructores de operaciones de cálculo y servicios, que pueden ser desarrollados para clientes HTML o Java indistintamente.

Se encuentra completamente integrado con Oracle JDeveloper para mejorar la productividad en el desarrollo de aplicaciones de Business Intelligence.

Oracle Developer Tools for Visual Studio

Es una herramienta para añadir a Microsoft Visual Studio y que se integra perfectamente con las siguientes versiones de Visual Studio: 2005, 2008 y 2010.

Las características que aporta esta herramienta a Microsoft Visual Studio son las siguientes:

- Generación automática de código .NET.
- Facilidad para el desarrollo de aplicaciones ASP.NET para la web.
- Herramientas para la mejora del rendimiento de las aplicaciones (Tuning).
- Generación de scripts SQL para creación de objetos del esquema de Oracle con integración en el panel de control del código fuente.
- Editor PL/SQL y debugger para corrección de errores.
- Desarrollo de procedimientos almacenados NET.
- Sistema integrado de ayuda online.
- Manejo de usuarios, roles y privilegios.
- Diseñadores avanzados de consultas.
- Definición y creación de tipos de usuarios.
- Asistentes para la importación de tablas.
- Edición de datos, testeo de procedimientos almacenados y ejecución de código SQL.

ORACLE SQL DEVELOPER

Es una herramienta gratuita que proporciona Oracle para el desarrollo contra bases de datos, que tiene un soporte gráfico e intuitivo.

Con esta herramienta se puede navegar por los objetos de la base de datos, ejecutar sentencias y scripts SQL y editar y depurar sentencias PL/SQL.

Esta herramienta se puede obtener directamente desde el siguiente enlace de Internet, previo registro gratuito dentro de la web de Oracle:

http://www.oracle.com/technetwork/developer-tools/sql-developer/overview/index.html

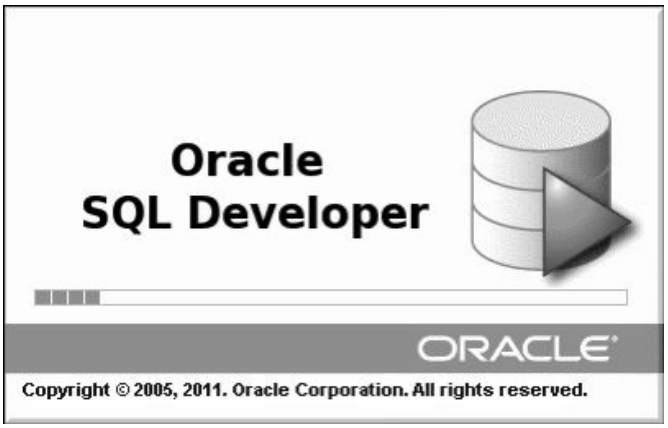
Oracle SQL Developer trabaja bajo la plataforma de Java, por lo que necesita que en el ordenador en el que se instale ya exista una instalación del JDK 1.6 o superior. Si no se dispone de dicho software o bien se desconoce su existencia en el equipo sobre el que se instalará, se deberá seleccionar la opción de descarga del producto que se indica a continuación:

Oracle SQL Developer for 32-bit Windows (This zip file includes the JDK1.6.0_11).

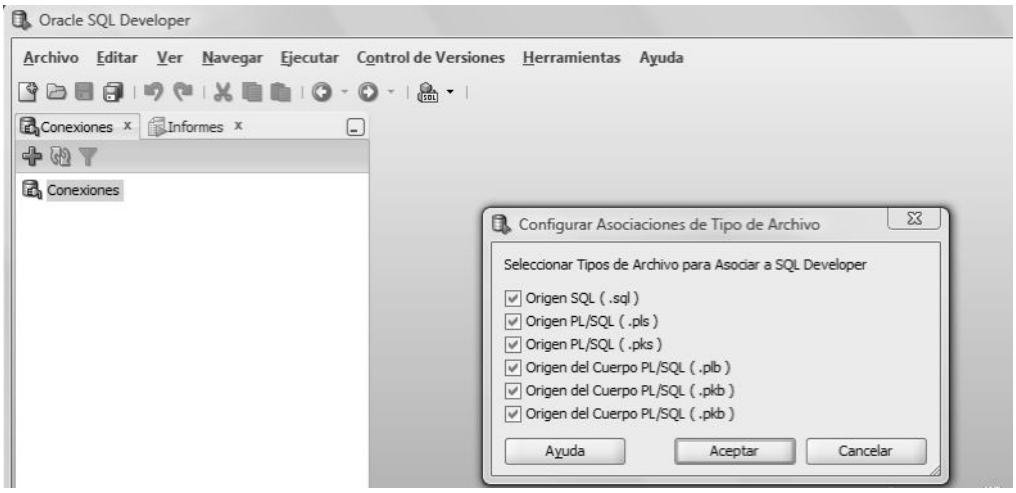
La instalación de la herramienta Oracle SQL Developer requiere unos pasos muy simples:

1. Descomprimir el archivo .zip que se descarga de la página web de Oracle.
2. Mover la carpeta sqldeveloper que se obtiene después de la descompresión desde la ubicación donde se realizó esta operación a la ruta en la que se quiera ubicar. O bien se puede ejecutar el producto directamente desde esta misma carpeta, en la ruta donde se ha descomprimido el producto.

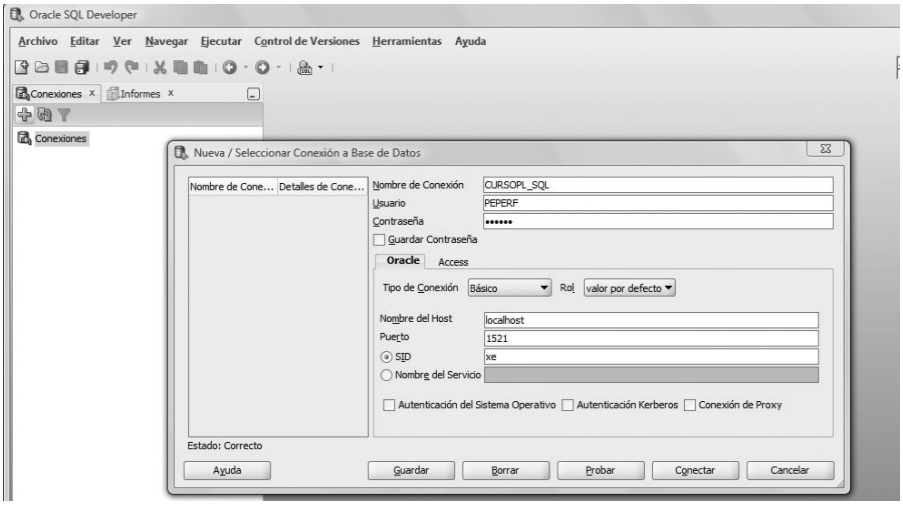
Para ejecutar la herramienta, hay que utilizar el fichero sqldeveloper.exe que se encuentra en la carpeta sqldeveloper. Al hacerlo, se mostrará una primera pantalla como la que se muestra a continuación, con el estado de progreso de la ejecución de la herramienta:



Una vez terminada la carga de la herramienta, se muestra el área de trabajo de la misma, con un recuadro en el que se solicita la asociación de extensiones de archivos que queremos que se abran directamente con Oracle SQL Developer. Normalmente, si no se dispone de otra herramienta para la edición de código SQL y PL/SQL, se marcarán todas las opciones como se muestra a continuación:



Por último, para poder trabajar contra una base de datos tenemos que crear una nueva conexión en la herramienta, para ello pulsaremos sobre el símbolo + que aparece en la solapa de Conexiones. Al hacerlo se mostrará una ventana en la que debemos introducir los datos de conexión contra la base de datos que vayamos a utilizar. En la pantalla que se muestra a continuación se indican los datos de conexión que se utilizan por defecto para el desarrollo de este curso:

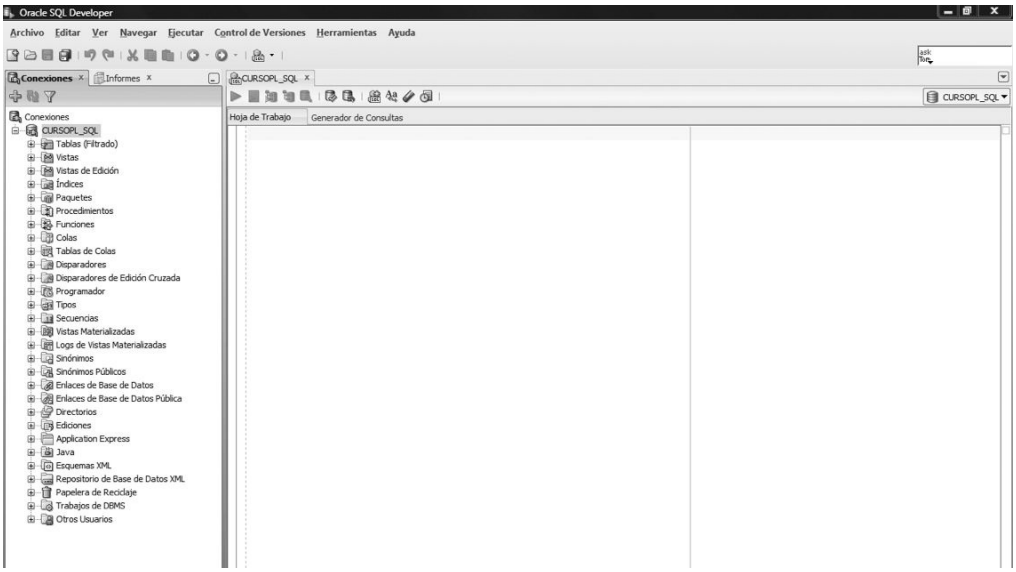


En concreto la información que se utiliza en la conexión es:

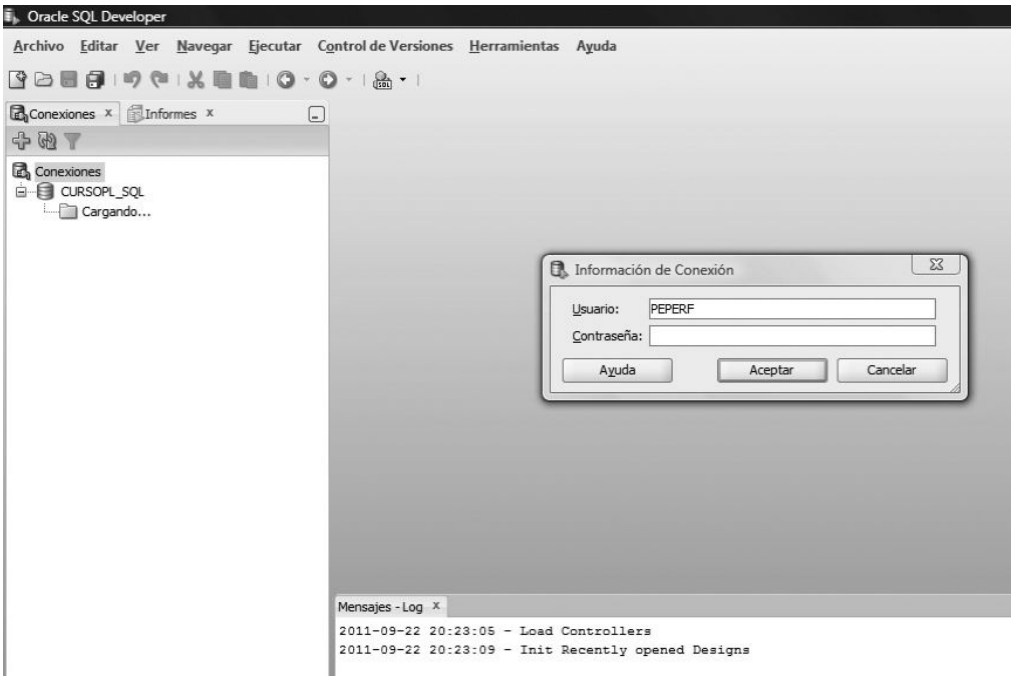
- Nombre de conexión: CURSOPL_SQL.
- Usuario: PEPERF.
- Contraseña: PEPITO.
- Tipo de conexión: Básico.
- Rol: Valor por defecto.
- Nombre del host: Localhost.
- Puerto: 1521.
- SID: xe.

Para comprobar que los datos de conexión son correctos y tenemos los servicios de la base de datos levantados (LISTENER e INSTANCIA), pulsamos sobre el botón Probar y en la ventana aparecerá junto a la palabra "Estado:" el texto Correcto que indica que se ha podido conectar contra la base de datos.

Únicamente nos queda por hacer persistente esta conexión dentro de Oracle SQL Developer para que cada vez que entremos no tengamos que realizar esta operación. Esto se consigue pulsando sobre el botón Conectar. A continuación, se crea una estructura en forma de árbol en la solapa "Conexiones" con los elementos de base de datos pertenecientes al usuario que hemos utilizado para la conexión, tal y como se muestra a continuación:



A partir de este momento cuando salgamos de la herramienta y volvamos a entrar, para conectarnos a alguna de las definiciones de conexión que tenemos en el árbol "Conexiones", únicamente tendremos que hacer doble clic sobre la misma y solo nos solicitará la contraseña del usuario, como se muestra en la siguiente figura:



Capítulo 15. CERTIFICACIONES DE ORACLE

introducción

Las certificaciones[1] de Oracle están reconocidas por la industria, y son la mejor carta de presentación para ayudarle a tener éxito en su carrera profesional dentro del sector de las Tecnologías de la Información.

Las certificaciones de Oracle son un certificado que prueba la educación y experiencia recibidas en los diversos productos de Oracle, y pueden acelerar su desarrollo profesional, mejorar su productividad y mejorar su credibilidad.

CERTIFICACIONES DE ORACLE DISPONIBLES

De acuerdo a la información que proporciona Oracle, a fecha de elaboración de este capítulo (22/5/2011), se pueden obtener las certificaciones en base de datos que se muestran en la tabla del siguiente apartado.

Para actualizar dichas certificaciones, o puede consultar el global de todas ellas, acudiendo al siguiente enlace de Internet:

http://education.oracle.com/pls/web_prod-plq-dad/db_pages.getpage?page_id=141&p_org_id=51&lang=E

Certificaciones de base de datos

En la siguiente tabla podrá encontrar las distintas certificaciones que pueden ayudarle a mejorar su perfil de cara a obtener el cargo que se indica en la tabla.

Asimismo, Oracle distingue cuatro tipos de perfiles relacionados con el cargo: Asociado, Profesional, Maestro y Especialización o experto. Dependiendo del tipo de perfil que quiera obtener, deberá realizar los exámenes de certificación que se indican en cada caso.

Producto	Cargo	Asociado	Profesional	Maestro	Especialización/ Experto
Cualquier versión del producto	Administrador de B.D.				Oracle Database: SQL Certified Expert
Cualquier versión del producto	Desarrollador				Oracle Application Express Developer Certified Expert
Oracle Database 11g	Administrador de B.D.				Oracle Database 11g Performance Tuning Certified Expert.
Oracle Database 11g	Administrador de B.D.				Oracle Certified Expert, Oracle Real Application Clusters 11g and Grid Infrastructure Administrator
Oracle Database 11g	Implementador				Oracle Database 11g Certified Implementation Specialist
Oracle Database 11g	Implementador				Oracle Database 11g Security Certified Implementation Specialist
Oracle Database 11g	Implementador				Oracle Data Warehousing 11g Certified Implementation Specialist
Oracle Database 11g	Implementador				Oracle Exadata 11g Certified Implementation Certification
Oracle Database 10g	Administrador de B.D.	Oracle 10g DBA OCA	Oracle 10g DBA OCP	Oracle 10g DBA OCM	
Oracle Database 10g	Administrador de B.D.				Oracle 10g: Managing Oracle on Linux Certified Expert
Oracle Database 10g	Administrador de B.D.				Oracle Database 10g: Administering Real Application Clusters Certified Expert.
Oracle Database 9i	Administrador de B.D.	Oracle 9i DBA OCA	Oracle 9i DBA OCP		

MySQL 5	Nivel de conexión para desarrollo o DBA	OCA, MySQL 5			
MySQL 5	Desarrollador		OCP, MySQL 5 Developer		
MySQL 5	Administrador de B.D.		OCP, MySQL 5 Database Administrator		
MySQL 5	Administrador de B.D.				OCE, MySQL 5.1 Cluster Database Administrator.
Oracle Enterprise Manager 11g	Implementador				Oracle Enterprise Manager 11g Certified Implementation Specialist
Oracle Enterprise Manager 11g	Implementador				Oracle Enterprise Manager 11g Application Quality Management Certified Implementation Specialist

INFORMACIÓN SOBRE EL PROCESO

La información que se expone en este apartado ha sido recopilada desde las propias páginas de certificación de Oracle.

Puede consultar información actualizada sobre todo el proceso de certificación en el siguiente enlace de Internet:

http://education.oracle.com/pls/web_prod-plq-dad/db_pages.getpage?page_id=39.

Identificar el camino correcto para la certificación deseada

La experiencia técnica es provechosa y recomendada a la hora de afrontar un examen de certificación. Una combinación de experiencia de uso del producto, la propia experiencia técnica y la certificación asegurarán más amplias oportunidades de carrera para usted.

Elija su camino preferido y repase las exigencias deseadas para afrontar la certificación que le gustaría obtener. Las exigencias de certificación varían para cada certificación. Acceda a la información completa sobre la certificación seleccionada desde la propia página de certificaciones de Oracle, donde se informa de todos los requisitos y el temario necesario para afrontar el examen de certificación.

Prepararse para el examen

Existe un buen número de exámenes de muestra para la preparación de las distintas certificaciones de Oracle que se pueden obtener online, introduciendo el código de examen en un buscador de Internet.

También, Oracle pone a disposición de los candidatos a obtener una certificación, un conjunto de recursos gratuitos para la preparación de sus certificaciones. Pueden acceder a estos recursos en el siguiente enlace de Internet:

http://education.oracle.com/pls/web_prod-plq-dad/db_pages.getpage?page_id=42&p_org_id=51&lang=E

Programar y completar el examen

Visite la página web de registro de exámenes para ver las instrucciones sobre la petición de una cita en un Centro examinador de Oracle (Oracle Testing Center), o a través de un Centro examinador autorizado Pearson VUE (Pearson VUE Authorized Test Center). El enlace de Internet a dicha página es el siguiente:

http://education.oracle.com/pls/web_prod-plq-dad/db_pages.getpage?page_id=51&p_org_id=51&lang=E.

Completar los requerimientos de tu certificación

La dirección que usted facilite a Pearson VUE, en el momento del registro del examen de certificación, será usada por Oracle para enviarle el kit del Programa de Éxito de la certificación de Oracle. Asegúrese de mantener al día los datos del domicilio y correo electrónico. Si necesita cambiar cualquier información de contacto, puede visitar la página www.pearsonvue.com/oracle.

Además, confirme que usted completó todos los requerimientos de la certificación correspondientes al número de identificación del test. Puede confirmar su histórico de requerimientos de certificación de manera on-line, en el enlace certview.oracle.com.

Dentro de las 6-8 semanas posteriores a pasar todos los exámenes requeridos, usted recibirá el kit del Programa de Éxito de la certificación de Oracle.

PREGUNTAS TIPO EXAMEN DE CERTIFICACIÓN SQL

En esta sección se ha recopilado de diversas fuentes públicas para la preparación de los exámenes de Certificación de Oracle, una serie de preguntas que le pueden orientar y servir de práctica a la hora de afrontar un examen de certificación en el lenguaje SQL.

En concreto, actualmente existen los siguientes códigos de examen de certificación en PL/SQL:

- 1Z0-144 (Oracle Database 11g: Program with PL/SQL).
- 1Z0-146 (Oracle Database 11g: Advanced PL/SQL).
- 1Z0-147 (Program with PL/SQL).

Es importante reseñar que todos los exámenes se realizan en inglés, por lo que debe de estar familiarizado con la terminología técnica en dicho idioma.

Las preguntas aquí recogidas en algunos casos se han traducido al castellano para facilitar su comprensión, pero no debe confundir al candidato que se presente al examen, dado que no las encontrará así.

La resolución a las preguntas que se plantean a continuación las podrá encontrar en el Anexo II de este libro.

Cuestión 1

¿Cuántas veces se ejecutará la sentencia de inserción dentro de la ejecución del FOR LOOP del siguiente bloque de PL/SQL?

```
DECLARE
        v_lower NUMBER := 2;
        v_upper NUMBER := 100;
        v_count NUMBER := 1;
BEGIN
        FOR i IN v_lower..v_lower LOOP
                INSERT INTO test(results)
        VALUES(v_count);
        v_count := v_count + 1;
        END LOOP;
END;
```

A. 0
B. 1
C. 2
D. 98
E. 100

Cuestión 2

Evalúe el siguiente código PL/SQL y responda a la pregunta: ¿Cuál será el valor de V_RESULT si los 3 registros son borrados?

```
DECLARE
        V_result BOOLEAN;
BEGIN
        DELETE FROM sale
        WHERE salesperson_id IN (25,35,45);
        V_result := SQL%ISOPEN;
        COMMIT;
END;
```

A. 0
B. 3
C. TRUE
D. NULL
E. FALSE

Cuestión 3

Dado el siguiente procedimiento, si V_BONUS = TRUE y V_RAISE = NULL, ¿qué valor se asigna a V_ISSUE_CHECK?

```
PROCEDURE dept_salary ( v_bonus IN BOOLEAN,
        v_raise IN BOOLEAN,
        v_issue_check IN OUT BOOLEAN)
IS
BEGIN
        V_issue_check := v_bonus OR v_raise;
END;
```

A. TRUE
B. FALSE
C. NULL
D. Ninguna de las anteriores

Cuestión 4

Los procedimientos y funciones son muy similares. ¿Qué razón nos hace elegir crear una función en vez de un procedimiento, en caso de que podamos hacer ambos?

A. Una función devuelve 1 valor.
B. Una función se puede usar en una sentencia SQL.
C. Una función solo se puede ejecutar desde un procedimiento creado previamente.
D. Una función garantiza que solo se consulta información, mientras que un procedimiento no garantiza esto.

Cuestión 5

Examine la siguiente función PL/SQL.

```
CREATE OR REPLACE FUNCTION get_budget(v_studio_id IN NUMBER)
RETURN number IS
        v_yearly_budget NUMBER;
BEGIN
        SELECT yearly_budget
        INTO v_yearly_budget
        FROM studio
        WHERE id = v_studio_id;

        RETURN v_yearly_budget;
END;
```

Este procedimiento es propiedad del usuario PROD. El usuario JSMITH debe ejecutar el mismo. ¿Qué clase de privilegios debe otorgar PROD a JSMITH?

A. GRANT EXECUTE ON get_budget TO jsmith;
B. GRANT EXECUTE, SELECT ON studio TO jsmith;
C. GRANT EXECUTE, SELECT ON get_budget TO jsmith;
D. GRANT SELECT ON Studio TO jsmith;
   GRANT EXECUTE ON get_budget TO jsmith;

Cuestión 6

Examine la siguiente función y determine qué conjunto de sentencias invocarán la función de forma correcta dentro de un intérprete como SQL*PLUS.

```
CREATE OR REPLACE FUNCTION get_budget(v_studio_id IN NUMBER)
RETURN number IS
        v_yearly_budget NUMBER;


BEGIN
SELECT yearly_budget
INTO v_yearly_budget
FROM studio
WHERE id = v_studio_id;

        RETURN v_yearly_budget;
END;
```

A. VARIABLE g_yearly_budget NUMBER;
: g_yearly_budget := GET_BUDGET(11);
B. VARIABLE g_yearly_budget NUMBER;
EXECUTE g_yearly_budget := GET_BUDGET(11);
C. VARIABLE g_yearly_budget NUMBER;
EXECUTE :g_yearly_budget := GET_BUDGET(11);
D. VARIABLE :g_yearly_budget NUMBER;
EXECUTE :g_yearly_budget := GET_BUDGET(11);

Cuestión 7

Como consecuencia de una alteración de una tabla concreta, se desea saber qué procedimientos y funciones se han podido ver afectados por dicha alteración. ¿Qué tabla del sistema se ha de consultar para saberlo?

A. USER_STATUS
B. USER_SOURCE
C. USER_OBJECTS
D. USER_CONSTRUCTS

Cuestión 8

Examine el siguiente código correspondiente a un paquete y responda a la pregunta siguiente: ¿Qué frase es verdadera respecto al valor de CURRENT_AVG_COST_TICKET?

```
CREATE OR REPLACE PACKAGE BODY theater_pck IS
current_avg_cost_per_ticket NUMBER := 0;

PROCEDURE find_cpt
(v_movie_id IN NUMBER, v_cost_per_ticket IN OUT NUMBER)IS


BEGIN
IF v_cost_per_ticket > current_avg_cost_per_ticket THEN
SELECT cost_per_ticket
INTO v_cost_per_ticket
FROM gross_receipt
WHERE movie_id = v_movie_id;
END IF;
END find_cpt;

PROCEDURE find_seats_sold
(v_movie_id IN NUMBER DEFAULT 34, v_theater_id IN NUMBER) IS
v_seats_sold gross_receipt.seats_sold%TYPE;
v_budget studio.yearly_budget%TYPE;
BEGIN
SELECT seats_sold
INTO v_seats_sold
FROM gross_receipt
WHERE movie_id = v_movie_id
AND theater_id = v_theater_id;
END find_seats_sold;

BEGIN
current_avg_cost_per_ticket := 8.50;
END theater_pck;
```

A. Es 0 hasta que sea referenciado explícitamente.
B. Se le asigna 8.50 cada vez que el paquete es referenciado.
C. Se le asigna 8.50 únicamente cuando es referenciado explícitamente.
D. Se le asigna 8.50 cuando el paquete es invocado por primera vez dentro de una sesión.

Cuestión 9

Examine el siguiente código correspondiente a un paquete y responda a la pregunta siguiente: ¿Cuál sería la correcta invocación del procedimiento FIND_SEATS_SOLD de dicho paquete si lo queremos hacer directamente desde el prompt de SQL*PLUS?.

```
CREATE OR REPLACE PACKAGE BODY theater_pck IS
        v_total_budget NUMBER;

PROCEDURE find_seats_sold
(v_movie_id IN NUMBER DEFAULT 34, v_theater_id IN NUMBER);
END theater_pck;


CREATE OR REPLACE PACKAGE BODY theater_pck IS
        current_avg_cost_per_ticket NUMBER;

PROCEDURE find_seats_sold
(v_movie_id IN NUMBER DEFAULT 34, v_theater_id IN NUMBER) IS
        v_seats_sold gross_receipt.seats_sold%TYPE;
        v_budget studio.yearly_budget%type;
BEGIN
        SELECT seats_sold
INTO v_seats_sold
FROM gross_receipt
WHERE movie_id = v_movie_id
AND theater_id = v_theater_id;
END find_seats_sold;

FUNCTION get_budget(v_studio_id IN NUMBER)
RETURN number IS
        v_yearly_budget NUMBER;
BEGIN
SELECT yearly_budget
INTO v_yearly_budget
FROM studio
WHERE id = v_studio_id;
        RETURN v_yearly_budget;
END get_budget;

END theater_pck;
```

A. EXECUTE find_seats_sold(500,11);

B. Theater_pck.find_seats_sold(500,11);
C. EXECUTE theater_pck.find_seats_sold(500,11);
D. SELECT find_seats_sold(movie_id, theatre_id) FROM gross_receipt;

Cuestión 10

Examine el siguiente código y seleccione una respuesta como única correcta.

```
CREATE OR REPLACE PACKAGE PROD_PACK IS
        G_TAX_RATE NUMBER := .08;
END PROD_PACK;
```

A. Esta especificación de cabecera de paquete puede existir sin un cuerpo de paquete.
B. Este cuerpo de paquete puede existir sin una especificación de cabecera de paquete.
C. Este cuerpo de paquete no puede existir sin una especificación de cabecera de paquete.
D. Esta especificación de cabecera de paquete no puede existir sin un cuerpo de paquete.

Cuestión 11

Examine el siguiente código y diga qué frase es correcta sobre los procedimientos definidos en esta especificación de cabecera de paquete:

```
CREATE OR REPLACE PACKAGE theater_package IS
PROCEDURE find_cpt (v_movie_id IN NUMBER,
                    v_cost_per_ticket IN OUT NUMBER);
PROCEDURE update_theater (v_name IN VARCHAR2);
PROCEDURE find_seats_sold (v_movie_id IN NUMBER DEFAULT 34,
v_theater_id IN NUMBER);
PROCEDURE add_theater;
END theater_package;
```

A. Todos los procedimientos son construcciones públicas.
B. Todos los procedimientos son construcciones privadas.
C. Cada construcción de procedimiento tiene omitido el código; por tanto, es ilegal.
D. Cada construcción de procedimiento contiene una lista de argumentos; por tanto, es ilegal.

Cuestión 12

Examine este trigger:

```
CREATE OR REPLACE TRIGGER audit_gross_receipt
AFTER DELETE OR UPDATE OF seats_sold, cost_per_ticket ON gross_receipt
BEGIN

{additional code}

END;
```

¿Cuántas veces se ejecutará el cuerpo del trigger por cada invocación?

A. Una vez.
B. Dos veces.
C. 1 vez por cada fila borrada o actualizada.
D. 1 vez por cada fila borrada o se actualice SEATS_SOLD o COST_PER_TICKET.

Cuestión 13

Examine este trigger:

```
CREATE OR REPLACE TRIGGER update_studio
BEFORE UPDATE OF yearly_budget ON STUDIO
FOR EACH ROW
```

¿Qué evento invocará al trigger?

A. Una actualización de la tabla STUDIO.
B. Cualquier modificación de la tabla STUDIO.
C. Una actualización de la columna YEARLY_BUDGET.
D. Una actualización de cualquier columna que no sea YEARLY_BUDGET.

Cuestión 14

Given this PL/SQL block:

```
BEGIN
INSERT INTO employee(salary, last_name, first_name)
VALUES(35000, 'Wagner', 'Madeline');
SAVEPOINT save_a;
INSERT INTO employee(salary, last_name, first_name)
VALUES(40000, 'Southall', 'David');
SAVEPOINT save_b;
DELETE FROM employee
WHERE dept_no = 10;
SAVEPOINT save_c;
INSERT INTO employee(salary, last_name, first_name)
VALUES(25000, 'Brown', 'Bert');
ROLLBACK TO SAVEPOINT save_c;
INSERT INTO employee(salary, last_name, first_name)
VALUE(32000, 'Dean', 'Mike');
ROLLBACK TO SAVEPOINT save_b;
COMMIT;
END;
```

Which two changes to the database will be made permanent? (Choose two.)

A. DELETE FROEM employee WHERE dept_no = 10;
B. INSERT INTO employee(salary, last_name, first_name)
   VALUES(32000, 'Dean', 'Mike');
C. INSERT INTO employee(salary, last_name, first_name)
   VALUES(25000, 'Brown', 'Bert');
D. INSERT INTO employee(salary, last_name, first_name)
   VALUES(40000, 'Southall', 'David');
E. INSERT INTO employee(salary, last_name, first_name)
   VALUES(35000, 'Wagner', 'Madeline');

Cuestión 15

Arrange the events that occur when an explicit cursor is opened and one row is fetched in the appropriate order.

When an explicit cursor is opened and one row is fetched the following events occur:

1. The PL/SQL variables are populated.
2. The active set is identified.
3. The pointer is advanced.
4. A query is executed.
5. The current row data is read.
6. The pointer is positioned before the first row.

A. 1,2,3,4,5,6
B. 5,3,2,4,6,1
C. 2,4,5,3,1,6
D. 4,2,6,3,5,1

Cuestión 16

Which does NOT happen when rows are found using a FETCH statement?

A. The cursor remains open after each fetch.
B. The active set is identified satisfying the search criteria.
C. Output PL/SQL variables are populated with the current row data.
D. The pointer identifying the next row in the active set is advanced.

Cuestión 17

Examine this database trigger:

```
CREATE OR REPLACE TRIGGER update_studio
BEFORE UPDATE OF yearly_budget ON STUDIO
FOR EACH ROW
DECLARE
v_max_budget NUMBER;
BEGIN
SELECT max(yearly_budget)
INTO v_max_budget
FROM studio;
        IF :new.yearly_budget > v_max_budget THEN
:new.yearly_budget := v_max_budget;
        END IF;
END;
```

After creating this database trigger successfully, you test it by updating the YEARLY_BUDGET column to a value greater than the maximum value already in the STUDIO table. What result can you expect?

A. The YEARLY_BUDGET column will be set to the maximun value.
B. The STUDIO table is mutating and therefore, an error is returned.
C. The STUDIO table is mutating and therefore, the trigger execution is ignored.
D. Referencing the NEW qualifier in a BEFORE trigger is illegal and therefore, an error is returned.

Cuestión 18

The script file containing the CHECK_SAL server procedure is lost. This procedure must be modified immediately and recreated. Which data dictionary view can you query to retrieve the source code of this procedure?

A. ALL_SOURCE
B. ALL_PROCEDURES
C. PROCEDURE_SOURCE
D. SERVER_PROCEDURES

Cuestión 19

¿Qué 2 sentencias referidas a la sobrecarga de paquetes son verdaderas?

A. Los subprogramas deben ser locales.
B. Los subprogramas pueden ser locales o remotos.
C. Está permitido exceder el máximo número de subprogramas.
D. Dos subprogramas con el mismo nombre deben diferir solo en el tipo que se devuelve.
E. Dos subprogramas con el mismo nombre y número de parámetros formales deben tener al menos un parámetro definido con el tipo de datos diferente.

Cuestión 20

Examine this database trigger:

```
CREATE OR REPLACE TRIGGER audit_gross_receipt
AFTER DELETE OR UPDATE OF seats_sold, cost_per_ticket ON gross_receipt
BEGIN

{additional code}

END;
```

If a user issues a DELETE statement against the GROSS_RECEIPT table, the procedure, AUDIT_DELETE_GR of the THEATER_PCK package, must be executed once for the entire DELETE statement. Which code, when added, will perform this successfully?

A. IF DELETING THEN theatre_pck.audit_delete_gr;
   END IF;
B. IF CHECK_DELETE THEN theatre_pck.audit_delete_gr;
   END IF;
C. IF DBMS_SQL('DELETE') THEN theatre_pck.audit_delete_gr;
   END IF;
D. IF DMBS_CHECK_MANIPULATION('DELETE') THEN
   theatre_pck.audit_delete_gr;
   END IF;

Cuestión 21

Which data dictionary table can you query to view all the dependencies between the objects that you own?

A. USER_OBJECTS
B. USER_RELATIONS
C. USER_DEPENDENCIES
D. USER_RELATIONSHIPS

Cuestión 22

Examine this function:

```
CREATE OR REPLACE FUNCTION set_budget (v_studio_id IN NUMBER,
                                       v_new_budget IN NUMBER)
RETURN number IS
BEGIN
UPDATE studio
SET yearly_budget = v_new_budget
WHERE id = v_studio_id;
COMMIT;
        RETURN SQL%ROWCOUNT;
END;
```

This function is executed from within a procedure called CALCULATE_BUDGET. The database administrator has just informed you that a new column has been added to the STUDIO table. What affect will this have?

A. Only SET_BUDGET will be marked invalid.
B. Only CALCULATE_BUDGET will be marked invalid.
C. Both SET_BUDGET and CALCULATE_BUDGET will be marked invalid.
D. SET_BUDGET, CALCULATE_BUDGET, and STUDIO will be marked invalid.

Cuestión 23

Which character function would you use to return a specified portion of a character string?

A. CONCAT
B. SUBSTR
C. LENGTH
D. INITCAP

Cuestión 24

You attempt to create the ALPHA_3000 table with this statement:

```
1. CREATE TABLE alpha_3000
2. (3000_id NUMBER(9)
3. CONSTRAINT alpha_3000_id_pk PRIMARY KEY,
4. name VARCHAR2(25),
5. title VARCHAR2(25),
6. idname VARCHAR2(25)
7. CONSTRAINT alpha_3000_idname_nn NOT NULL);
```

Which line in the statement causes a syntax error?

A. 1
B. 2
C. 3
D. 7

Cuestión 25

Evaluate this PL/SQL block:

```
DECLARE
v_result BOOLEAN;
BEGIN
DELETE FROM sale WHERE salesperson_id IN (25, 35, 45);
v_result := SQL%ISOPEN;
COMMIT;
END;
```

What will be the value of V_RESULT if three rows are deleted?

A. 0
B. 3
C. TRUE
D. NULL
E. FALSE

Cuestión 26

Which SELECT statement is an equijoin query between two tables?

A. SELECT region.region_name, employee.salary
FROM region, employee
WHERE region.id = employee.region_no;
B. SELECT region.region_name, employee.salary
FROM region, employee
WHERE region.id = employee.region_no(+);
C. SELECT region.region_name, employee.salary
FROM region, employee
WHERE employee.salary BETWEEN region.avq_salary AND region.max_salary;
D. SELECT region.region_name, employeeinfo.last_name
FROM employee region, employee.employeeinfo
WHERE employeeinfo.id >= region.manager_id;

Cuestión 27

The CUSTOMER table has been created. You attempt to create the SALE table with this command:

```
1. CREATE TABLE sale
2. (purchase_no NUMBER(9),
3. customer_no NUMBER(9)
4. CONSTRAINT sale_customer_id_fk REFERENCES
5. customer (id),
6. CONSTRAINT sale_purchase_no_pk PRIMARY KEY (purchase_no),
7. CONSTRAINT sale_customer_no_nn NOT NULL (customer_no));
```

Which line in the statement will cause an error?

A. 2
B. 3
C. 4
D. 6
E. 7

Cuestión 28

The TRANSACTION table has six columns. Since you often query the table with a join to the SALE table, you created an index on five of the columns in the TRANSACTION table. Which result will occur?

A. Inserts to the table will be slower.
B. The speed of deletes will be increased.
C. The size of the TRANSACTION table will be increased.
D. All queries on the TRANSACTION table will be faster if it does contain a large number of NULL values.

Cuestión 29

You alter the database with this command:

RENAME streets TO city;

Which task is accomplished?

A. The STREETS user is renamed CITY.
B. The STREETS table is renamed CITY.
C. The STREETS column is renamed CITY.
D. The STREETS constraint is renamed CITY.

Cuestión 30

You query the database with this command:

```
SELECT name, salary, dept_id
FROM employee
WHERE salary >
(SELECT AVG(salary)
FROM employee
WHERE dept_no =
(SELECT dept_no
FROM employee
WHERE last_name =
(SELECT last_name
FROM employee
WHERE salary > 50000)));
```

Which SELECT clause is evaluated first?

A. SELECT dept_no
B. SELECT last_name
C. SELECT AVG(salary)
D. SELECT name, salary, dept_id

Cuestión 31

Evaluate this PL/SQL block:

```
BEGIN
        FOR i IN 1..6 LOOP
IF i = 1 THEN
COMMIT;
ELSE
IF i = 3 THEN
ROLLBACK;
ELSE
IF i = 5 THEN
COMMIT;
ELSE
INSERT INTO exam(id)
VALUES (i);
END IF;
END IF;
END IF;
        END LOOP;
        COMMIT;
END;
```

How many values will be inserted into the EXAM table?

A. 0
B. 1
C. 2
D. 3
E. 5
F. 6

Cuestión 32

You query the database with this command:

```
SELECT name
FROM employee
WHERE name LIKE '_a%';
```

Which names are displayed?

A. Names starting with "a".
B. Names starting with "a" or "A".
C. Names containing "a" as second letter.
D. Names containing "a" as any letter except the first.

Cuestión 33

Evaluate this SQL statement:

```
SELECT id, (2 * cost) / (2 * sale_price) + 10 price
FROM product;
```

All of the COST and SALE_PRICE values in the PRODUCT table are greater than one. What would happen if you removed all the parentheses from the calculation?

A. The statement would generate a syntax error.
B. The statement would achieve the same results.
C. The results of the PRICE values would be lower.
D. The results of the PRICE values would be higher.

Cuestión 34

Evaluate this IF statement. You issue this command:

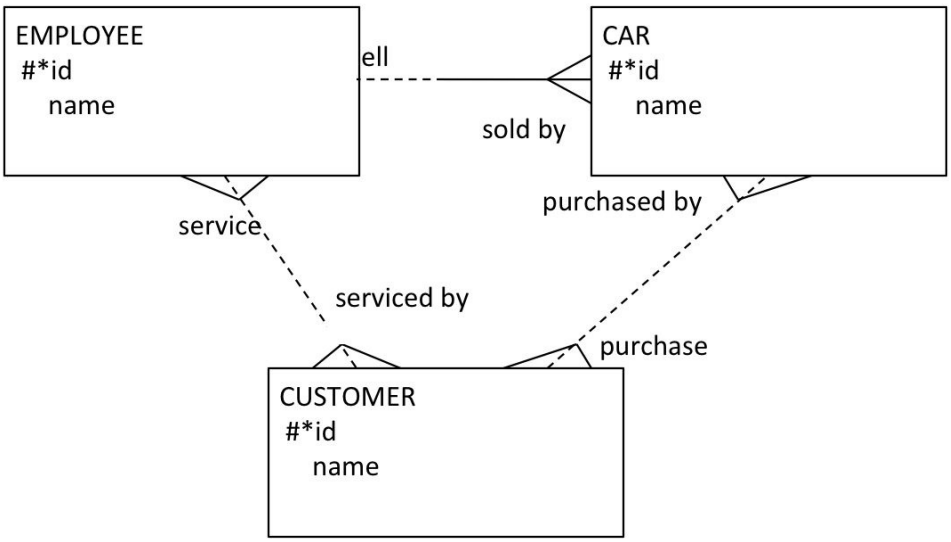
```
GRANT update
ON employee
TO ed
WITH GRANT OPTION;
```

Which task could Ed perform on the EMPLOYEE table?

A. View data.
B. Delete data.
C. Modify constraint.
D. Give update access to other users.

Cuestión 35

Based on the diagram, which relationship is mandatory?



A. Employees sell cars.
B. Customers purchase cars.
C. Cars are sold by employees.
D. Employees service customers.

Cuestión 36

You query the database with this command:

```
SELECT object_name
FROM all_objects
WHERE object_type = 'TABLE';
```

Which values are displayed?

A. Only the names of the tables you own.
B. Only the names of the objects you own.
C. Only the names of all the objects you can access.
D. Only the names of all the tables you can access.

Cuestión 37

What happens when rows are found using a FETCH statement?

A. The cursor opens.
B. The cursor closes.
C. The current row values are loaded into variables.
D. Variables are created to hold the current row values.

Cuestión 38

What is the maximum number of handlers processed before the PL/SQL block is exited when an exception occurs?

A. Only one.
B. All referenced.
C. All that apply.
D. None.

Cuestión 39

Evaluate this SQL script:

```
CREATE ROLE payroll;
CREATE ROLE sales_dept;
CREATE ROLE inventory;
CREATE USER scott IDENTIFIED BY tiger;
GRANT SELECT ON employee TO payroll;
GRANT SELECT ON sale TO sales_dept;
GRANT payroll TO sales_dept;
GRANT sales_dept TO inventory;
GRANT inventory TO scott
/
```

Which tables can user SCOTT query?

A. Only SALE.
B. Only EMPLOYEE.
C. Both SALE and EMPLOYEE.
D. Neither SALE nor EMPLOYEE.

Cuestión 40

You issue this command:

```
SELECT emp_id_seq.CURRVAL
FROM SYS.dual;
```

Which value(s) is displayed?

A. Values of the EMP_ID_SEQ column.
B. Current value of the EMP_ID_SEQ index.
C. Current value of the EMP_ID_SEQ cursor.
D. Current value of the EMP_ID_SEQ sequence.

Cuestión 41

Which program construct must return a value?

A. Package.
B. Function.
C. Anonymous block.
D. Stored procedure.
E. Application procedure.

Cuestión 42

You issue this command:

```
ALTER USER ed IDENTIFIED BY wvu88;
```

Which task has been accomplished?

A. A new user has been added.
B. The user name has been changed.
C. The user password has been changed.
D. A password has been added to the user account.

Cuestión 43

Peter works as a Database Administrator. He create a table named ACCOUNTS that contains the accounts information of the company. Peter wants to validate data automatically, before it is inserted into the ACCOUNTS table. Which of the following will Peter use to accomplish this task?

A. Stored procedure.
B. Anonymous block
C. SELECT statement
D. Trigger

Cuestión 44

You work as an Application Developer in a company. The company uses an Oracle database. The database contains a table names EMPLOYEES. The EMPLOYEES table contains a column of the LONG datatype. You want to change the datatype of the column from LOG to BLOB. What will you do to accomplish this?

A. Use the DBMS_LOG.MIGRATE stored procedure.
B. Use the ALTER DATABASE statement.
C. Use the ALTER TABLE statement.
D. You cannot accomplish the task.

Cuestión 45

David is an employee in Technet Inc. The company uses an Oracle database. David has been granted a role names SALES. David wants to DROP the role. He executes the following statement to accomplish this:

```
DROP ROLE Sales;
```

When he executes the statement, it returns an error. What is the most likely cause of the issue?

Each correct answer represents a complete solution. Choose two.

A. David is not granted the role with the GRANT OPTION.
B. David is not granted the role with de ADMIN OPTION.
C. Only the database administrator can drop the role.
D. David does not have the DROP ANY ROLE system privilege.

Cuestión 46

You work as an Application Developer for Technet Inc. The company uses an Oracle database. The database contains a table names EMPLOYEES. You have defined a database trigger named RAISE_SALARY on the EMPLOYEES table. You want to remove the trigger from the database. Which of the following SQL statements will you use to accomplish this?

A. ALTER TRIGGER Raise_Salary REMOVE;
B. DELETE TRIGGER Raise_Salary;
C. DROP TRIGGER Raise_Salary;
D. REMOVE TRIGGER Raise_Salary;

Cuestión 47

Under which of the following conditions is the use of an explicit cursor necessary?

A. When any SQL data manipulation language (DML) statement is used.
B. When a query returns only one row.
C. When a query does not return any row.
D. When a query returns more than one row.

Cuestión 48

Which of the following statements about a package are true?

Each correct answer represents a complete solution. Choose two.

A. The specification of a package declares the package constructs, whereas the body of the package defines them.
B. A package has three parts.
C. A package itself cannot be called or nested.
D. A package is loaded into the memory every time a package construct is called.

Cuestión 49

Which of the following functions is not available in procedural statements?

A. MOD
B. TRUNC
C. TO_DATE
D. TO_CHAR
E. LOWER
F. DECODE

Cuestión 50

Which of the following is a cursor attribute that yields the number of rows processed by the last DML statement?

A. SQL%ROWRECKON
B. SQL%ROWTALLY
C. SQL%ROWCOUNT
D. SQL%ROWADD

Cuestión 51

Identify whether the given statement is true or false?

"Only the IN parameters can be assigned a default value. The OUT and IN OUT parameters cannot be assigned a default value."

A. False
B. True

Cuestión 52

Which of the following SQL*Plus commands is used to invoke a procedure from iSQL*Plus?

A. RUN
B. EXECUTE
C. CALLL
D. REVOKE
E. START
F. STARTUP

Cuestión 53

Identify whether the given statement is true or false? "A stored procedure cannot be executed as a stand-alone statement."

A. True
B. False

Cuestión 54

Which of the following DBMS_SQL Subprograms combine a given value to a given collection?

A. ADD_ARRAY Procedures
B. UNITE_ARRAY Procedures
C. COMBINE_ARRAY Procedures
D. BIND_ARRAY Procedures

Cuestión 55

Which of the following methods removes one element form the end of a collection?

A. DELETE(m,n)
B. TRIM(n)
C. DELETE(n)
D. DELETE
E. TRIM

Anexo I. RESOLUCIÓN DE SUPUESTOS PRÁCTICOS

SUPUESTO PRÁCTICO 0

Si usted no ha realizado previamente el curso: "Oracle 11g SQL. Curso práctico de formación" de esta misma editorial, tendrá que cargar la estructura de base de datos de un HOSPITAL que se utilizará como ejemplo durante este curso. Para ello, deberá realizar las siguientes operaciones:

- Ejecutar en SQL*PLUS el script script_bdhospital.sql incluido en el anexo III de este libro. También puede encontrarlo en la web de la editorial: http://www.rclibros.es, donde podrá descargarlo y ejecutarlo contra la base de datos.
- Una vez ejecutado este script se habrá cargado toda la configuración del esquema HOSPITAL (tablas, secuencias, datos…) en el usuario PEPERF con contraseña PEPITO.
- Conéctese con el usuario PEPERF y consulte la información de los objetos creados para este usuario dentro de la tabla USER_OBJECTS, cuyo nombre no empiece por 'BIN'.

Resolución del supuesto

La resolución que se ofrece a este supuesto parte de la base de que el fichero script_bdhospital.sql se ha ubicado en la unidad C:\ del disco al que se va a acceder desde SQL*PLUS para su ejecución.

```
C:\> SQLPLUS SYS/CURSOSQL AS SYSDBA;

SQL> START c:\script_bdhospital.sql;
SQL> CONNECT PEPERF/PEPITO;
SQL> SELECT OBJECT_NAME, OBJECT_TYPE FROM USER_OBJECTS
2 > WHERE OBJECT_NAME NOT LIKE 'BIN%';
```

Los resultados que se mostrarían al ejecutar la consulta serían los siguientes:

```
OBJECT_NAME OBJECT_TYPE
------------------------------------------- ----------------
SEQ_INSCRIPCION SEQUENCE
IX1_DEPARTAMENTO INDEX
IX2_ENFERMO INDEX
IX1_ENFERMO INDEX
PK_PERSONAL INDEX
PERSONA TABLE
PK_ESTADOS_CIVILES INDEX
ESTADOS_CIVILES TABLE
DEPT_EMP_PK INDEX
DEPARTAMENTO_EMPLEADO TABLE
DEPT_PK INDEX
DEPARTAMENTO TABLE
EMP_HOSP_PK INDEX
EMPLEADO_HOSPITAL TABLE
DOC_HOSP_PK INDEX
DOCTOR_HOSPITAL TABLE
DOC_PK INDEX
DOCTOR TABLE
HOSP_ENF_PK INDEX
HOSPITAL_ENFERMO TABLE
PLAN_SALA_PK INDEX
PLANTILLA_SALA TABLE
SALA_PK INDEX
SALA TABLE
PLAN_PK INDEX
PLANTILLA TABLE
ENF_PK INDEX
```

ENFERMO TABLE
HOSP_PK INDEX
HOSPITAL TABLE
EMP_PK INDEX
EMPLEADO TABLE

32 filas seleccionadas

Supuesto Práctico 1

Crear un bloque sin nominar en SQL*PLUS que realice las siguientes operaciones:

- Solicitar el siguiente literal por pantalla: *nombre_de_enfermo*, y almacenarlo en una variable del mismo tipo y tamaño que el campo que almacena el nombre en la tabla enfermo.
- Solicitar el siguiente literal por pantalla: *apellidos_del_mismo*, y almacenarlo en una variable del mismo tipo y tamaño que el campo que almacena el apellido en la tabla enfermo.
- Solicitar el siguiente literal por pantalla: *dirección_donde_reside*, y almacenarlo en una variable del mismo tipo y tamaño que el campo que almacena la dirección en la tabla enfermo.
- Una vez introducidos todos estos datos, se deberá ejecutar el siguiente código:

```
DBMS_OUTPUT.PUT_LINE('DATOS DEL ENFERMO');
DBMS_OUTPUT.PUT_LINE ('----------'||CHR(10));
```

- Mostrar en una primera línea el nombre del enfermo introducido por pantalla seguido de una coma (,) y el apellido.
- Mostrar en una segunda línea la dirección del enfermo introducida por pantalla.

```
/* Activación en SQL*PLUS de la variable de
entorno que posibilita ver resultados por
pantalla mediante el paquete DBMS_OUTPUT de
PL/SQL */

SET SERVEROUTPUT ON
```

Resolución del supuesto

```
SET SERVEROUTPUT ON

-- Comienzo del bloque Supuesto 1
DECLARE
V_NOMBRE ENFERMO.NOMBRE%TYPE := '&nombre_de_enfermo';
V_APELLIDOS ENFERMO.APELLIDOS%TYPE := '&apellidos_del_mismo';
V_DIRECCION ENFERMO.DIRECCION%TYPE := &direccion_donde_reside';
BEGIN
DBMS_OUTPUT.PUT_LINE('DATOS DEL ENFERMO');
DBMS_OUTPUT.PUT_LINE('----------------'||CHR(10));
DBMS_OUTPUT.PUT_LINE(V_NOMBRE||','||V_APELLIDOS||CHR(10)||
                                    V_DIRECCION);
END;
/
```

Cuando se ejecuta el código, solicita unos valores por pantalla para satisfacer las variables que se han definido en el bloque sin nominar. Si se introducen los siguientes valores para cada una de ellas:

- &nombre_de_enfermo: ANTOLIN
- &apellidos_del_mismo: MUÑOZ CHAPARRO
- &dirección_donde_reside: MADRID

El resultado que se obtendría sería el siguiente:

```
DATOS DEL ENFERMO
----------------
ANTOLIN,MUÑOZ CHAPARRO
MADRID

Procedimiento PL/SQL terminado correctamente.
```

Supuesto Práctico 2

Crear un bloque sin nominar en SQL*PLUS que realice las siguientes operaciones:

- Solicitar el siguiente literal por pantalla: *nombre_de_enfermo*, y almacenarlo en una variable del mismo tipo y tamaño que el campo que almacena el nombre en la tabla enfermo.
- Solicitar el siguiente literal por pantalla: *apellidos_del_mismo*, y almacenarlo en una variable del mismo tipo y tamaño que el campo que almacena el apellido en la tabla enfermo.
- Solicitar el siguiente literal por pantalla: *dirección_donde_reside*, y almacenarlo en una variable del mismo tipo y tamaño que el campo que almacena la dirección en la tabla enfermo.

- Mostrar un texto con la siguiente información: 'Nº de filas en tabla enfermo antes de insertar =', concatenado con el resultado de contar el número de filas que hay en la tabla enfermo.
- Insertar en la tabla ENFERMO un registro con la siguiente información:

  NUMSEGSOCIAL = 280862486
  NOMBRE= el nombre que se ha insertado por pantalla
  APELLIDOS= los apellidos introducidos por pantalla
  DIRECCION= la que se ha introducido por pantalla
  SEXO=M

- Insertar, mediante un bucle repetitivo, diez líneas en la tabla HOSPITAL_ENFERMO con la siguiente información para el mismo:

HOSP_CODIGO = 6
INSCRIPCION = seq_inscripcion.nextval
ENF_NUMSEGSOCIAL = 280862486
FINSCRIPCION = TO_DATE('01012000','DDMMYYYY') + seq_inscripcion.nextval

- Rechazar las filas insertadas.
- Mostrar un texto con la siguiente información: 'Nº de filas en la tabla enfermo después de insertar y rechazar filas=' concatenado con el resultado de contar el número de filas que hay en la tabla enfermo.

Resolución del supuesto

```
SET SERVEROUTPUT ON
DECLARE
V_NOMBRE    ENFERMO.NOMBRE%TYPE := '&nombre_de_enfermo';
V_APELLIDOS ENFERMO.APELLIDOS%TYPE := '&apellidos_del_mismo';
V_DIRECCION ENFERMO.DIRECCION%TYPE := &direccion_donde_reside';
CUENTA      NUMBER(3);
BEGIN
  SELECT COUNT(*) INTO cuenta FROM enfermo;
  DBMS_OUTPUT.PUT_LINE(CHR(10)||CHR(10)||
         'Nº de filas en tabla enfermo antes de insertar = '||
         cuenta);
     INSERT INTO ENFERMO (NUMSEGSOCIAL,NOMBRE,APELLIDOS,DIRECCION,SEXO)
  VALUES (280862486,v_nombre,v_apellidos,v_direccion,'M');
  FOR I IN 1..10 LOOP
    INSERT INTO HOSPITAL_ENFERMO
         (HOSP_CODIGO, INSCRIPCION, ENF_NUMSEGSOCIAL,
          FINSCRIPCION)
    VALUES (6,seq_inscripcion.nextval,280862486,
                  TO_DATE('01012000','DDMMYYYY') +
                  seq_inscripcion.nextval);
  END LOOP;
  ROLLBACK;
  SELECT COUNT(*) INTO cuenta FROM enfermo;
  DBMS_OUTPUT.PUT_LINE('Nº de filas en tabla enfermo después de insertar y rechazar filas =
'||cuenta);
END;
/
```

Cuando se ejecuta el código, solicita unos valores por pantalla para satisfacer las variables que se han definido en el bloque sin nominar. Si se introducen los siguientes valores para cada una de ellas:

- &nombre_de_enfermo: ANTOLIN
- &apellidos_del_mismo: MUÑOZ CHAPARRO
- &dirección_donde_reside: MADRID

El resultado que se obtendría sería el siguiente:

```
Nº de filas en tabla enfermo antes de insertar = 6
```

Nº de filas en tabla enfermo después de insertar y rechazar filas = 6

Procedimiento PL/SQL terminado correctamente.

Supuesto Práctico 3

1. Realizar una consulta de la tabla DEPARTAMENTO que saque el NOMBRE de todos los departamentos distintos que haya en la misma.

2. Crear un bloque sin nominar en SQL*PLUS que realice las siguientes operaciones:

- Declarar un tipo registro que se componga de los siguientes campos:
  - Un campo que almacene el nombre del departamento y que sea del mismo tipo y tamaño que su homólogo en la tabla departamentos.
  - Un campo que almacene el número de empleados que trabajan en el departamento de tipo numérico de 5 dígitos.
  - Un campo que almacene el tipo de ocupación con formato VARCHAR2(20).
- Declarar un tipo tabla que almacene registros como el que se ha creado en el punto anterior.

- Realizar una consulta que solicite por pantalla el nombre de un departamento y que para el mismo, extraiga el nombre y el número de empleados que trabajan en él. El nombre del departamento se almacenará en el primer campo del registro declarado y el número de empleados en el segundo campo de registro.
- En el tercer campo del registro declarado se almacenarán algunos de los siguientes literales (*BAJA*, *MEDIA*, *ALTA*) dependiendo de las siguientes consideraciones:
  - Si el número de empleados de un departamento es menor que 10, entonces se almacenará la palabra *BAJA*.
  - Si el número de empleados de un departamento se encuentra entre 10 y 19, entonces se almacenará la palabra *MEDIA*.
  - En el resto de casos se almacenará la palabra *ALTA*.
- Después de esto se deberá mostrar la siguiente línea:
  (CHR(10)||CHR(10)||'Datos de ocupación del departamento' concatenado con el nombre del departamento que se ha almacenado en el primer elemento del tipo tabla declarado.

- A continuación, mostrará la siguiente línea:
  ('Ocupación actual (Tipo/nº empleados):' concatenado con el símbolo '/' y por último concatenado con el segundo campo del tipo tabla declarado.

Resolución del supuesto (punto 1)

```
SELECT DISTINCT NOMBRE FROM DEPARTAMENTO;
```

El resultado que se obtendría sería el siguiente:

```
NOMBRE
--------------------
ADMINISTRACION
INVESTIGACION
CONTABILIDAD
FACTURACION
FARMACIA
LIMPIEZA

6 filas seleccionadas
```

Resolución del supuesto (punto 2)

```
SET SERVEROUTPUT ON
DECLARE
TYPE R_Ocupacion_Departamentos IS RECORD
(Nombre_dpto DEPARTAMENTO.NOMBRE%TYPE,
Ocupacion NUMBER(5),
Tipo_Ocupacion VARCHAR2(20));

TYPE T_Ocupacion_Departamento IS TABLE OF
        R_Ocupacion_Departamentos
INDEX BY BINARY_INTEGER;

V_Ocupacion T_Ocupacion_Departamento;
BEGIN
SELECT D.NOMBRE, COUNT(E.EMP_NIF)
INTO V_Ocupacion(1).Nombre_dpto, V_Ocupacion(1).Ocupacion
FROM DEPARTAMENTO D, DEPARTAMENTO_EMPLEADO E
WHERE UPPER(D.NOMBRE) = UPPER('&Indique_departamento')
AND E.DEPT_CODIGO = D.CODIGO
GROUP BY D.NOMBRE;

IF V_Ocupacion(1).Ocupacion < 10 THEN
V_Ocupacion(1).Tipo_Ocupacion := 'BAJA';
ELSIF V_Ocupacion(1).Ocupacion BETWEEN 10 AND 19 THEN
V_Ocupacion(1).Tipo_Ocupacion := 'MEDIA';
ELSE
V_Ocupacion(1).Tipo_Ocupacion := 'ALTA';
END IF;

DBMS_OUTPUT.PUT_LINE(CHR(10)||CHR(10)||
        'Datos de ocupación del departamento '||
        V_Ocupacion(1).Nombre_dpto);
DBMS_OUTPUT.PUT_LINE('-------------------------------------------------'||CHR(10)||CHR(10));
DBMS_OUTPUT.PUT_LINE('Ocupación actual Tipo/nºempleados):
'||V_Ocupacion(1).Tipo_Ocupacion||'/'||
V_Ocupacion(1).Ocupacion);

END;
/
```

El resultado que se obtendría sería el siguiente si introducimos para la variable &Indique_departamento el valor: ADMINISTRACION.

Datos de ocupación del departamento ADMINISTRACION

Ocupación actual (Tipo/nºempleados): BAJA/7

Procedimiento PL/SQL terminado correctamente.

Supuesto Práctico 4

Crear un bloque sin nominar en SQL*PLUS que realice las siguientes operaciones:

- Se necesita incluir a todos los doctores dentro de la tabla de plantilla sanitaria para conseguir una estructura de información más lógica. Para realizar dicha operación, se deberá definir un cursor explícito que recupere, mediante una consulta sobre la tabla DOCTOR_HOSPITAL, todas las filas y columnas que tenga.
- Para insertar los doctores en la tabla PLANTILLA_SALA, se necesita un dato fundamental que no posee la tabla DOCTOR: el código de sala. Para solucionar este problema, se van a insertar todos los doctores en la tabla PLANTILLA_SALA con el mismo número de sala: 99.
- Para poder insertar información en la tabla PLANTILLA_SALA, aparte del código del hospital y de la sala, se necesita el nif del doctor en cuestión que previamente deberá haberse insertado en la tabla PLANTILLA. Antes de insertarlo en esta última comprobaremos la existencia de dicho NIF en la tabla. En caso de existir no se realizará la inserción. Si es necesario hacer la inserción, los datos a introducir serán los siguientes:

NIF = el nif del doctor
SALARIO = 0
NOMBRE = el nombre del doctor
TURNO = M
APELLIDOS = el apellido del doctor
FUNCION = la especialidad del doctor

- También hay que indicar que el código de sala 99 deberá previamente existir en la tabla SALA para el hospital concreto del doctor, por lo que antes de insertar el doctor en la tabla PLANTILLA_SALA, se deberá comprobar la existencia de una sala 99 para el hospital al que pertenece el doctor en la tabla SALA. En caso de que no exista dicha información habrá de insertarse una nueva fila en la tabla sala con la siguiente información:
  HOSP_CODIGO = código del hospital al que pertenece el doctor
  CODIGO = 99
  NUMCAMAS = 0

NOMBRE = 'SALA DOCTORES'

- Cuando se necesite insertar en la tabla SALA, después de la inserción, se deberá mostrar por pantalla un mensaje con el siguiente formato ('Se ha introducido una sala 99 para el hospital'|| código del hospital al que pertenece el doctor que se ha insertado).
- Una vez que se hayan insertado todos los doctores en la tabla plantilla se deberá mostrar por pantalla un mensaje con el siguiente formato: (CHR(10)||'Se han insertado '||número efectivo de doctores insertados||' doctores en la tabla de empleados sanitarios de '||número posible de doctores a insertar||' posibles.').
- Se confirmarán las inserciones realizadas.
- Tras estas operaciones queremos visualizar en pantalla un listado con el nombre, apellidos y función de cada uno de los empleados sanitarios. Este listado se va a visualizar dentro del propio bloque mediante una consulta sobre la tabla correspondiente, pero para poderlo conseguir antes habrá de crearse un tipo tabla que almacene registros de 3 campos (nombre, apellidos y función) con el mismo tamaño y tipo que sus homólogos en la tabla de plantilla sanitaria. Y además un cursor que seleccione la información de la tabla de plantilla sanitaria y que se incluya dentro de cada celda del tipo tabla.
- Previo al propio listado se deberá mostrar una línea con el siguiente formato: (CHR(10)||'Se van a visualizar '||número de filas que posee la tabla de plantilla sanitaria||' empleados sanitarios.').
- Como línea de encabezado del listado se mostrará la siguiente: ('NOMBRE'||CHR(9)||'APELLIDOS'||CHR(9)||'FUNCIÓN').
- Para el correcto visionado del listado se deberá hacer por cada fila a visualizar una comprobación previa: comprobar la longitud del apellido de la persona. Si la longitud es = 4, se almacenará en una variable de tipo texto dos tabulaciones CHR(9). En caso contrario una sola.
- Cada línea mostrada seguirá el siguiente formato: Los 6 primeros caracteres del nombre||CHR(9)||los apellidos||las tabulaciones correspondientes del punto j||la función).

Resolución del supuesto

```
SET SERVEROUTPUT ON

DECLARE
V_DOCTOR_HOSPITAL DOCTOR_HOSPITAL%ROWTYPE;
CONTADOR NUMBER;
CONTAINSERTA NUMBER := 0;
TEXTO VARCHAR2(255);
TYPE R_DOCTOR IS RECORD
(NOMBRE DOCTOR.NOMBRE%TYPE,
APELLIDOS DOCTOR.APELLIDOS%TYPE,
ESPECIALIDAD DOCTOR.ESPECIALIDAD%TYPE);
V_DOCTOR R_DOCTOR;
TYPE R_PLANTILLA IS RECORD
(NOMBRE PLANTILLA.NOMBRE%TYPE,
APELLIDOS PLANTILLA.APELLIDOS%TYPE,
FUNCION PLANTILLA.FUNCION%TYPE);
TYPE T_PLANTILLA IS TABLE OF R_PLANTILLA
INDEX BY BINARY_INTEGER;
V_PLANTILLA T_PLANTILLA;
CURSOR C_DOCTOR_HOSPITAL IS
SELECT * FROM DOCTOR_HOSPITAL;
CURSOR C_PLANTILLA IS
SELECT DISTINCT NOMBRE, APELLIDOS, FUNCION
FROM PLANTILLA;
BEGIN
OPEN C_DOCTOR_HOSPITAL;
LOOP
FETCH C_DOCTOR_HOSPITAL INTO V_DOCTOR_HOSPITAL;
EXIT WHEN C_DOCTOR_HOSPITAL%NOTFOUND;

SELECT COUNT(*) INTO CONTADOR FROM SALA
WHERE HOSP_CODIGO = V_DOCTOR_HOSPITAL.HOSP_CODIGO
AND CODIGO = 99;

IF CONTADOR = 0 THEN
INSERT INTO SALA VALUES
        (V_DOCTOR_HOSPITAL.HOSP_CODIGO,99,0,
                'SALA DOCTORES');
DBMS_OUTPUT.PUT_LINE('Se ha introducido una sala 99 para el hospital
'||V_DOCTOR_HOSPITAL.HOSP_CODIGO);
END IF;
```

```
        SELECT COUNT(*) INTO CONTADOR FROM PLANTILLA
WHERE NIF = V_DOCTOR_HOSPITAL.DOC_NIF;

IF CONTADOR = 0 THEN
SELECT NOMBRE, APELLIDOS, ESPECIALIDAD
        INTO V_DOCTOR.NOMBRE, V_DOCTOR.APELLIDOS,
        V_DOCTOR.ESPECIALIDAD
        FROM DOCTOR
WHERE NIF = V_DOCTOR_HOSPITAL.DOC_NIF;

INSERT INTO PLANTILLA VALUES
                (V_DOCTOR_HOSPITAL.DOC_NIF, 0, V_DOCTOR.NOMBRE,
                'M', V_DOCTOR.APELLIDOS, V_DOCTOR.ESPECIALIDAD);
CONTAINSERTA := CONTAINSERTA + 1;
END IF;
END LOOP;
DBMS_OUTPUT.PUT_LINE(CHR(10)||
'Se han insertado '||CONTAINSERTA||
' doctores en la tabla de empleados sanitarios de '||C_DOCTOR_HOSPITAL%ROWCOUNT||'
posibles.');
COMMIT;
CLOSE C_DOCTOR_HOSPITAL;

SELECT COUNT(*) INTO CONTADOR FROM PLANTILLA;
DBMS_OUTPUT.PUT_LINE(CHR(10)||'Se van a visualizar '||CONTADOR||' empleados sanitarios.');
DBMS_OUTPUT.PUT_LINE('-----------------------------------------------------'||CHR(10)||CHR(10));
DBMS_OUTPUT.PUT_LINE('NOMBRE'||CHR(9)||'APELLIDOS'||
        CHR(9)||'FUNCIÓN');
DBMS_OUTPUT.PUT_LINE('-----------------------------------------------------');
CONTADOR := 1;
OPEN C_PLANTILLA;
LOOP
FETCH C_PLANTILLA INTO V_PLANTILLA(CONTADOR);
EXIT WHEN C_PLANTILLA%NOTFOUND;
SELECT DECODE(LENGTH(V_PLANTILLA(CONTADOR).APELLIDOS),4,
                        CHR(9)||CHR(9),CHR(9))
INTO TEXTO FROM DUAL;
DBMS_OUTPUT.PUT_LINE(SUBSTR(V_PLANTILLA(CONTADOR).NOMBRE,1,6)||CHR(9)||V_PLAN
V_PLANTILLA(CONTADOR).FUNCION);
CONTADOR := CONTADOR + 1;
END LOOP;
CLOSE C_PLANTILLA;
END;
/
```

El resultado que se obtendría sería el siguiente:

Se ha introducido una sala 99 para el hospital 1
Se ha introducido una sala 99 para el hospital 2
Se ha introducido una sala 99 para el hospital 3
Se ha introducido una sala 99 para el hospital 4
Se ha introducido una sala 99 para el hospital 5
Se ha introducido una sala 99 para el hospital 6
Se ha introducido una sala 99 para el hospital 7

Se han insertado 33 doctores en la tabla de empleados sanitarios de 36 posibles.

Se van a visualizar 48 empleados sanitarios.

NOMBRE APELLIDOS FUNCIÓN

Amelia A.A. ENFERMERA
Bony B.B. ENFERMERA
Iñaqui Gutierrez T. Cardiologia
Ines Acaso B. Ginecologia
Toñin Torres B. Ginecologia
Carlos C.C. ENFERMERO
Bibian B.B. ENFERMERA
Casild C.C. ENFERMERA
Benici B.B. ENFERMERO
Carlos Lopez T. Cardiologia
Bartol Casas B. Ginecologia
Iñaqui Lopez T. Cardiologia
Jorge Fermin J. Cardiologia
Fede Fermin J. Cardiologia
Loles Lopez T. Cardiologia
Rocio Fernandez J. Cardiologia
JuanMa Lopez T. Cardiologia
Albert A.A. ENFERMERO
Bonifa B.B. ENFERMERO
Casand C.C. ENFERMERA
Cicero C.C. ENFERMERO
Samuel Lopez T. Cardiologia
Maria Acaso B. Ginecologia
Maica Acaso B. Ginecologia
Casimi C.C. ENFERMERO
Maria Moreno D. Pediatria
Isidor Moreno D. Pediatria
Bartol Torres B. Ginecologia
Loli Fernandez J. Cardiologia
Tirano Torres B. Ginecologia
Armand A.A. ENFERMERO
Juan Torres B. Ginecologia
Alejan A.A. ENFERMERO
Adrian A.A. ENFERMERA
Ines Soledad B. Ginecologia

Antoni Moreno D. Pediatria
Ramiro Del Toro D. Psiquiatria
Edmunt Fermin J. Cardiologia
Ramon Fernandez J. Cardiologia
Bartol B.B. ENFERMERO
Raimun Gutierrez J. Cardiologia
Perico Gutierrez J. Cardiologia
Rosa Moreno D. Pediatria

Procedimiento PL/SQL terminado correctamente.

Supuesto Práctico 5

Crear un bloque sin nominar en SQL*PLUS que realice las siguientes operaciones:

- Realizar, mediante un bucle repetitivo, una inserción de 6 registros sobre la tabla DEPARTAMENTO_EMPLEADO, con la siguiente información:
  EMP_NIF = 10000000A
  DEPT_CODIGO = el valor de la variable que controla el bucle.

  En este bucle se controlará que cuando el código de departamento sea el 4, no se realice inserción en la tabla.
- Confirmar las inserciones realizadas.
- Diseñar un cursor que, para un código de departamento que se habrá de introducir por pantalla, seleccione los nombres de los empleados que trabajan en los mismos. Será necesario acceder a las tablas DEPARTAMENTO_EMPLEADO y EMPLEADO.
- Diseñar un tipo tabla para almacenar el nombre de todos aquellos empleados que se recuperan mediante el cursor indicado en el punto anterior.
- Si no se encuentra ningún empleado en el departamento que se indique por pantalla, habrá de mostrarse la siguiente línea: (CHR(10)||'No se recuperó ningún empleado en el departamento con código '||código del departamento introducido); y terminará el programa.
- En caso contrario se mostrará una línea con la siguiente información (CHR(10)||'Se han encontrado '||número de empleados de la tabla empleados que pertenecen al departamento con código introducido||' empleados pertenecientes al departamento '||nombre del departamento cuyo código se ha introducido);.
- Además, habrá de mostrarse debajo todos los nombres que se han almacenado en el tipo tabla.
- Por último, habrán de realizarse 2 actualizaciones:
  - Sobre la tabla EMPLEADO (empleados no sanitarios) se actualizarán las columnas salario y comisión a valor 0 para cada uno de los nombres

almacenados en el tipo tabla.

- Sobre la tabla PLANTILLA (empleados sanitarios), se actualizará la columna salario a valor 0 para cada uno de los nombres coincidentes en el tipo tabla. En este último caso, se deberá controlar mediante un cursor implícito la posibilidad de que no existan empleados que coincidan en el nombre almacenado en el tipo tabla, en este último caso se mostrará la siguiente línea: (CHR(10)||'No se encontraron empleados sanitarios con el nombre de '||nombre almacenado en la celda correspondiente del tipo tabla). Para el caso de que si exista, se mostrará el siguiente mensaje: (CHR(10)||'Se han actualizado '||número de filas que se actualizan||' empleados sanitarios con el nombre de '||nombre almacenado en la celda correspondiente al tipo tabla.

- En el caso de que se ejecute el punto anterior, rechazar las actualizaciones que se hayan producido.

Resolución del supuesto

```
SET SERVEROUTPUT ON
DECLARE
contador NUMBER := 0;
SUBTYPE nombre IS empleado.nombre%TYPE;
nombre_empleado nombre;
nombre_dpto nombre;
v_codigo departamento.codigo%TYPE;
TYPE t_nombre IS TABLE OF nombre INDEX BY BINARY_INTEGER;
v_nombre t_nombre;
CURSOR c_departamento (valor departamento.codigo%TYPE)
IS select nombre from departamento_empleado, empleado
where dept_codigo = v_codigo
and nif = emp_nif;

BEGIN
FOR I IN 1..6 LOOP
IF I != 4 THEN
INSERT INTO DEPARTAMENTO_EMPLEADO
                    VALUES (I,'10000000A');
END IF;
END LOOP;
COMMIT;

v_codigo := &Codigo_departamento;
OPEN c_departamento(v_codigo);
LOOP
FETCH c_departamento INTO nombre_empleado;
EXIT WHEN c_departamento%NOTFOUND;
contador := contador + 1;
v_nombre(contador) := nombre_empleado;
END LOOP;
IF c_departamento%ROWCOUNT < 1 THEN
DBMS_OUTPUT.PUT_LINE(CHR(10)||'No se recuperó ningun empleado en el departamento con
código '||v_codigo);
ELSE
SELECT COUNT(*) INTO contador FROM DEPARTAMENTO_EMPLEADO
WHERE DEPT_CODIGO = v_codigo;

SELECT NOMBRE INTO nombre_dpto FROM DEPARTAMENTO
WHERE CODIGO = v_codigo;
DBMS_OUTPUT.PUT_LINE(CHR(10)||'Se han encontrado '||contador||' empleados pertenecientes al
departamento '||nombre_dpto);
```

```
DBMS_OUTPUT.PUT_LINE(CHR(10)||'Relación de empleados del departamento '||nombre_dpto||'.');
DBMS_OUTPUT.PUT_LINE(CHR(10)||'----------------------------------------------'||CHR(10));

FOR indice IN v_nombre.FIRST..v_nombre.LAST LOOP
IF v_nombre.EXISTS(indice) THEN
DBMS_OUTPUT.PUT_LINE(v_nombre(indice));
END IF;
END LOOP;

DBMS_OUTPUT.PUT_LINE(CHR(10));
FOR indice IN v_nombre.FIRST..v_nombre.LAST LOOP
IF v_nombre.EXISTS(indice) THEN
UPDATE EMPLEADO
SET SALARIO = 0
, COMISION = 0
WHERE NOMBRE = v_nombre(indice);

UPDATE PLANTILLA
SET SALARIO = 0
WHERE NOMBRE = v_nombre(indice);

IF SQL%NOTFOUND THEN
DBMS_OUTPUT.PUT_LINE(CHR(10)||'No se encontraron empleados sanitarios con el nombre de '||v_nombre(indice));
ELSE
DBMS_OUTPUT.PUT_LINE(CHR(10)||'Se han actualizado '||SQL%ROWCOUNT||' empleados sanitarios con el nombre de '||v_nombre(indice));
END IF;
END IF;
END LOOP;
ROLLBACK;
END IF;
CLOSE c_departamento;
END;
/
```

El resultado que se obtendría sería el siguiente en caso de utilizar el valor 1 como departamento asociado a la variable &Codigo_departamento que se solicita durante la ejecución del bloque PL/SQL:

```
Se han encontrado 4 empleados pertenecientes al departamento CONTABILIDAD

Relación de empleados del departamento CONTABILIDAD.
----------------------------------------------

Jorge
```

Carlos
Lola
Angel

No se encontraron empleados sanitarios con el nombre de Jorge

Se han actualizado 1 empleados sanitarios con el nombre de Carlos

No se encontraron empleados sanitarios con el nombre de Lola

No se encontraron empleados sanitarios con el nombre de Angel

Procedimiento PL/SQL terminado correctamente.

Supuesto Práctico 6

1. Diseñar un procedimiento que convierta el salario anual que existe en la tabla plantilla en un salario mensual, teniendo en cuenta que los miembros de la tabla plantilla solo cobran 12 pagas. Para el diseño de este procedimiento, se seguirán estas pautas:
    - Solo se podrá actualizar el importe a aquellos miembros de la tabla plantilla cuyo salario anual sea igual o superior al máximo que haya en la tabla plantilla.
    - La actualización no se hará a todos los miembros de la tabla plantilla cada vez, sino a aquellos que comiencen por una letra determinada o en caso contrario pertenezcan a un turno concreto. Estos datos serán pasados al procedimiento mediante parámetros.
    - El procedimiento devolverá el total de filas actualizadas o 0 en caso de que no se actualice ninguna.
    - Se mostrarán los siguientes mensajes dependiendo de que se realice una u otra acción:
        - **Si se actualizan filas por la letra de comienzo del nombre**: 'Se han actualizado '||nº de filas actualizadas||' filas, correspondientes a la plantilla sanitaria cuya letra comienza por la letra '||la letra que se le ha pasado al procedimiento.
        - **Si se actualizan filas por el turno**: 'Se han actualizado '||nº de filas actualizadas||' filas, correspondientes a la plantilla sanitaria cuyo turno es '||el turno que se le pasó al procedimiento.
        - **Si no se actualizan filas porque no se cumple ninguna de las condiciones anteriores de acuerdo con los datos de la tabla plantilla**: 'No se ha podido actualizar ningún empleado de la plantilla sanitaria cuya letra comience por '||la letra que se le pasó al procedimiento y el mensaje
        - **En caso contrario**: 'No se ha podido actualizar ningún empleado de la plantilla sanitaria cuya turno sea '||el turno que se le pasó al procedimiento.

2. Diseñar un bloque sin nominar en SQL*PLUS que realice la llamada al procedimiento creado anteriormente pasándole como parámetros los siguientes valores:

   LETRA = 'B'
   TURNO = 'T'

Resolución del supuesto (punto 1)

```
CREATE OR REPLACE PROCEDURE supuesto6 (letra IN CHAR,
el_turno IN CHAR,
filas OUT NUMBER)
IS
texto_turno VARCHAR2(20);
BEGIN
SELECT DECODE(el_turno,'T','Tarde',
                        'N','Noche',
                        'M','Mañana',
                        'Desconodido')
INTO texto_turno FROM DUAL;
/* Atención, aunque un cursor implícito SQL no permite solucionar el problema de que un SELECT no devuelva filas
o devuelva más de una fila, sí que cuenta el número de filas que selecciona cuando no da error, es decir, cuando un SELECT devuelve 1 fila => SQL%ROWCOUNT = 1 */



UPDATE plantilla a
SET a.salario = a.salario / 12
WHERE Upper(a.nombre) like upper(letra)||'%'
AND a.salario >= (SELECT MAX(b.salario) FROM plantilla b);

IF SQL%NOTFOUND THEN
UPDATE plantilla a
SET a.salario = a.salario / 12
WHERE Upper(a.turno) = upper(el_turno)
AND a.salario >= (SELECT MAX(b.salario)
                        FROM plantilla b);

IF SQL%NOTFOUND THEN
filas := 0;
DBMS_OUTPUT.PUT_LINE('No se ha podido actualizar ningun empleado de la plantilla sanitaria cuya letra comience por '||letra);
DBMS_OUTPUT.PUT_LINE('No se ha podido actualizar ningun empleado de la plantilla sanitaria cuya turno sea '||texto_turno);
ELSE
filas := SQL%ROWCOUNT;
DBMS_OUTPUT.PUT_LINE('Se han actualizado '||SQL%ROWCOUNT||' filas, correspondientes a la plantilla sanitaria cuyo turno es '||texto_turno);
END IF;
ELSE
filas := SQL%ROWCOUNT;
```

```
DBMS_OUTPUT.PUT_LINE('Se han actualizado '||SQL%ROWCOUNT||' filas, correspondientes a la plantilla sanitaria cuya letra comienza por la letra '||letra);
END IF;
ROLLBACK;
END supuesto6;
```

Resolución del supuesto (punto 2)

```
SET SERVEROUTPUT ON
DECLARE
num_empleados NUMBER(2);
BEGIN
SUPUESTO6('B','T',num_empleados);
END;
```

El resultado que se obtendría sería el siguiente:

```
Se han actualizado 5 filas, correspondientes a la plantilla sanitaria cuya letra comienza por la letra B

Procedimiento PL/SQL terminado correctamente.
```

Supuesto Práctico 7

1. Diseñar una función recursiva para el cálculo del factorial de un número:
   - Solo admitirá valores enteros.
   - En caso de introducir un valor negativo, deberá mostrar el siguiente mensaje 'No se consideran valores negativos para el cálculo del factorial'. Además, la función devolverá como valor -1.
   - En el resto de los casos devolverá el resultado del cálculo del factorial.
   - El máximo número que se admitirá para el cálculo del factorial será el 33. Números superiores a este se rechazarán con el siguiente mensaje: 'No se consideran valores SUPERIORES a 33 para el cálculo del factorial' y la función devolverá valor 0.
2. Para probar la función, se pueden utilizar los 3 métodos posibles:
   a) Mediante un consulta SELECT.
   b) Ejecutándose mediante un bloque sin nominar con declaración de variables privadas.
   c) Ejecutándose mediante un bloque sin nominar con declaración de variables globales.

Resolución del supuesto (punto 1)

```
CREATE OR REPLACE FUNCTION factorial (n INTEGER) RETURN INTEGER IS
BEGIN
IF n < 0 THEN
DBMS_OUTPUT.PUT_LINE('No se consideran valores negativos para el cálculo del factorial');
RETURN -1;
ELSE

IF n > 33 THEN
DBMS_OUTPUT.PUT_LINE('No se consideran valores SUPERIORES a 33 para el cálculo del factorial');
RETURN 0;
ELSE
IF (n = 1) OR (n = 0) THEN
RETURN 1;
ELSE
RETURN n * factorial (n - 1);
END IF;
END IF;
END IF;
END factorial;
```

Resolución del supuesto (punto 2a)

```
SELECT FACTORIAL(10) FROM DUAL;
```

El resultado que se obtendría sería el siguiente:

```
FACTORIAL(10)
----------
3628800
```

Resolución del supuesto (punto 2b)

```
DECLARE
I INTEGER;
BEGIN
I := FACTORIAL(34);
END;
/
```

Resolución del supuesto (punto 2c)

```
variable i number;
/
begin
:i := factorial(-1);
end;
/
print i;
/
```

El resultado que se obtendría sería el siguiente:

```
I
--
-1

Procedimiento PL/SQL terminado correctamente.
```

Supuesto Práctico 8

Crear un paquete (cabecera y cuerpo) denominado SUPUESTO8 que incluya en su interior la funcionalidad capaz de realizar las operaciones siguientes en funciones o procedimientos separados:

- Calcular el factorial de un número.
- Dado un número en pesetas convertirlo en euros.
- Dado un número en euros convertirlo a pesetas.
- Dado un número en dígitos (de un máximo de 7 dígitos y 3 decimales) convertirlo a carácter con los separadores de miles y decimales correspondientes, teniendo en cuenta lo siguiente:
  - **9**: Indica que se sustituirá por un dígito.
  - **G**: Indica que se sustituirá por el separador de miles.
  - **D**: Indica que se sustituirá por el separador decimal.

  Ejemplo:

  Dado 5325.33 el paso a carácter sería TO_CHAR(5325.33, '9G999D99') con resultado 5.325,33.
- Indicar una variable constante con el valor de conversión de pesetas a euros, y viceversa: 166.386.

  Nota: En las conversiones se utilizará el redondeo y para ello se utilizará la función ROUND que opera de la siguiente manera: ROUND (numero_a_redondear, dígitos_decimales_del_resultado).

  Ej: ROUND (166.386,2) = 166.39

Resolución del supuesto

```
    create or replace package supuesto8 is
function factorial(n POSITIVE) return INTEGER;
function ptas_euros(n NUMBER) return NUMBER;
function euros_ptas(n NUMBER) return NUMBER;
function num_carac (n NUMBER) return VARCHAR2;
conversion CONSTANT NUMBER(6,3) := 166.386;
end supuesto8;
/

create or replace package body supuesto8 is
FUNCTION factorial (n POSITIVE) RETURN INTEGER IS
BEGIN
IF (n = 1) or (n = 0)THEN
RETURN 1;
ELSE
RETURN n * factorial (n - 1);
END IF;
END factorial;

FUNCTION ptas_euros (n NUMBER) RETURN NUMBER IS
BEGIN
RETURN ROUND(n / conversion,2);
END ptas_euros;

FUNCTION euros_ptas (n NUMBER) RETURN NUMBER IS
BEGIN
RETURN ROUND(n * conversion,0);
END euros_ptas;

FUNCTION num_carac(n NUMBER) RETURN VARCHAR2 IS
BEGIN
RETURN TO_CHAR(n,'9G999G999D999');
END num_carac;
END supuesto8;
```

Supuesto Práctico 9

Crear un paquete (cabecera y cuerpo) denominado DEVUELVEREGISTRO, que incluya en su interior, función y un procedimiento para realizar las siguientes operaciones:

- La función devolverá como resultado un elemento con formato tipo tabla. Para ello, consultará todos los miembros de plantilla sanitaria cuyo código de hospital coincida con el que se le haya pasado como parámetro de entrada a la función. Cada miembro se irá almacenando con toda su información (el registro o fila entera) dentro de una celda distinta del tipo tabla. Y una vez recorrido todo el cursor se devolverá el tipo tabla entero.
- Un procedimiento que admita como parámetro de entrada un código de hospital y que con el mismo ejecute la función anterior. El resultado de la función (un tipo tabla entero) lo manejará este procedimiento con objeto de listar el nombre y apellidos de todos los miembros de la plantilla que pertenecían al hospital indicado.
- Para ejecutar este paquete, se ejecutará el procedimiento anterior indicando el código del hospital que se quiere consultar para obtener la lista de su plantilla sanitaria.

Resolución del supuesto

```
CREATE OR REPLACE PACKAGE DevuelveRegistro IS

TYPE t_registros IS TABLE OF PLANTILLA%ROWTYPE
INDEX BY BINARY_INTEGER;

FUNCTION PLANTILLA_POR_HOSPITAL(CODIGO IN NUMBER)
                RETURN t_registros;
PROCEDURE VER_PLANTILLA_HOSPITAL(CODIGO IN NUMBER);

END DevuelveRegistro;
/


CREATE OR REPLACE PACKAGE BODY DevuelveRegistro IS

FUNCTION PLANTILLA_POR_HOSPITAL(CODIGO IN NUMBER)
RETURN T_REGISTROS IS

CURSOR C_PLANTILLA IS
SELECT A.* FROM PLANTILLA A, PLANTILLA_SALA S
        WHERE S.SALA_HOSP_CODIGO = 1
        AND A.NIF = S.PLAN_NIF;

V_REGISTROS T_REGISTROS;
BEGIN
OPEN C_PLANTILLA;
LOOP
FETCH C_PLANTILLA INTO
V_REGISTROS(C_PLANTILLA%ROWCOUNT);
EXIT WHEN C_PLANTILLA%NOTFOUND;
END LOOP;
CLOSE C_PLANTILLA;

RETURN V_REGISTROS;
END PLANTILLA_POR_HOSPITAL;

PROCEDURE VER_PLANTILLA_HOSPITAL(codigo IN NUMBER) IS
V_TABLA T_REGISTROS;
BEGIN
V_TABLA := PLANTILLA_POR_HOSPITAL(CODIGO);
FOR I IN V_TABLA.FIRST..V_TABLA.LAST LOOP
DBMS_OUTPUT.PUT_LINE(V_TABLA(I).NIF||
                                        '-'||V_TABLA(I).NOMBRE);
END LOOP;
```

```
END VER_PLANTILLA_HOSPITAL;
END DevuelveRegistro;
/
```

Supuesto Práctico 10

Comprobar el funcionamiento de un TRIGGER ejecutando un código dado (omitiendo las instrucciones correspondientes a los comentarios):

```
SQL> /* Creamos una tabla denominada ESTUDIANTE */
SQL> CREATE TABLE ESTUDIANTE
2 (NOMBRE VARCHAR2(10));

Tabla creada.

SQL> /* Creamos una tabla denominada SUCESOS */
SQL> CREATE TABLE SUCESOS
2 (NOMBRE VARCHAR2(50),
3 EN_TABLA VARCHAR2(10),
4 DIA DATE);

Tabla creada.

SQL> /* Creamos un trigger asociado a la tabla estudiante
2 que se dispara cuando se produce una operación de INSERT
3 sobre dicha tabla y que tiene como misión insertar una fila */
SQL> CREATE OR REPLACE TRIGGER GRABA_SUCESO
2 BEFORE INSERT ON ESTUDIANTE
3 BEGIN
4 INSERT INTO SUCESOS VALUES ('TRIGGER','ESTUDIANTE',SYSDATE);
5 END GRABA_SUCESO;
6 /

Disparador creado.

SQL> /* Seleccionamos todas las filas de la tabla ESTUDIANTE */
SQL> SELECT * FROM ESTUDIANTE;

ninguna fila seleccionada

SQL> /* Seleccionamos todas las filas de la tabla SUCESOS */
SQL> SELECT * FROM SUCESOS;

ninguna fila seleccionada

SQL> /* Realizamos un select sobre la tabla del sistema USER_OBJECTS
2 que nos devuelve el nombre, tipo y estado del trigger cuyo nombre
3 empieza por GRA para ver en que estado se encuentran */
SQL> SELECT OBJECT_NAME, OBJECT_TYPE, STATUS FROM USER_OBJECTS
2 WHERE OBJECT_NAME LIKE 'GRA%';
```

```
OBJECT_NAME
-------------------------------------------------------------OBJECT_TYPE STATUS
------------ -------
GRABA_SUCESO
TRIGGER VALID

SQL> /* Insertamos una fila en la tabla ESTUDIANTE lo cual provoca que
2 se dispare nuestro trigger GRABA_SUCESO */


SQL> INSERT INTO ESTUDIANTE VALUES ('PEPITO');

1 fila creada.

SQL> /* Seleccionamos todas las filas de la tabla ESTUDIANTE para comprobar
2 que se ha ejecutado el INSERT anterior. */
SQL> SELECT * FROM ESTUDIANTE;

NOMBRE
----------
PEPITO
SQL> /* Seleccionamos todas las filas de la tabla SUCESOS para comprobar
2 que se ha disparado nuestro trigger GRABA_SUCESO */
SQL> SELECT * FROM SUCESOS;

NOMBRE EN_TABLA
DIA
---------------------------------------------- ----------
TRIGGER ESTUDIANTE
22/11/00

SQL> /* Nos arrepentimos y deshacemos el INSERT sobre la tabla ESTUDIANTES
2 ¿que ocurre entonces con la tabla SUCESOS y su información generada
3 por el trigger GRABA_SUCESO? */
SQL> ROLLBACK;

Rollback terminado.

SQL> /* Seleccionamos todas las filas de la tabla ESTUDIANTE y vemos que tras
2 el ROLLBACK volvemos al estado inicial que era una tabla sin filas */
SQL> SELECT * FROM ESTUDIANTE;

ninguna fila seleccionada

SQL> /* El ROLLBACK deshace cualquier operación de actualización realizada o sea
2 que también deja la tabla SUCESOS como estaba al principio, sin filas
3 para ello ejecutamos la select que devuelve todos los datos y vemos que
4 está vacía */
```

```
SQL> SELECT * FROM SUCESOS;

ninguna fila seleccionada

SQL> /* Volvemos a ejecutar la operación de inserción sobre tabla ESTUDIANTE */
SQL> INSERT INTO ESTUDIANTE VALUES ('PEPITO');

1 fila creada.

SQL> /* Confirmamos la inserción y la hacemos definitiva */
SQL> COMMIT;

Validación terminada.

SQL> /* Consultamos las filas de la tabla ESTUDIANTE */
SQL> SELECT * FROM ESTUDIANTE;

NOMBRE
----------
PEPITO

SQL> /* Consultamos las filas de la tabla SUCESOS y vemos que se ha ejecutado
2 el trigger */
SQL> SELECT * FROM SUCESOS;

NOMBRE EN_TABLA
DIA
-------------------------------------------------- ----------
TRIGGER ESTUDIANTE
22/11/00

SQL> /* Borramos la tabla ESTUDIANTE */
SQL> DROP TABLE ESTUDIANTE;

Tabla borrada.

SQL> /* Consultamos la tabla SUCESOS para ver si su estado ha variado al borrar
2 la tabla ESTUDIANTE y vemos que no ha variado */
SQL> SELECT * FROM SUCESOS;



NOMBRE EN_TABLA
DIA
-------------------------------------------------- ----------
TRIGGER ESTUDIANTE
22/11/00

SQL> /* Consultamos la tabla del sistema USER_OBJECTS para ver que ha ocurrido
2 con el trigger al borrar la tabla sobre la que estaba asociado el mismo. El
```

```
3 resultado es que se ha borrado también porque un trigger depende de una tabla
4 y si la misma se borra, se eliminan automáticamente todos los elementos asociados
5 a ella: Indices, Vistas y Triggers */
SQL> SELECT * FROM USER_OBJECTS WHERE OBJECT_NAME LIKE 'GR%';

ninguna fila seleccionada

SQL> /* Borramos la tabla sucesos */
SQL> drop table sucesos;

Tabla borrada.

SQL> /* Fin del ejemplo de utilización de TRIGGERS */
```

Supuesto Práctico 11

Comprobar el funcionamiento de un TRIGGER ejecutando un código dado (omitiendo las instrucciones correspondientes a los comentarios):

```
SQL> /* Creamos una tabla estudiantes */
SQL> CREATE TABLE estudiantes
2 (ID NUMBER PRIMARY KEY,
3 NOMBRE VARCHAR2(25),
4 APELLIDOS VARCHAR2(30));

Table created.

SQL> /* Creamos una secuencia */
SQL> CREATE SEQUENCE contador
2 START WITH 1
3 INCREMENT BY 1
4 NOCACHE;
Sequence created.

SQL> CREATE OR REPLACE TRIGGER GeneraID
2 BEFORE INSERT OR UPDATE ON estudiantes
3 FOR EACH ROW
4 BEGIN
5 SELECT CONTADOR.NEXTVAL INTO :NEW.ID FROM DUAL;
6 END;
7 /

Trigger created.

SQL> /* Hacemos una primera inserción en la tabla estudiantes */
SQL> INSERT INTO estudiantes VALUES(10,'Juan','Gabriel Sanchez');

1 row created.


SQL> /* Vemos que se ha insertado en realidad en tabla ESTUDIANTES */
SQL> SELECT * FROM estudiantes;

ID NOMBRE APELLIDOS
--------- ------------------------ ----------------------
1 Juan Gabriel Sanchez

SQL> /* Realizamos otro insert en tabla ESTUDIANTES */
SQL> INSERT INTO estudiantes (NOMBRE, APELLIDOS) VALUES ('Pepe','Domingo Castro');
```

```
1 row created.

SQL> /* Vemos que se ha insertado */
SQL> SELECT * FROM estudiantes;

ID NOMBRE APELLIDOS
--------- ------------------------ ----------------------
1 Juan Gabriel Sanchez
2 Pepe Domingo Castro

SQL> /* Como se puede ver por las 2 inserciones el trigger lo que hace
2> es dar un valor para el campo ID (el valor del siguiente número de la secuencia)
3>independientemente de que se inserte o no un valor para dicho campo */
SQL> /* Fin de la ejecución del trigger */
```

Supuesto Práctico 12

Diseñar un trigger que haga las funciones de la restricción de integridad referencial ON UPDATE CASCADE, que no implemente directamente el lenguaje SQL de Oracle. Se creará este trigger sobre la tabla *departamento*, de manera que cuando se actualice algún código de departamento, se actualicen todas las filas de la tabla empleado que contengan dicho código.

- Además, por cada operación de manipulación que se lleve a cabo en la tabla departamento y que afecte a la columna código de departamento se producirá una inserción en la tabla HISTO_CAMBIOS_DEPARTAMENTO cuya estructura se define más abajo y que habrá de crearse previamente. Los datos a introducir en dicha tabla por cada operación que se produzca son los siguientes:
- En caso de producirse una inserción en la tabla departamento, habrá de introducirse un registro en la tabla HISTO_CAMBIOS_DEPARTAMENTO con los siguientes valores:

  FECHAYHORA = *la fecha y hora actual*
  TEXTO= 'Se ha insertado una fila con datos: '||*nuevo código de departamento introducido*||'-'|| *nuevo nombre del departamento*

- En caso de producirse un borrado en la tabla departamento, habrá de comprobarse si existen datos de empleados que pertenezcan a ese departamento, en cuyo caso se introducirá un registro en la tabla HISTO_CAMBIOS_DEPARTAMENTO con los siguientes valores:

  FECHAYHORA = *la fecha y hora actual*
  TEXTO= 'Se ha intentado borrar el departamento: '|| *código de departamento que se intenta borrar*

  En caso de que no existan datos de empleados para ese departamento, se introducirá un registro en la tabla HISTO_CAMBIOS_DEPARTAMENTO con los siguientes valores:

  FECHAYHORA = *la fecha y hora actual*
  TEXTO= 'Se ha borrado el departamento: '|| *código de departamento que se intenta borrar*

- En caso de producirse una actualización en la tabla departamento, habrá de comprobarse si existen datos de empleados que pertenezcan a ese departamento, en cuyo caso primero se actualizará el departamento a dichos

empleados.

- Tanto si existen datos de empleados pertenecientes a ese departamento como si no, se introducirá un registro en la tabla HISTO_CAMBIOS_DEPARTAMENTO con los siguientes valores:

FECHAYHORA = *la fecha y hora actual*

TEXTO= 'Se ha actualizado el departamento: '|| *código del departamento antes de que se actualizase*||' al valor '|| *nuevo código del departamento.*

HISTO_CAMBIOS_DEPARTAMENTO
DIAYHORA DATE,
TEXTO VARCHAR2(500)
PRIMARY KEY(diayhora)

Resolución del supuesto

```
-- Creación de la tabla histo_cambios_departamento
CREATE TABLE HISTO_CAMBIOS_DEPARTAMENTO
(DIAYHORA DATE,
TEXTO VARCHAR2(500),
CONSTRAINT pk_histo_cambios_departamento PRIMARY KEY(diayhora));
/

-- Creación del trigger tr_departamento
CREATE OR REPLACE TRIGGER tr_departamento
BEFORE DELETE OR INSERT OR UPDATE OF codigo ON departamento
FOR EACH ROW
DECLARE
        Contador NUMBER;
BEGIN
IF INSERTING THEN
INSERT INTO histo_cambios_departamento
VALUES(sysdate,'Se ha insertado una fila con datos: '||TO_CHAR(:new.codigo)||'-'||:new.nombre);
ELSIF DELETING THEN
SELECT COUNT(*) INTO contador
                FROM departamento_empleado
WHERE dept_codigo = :old.codigo;
IF contador > 0 THEN
INSERT INTO histo_cambios_departamento
VALUES (sysdate,'Se ha intentado borrar el departamento: '||TO_CHAR(:old.codigo));
ELSE
INSERT INTO histo_cambios_departamento
VALUES (sysdate,'Se ha borrado el departamento: '||TO_CHAR(:old.codigo));
END IF;
ELSE
SELECT COUNT(*) INTO contador
                FROM departamento_empleado
WHERE dept_codigo = :old.codigo;

IF contador > 0 THEN
UPDATE departamento_empleado
SET dept_codigo = :new.codigo
WHERE dept_codigo = :old.codigo;
END IF;
INSERT INTO histo_cambios_departamento
VALUES (sysdate,'Se ha actualizado el departamento '||TO_CHAR(:old.codigo)||' al valor
'||TO_CHAR(:new.codigo));
END IF;
END;
```

/

Supuesto Práctico 13

1. Diseñar un procedimiento que sea capaz de dividir 2 números que se soliciten como argumentos del procedimiento. En dicho bloque queremos realizar las siguientes operaciones:
   - Realizar la operación de división.
   - Mostrar el resultado de la división de la siguiente manera: 'El resultado de la división es: '||<resultado>.
   - Controlar los supuestos de error que puedan producirse en este bloque utilizando las excepciones por defecto que se han aprendido en el curso (al menos hay que controlar 1 supuesto de error que se puede producir).
   - En caso de producirse un error se deberá mostrar lo siguiente:

     - Un texto que diga: 'Número de error ORACLE: '||<el código de error de Oracle>.
     - Un texto que diga: 'El mensaje de error ORACLE es: '||<el mensaje de error de Oracle>
     - El texto que diga: 'El valor del dividendo es: '||<valor>
     - El texto que diga: 'El valor del divisor es: '||<valor>
     - Controlar mediante una excepción creada por el usuario en el bloque, la posibilidad de que se introduzca valores negativos para el dividendo o el divisor, mostrando el siguiente mensaje: 'Los valores introducidos para el dividendo: '||<dividendo>||' o el divisor: '|| <divisor>||' son negativos.'

2. Ejecutar el procedimiento con la instrucción EXECUTE para los siguientes ejemplos:
   a) Dividendo = -1, divisor = -5
   b) Dividendo = 15, divisor = 0
   c) Dividendo = 32.759, divisor = 28
   d) Dividendo = 5.879, divisor = -4

Resolución del supuesto (punto 1)

```
CREATE OR REPLACE PROCEDURE DIVISION (dividendo IN NUMBER,
                                      divisor IN NUMBER)
IS
Resultado NUMBER;
Valores_negativos EXCEPTION;
BEGIN
If (dividendo < 0) or (divisor < 0) then
RAISE valores_negativos;
Else
Resultado := dividendo / divisor;
End if;
DBMS_OUTPUT.PUT_LINE('El resultado de la división es: '||resultado);
EXCEPTION
WHEN ZERO_DIVIDE THEN
DBMS_OUTPUT.PUT_LINE('Número de error ORACLE: '||sqlcode);
DBMS_OUTPUT.PUT_LINE('El mensaje de error ORACLE es: '||sqlerrm);
DBMS_OUTPUT.PUT_LINE('El valor del dividendo es: '||dividendo);
DBMS_OUTPUT.PUT_LINE('El valor del divisor es: '||divisor);
WHEN valores_negativos THEN
DBMS_OUTPUT.PUT_LINE('Los valores introducidos para el dividendo: '||dividendo||' o divisor:
'||divisor||' son negativos.');
END;
```

Resolución del supuesto (punto 2a)

```
SET SERVEROUTPUT ON
EXECUTE DIVISION(-1,-5);
```

El resultado que se obtendría sería el siguiente:

```
Los valores introducidos para el dividendo: -1 o divisor: -5 son negativos.
```

Resolución del supuesto (punto 2b)

```
SET SERVEROUTPUT ON
EXECUTE DIVISION(15,0);
```

El resultado que se obtendría sería el siguiente:

```
Número de error ORACLE: -1476
El mensaje de error ORACLE es: ORA-01476: divisor is equal to zero
El valor del dividendo es: 15
El valor del divisor es: 0
```

Resolución del supuesto (punto 2c)

```
SET SERVEROUTPUT ON
EXECUTE DIVISION(32759,28);
```

El resultado que se obtendría sería el siguiente:

```
El resultado de la división es: 1169,964285714285714285714285714285714286
```

Resolución del supuesto (punto 2d)

```
SET SERVEROUTPUT ON
EXECUTE DIVISION(5879,-4);
```

El resultado que se obtendría sería el siguiente:

Los valores introducidos para el dividendo: 5879 o divisor: -4 son negativos.

Supuesto Práctico 14

1. Diseñar un procedimiento que tome como único argumento un código de hospital y que realice las siguientes operaciones:
   - Detecte mediante un control de errores la no existencia de enfermos en dicho hospital y que muestre un mensaje de notificación de dicha situación.
   - Detecte mediante un control de errores la existencia de más de un enfermo en dicho hospital, y en ese caso muestre el número total de enfermos que hay en el mismo, mediante un mensaje.

2. Probar la ejecución del procedimiento para los siguientes supuestos:

3.
   a) Código de hospital = 1
   b) Código de hospital = 6
   c) Código de hospital = 7

Resolución del supuesto (punto 1)

```
CREATE OR REPLACE PROCEDURE BUSCA_ENFERMOS(codigo IN NUMBER)
IS
Cuenta NUMBER;
V_nombre enfermo.nombre%TYPE;
V_apellidos enfermo.apellidos%TYPE;
BEGIN
SELECT NOMBRE, APELLIDOS INTO v_nombre, v_apellidos
FROM ENFERMO, HOSPITAL_ENFERMO
WHERE HOSP_CODIGO = CODIGO
AND NUMSEGSOCIAL = ENF_NUMSEGSOCIAL;
DBMS_OUTPUT.PUT_LINE('El único enfermo del hospital '||TO_CHAR(CODIGO)||' es '||v_nombre||','||v_apellidos);
EXCEPTION
WHEN NO_DATA_FOUND THEN
DBMS_OUTPUT.PUT_LINE('No hay enfermos en el hospital con código '||TO_CHAR(codigo));
WHEN TOO_MANY_ROWS THEN
SELECT COUNT(DISTINCT ENF_NUMSEGSOCIAL) INTO cuenta
        FROM HOSPITAL_ENFERMO
WHERE HOSP_CODIGO = codigo;
DBMS_OUTPUT.PUT_LINE('El hospital con código '||TO_CHAR(codigo)||' tiene '||TO_CHAR(cuenta)||' enfermos.');
END;
```

Resolución del supuesto (punto 2a)

```
SET SERVEROUTPUT ON
EXECUTE busca_enfermos(1);
```

El resultado que se obtendría sería el siguiente:

```
El hospital con código 1 tiene 6 enfermos.
```

Resolución del supuesto (punto 2b)

```
SET SERVEROUTPUT ON
EXECUTE busca_enfermos(6);
```

El resultado que se obtendría sería el siguiente:

```
El único enfermo del hospital 6 es Beatriz,A.A.
```

Resolución del supuesto (punto 2c)

```
SET SERVEROUTPUT ON
EXECUTE busca_enfermos(7);
```

El resultado que se obtendría sería el siguiente:

```
No hay enfermos en el hospital con código 7
```

Supuesto Práctico 15

1. Diseñar un procedimiento que sea capaz mediante un cursor del tipo FOR UPDATE, de actualizar única y exclusivamente el campo SALARIO de la tabla plantilla sanitaria de todos aquellos doctores (para determinar si es un doctor se deberá comparar las tablas plantilla sanitaria y doctores).
   - El nuevo salario que se deberá asignar a todos los doctores de la tabla plantilla sanitaria será el máximo que exista en dicha tabla.
2. Ejecutar una consulta que nos muestre el nombre y salario de todos los doctores de la tabla plantilla (comparándolos con la tabla doctores) y observar el salario de todos ellos antes de actualizarlo.
3. Ejecutar el procedimiento creado en el punto 1.
4. Volver a ejecutar la consulta del punto 2 y comprobar que se han actualizado los salarios.

Resolución del supuesto (punto 1)

```
CREATE OR REPLACE PROCEDURE EQUIPARAR_SALARIOS IS
V_salariomax plantilla.salario%TYPE;
CURSOR mi_cursor IS
SELECT * FROM plantilla
WHERE EXISTS (SELECT NULL FROM doctor
WHERE doctor.nif = plantilla.nif
AND doctor.especialidad =
plantilla.funcion)
FOR UPDATE OF salario;
BEGIN
FOR v_empleado_sanitario IN mi_cursor LOOP
SELECT max(salario) INTO v_salariomax
FROM plantilla;

UPDATE plantilla
SET salario = v_salariomax
WHERE CURRENT OF mi_cursor;
END LOOP;
COMMIT;
END equiparar_salarios;
```

Resolución del supuesto (punto 2)

```
SELECT NOMBRE, SALARIO FROM plantilla
WHERE EXISTS (SELECT NULL FROM DOCTOR
WHERE DOCTOR.NIF = PLANTILLA.NIF
AND DOCTOR.ESPECIALIDAD =
PLANTILLA.FUNCION);
```

El resultado que se obtendría sería el siguiente:

```
NOMBRE SALARIO
-------------------- ----------------------
Raimundo 0
Perico 0
Iñaqui 0
Ines 0
Bartolome 0
Rosa 0
Maria 0
Isidoro 0
Antonio 0
Ramiro 0
Edmunto 0
Iñaqui 0
Ines 0
Bartolome 0
Rosa 0
Loli 0
Jorge 0
Samuel 0
Maria 0
Tirano 0
Ramon 0
Fede 0
Loles 0
Maica 0
Toñin 0
Ramon 0
Fede 0
Loles 0
Maica 0
Rocio 0
Carlos 0
Juan 0
JuanMa 0
```

33 filas seleccionadas

Resolución del supuesto (punto 3)

```
EXECUTE EQUIPARAR_SALARIOS;
```

Resolución del supuesto (punto 4)

```
SELECT NOMBRE, SALARIO FROM plantilla
WHERE EXISTS (SELECT NULL FROM DOCTOR
WHERE DOCTOR.NIF = PLANTILLA.NIF
AND DOCTOR.ESPECIALIDAD =
PLANTILLA.FUNCION);
```

El resultado que se obtendría sería el siguiente:

```
NOMBRE SALARIO
-------------------- ----------------------
Raimundo 15000,22
Perico 15000,22
Iñaqui 15000,22
Ines 15000,22
Bartolome 15000,22
Rosa 15000,22
Maria 15000,22
Isidoro 15000,22
Antonio 15000,22
Ramiro 15000,22
Edmunto 15000,22
Iñaqui 15000,22
Ines 15000,22
Bartolome 15000,22
Rosa 15000,22
Loli 15000,22
Jorge 15000,22
Samuel 15000,22
Maria 15000,22
Tirano 15000,22
Ramon 15000,22
Fede 15000,22
Loles 15000,22
Maica 15000,22
Toñin 15000,22
Ramon 15000,22
Fede 15000,22
Loles 15000,22
Maica 15000,22
Rocio 15000,22
Carlos 15000,22
Juan 15000,22
JuanMa 15000,22
```

33 filas seleccionadas

Anexo II. RESOLUCIÓN DE CUESTIONES DE CERTIFICACIÓN

CUESTIÓN 1

¿Cuántas veces se ejecutará la sentencia de inserción dentro de la ejecución del FOR LOOP del siguiente bloque de PL/SQL?

```
DECLARE
        v_lower NUMBER := 2;
        v_upper NUMBER := 100;
        v_count NUMBER := 1;
BEGIN
        FOR i IN v_lower..v_lower LOOP
                INSERT INTO test(results)
        VALUES(v_count);
        v_count := v_count + 1;
        END LOOP;
END;
```

A. 0
B. 1
C. 2
D. 98
E. 100

CUESTIÓN 2

Evalúe el siguiente código PL/SQL y responda a la pregunta: ¿cuál será el valor de V_RESULT si los 3 registros son borrados?

```
DECLARE
        V_result BOOLEAN;
BEGIN
        DELETE FROM sale
        WHERE salesperson_id IN (25,35,45);
        V_result := SQL%ISOPEN;
        COMMIT;
END;
```

A. 0
B. 3
C. TRUE
D. NULL
E. FALSE

CUESTIÓN 3

Dado el siguiente procedimiento, si V_BONUS = TRUE y V_RAISE = NULL, ¿qué valor se asigna a V_ISSUE_CHECK?

```
PROCEDURE dept_salary ( v_bonus IN BOOLEAN,
        v_raise IN BOOLEAN,
        v_issue_check IN OUT BOOLEAN)
IS
BEGIN
        V_issue_check := v_bonus OR v_raise;
END;
```

A. TRUE
B. FALSE
C. NULL
D. Ninguna de las anteriores

CUESTIÓN 4

Los procedimientos y funciones son muy similares. ¿Qué razón nos hace elegir crear una función en vez de un procedimiento, en caso de que podamos hacer ambos?

A. Una función devuelve 1 valor.
B. Una función se puede usar en una sentencia SQL.
C. Una función solo se puede ejecutar desde un procedimiento creado previamente.
D. Una función garantiza que solo se consulta información, mientras que un procedimiento no garantiza esto.

CUESTIÓN 5

Examine la siguiente función PL/SQL.

```
CREATE OR REPLACE FUNCTION get_budget(v_studio_id IN NUMBER)
RETURN number IS
        v_yearly_budget NUMBER;
BEGIN
        SELECT yearly_budget
        INTO v_yearly_budget
        FROM studio
        WHERE id = v_studio_id;

        RETURN v_yearly_budget;
END;
```

Este procedimiento es propiedad del usuario PROD. El usuario JSMITH debe ejecutar el mismo. ¿Qué clase de privilegios debe otorgar PROD a JSMITH?

A. GRANT EXECUTE ON get_budget TO jsmith;
B. GRANT EXECUTE, SELECT ON studio TO jsmith;
C. GRANT EXECUTE, SELECT ON get_budget TO jsmith;
D. GRANT SELECT ON Studio TO jsmith;
GRANT EXECUTE ON get_budget TO jsmith;

Cuestión 6

Examine la siguiente función y determine qué conjunto de sentencias invocarán la función de forma correcta dentro de un intérprete como SQL*PLUS.

```
CREATE OR REPLACE FUNCTION get_budget(v_studio_id IN NUMBER)
RETURN number IS
        v_yearly_budget NUMBER;
BEGIN
SELECT yearly_budget
INTO v_yearly_budget
FROM studio
WHERE id = v_studio_id;

        RETURN v_yearly_budget;
END;
```

A. VARIABLE g_yearly_budget NUMBER;
: g_yearly_budget := GET_BUDGET(11);
B. VARIABLE g_yearly_budget NUMBER;
EXECUTE g_yearly_budget := GET_BUDGET(11);
C. VARIABLE g_yearly_budget NUMBER;
EXECUTE :g_yearly_budget := GET_BUDGET(11);
D. VARIABLE :g_yearly_budget NUMBER;
EXECUTE :g_yearly_budget := GET_BUDGET(11);

Cuestión 7

Como consecuencia de una alteración de una tabla concreta, se desea saber qué procedimientos y funciones se han podido ver afectados por dicha alteración. ¿Qué tabla del sistema se ha de consultar para saberlo?

A. USER_STATUS
B. USER_SOURCE
C. USER_OBJECTS
D. USER_CONSTRUCTS

Cuestión 8

Examine el siguiente código correspondiente a un paquete y responda a la pregunta siguiente: ¿Qué frase es verdadera respecto al valor de CURRENT_AVG_COST_TICKET?

```
CREATE OR REPLACE PACKAGE BODY theater_pck IS
current_avg_cost_per_ticket NUMBER := 0;

PROCEDURE find_cpt
(v_movie_id IN NUMBER, v_cost_per_ticket IN OUT NUMBER)
IS
BEGIN
IF v_cost_per_ticket > current_avg_cost_per_ticket THEN
SELECT cost_per_ticket
INTO v_cost_per_ticket
FROM gross_receipt
WHERE movie_id = v_movie_id;
END IF;
END find_cpt;

PROCEDURE find_seats_sold
(v_movie_id IN NUMBER DEFAULT 34, v_theater_id IN NUMBER) IS
v_seats_sold gross_receipt.seats_sold%TYPE;
v_budget studio.yearly_budget%TYPE;
BEGIN
SELECT seats_sold
INTO v_seats_sold
FROM gross_receipt
WHERE movie_id = v_movie_id
AND theater_id = v_theater_id;
END find_seats_sold;

BEGIN
current_avg_cost_per_ticket := 8.50;
END theater_pck;
```

A. Es 0 hasta que sea referenciado explícitamente.
B. Se le asigna 8.50 cada vez que el paquete es referenciado.
C. Se le asigna 8.50 únicamente cuando es referenciado explícitamente.
D. Se le asigna 8.50 cuando el paquete es invocado por primera vez dentro de una sesión.

Cuestión 9

Examine el siguiente código correspondiente a un paquete y responda a la pregunta siguiente: ¿Cuál sería la correcta invocación del procedimiento FIND_SEATS_SOLD de dicho paquete si lo queremos hacer directamente desde el prompt de SQL*PLUS?.

```
CREATE OR REPLACE PACKAGE BODY theater_pck IS
        v_total_budget NUMBER;

PROCEDURE find_seats_sold
(v_movie_id IN NUMBER DEFAULT 34, v_theater_id IN NUMBER);
END theater_pck;


CREATE OR REPLACE PACKAGE BODY theater_pck IS
        current_avg_cost_per_ticket NUMBER;

PROCEDURE find_seats_sold
(v_movie_id IN NUMBER DEFAULT 34, v_theater_id IN NUMBER) IS
        v_seats_sold gross_receipt.seats_sold%TYPE;
        v_budget studio.yearly_budget%type;
BEGIN
        SELECT seats_sold
INTO v_seats_sold
FROM gross_receipt
WHERE movie_id = v_movie_id
AND theater_id = v_theater_id;
END find_seats_sold;

FUNCTION get_budget(v_studio_id IN NUMBER)
RETURN number IS
        v_yearly_budget NUMBER;
BEGIN
SELECT yearly_budget
INTO v_yearly_budget
FROM studio
WHERE id = v_studio_id;
        RETURN v_yearly_budget;
END get_budget;

END theater_pck;
```

A. EXECUTE find_seats_sold(500,11);
B. Theater_pck.find_seats_sold(500,11);

C. EXECUTE theater_pck.find_seats_sold(500,11);
D. SELECT find_seats_sold(movie_id, theatre_id) FROM gross_receipt;

Cuestión 10

Examine el siguiente código y seleccione una respuesta como única correcta.

```
CREATE OR REPLACE PACKAGE PROD_PACK IS
        G_TAX_RATE NUMBER := .08;
END PROD_PACK;
```

A. Esta especificación de cabecera de paquete puede existir sin un cuerpo de paquete.
B. Este cuerpo de paquete puede existir sin una especificación de cabecera de paquete.
C. Este cuerpo de paquete no puede existir sin una especificación de cabecera de paquete.
D. Esta especificación de cabecera de paquete no puede existir sin un cuerpo de paquete.

Cuestión 11

Examine el siguiente código y diga qué frase es correcta sobre los procedimientos definidos en esta especificación de cabecera de paquete:

```
CREATE OR REPLACE PACKAGE theater_package IS
PROCEDURE find_cpt (v_movie_id IN NUMBER,
                    v_cost_per_ticket IN OUT NUMBER);
PROCEDURE update_theater (v_name IN VARCHAR2);
PROCEDURE find_seats_sold (v_movie_id IN NUMBER DEFAULT 34,
v_theater_id IN NUMBER);
PROCEDURE add_theater;
END theater_package;
```

A. Todos los procedimientos son construcciones públicas.
B. Todos los procedimientos son construcciones privadas.
C. Cada construcción de procedimiento tiene omitido el código; por tanto, es ilegal.
D. Cada construcción de procedimiento contiene una lista de argumentos; por tanto, es ilegal.

Cuestión 12

Examine este trigger:

```
CREATE OR REPLACE TRIGGER audit_gross_receipt
AFTER DELETE OR UPDATE OF seats_sold, cost_per_ticket ON gross_receipt
BEGIN

{additional code}

END;
```

¿Cuántas veces se ejecutará el cuerpo del trigger por cada invocación?

A. Una vez.
B. Dos veces.
C. 1 vez por cada fila borrada o actualizada.
D. 1 vez por cada fila borrada o se actualice SEATS_SOLD o COST_PER_TICKET.

Cuestión 13

Examine este trigger:

```
CREATE OR REPLACE TRIGGER update_studio
BEFORE UPDATE OF yearly_budget ON STUDIO
FOR EACH ROW
```

¿Qué evento invocará al trigger?

A. Una actualización de la tabla STUDIO.
B. Cualquier modificación de la tabla STUDIO.
C. Una actualización de la columna YEARLY_BUDGET.
D. Una actualización de cualquier columna que no sea YEARLY_BUDGET.

Cuestión 14

Given this PL/SQL block:

```
BEGIN
INSERT INTO employee(salary, last_name, first_name)
VALUES(35000, 'Wagner', 'Madeline');
SAVEPOINT save_a;
INSERT INTO employee(salary, last_name, first_name)
VALUES(40000, 'Southall', 'David');
SAVEPOINT save_b;
DELETE FROM employee
WHERE dept_no = 10;
SAVEPOINT save_c;
INSERT INTO employee(salary, last_name, first_name)
VALUES(25000, 'Brown', 'Bert');
ROLLBACK TO SAVEPOINT save_c;
INSERT INTO employee(salary, last_name, first_name)
VALUE(32000, 'Dean', 'Mike');
ROLLBACK TO SAVEPOINT save_b;
COMMIT;
END;
```

Which two changes to the database will be made permanent? (Choose two.)

A. DELETE FROEM employee WHERE dept_no = 10;
B. INSERT INTO employee(salary, last_name, first_name)
   VALUES(32000, 'Dean', 'Mike');
C. INSERT INTO employee(salary, last_name, first_name)
   VALUES(25000, 'Brown', 'Bert');
D. INSERT INTO employee(salary, last_name, first_name)
   VALUES(40000, 'Southall', 'David');
E. INSERT INTO employee(salary, last_name, first_name)
   VALUES(35000, 'Wagner', 'Madeline');

Cuestión 15

Arrange the events that occur when an explicit cursor is opened and one row is fetched in the appropriate order.

When an explicit cursor is opened and one row is fetched the following events occur:

1. The PL/SQL variables are populated.
2. The active set is identified.
3. The pointer is advanced.
4. A query is executed.
5. The current row data is read.
6. The pointer is positioned before the first row.

A. 1,2,3,4,5,6
B. 5,3,2,4,6,1
C. 2,4,5,3,1,6
D. 4,2,6,3,5,1

Cuestión 16

Which does NOT happen when rows are found using a FETCH statement?

A. The cursor remains open after each fetch.
B. The active set is identified satisfying the search criteria.
C. Output PL/SQL variables are populated with the current row data.
D. The pointer identifying the next row in the active set is advanced.

Cuestión 17

Examine this database trigger:

```
CREATE OR REPLACE TRIGGER update_studio
BEFORE UPDATE OF yearly_budget ON STUDIO
FOR EACH ROW
DECLARE
v_max_budget NUMBER;
BEGIN
SELECT max(yearly_budget)
INTO v_max_budget
FROM studio;
        IF :new.yearly_budget > v_max_budget THEN
:new.yearly_budget := v_max_budget;
        END IF;
END;
```

After creating this database trigger successfully, you test it by updating the YEARLY_BUDGET column to a value greater than the maximum value already in the STUDIO table. What result can you expect?

A. The YEARLY_BUDGET column will be set to the maximun value.
B. The STUDIO table is mutating and therefore, an error is returned.
C. The STUDIO table is mutating and therefore, the trigger execution is ignored.
D. Referencing the NEW qualifier in a BEFORE trigger is illegal and therefore, an error is returned.

Cuestión 18

The script file containing the CHECK_SAL server procedure is lost. This procedure must be modified immediately and recreated. Which data dictionary view can you query to retrieve the source code of this procedure?

A. ALL_SOURCE
B. ALL_PROCEDURES
C. PROCEDURE_SOURCE
D. SERVER_PROCEDURES

Cuestión 19

¿Qué 2 sentencias referidas a la sobrecarga de paquetes son verdaderas?

A. Los subprogramas deben de ser locales.
B. Los subprogramas pueden ser locales o remotos.
C. Está permitido exceder el máximo número de subprogramas.
D. Dos subprogramas con el mismo nombre deben diferir solo en el tipo que se devuelve.
E. Dos subprogramas con el mismo nombre y número de parámetros formales deben tener al menos un parámetro definido con el tipo de datos diferente.

Cuestión 20

Examine this database trigger:

```
CREATE OR REPLACE TRIGGER audit_gross_receipt
AFTER DELETE OR UPDATE OF seats_sold, cost_per_ticket ON gross_receipt
BEGIN
{additional code}
END;
```

If a user issues a DELETE statement against the GROSS_RECEIPT table, the procedure, AUDIT_DELETE_GR of the THEATER_PCK package, must be executed once for the entire DELETE statement. Which code, when added, will perform this successfully?

A. IF DELETING THEN theatre_pck.audit_delete_gr;
END IF;
B. IF CHECK_DELETE THEN theatre_pck.audit_delete_gr;
END IF;
C. IF DBMS_SQL('DELETE') THEN theatre_pck.audit_delete_gr;
END IF;
D. IF DMBS_CHECK_MANIPULATION('DELETE') THEN
theatre_pck.audit_delete_gr;
END IF;

Cuestión 21

Which data dictionary table can you query to view all the dependencies between the objects that you own?

A. USER_OBJECTS
B. USER_RELATIONS
C. USER_DEPENDENCIES
D. USER_RELATIONSHIPS

Cuestión 22

Examine this function:

```
CREATE OR REPLACE FUNCTION set_budget (v_studio_id IN NUMBER,
                                       v_new_budget IN NUMBER)
RETURN number IS
BEGIN
UPDATE studio
SET yearly_budget = v_new_budget
WHERE id = v_studio_id;
COMMIT;
        RETURN SQL%ROWCOUNT;
END;
```

This function is executed from within a procedure called CALCULATE_BUDGET. The database administrator has just informed you that a new column has been added to the STUDIO table. What affect will this have?

A. Only SET_BUDGET will be marked invalid.
B. Only CALCULATE_BUDGET will be marked invalid.
C. Both SET_BUDGET and CALCULATE_BUDGET will be marked invalid.
D. SET_BUDGET, CALCULATE_BUDGET, and STUDIO will be marked invalid.

Cuestión 23

Which character function would you use to return a specified portion of a character string?

A. CONCAT
B. SUBSTR
C. LENGTH
D. INITCAP

Cuestión 24

You attempt to create the ALPHA_3000 table with this statement:

```
1. CREATE TABLE alpha_3000
2. (3000_id NUMBER(9)
3. CONSTRAINT alpha_3000_id_pk PRIMARY KEY,
4. name VARCHAR2(25),
5. title VARCHAR2(25),
6. idname VARCHAR2(25)
7. CONSTRAINT alpha_3000_idname_nn NOT NULL);
```

Which line in the statement causes a syntax error?

A. 1
B. 2
C. 3
D. 7

Cuestión 25

Evaluate this PL/SQL block:

```
DECLARE
v_result BOOLEAN;
BEGIN
DELETE FROM sale WHERE salesperson_id IN (25, 35, 45);
v_result := SQL%ISOPEN;
COMMIT;
END;
```

What will be the value of V_RESULT if three rows are deleted?

A. 0
B. 3
C. TRUE
D. NULL
E. FALSE

Cuestión 26

Which SELECT statement is an equijoin query between two tables?

A. SELECT region.region_name, employee.salary
FROM region, employee
WHERE region.id = employee.region_no;
B. SELECT region.region_name, employee.salary
FROM region, employee
WHERE region.id = employee.region_no(+);
C. SELECT region.region_name, employee.salary
FROM region, employee
WHERE employee.salary BETWEEN region.avq_salary AND region.max_salary;
D. SELECT region.region_name, employeeinfo.last_name
FROM employee region, employee.employeeinfo
WHERE employeeinfo.id >= region.manager_id;

Cuestión 27

The CUSTOMER table has been created. You attempt to create the SALE table with this command:

```
1. CREATE TABLE sale
2. (purchase_no NUMBER(9),
3. customer_no NUMBER(9)
4. CONSTRAINT sale_customer_id_fk REFERENCES
5. customer (id),
6. CONSTRAINT sale_purchase_no_pk PRIMARY KEY (purchase_no),
7. CONSTRAINT sale_customer_no_nn NOT NULL (customer_no));
```

Which line in the statement will cause an error?

A. 2
B. 3
C. 4
D. 6
E. 7

Cuestión 28

The TRANSACTION table has six columns. Since you often query the table with a join to the SALE table, you created an index on five of the columns in the TRANSACTION table. Which result will occur?

A. Inserts to the table will be slower.
B. The speed of deletes will be increased.
C. The size of the TRANSACTION table will be increased.
D. All queries on the TRANSACTION table will be faster if it does contain a large number of NULL values.

Cuestión 29

You alter the database with this command:

RENAME streets TO city;

Which task is accomplished?

A. The STREETS user is renamed CITY.
B. The STREETS table is renamed CITY.
C. The STREETS column is renamed CITY.
D. The STREETS constraint is renamed CITY.

Cuestión 30

You query the database with this command:

```
SELECT name, salary, dept_id
FROM employee
WHERE salary >
(SELECT AVG(salary)
FROM employee
WHERE dept_no =
(SELECT dept_no
FROM employee
WHERE last_name =
(SELECT last_name
FROM employee
WHERE salary > 50000)));
```

Which SELECT clause is evaluated first?

A. SELECT dept_no
B. SELECT last_name
C. SELECT AVG(salary)
D. SELECT name, salary, dept_id

Cuestión 31

Evaluate this PL/SQL block:

```
BEGIN
        FOR i IN 1..6 LOOP
IF i = 1 THEN
COMMIT;
ELSE
IF i = 3 THEN
ROLLBACK;
ELSE
IF i = 5 THEN
COMMIT;
ELSE
INSERT INTO exam(id)
VALUES (i);
END IF;
END IF;
END IF;
        END LOOP;
        COMMIT;
END;
```

How many values will be inserted into the EXAM table?

A. 0
B. 1
C. 2
D. 3
E. 5
F. 6

Cuestión 32

You query the database with this command:

```
SELECT name
FROM employee
WHERE name LIKE '_a%';
```

Which names are displayed?

A. Names starting with "a".
B. Names starting with "a" or "A".
C. Names containing "a" as second letter.
D. Names containing "a" as any letter except the first.

Cuestión 33

Evaluate this SQL statement:

```
SELECT id, (2 * cost) / (2 * sale_price) + 10 price
FROM product;
```

All of the COST and SALE_PRICE values in the PRODUCT table are greater than one. What would happen if you removed all the parentheses from the calculation?

A. The statement would generate a syntax error.
B. The statement would achieve the same results.
C. The results of the PRICE values would be lower.
D. The results of the PRICE values would be higher.

Cuestión 34

Evaluate this IF statement. You issue this command:

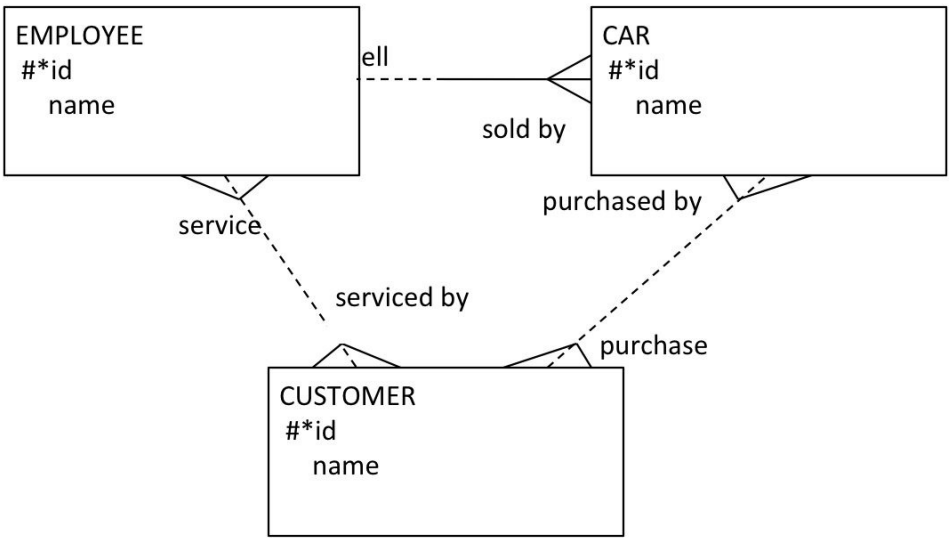
```
GRANT update
ON employee
TO ed
WITH GRANT OPTION;
```

Which task could Ed perform on the EMPLOYEE table?

A. View data.
B. Delete data.
C. Modify constraint.
D. Give update access to other users.

Cuestión 35

Based on the diagram, which relationship is mandatory?



A. Employees sell cars.
B. Customers purchase cars.
C. Cars are sold by employees.
D. Employees service customers.

Cuestión 36

You query the database with this command:

```
SELECT object_name
FROM all_objects
WHERE object_type = 'TABLE';
```

Which values are displayed?

A. Only the names of the tables you own.
B. Only the names of the objects you own.
C. Only the names of all the objects you can access.
D. Only the names of all the tables you can access.

Cuestión 37

What happens when rows are found using a FETCH statement?

A. The cursor opens.
B. The cursor closes.
C. The current row values are loaded into variables.
D. Variables are created to hold the current row values.

Cuestión 38

What is the maximum number of handlers processed before the PL/SQL block is exited when an exception occurs?

A. Only one.
B. All referenced.
C. All that apply.
D. None.

Cuestión 39

Evaluate this SQL script:

```
CREATE ROLE payroll;
CREATE ROLE sales_dept;
CREATE ROLE inventory;
CREATE USER scott IDENTIFIED BY tiger;
GRANT SELECT ON employee TO payroll;
GRANT SELECT ON sale TO sales_dept;
GRANT payroll TO sales_dept;
GRANT sales_dept TO inventory;
GRANT inventory TO scott
/
```

Which tables can user SCOTT query?

A. Only SALE.
B. Only EMPLOYEE.
C. Both SALE and EMPLOYEE.
D. Neither SALE nor EMPLOYEE.

Cuestión 40

You issue this command:

```
SELECT emp_id_seq.CURRVAL
FROM SYS.dual;
```

Which value(s) is displayed?

A. Values of the EMP_ID_SEQ column.
B. Current value of the EMP_ID_SEQ index.
C. Current value of the EMP_ID_SEQ cursor.
D. Current value of the EMP_ID_SEQ sequence.

Cuestión 41

Which program construct must return a value?

A. Package.
B. Function.
C. Anonymous block.
D. Stored procedure.
E. Application procedure.

Cuestión 42

You issue this command:

```
ALTER USER ed IDENTIFIED BY wvu88;
```

Which task has been accomplished?

A. A new user has been added.
B. The user name has been changed.
C. The user password has been changed.
D. A password has been added to the user account.

Cuestión 43

Peter works as a Database Administrator. He create a table named ACCOUNTS that contains the accounts information of the company. Peter wants to validate data automatically, before it is inserted into the ACCOUNTS table. Which of the following will Peter use to accomplish this task?

A. Stored procedure.
B. Anonymous block
C. SELECT statement
D. Trigger

Cuestión 44

You work as an Application Developer in a company. The company uses an Oracle database. The database contains a table names EMPLOYEES. The EMPLOYEES table contains a column of the LONG datatype. You want to change the datatype of the column from LOG to BLOB. What will you do to accomplish this?

A. Use the DBMS_LOG.MIGRATE stored procedure.
B. Use the ALTER DATABASE statement.
C. Use the ALTER TABLE statement.
D. You cannot accomplish the task.

Cuestión 45

David is an employee in Technet Inc. The company uses an Oracle database. David has been granted a role names SALES. David wants to DROP the role. He executes the following statement to accomplish this:

```
DROP ROLE Sales;
```

When he executes the statement, it returns an error. What is the most likely cause of the issue?

Each correct answer represents a complete solution. Choose two.

A. David is not granted the role with the GRANT OPTION.
B. David is not granted the role with de ADMIN OPTION.
C. Only the database administrator can drop the role.
D. David does not have the DROP ANY ROLE system privilege.

Cuestión 46

You work as an Application Developer for Technet Inc. The company uses an Oracle database. The database contains a table names EMPLOYEES. You have defined a database trigger named RAISE_SALARY on the EMPLOYEES table. You want to remove the trigger from the database. Which of the following SQL statements will you use to accomplish this?

A. ALTER TRIGGER Raise_Salary REMOVE;
B. DELETE TRIGGER Raise_Salary;
C. DROP TRIGGER Raise_Salary;
D. REMOVE TRIGGER Raise_Salary;

Cuestión 47

Under which of the following conditions is the use of an explicit cursor necessary?

A. When any SQL data manipulation language (DML) statement is used.
B. When a query returns only one row.
C. When a query does not return any row.
D. When a query returns more than one row.

Cuestión 48

Which of the following statements about a package are true?

Each correct answer represents a complete solution. Choose two.

A. The specification of a package declares the package constructs, whereas the body of the package defines them.
B. A package has three parts.
C. A package itself cannot be called or nested.
D. A package is loaded into the memory every time a package construct is called.

Cuestión 49

Which of the following functions is not available in procedural statements?

A. MOD
B. TRUNC
C. TO_DATE
D. TO_CHAR
E. LOWER
F. DECODE

Cuestión 50

Which of the following is a cursor attribute that yields the number of rows processed by the last DML statement?

A. SQL%ROWRECKON
B. SQL%ROWTALLY
C. SQL%ROWCOUNT
D. SQL%ROWADD

Cuestión 51

Identify whether the given statement is true or false?

"Only the IN parameters can be assigned a default value. The OUT and IN OUT parameters cannot be assigned a default value."

A. False
B. True

Cuestión 52

Which of the following SQL*Plus commands is used to invoke a procedure from iSQL*Plus?

A. RUN
B. EXECUTE
C. CALLL
D. REVOKE
E. START
F. STARTUP

Cuestión 53

Identify whether the given statement is true or false? "A stored procedure cannot be executed as a stand-alone statement."

A. True
B. False

Cuestión 54

Which of the following DBMS_SQL Subprograms combine a given value to a given collection?

A. ADD_ARRAY Procedures
B. UNITE_ARRAY Procedures
C. COMBINE_ARRAY Procedures
D. BIND_ARRAY Procedures

Cuestión 55

Which of the following methods removes one element form the end of a collection?

A. DELETE(m,n)
B. TRIM(n)
C. DELETE(n)
D. DELETE
E. TRIM

Anexo 3. FICHERO SCRIPT_BDHOSPITAL

introducción

El fichero script_bdhospital.sql crea un completo esquema de un hospital junto con la información que se incluye en cada una de las tablas del esquema.

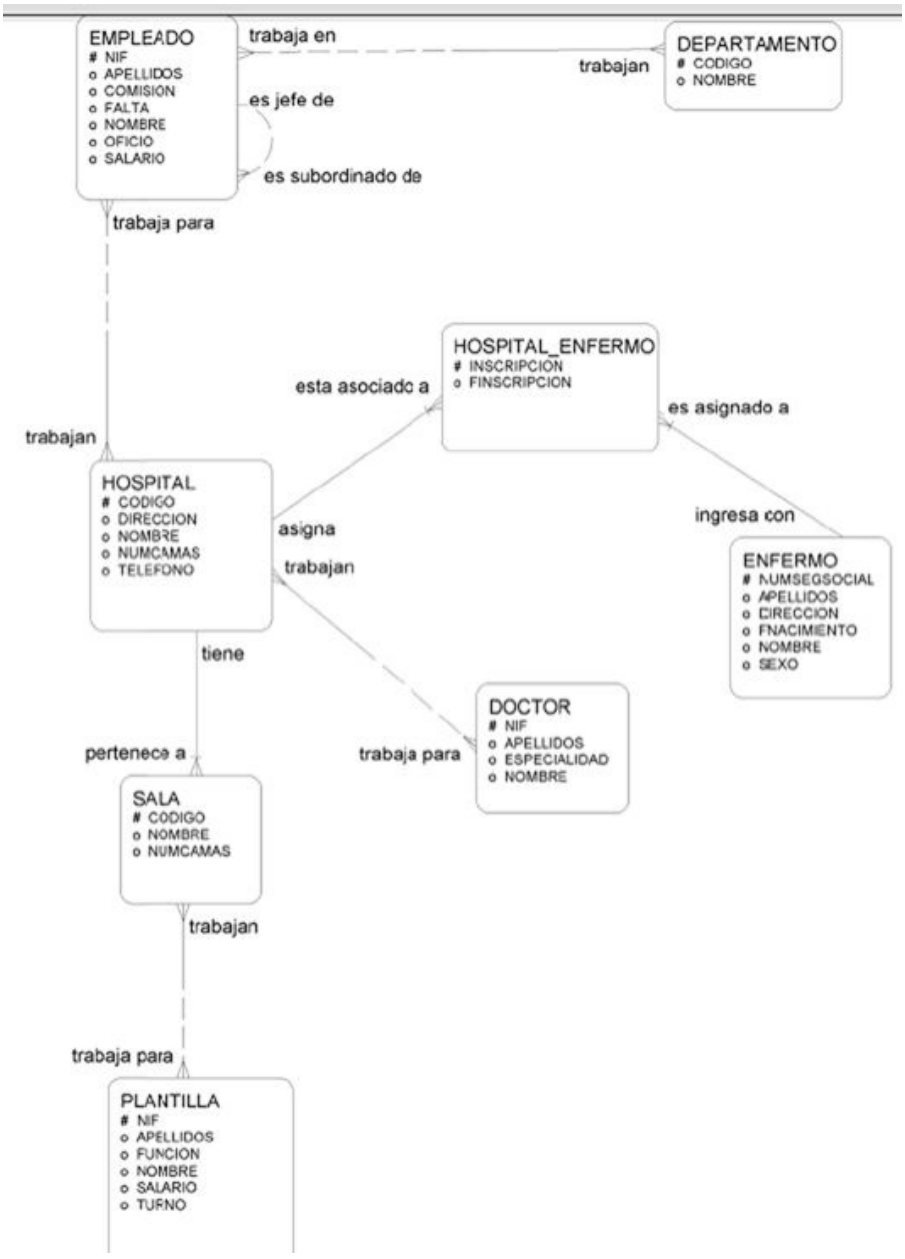
En este anexo podrá encontrar información sobre la estructura del esquema, el código de creación de los objetos que lo componen y la información que se añade a cada una de las tablas que lo conforman.

Todo el esquema del hospital se crea asociado a un nuevo usuario que también se crea en el script con nombre PEPERF y con contraseña PEPITO.

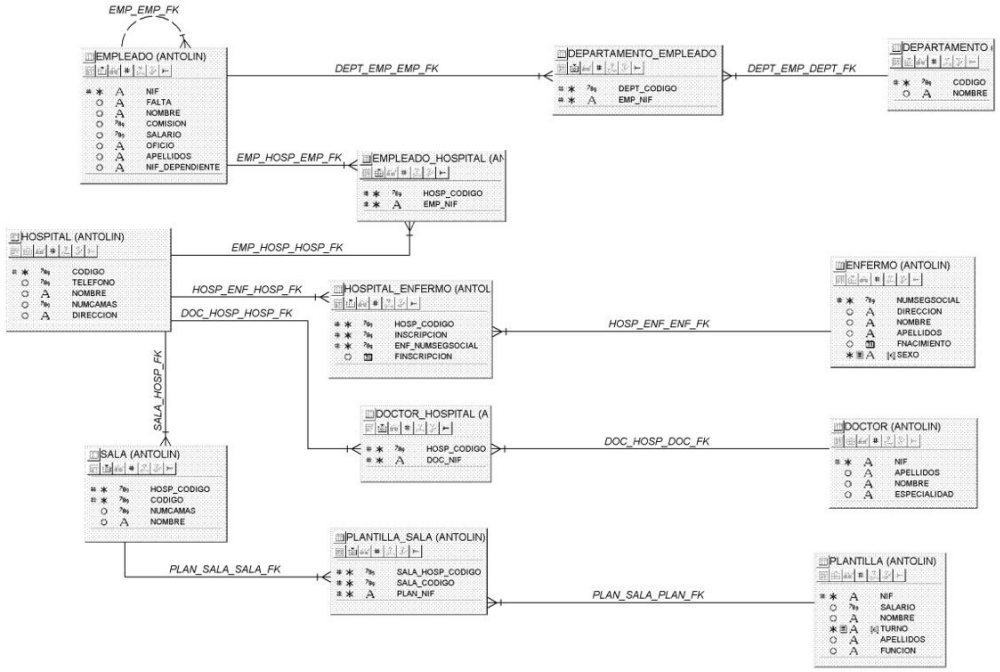
Si no quiere escribir directamente todo el código que se va a mostrar en este anexo, puede descargarse el fichero script_bdhospital.sql desde la web de la editorial:

http://www.rclibros.es

DIAGRAMA ENTIDAD/RELACIÓN DEL ESQUEMA DEL HOSPITAL



modelo relacional del esquema del hospital



contenido del fichero con comentarios

A continuación, se describe el contenido del fichero script_bdhospital.sql.

La ejecución de este fichero se deberá hacer con un usuario de base de datos con permisos de administración: por ejemplo, SYS o SYSTEM.

Creación del usuario propietario de los objetos

```
/* Creacion de un usuario para el curso con nombre de usuario PEPERF con contraseña PEPITO y asignacion de permisos */
/* Los datos de usuario y contraseña pueden ser variados por el alumno si lo desea */
CREATE USER PEPERF IDENTIFIED BY PEPITO;
GRANT CONNECT, RESOURCE TO PEPERF;
GRANT CREATE VIEW TO PEPERF;
```

Conexión a la base de datos con el nuevo usuario

```
/* Se cambia la conexion, para acceder con el nuevo usuario */
CONNECT PEPERF/PEPITO;
```

Creación de las tablas del esquema hospital

```
/* Creacion de la base de datos de una estructura hospitalaria */
CREATE TABLE EMPLEADO
 (NIF              VARCHAR2(9) NOT NULL
 ,FALTA          VARCHAR2(240)
 ,NOMBRE             VARCHAR2(20)
 ,COMISION           NUMBER(6,2)
 ,SALARIO            NUMBER(6,2)
 ,OFICIO             VARCHAR2(20)
 ,APELLIDOS          VARCHAR2(40)
 ,NIF_DEPENDIENTE VARCHAR2(9)
 ,CONSTRAINT EMP_PK PRIMARY KEY (NIF)
 ,CONSTRAINT EMP_EMP_FK FOREIGN KEY (NIF_DEPENDIENTE)
                 REFERENCES EMPLEADO (NIF))
/

CREATE TABLE HOSPITAL
 (CODIGO       NUMBER(2,0) NOT NULL
 ,TELEFONO  NUMBER(9)
 ,NOMBRE     VARCHAR2(12)
 ,NUMCAMAS  NUMBER(4,0)
 ,DIRECCION VARCHAR2(50)
,CONSTRAINT HOSP_PK PRIMARY KEY (CODIGO))
/

CREATE TABLE ENFERMO
 (NUMSEGSOCIAL  NUMBER(9,0) NOT NULL
 ,DIRECCION        VARCHAR2(50)
 ,NOMBRE           VARCHAR2(20)
 ,APELLIDOS        VARCHAR2(40)
 ,FNACIMIENTO   DATE
 ,SEXO          CHAR(1) NOT NULL
 ,CONSTRAINT ENF_PK PRIMARY KEY (NUMSEGSOCIAL)
 ,CONSTRAINT AVCON_1247567854_SEXO_000 CHECK (SEXO IN ('M', 'F')))
/

CREATE TABLE PLANTILLA
 (NIF        VARCHAR2(9) NOT NULL
 ,SALARIO     NUMBER(7,2)
 ,NOMBRE     VARCHAR2(20)
 ,TURNO        CHAR(1) NOT NULL
 ,APELLIDOS VARCHAR2(40)
 ,FUNCION     VARCHAR2(20)
 ,CONSTRAINT PLAN_PK PRIMARY KEY (NIF)
 ,CONSTRAINT AVCON_1247567854_TURNO_000 CHECK (TURNO IN ('M','T','N')))
```

```
/

CREATE TABLE SALA
 (HOSP_CODIGO  NUMBER(2,0) NOT NULL
 ,CODIGO       NUMBER(2,0) NOT NULL
 ,NUMCAMAS     NUMBER(2,0)
 ,NOMBRE       VARCHAR2(30)
 ,CONSTRAINT SALA_PK PRIMARY KEY (CODIGO,HOSP_CODIGO)
 ,CONSTRAINT SALA_HOSP_FK FOREIGN KEY (HOSP_CODIGO)
              REFERENCES HOSPITAL (CODIGO))
/

CREATE TABLE PLANTILLA_SALA
 (SALA_HOSP_CODIGO     NUMBER(2,0) NOT NULL
 ,SALA_CODIGO          NUMBER(2,0) NOT NULL
 ,PLAN_NIF             VARCHAR2(9) NOT NULL
 ,CONSTRAINT PLAN_SALA_PK PRIMARY KEY
(PLAN_NIF,SALA_CODIGO,SALA_HOSP_CODIGO)
 ,CONSTRAINT PLAN_SALA_SALA_FK FOREIGN KEY (SALA_CODIGO,SALA_HOSP_CODIGO)
REFERENCES SALA (CODIGO,HOSP_CODIGO)
 ,CONSTRAINT PLAN_SALA_PLAN_FK FOREIGN KEY (PLAN_NIF)
REFERENCES PLANTILLA (NIF))
/

create table estados_civiles
(codigo          char(1),
descripcion      varchar2(50) constraint nn_estados_civiles not null,
constraint pk_estados_civiles primary key(codigo));

create table persona
(nif             varchar2(9),
codestadocivil number(1),
nombre           varchar2(100) constraint nn_nombre_persona not null,
constraint pk_personal primary key(nif));
```

Alteración de tablas del esquema hospital y creación de índices secundarios

```
/* Alteracion de tablas y crecion de indices */
alter table estados_civiles
add constraint ck_codigo_estados_civiles check (codigo in ('S','C','V','O'));

alter table persona
modify (codestadocivil   char(1));

alter table persona
add constraint fk_persona_estados_civiles foreign key (codestadocivil) references estados_civiles(codigo);

alter table persona
disable constraint fk_persona_estados_civiles;

CREATE INDEX ix1_enfermo ON enfermo(nombre);

CREATE INDEX ix2_enfermo ON enfermo(apellidos);

CREATE UNIQUE INDEX ix1_departamento ON departamento(nombre);
```

Creación de una secuencia para manejo del código de inscripción en un hospital

```
/* Creacion de una secuencia */
CREATE SEQUENCE seq_inscripcion
START WITH 1
INCREMENT BY 1
MINVALUE 1
MAXVALUE 99999999
NOCYCLE
NOCACHE;
```

Inserción de datos en las tablas del esquema

```
/* Insercion de datos sobre las tablas de la estructura hospitalaria */
-- Inserciones en la tabla HOSPITAL

INSERT INTO hospital
values (1,'916644264','Provincial',502,'O Donell 50');

INSERT INTO hospital
values (2,'915953111','General',987,'Atocha s/n');

INSERT INTO hospital
values (3,'919235411','La Paz',412,'Castellana 100');

INSERT INTO hospital
values (4,'915971500','San Carlos',845,'Ciudad Universitaria');

INSERT INTO hospital
values (5,'915971500','Gr. Marañon',300,'Francisco Silvela');

INSERT INTO hospital
values (6,'915971500','Doce Octubre',200,'Avda. Cordoba');

INSERT INTO hospital
values (7,'915971500','La Zarzuela',100,'Moncloa');

-- Inserciones en la tabla ENFERMO

INSERT INTO enfermo
VALUES(280862482,'Goya20','Jose','M.M.',to_date('16051956','ddmmyyyy'),'M');

INSERT INTO enfermo
VALUES(280862481,'Granada 35','Javier','R.R.',to_date('16081970','ddmmyyyy'),'M');

INSERT INTO enfermo
VALUES(280862480,'Sevilla 32','Ruben','S.S.',to_date('10091971','ddmmyyyy'),'M');

INSERT INTO enfermo
VALUES(280862483,'Toledo 1','Rocio','K.K.',to_date('10091968','ddmmyyyy'),'F');

INSERT INTO enfermo
VALUES(280862484,'Malaga 2','Laura','J.J.',to_date('10091971','ddmmyyyy'),'F');

INSERT INTO enfermo
VALUES(280862485,'Barcelona 2','Beatriz','A.A.',to_date('10091988','ddmmyyyy'),'M');
```

-- Inserciones en la tabla HOSPITAL_ENFERMO

```
INSERT INTO hospital_enfermo
VALUES(1,seq_inscripcion.nextval,280862482,to_date('01012002','ddmmyyyy'));
INSERT INTO hospital_enfermo
VALUES(1,seq_inscripcion.nextval,280862482,to_date('02012002','ddmmyyyy'));

INSERT INTO hospital_enfermo
VALUES(1,seq_inscripcion.nextval,280862482,to_date('03012002','ddmmyyyy'));

INSERT INTO hospital_enfermo
VALUES(1,seq_inscripcion.nextval,280862482,to_date('04012002','ddmmyyyy'));

INSERT INTO hospital_enfermo
VALUES(1,seq_inscripcion.nextval,280862482,to_date('05012002','ddmmyyyy'));

INSERT INTO hospital_enfermo
VALUES(1,seq_inscripcion.nextval,280862481,to_date('01012002','ddmmyyyy'));

INSERT INTO hospital_enfermo
VALUES(1,seq_inscripcion.nextval,280862481,to_date('02012002','ddmmyyyy'));

INSERT INTO hospital_enfermo
VALUES(1,seq_inscripcion.nextval,280862481,to_date('03012002','ddmmyyyy'));

INSERT INTO hospital_enfermo
VALUES(1,seq_inscripcion.nextval,280862480,to_date('01102002','ddmmyyyy'));

INSERT INTO hospital_enfermo
VALUES(1,seq_inscripcion.nextval,280862480,to_date('02102002','ddmmyyyy'));

INSERT INTO hospital_enfermo
VALUES(1,seq_inscripcion.nextval,280862480,to_date('03102002','ddmmyyyy'));

INSERT INTO hospital_enfermo
VALUES(1,seq_inscripcion.nextval,280862483,to_date('03102002','ddmmyyyy'));

INSERT INTO hospital_enfermo
VALUES(1,seq_inscripcion.nextval,280862484,to_date('04102002','ddmmyyyy'));


INSERT INTO hospital_enfermo
VALUES(1,seq_inscripcion.nextval,280862485,to_date('03112002','ddmmyyyy'));
INSERT INTO hospital_enfermo
VALUES(2,seq_inscripcion.nextval,280862482,to_date('02012002','ddmmyyyy'));

INSERT INTO hospital_enfermo
VALUES(2,seq_inscripcion.nextval,280862482,to_date('03012002','ddmmyyyy'));

INSERT INTO hospital_enfermo
```

```
VALUES(2,seq_inscripcion.nextval,280862482,to_date('04012002','ddmmyyyy'));

INSERT INTO hospital_enfermo
VALUES(2,seq_inscripcion.nextval,280862482,to_date('05012002','ddmmyyyy'));

INSERT INTO hospital_enfermo
VALUES(2,seq_inscripcion.nextval,280862481,to_date('02012002','ddmmyyyy'));

INSERT INTO hospital_enfermo
VALUES(2,seq_inscripcion.nextval,280862481,to_date('03012002','ddmmyyyy'));

INSERT INTO hospital_enfermo
VALUES(2,seq_inscripcion.nextval,280862480,to_date('02102002','ddmmyyyy'));

INSERT INTO hospital_enfermo
VALUES(2,seq_inscripcion.nextval,280862480,to_date('03102002','ddmmyyyy'));

INSERT INTO hospital_enfermo
VALUES(2,seq_inscripcion.nextval,280862484,to_date('04102002','ddmmyyyy'));

INSERT INTO hospital_enfermo
VALUES(2,seq_inscripcion.nextval,280862485,to_date('03112002','ddmmyyyy'));

INSERT INTO hospital_enfermo
VALUES(3,seq_inscripcion.nextval,280862482,to_date('03012002','ddmmyyyy'));


INSERT INTO hospital_enfermo
VALUES(3,seq_inscripcion.nextval,280862482,to_date('04012002','ddmmyyyy'));

INSERT INTO hospital_enfermo
VALUES(3,seq_inscripcion.nextval,280862482,to_date('05012002','ddmmyyyy'));

INSERT INTO hospital_enfermo
VALUES(3,seq_inscripcion.nextval,280862480,to_date('03102002','ddmmyyyy'));

INSERT INTO hospital_enfermo
VALUES(3,seq_inscripcion.nextval,280862485,to_date('03112002','ddmmyyyy'));

INSERT INTO hospital_enfermo
VALUES(4,seq_inscripcion.nextval,280862482,to_date('04012002','ddmmyyyy'));

INSERT INTO hospital_enfermo
VALUES(4,seq_inscripcion.nextval,280862482,to_date('05012002','ddmmyyyy'));

INSERT INTO hospital_enfermo
VALUES(4,seq_inscripcion.nextval,280862485,to_date('03112002','ddmmyyyy'));

INSERT INTO hospital_enfermo
```

```
VALUES(5,seq_inscripcion.nextval,280862482,to_date('05012002','ddmmyyyy'));

INSERT INTO hospital_enfermo
VALUES(5,seq_inscripcion.nextval,280862485,to_date('03112002','ddmmyyyy'));

INSERT INTO hospital_enfermo
VALUES(6,seq_inscripcion.nextval,280862485,to_date('03112002','ddmmyyyy'));

-- Inserciones en la tabla SALA

INSERT INTO sala
VALUES (1,1,24,'Maternidad');
INSERT INTO sala
VALUES (1,2,21,'Cuidados intensivos');

INSERT INTO sala
VALUES (1,3,67,'Psiquiatrico');
INSERT INTO sala
VALUES (1,4,53,'Cardiologia');

INSERT INTO sala
VALUES (1,5,10,'Recuperacion');

INSERT INTO sala
VALUES (2,1,88,'Maternidad');

INSERT INTO sala
VALUES (2,2,88,'Cuidados intensivos');

INSERT INTO sala
VALUES (2,3,88,'Psiquiatrico');

INSERT INTO sala
VALUES (2,4,88,'Cardiologia');

INSERT INTO sala
VALUES (2,5,88,'Recuperacion');

INSERT INTO sala
VALUES (3,5,99,'Maternidad');

INSERT INTO sala
VALUES (3,4,99,'Cuidados intensivos');

INSERT INTO sala
VALUES (3,3,99,'Psiquiatrico');

INSERT INTO sala
VALUES (3,2,99,'Cardiologia');
```

```
INSERT INTO sala
VALUES (3,1,99,'Recuperacion');

INSERT INTO sala
VALUES (4,1,10,'Maternidad');
INSERT INTO sala
VALUES (4,2,11,'Cuidados intensivos');

INSERT INTO sala
VALUES (4,3,12,'Psiquiatrico');

INSERT INTO sala
VALUES (4,4,13,'Cardiologia');

INSERT INTO sala
VALUES (5,1,10,'Maternidad');
INSERT INTO sala
VALUES (5,2,11,'Cuidados intensivos');

INSERT INTO sala
VALUES (5,3,12,'Psiquiatrico');

INSERT INTO sala
VALUES (6,1,10,'Maternidad');

INSERT INTO sala
VALUES (6,2,11,'Cuidados intensivos');

INSERT INTO sala
VALUES (7,1,99,'Maternidad');

-- Inserciones en la tabla DOCTOR

INSERT INTO doctor
VALUES ('12345678A','Gutierrez J.','Raimundo','Cardiologia');

INSERT INTO doctor
VALUES ('12345678F','Gutierrez J.','Perico','Cardiologia');

INSERT INTO doctor
VALUES ('12345678J','Gutierrez T.','Iñaqui','Cardiologia');

INSERT INTO doctor
VaLUES ('12345678B','Soledad B.','Ines','Ginecologia');
INSERT INTO doctor
VaLUES ('12345678K','Casas B.','Bartolome','Ginecologia');

INSERT INTO doctor
VALUES ('12345678C','Moreno D.','Rosa','Pediatria');
```

```
INSERT INTO doctor
VALUES ('12345678L','Moreno D.','Maria','Pediatria');

INSERT INTO doctor
VALUES ('12345678M','Moreno D.','Isidoro','Pediatria');

INSERT INTO doctor
VALUES ('12345678N','Moreno D.','Antonio','Pediatria');

INSERT INTO doctor
VALUES ('12345678D','Del Toro D.','Ramiro','Psiquiatria');

INSERT INTO doctor
VALUES ('22345678A','Fermin J.','Edmunto','Cardiologia');

INSERT INTO doctor
VALUES ('22345678J','Lopez T.','Iñaqui','Cardiologia');

INSERT INTO doctor
VaLUES ('22345678B','Acaso B.','Ines','Ginecologia');

INSERT INTO doctor
VaLUES ('22345678K','Torres B.','Bartolome','Ginecologia');

INSERT INTO doctor
VALUES ('22345678C','Moreno D.','Rosa','Pediatria');

INSERT INTO doctor
VALUES ('32345678A','Fernandez J.','Loli','Cardiologia');

INSERT INTO doctor
VALUES ('32345678P','Fermin J.','Jorge','Cardiologia');

INSERT INTO doctor
VALUES ('32345678J','Lopez T.','Samuel','Cardiologia');

INSERT INTO doctor
VaLUES ('32345678B','Acaso B.','Maria','Ginecologia');

INSERT INTO doctor
VaLUES ('32345678K','Torres B.','Tirano','Ginecologia');

INSERT INTO doctor
VALUES ('42345678A','Fernandez J.','Ramon','Cardiologia');

INSERT INTO doctor
VALUES ('42345678M','Fermin J.','Fede','Cardiologia');

INSERT INTO doctor
VALUES ('42345678J','Lopez T.','Loles','Cardiologia');
```

```
INSERT INTO doctor
VaLUES ('42345678B','Acaso B.','Maica','Ginecologia');

INSERT INTO doctor
VaLUES ('42345678K','Torres B.','Toñin','Ginecologia');

INSERT INTO doctor
VALUES ('52345678A','Fernandez J.','Ramon','Cardiologia');

INSERT INTO doctor
VALUES ('52345678T','Fermin J.','Fede','Cardiologia');


INSERT INTO doctor
VALUES ('52345678J','Lopez T.','Loles','Cardiologia');

INSERT INTO doctor
VaLUES ('52345678B','Acaso B.','Maica','Ginecologia');
INSERT INTO doctor
VALUES ('62345678A','Fernandez J.','Rocio','Cardiologia');

INSERT INTO doctor
VALUES ('62345678J','Lopez T.','Carlos','Cardiologia');

INSERT INTO doctor
VaLUES ('62345678K','Torres B.','Juan','Ginecologia');

INSERT INTO doctor
VALUES ('72345678J','Lopez T.','JuanMa','Cardiologia');

-- Inserciones en la tabla DOCTOR_HOSPITAL

INSERT INTO doctor_hospital
VALUES (1,'12345678A');

INSERT INTO doctor_hospital
VALUES (1,'12345678F');

INSERT INTO doctor_hospital
VALUES (1,'12345678J');

INSERT INTO doctor_hospital
VaLUES (1,'12345678B');

INSERT INTO doctor_hospital
VaLUES (1,'12345678K');

INSERT INTO doctor_hospital
VALUES (1,'12345678C');
```

```
INSERT INTO doctor_hospital
VALUES (1,'12345678L');

INSERT INTO doctor_hospital
VALUES (1,'12345678M');

INSERT INTO doctor_hospital
VALUES (1,'12345678N');

INSERT INTO doctor_hospital
VALUES (1,'12345678D');

INSERT INTO doctor_hospital
VALUES (2,'12345678A');

INSERT INTO doctor_hospital
VALUES (2,'22345678A');

INSERT INTO doctor_hospital
VALUES (2,'22345678J');

INSERT INTO doctor_hospital
VaLUES (2,'22345678B');

INSERT INTO doctor_hospital
VaLUES (2,'22345678K');

INSERT INTO doctor_hospital
VALUES (2,'22345678C');

INSERT INTO doctor_hospital
VALUES (3,'32345678A');

INSERT INTO doctor_hospital
VALUES (3,'32345678P');

INSERT INTO doctor_hospital
VALUES (3,'32345678J');

INSERT INTO doctor_hospital
VaLUES (3,'32345678B');

INSERT INTO doctor_hospital
VaLUES (3,'32345678K');

INSERT INTO doctor_hospital
VALUES (3,'22345678C');

INSERT INTO doctor_hospital
```

```
VALUES (4,'42345678A');

INSERT INTO doctor_hospital
VALUES (4,'42345678M');
INSERT INTO doctor_hospital
VALUES (4,'42345678J');

INSERT INTO doctor_hospital
VaLUES (4,'42345678B');

INSERT INTO doctor_hospital
VaLUES (4,'42345678K');
INSERT INTO doctor_hospital
VALUES (5,'52345678A');

INSERT INTO doctor_hospital
VALUES (5,'52345678T');

INSERT INTO doctor_hospital
VALUES (5,'52345678J');

INSERT INTO doctor_hospital
VaLUES (5,'52345678B');

INSERT INTO doctor_hospital
VALUES (6,'62345678A');

INSERT INTO doctor_hospital
VALUES (6,'62345678J');

INSERT INTO doctor_hospital
VaLUES (6,'62345678K');

INSERT INTO doctor_hospital
VALUES (7,'62345678A');

INSERT INTO doctor_hospital
VALUES (7,'72345678J');

-- Inserciones en la tabla DEPARTAMENTO

INSERT INTO departamento
VALUES(1,'CONTABILIDAD');

INSERT INTO departamento
VALUES(2,'INVESTIGACION');

INSERT INTO departamento
VALUES(3,'FACTURACION');
```

```
INSERT INTO departamento
VALUES(4,'ADMINISTRACION');

INSERT INTO departamento
VALUES(5,'FARMACIA');

INSERT INTO departamento
VALUES(6,'LIMPIEZA');

-- Inserciones en la tabla EMPLEADO

INSERT INTO empleado
VALUES('10000000A',TO_DATE('10012002','DDMMYYYY'),'Jorge',1000.22,3000.11,'DIRECTOR','Perez
 Sala',NULL);

INSERT INTO empleado
VALUES('20000000B',TO_DATE('11012002','DDMMYYYY'),'Javier',500.22,2000.22,'GERENTE','Sala
Rodriguez','10000000A');

INSERT INTO empleado
VALUES('30000000C',TO_DATE('11012002','DDMMYYYY'),'Soledad',500.33,2000.33,'ADMISTRADOR
 J.','10000000A');

INSERT INTO empleado
VALUES('40000000D',TO_DATE('12012002','DDMMYYYY'),'Sonia',NULL,1800.44,'JEFE
FARMACIA','Moldes R.','20000000B');

INSERT INTO empleado
VALUES('50000000E',TO_DATE('12012002','DDMMYYYY'),'Antonio',300.44,1800.44,'JEFE
LABORATORIO','Lopez A.','20000000B');

INSERT INTO empleado
VALUES('60000000F',TO_DATE('12012002','DDMMYYYY'),'Carlos',500.55,1800.55,'CONTABLE','Roa
D.','30000000C');

INSERT INTO empleado
VALUES('70000000G',TO_DATE('13012002','DDMMYYYY'),'Lola',NULL,1000,'ADMINISTRATIVO','S
 D.','60000000F');

INSERT INTO empleado
VALUES('80000000L',TO_DATE('13012002','DDMMYYYY'),'Angel',NULL,1000,'ADMINISTRATIVO','

INSERT INTO empleado
VALUES('90000000M',TO_DATE('12012002','DDMMYYYY'),'Ramon',NULL,1500,'JEFE
LIMPIEZA','Maria Casas','20000000B');

INSERT INTO empleado
VALUES('11000000P',TO_DATE('14012002','DDMMYYYY'),'Luis',NULL,700,'HIGIENE','Sanchez
D.','90000000M');
```

```
INSERT INTO empleado
VALUES('12000000Q',TO_DATE('14012002','DDMMYYYY'),'Rosa',NULL,700,'HIGIENE','Torres
A.','90000000M');

INSERT INTO empleado
VALUES('10000000N',TO_DATE('14012002','DDMMYYYY'),'Sara',200,1000,'INVESTIGADOR','Gomez
A.','50000000E');

-- Inserciones en la tabla EMPLEADO_HOSPITAL
INSERT INTO empleado_hospital
VALUES(1,'10000000A');

INSERT INTO empleado_hospital
VALUES(1,'20000000B');

INSERT INTO empleado_hospital
VALUES(1,'30000000C');

INSERT INTO empleado_hospital
VALUES(1,'40000000D');

INSERT INTO empleado_hospital
VALUES(1,'50000000E');

INSERT INTO empleado_hospital
VALUES(1,'60000000F');

INSERT INTO empleado_hospital
VALUES(1,'70000000G');

INSERT INTO empleado_hospital
VALUES(1,'80000000L');

INSERT INTO empleado_hospital
VALUES(1,'90000000M');

INSERT INTO empleado_hospital
VALUES(1,'11000000P');

INSERT INTO empleado_hospital
VALUES(1,'12000000Q');

INSERT INTO empleado_hospital
VALUES(1,'10000000N');

INSERT INTO empleado_hospital
VALUES(2,'10000000A');

INSERT INTO empleado_hospital
VALUES(2,'20000000B');
```

```
INSERT INTO empleado_hospital
VALUES(2,'30000000C');

INSERT INTO empleado_hospital
VALUES(2,'40000000D');


INSERT INTO empleado_hospital
VALUES(2,'50000000E');

INSERT INTO empleado_hospital
VALUES(2,'70000000G');

INSERT INTO empleado_hospital
VALUES(2,'80000000L');

INSERT INTO empleado_hospital
VALUES(2,'90000000M');

INSERT INTO empleado_hospital
VALUES(2,'11000000P');

INSERT INTO empleado_hospital
VALUES(2,'12000000Q');

INSERT INTO empleado_hospital
VALUES(2,'10000000N');

INSERT INTO empleado_hospital
VALUES(3,'10000000A');

INSERT INTO empleado_hospital
VALUES(3,'20000000B');

INSERT INTO empleado_hospital
VALUES(3,'30000000C');

INSERT INTO empleado_hospital
VALUES(3,'40000000D');

INSERT INTO empleado_hospital
VALUES(3,'80000000L');

INSERT INTO empleado_hospital
VALUES(3,'90000000M');

INSERT INTO empleado_hospital
VALUES(3,'11000000P');
```

```
INSERT INTO empleado_hospital
VALUES(3,'12000000Q');

INSERT INTO empleado_hospital
VALUES(3,'10000000N');


INSERT INTO empleado_hospital
VALUES(4,'10000000A');

INSERT INTO empleado_hospital
VALUES(4,'20000000B');

INSERT INTO empleado_hospital
VALUES(4,'30000000C');

INSERT INTO empleado_hospital
VALUES(4,'40000000D');

INSERT INTO empleado_hospital
VALUES(4,'80000000L');

INSERT INTO empleado_hospital
VALUES(4,'90000000M');

INSERT INTO empleado_hospital
VALUES(4,'12000000Q');

INSERT INTO empleado_hospital
VALUES(4,'10000000N');

INSERT INTO empleado_hospital
VALUES(5,'10000000A');

INSERT INTO empleado_hospital
VALUES(5,'20000000B');

INSERT INTO empleado_hospital
VALUES(5,'30000000C');

INSERT INTO empleado_hospital
VALUES(5,'40000000D');

INSERT INTO empleado_hospital
VALUES(5,'80000000L');

INSERT INTO empleado_hospital
VALUES(5,'12000000Q');

INSERT INTO empleado_hospital
```

```
VALUES(5,'10000000N');

INSERT INTO empleado_hospital
VALUES(6,'10000000A');


INSERT INTO empleado_hospital
VALUES(6,'20000000B');

INSERT INTO empleado_hospital
VALUES(6,'30000000C');

INSERT INTO empleado_hospital
VALUES(6,'40000000D');

INSERT INTO empleado_hospital
VALUES(6,'12000000Q');

INSERT INTO empleado_hospital
VALUES(6,'10000000N');

-- Inserciones en la tabla PLANTILLA

INSERT INTO plantilla
VALUES('11111111A',15000.22,'Alejandro','M','A.A.','ENFERMERO');

INSERT INTO plantilla
VALUES('11111111B',15000.22,'Bartolome','T','B.B.','ENFERMERO');

INSERT INTO plantilla
VALUES('11111111C',15000.22,'Carlos','N','C.C.','ENFERMERO');

INSERT INTO plantilla
VALUES('22222222A',15000.22,'Adriana','M','A.A.','ENFERMERA');

INSERT INTO plantilla
VALUES('22222222B',15000.22,'Bibiana','T','B.B.','ENFERMERA');

INSERT INTO plantilla
VALUES('22222222C',15000.22,'Casilda','N','C.C.','ENFERMERA');

INSERT INTO plantilla
VALUES('33333333A',15000.22,'Alberto','M','A.A.','ENFERMERO');
INSERT INTO plantilla
VALUES('33333333B',15000.22,'Bonifacio','T','B.B.','ENFERMERO');

INSERT INTO plantilla
VALUES('33333333C',15000.22,'Casimiro','N','C.C.','ENFERMERO');

INSERT INTO plantilla
```

```
VALUES('44444444A',15000.22,'Amelia','M','A.A.','ENFERMERA');

INSERT INTO plantilla
VALUES('44444444B',15000.22,'Bony','T','B.B.','ENFERMERA');

INSERT INTO plantilla
VALUES('44444444C',15000.22,'Casandra','N','C.C.','ENFERMERA');

INSERT INTO plantilla
VALUES('55555555A',15000.22,'Armando','M','A.A.','ENFERMERO');

INSERT INTO plantilla
VALUES('55555555B',15000.22,'Benicio','T','B.B.','ENFERMERO');

INSERT INTO plantilla
VALUES('55555555C',15000.22,'Ciceron','N','C.C.','ENFERMERO');

-- Inserciones en la tabla PLANTILLA_SALA

INSERT INTO plantilla_sala
VALUES(1,1,'11111111A');

INSERT INTO plantilla_sala
VALUES(1,1,'11111111B');

INSERT INTO plantilla_sala
VALUES(1,1,'11111111C');

INSERT INTO plantilla_sala
VALUES(1,2,'22222222A');

INSERT INTO plantilla_sala
VALUES(1,2,'22222222B');

INSERT INTO plantilla_sala
VALUES(1,2,'22222222C');

INSERT INTO plantilla_sala
VALUES(1,3,'33333333A');

INSERT INTO plantilla_sala
VALUES(1,3,'33333333B');

INSERT INTO plantilla_sala
VALUES(1,3,'33333333C');

INSERT INTO plantilla_sala
VALUES(1,4,'44444444A');

INSERT INTO plantilla_sala
```

```
VALUES(1,4,'44444444B');
INSERT INTO plantilla_sala
VALUES(1,4,'44444444C');

INSERT INTO plantilla_sala
VALUES(1,5,'55555555A');

INSERT INTO plantilla_sala
VALUES(1,5,'55555555B');

INSERT INTO plantilla_sala
VALUES(1,5,'55555555C');

INSERT INTO plantilla_sala
VALUES(2,1,'11111111A');

INSERT INTO plantilla_sala
VALUES(2,1,'11111111B');

INSERT INTO plantilla_sala
VALUES(2,2,'22222222A');
INSERT INTO plantilla_sala
VALUES(2,2,'22222222B');

INSERT INTO plantilla_sala
VALUES(2,3,'33333333A');

INSERT INTO plantilla_sala
VALUES(2,3,'33333333B');

INSERT INTO plantilla_sala
VALUES(2,4,'44444444A');

INSERT INTO plantilla_sala
VALUES(2,4,'44444444B');

INSERT INTO plantilla_sala
VALUES(2,5,'55555555A');

INSERT INTO plantilla_sala
VALUES(2,5,'55555555B');

INSERT INTO plantilla_sala
VALUES(3,1,'11111111A');

INSERT INTO plantilla_sala
VALUES(3,2,'22222222A');

INSERT INTO plantilla_sala
VALUES(3,3,'33333333A');
```

```
INSERT INTO plantilla_sala
VALUES(3,4,'44444444A');

INSERT INTO plantilla_sala
VALUES(3,5,'55555555A');

INSERT INTO plantilla_sala
VALUES(4,1,'11111111A');

INSERT INTO plantilla_sala
VALUES(4,2,'22222222A');

INSERT INTO plantilla_sala
VALUES(4,3,'33333333A');
INSERT INTO plantilla_sala
VALUES(4,4,'44444444A');

INSERT INTO plantilla_sala
VALUES(6,1,'11111111A');

INSERT INTO plantilla_sala
VALUES(6,2,'22222222A');

INSERT INTO plantilla_sala
VALUES(7,1,'11111111A');

-- Inserciones en la tabla DEPARTAMENTO_EMPLEADO

INSERT INTO departamento_empleado
VALUES(4,'10000000A');

INSERT INTO departamento_empleado
VALUES(4,'20000000B');

INSERT INTO departamento_empleado
VALUES(4,'30000000C');

INSERT INTO departamento_empleado
VALUES(5,'40000000D');

INSERT INTO departamento_empleado
VALUES(2,'50000000E');

INSERT INTO departamento_empleado
VALUES(1,'60000000F');

INSERT INTO departamento_empleado
VALUES(1,'70000000G');

INSERT INTO departamento_empleado
```

```
VALUES(1,'80000000L');

INSERT INTO departamento_empleado
VALUES(6,'90000000M');

INSERT INTO departamento_empleado
VALUES(6,'11000000P');

INSERT INTO departamento_empleado
VALUES(6,'12000000Q');

INSERT INTO departamento_empleado
VALUES(2,'10000000N');
commit;

-- Inserciones en la tabla EMPLEADO

INSERT INTO empleado
VALUES('12345678B',TO_DATE('11011970','DDMMYYYY'),'Juan',NULL,3000,'DIRECTOR','Lopez
Z.',NULL);

INSERT INTO empleado
VALUES('87654321A',TO_DATE('12011975','DDMMYYYY'),'Fermin',1000,2000,'GERENTE','Garcia
L.','12345678B');

INSERT INTO empleado
VALUES('64328285C',TO_DATE('13011979','DDMMYYYY'),'Rosa',NULL,1500,'ADMINISTRADOR','M
 R.','87654321A');

INSERT INTO empleado
VALUES('83253235F',TO_DATE('14011980','DDMMYYYY'),'Miguel',300,1000,'CONTABLE','Soria
T.','64328285C');

-- Inserciones en la tabla DEPARTAMENTO_EMPLEAO

INSERT INTO departamento_empleado
VALUES(4,'12345678B');

INSERT INTO departamento_empleado
VALUES(4,'87654321A');

INSERT INTO departamento_empleado
VALUES(4,'64328285C');

INSERT INTO departamento_empleado
VALUES(4,'83253235F');
Commit;
```

Anexo IV. REFERENCIAS Y MATERIAL ANEXO EN INTERNET

REFERENCIAS UTILIZADAS PARA EL CURSO

Para la elaboración de la documentación contenida en este libro se han utilizado como fuentes de referencia técnica:

- Documentación oficial que proporciona Oracle en su web para la versión 11g Release 2. En el siguiente apartado se indican enlaces a la misma.

- Preguntas y respuestas de ejemplo para la presentación a las certificaciones de Oracle.

- Documentación elaborada durante el desempeño de la actividad profesional, fruto de la experiencia de más de 20 años, adquirida en los distintos puestos de trabajo desempeñados con el manejo de diversas versiones de los productos de Oracle desde la versión Oracle 7 a la versión Oracle 11g.

- Documentación que recoge la experiencia formativa, sugerencias y preguntas técnicas surgidas durante la impartición de este curso en distintas academias de formación técnica, y en empresas privadas.

enlaces a oracle

Si necesita ampliar información, conocer nuevos productos y descargar software de Oracle, puede acudir a los enlaces oficiales de la web de Oracle que se indican a continuación.

Página oficial de Oracle Internacional

La web oficial de Oracle a nivel internacional se encuentra en el enlace siguiente:

http://www.oracle.com/

Página oficial de Oracle España

La web oficial de Oracle en España se encuentra en el enlace siguiente:

http://www.oracle.com/es/index.html

Red tecnológica de Oracle

La web que aglutina la comunidad de desarrolladores, DBAs y arquitectos de Oracle se encuentra en el enlace siguiente:

http://www.oracle.com/technetwork/index.html

Esta página es el mejor punto para ampliar información y descargar software oficial de Oracle. Asimismo, encontrará tutoriales y ejemplos de scripts y código de los diversos productos de Oracle. También podrá compartir información y experiencias en los foros disponibles para desarrolladores, DBAs y arquitectos.

Programa de certificación en productos de Oracle

La web que informa sobre todo el proceso de certificación en los productos de Oracle se encuentra en el siguiente enlace:

http://education.oracle.com/pls/web_prod-plq-dad/db_pages.getpage?page_id=39

Universidad de Oracle

La web para la formación oficial de Oracle en sus productos se encuentra en el siguiente enlace:

http://education.oracle.com/pls/web_prod-plq-dad/db_pages.getpage?page_id=3&p_org_id=51&lang=E

Documentación oficial de Oracle de la versión 11g Rel.2

La web donde puede encontrar toda la documentación oficial de Oracle sobre la versión 11g Release 2 de base de datos, se encuentra en el enlace siguiente:

http://www.oracle.com/pls/db112/homepage

web del autor

En la web del autor cuyo enlace se indica al final de este apartado, podrá encontrar las soluciones a todos los supuestos prácticos incluidos en este libro, en ficheros individualizados en formato SQL para ser ejecutados directamente contra el servidor de base de datos.

También podrá encontrar información sobre los libros publicados por el autor y los enlaces a las páginas web de Oracle que se han indicado anteriormente.

http://www.murope.jazztel.es/antolin

web de lA EDITORIAL

Los archivos para la resolución de los supuestos se encuentran disponibles en la página individual del libro, accediendo a la web: *www.rclibros.es*

Anexo V. GUÍA DE INSTALACIÓN DE ORACLE 11G XE

INTRODUCCIÓN

Para el mejor aprovechamiento de este curso, se aconseja la instalación de la base de datos de Oracle 11g Express Edition, dado que es una versión reducida de libre distribución que consume pocos recursos de máquina y que permite llevar a cabo todas las prácticas y supuestos que se especifican en este libro.

Puede descargar este producto desde el siguiente enlace de la web de Oracle previo registro gratuito:

http://www.oracle.com/technetwork/database/express-edition/downloads/index.html

A la finalización de la redacción de este libro, la última versión que ofrece Oracle sobre la base de datos Express Edition es la 11.2 Express Edition Beta.

En este anexo podrá encontrar una guía práctica para la instalación de este producto Oracle en su PC.

REQUERIMIENTOS MÍNIMOS

Los requerimientos de máquina para la instalación de Oracle 11.2 Express Edition Beta son los siguientes:

- Sistema operativo Windows 32 bits:
    - Windows 2000 Service Pack 4 o posterior.
    - Windows Server 2003.
    - Windows XP Professional Service Pack 1 o posterior.
    - Compatible con Windows Vista y Windows 7.
- Protocolo TCP/IP:
    - El equipo debe de disponer de una tarjeta de red fija o inalámbrica.
- Espacio en disco necesario:
    - 1,6 GB.
- Memoria RAM:
    - 265 MB mínimo. Se recomienda 512 MB.
- Microsoft Windows Installer (MSI) disponible en el equipo:
    - MSI versión 2.0 o posterior.
    - Si no dispone de este componente, lo puede descargar desde la página web http://msdn.microsoft.com.

tutorial de instalación

A continuación, se muestran los pasos a seguir para la instalación de Oracle 11.2 Express Edition Beta.

Paso 1: Descarga del producto

Descargue Oracle 11.2 Express Edition Beta desde la web de Oracle.

Paso 2: Configuración del equipo para la instalación

Antes de comenzar la instalación, debe de seguir estos pasos para preparar su equipo para la instalación:

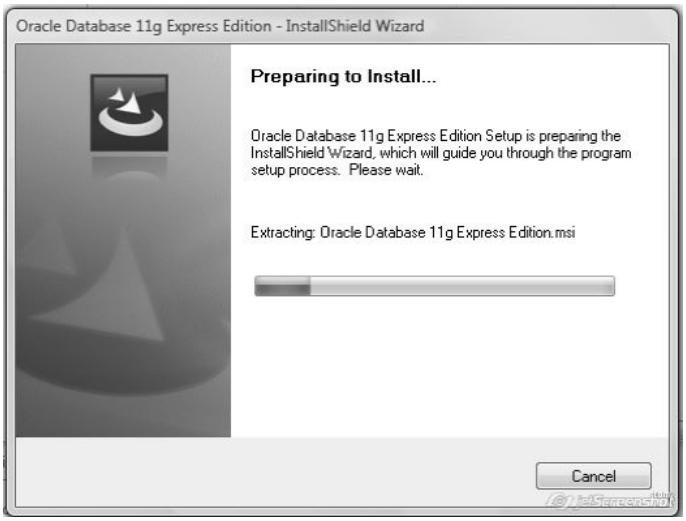
- Cree una carpeta con el nombre *OracleXE11gR2*.
- Extraiga el fichero de instalación *win32_11gR2_OracleXE.zip* en la carpeta *OracleXE11gR2* creada en el punto anterior.

Paso 3: Ejecute la instalación

Para comenzar la instalación debe de ejecutar el fichero *setup.exe* que encontrará en la carpeta *\OracleXE11gR2\DISK1*.

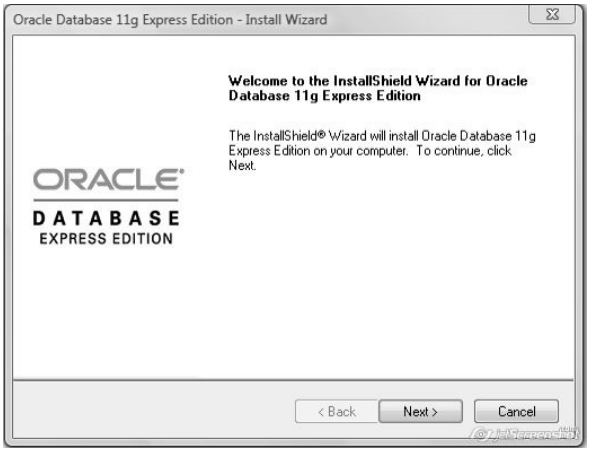
Paso 4: Comienza el proceso de instalación

Una vez ejecutado el fichero de instalación se mostrará una pantalla con el progreso de la preparación para la instalación.



Paso 5: Pantalla de bienvenida

Concluida la preparación para comenzar la instalación se mostrará una pantalla de bienvenida en la que deberá pulsar el botón *Next* para comenzar el proceso de instalación de la base de datos en su equipo.

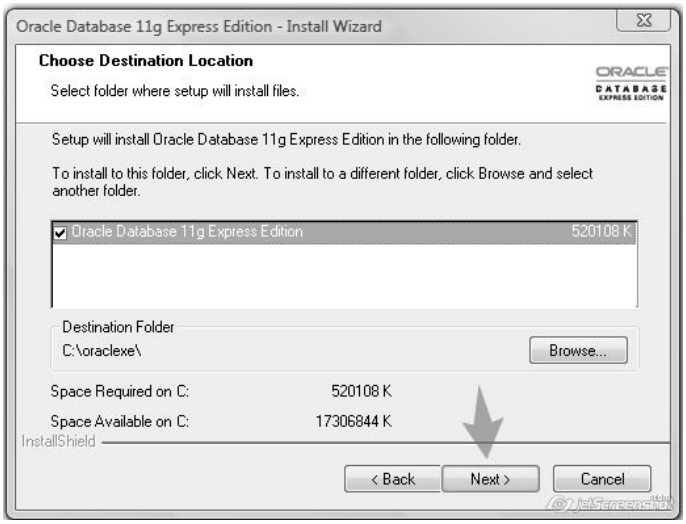


Paso 6: Términos de la licencia

A continuación, se le mostrará los términos de la licencia de uso del producto para su aceptación. Debe marcar la opción *I accept the terms in the license agreement* y posteriormente pulsar el botón *Next.*

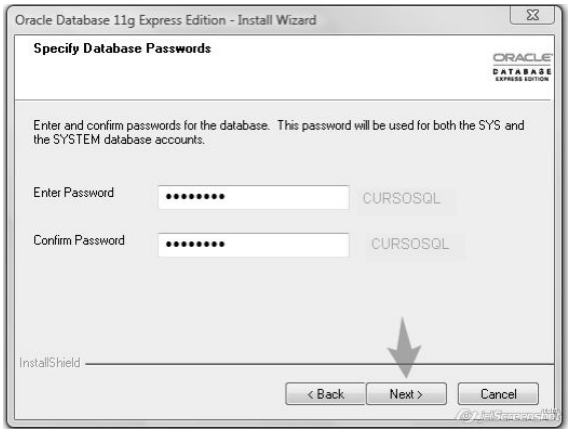
Paso 7: Ubicación para la instalación

Mantenga la ubicación por defecto de la instalación y pulse el botón *Next.*



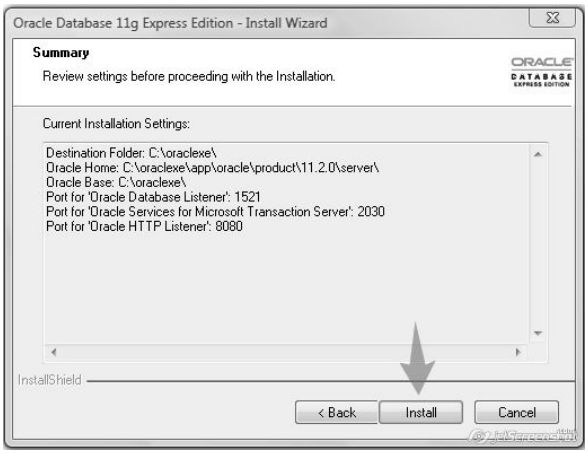
Paso 8: Contraseña para los usuarios SYS y SYSTEM

En este punto de la instalación el sistema nos solicita que introduzcamos una contraseña para el acceso a la base de datos de los usuarios administradores SYS y SYSTEM. Debe introducir la misma contraseña 2 veces. Se recomienda para el seguimiento de este curso que la contraseña que introduzca sea *CURSOSQL* y a continuación pulse el botón *Next.*



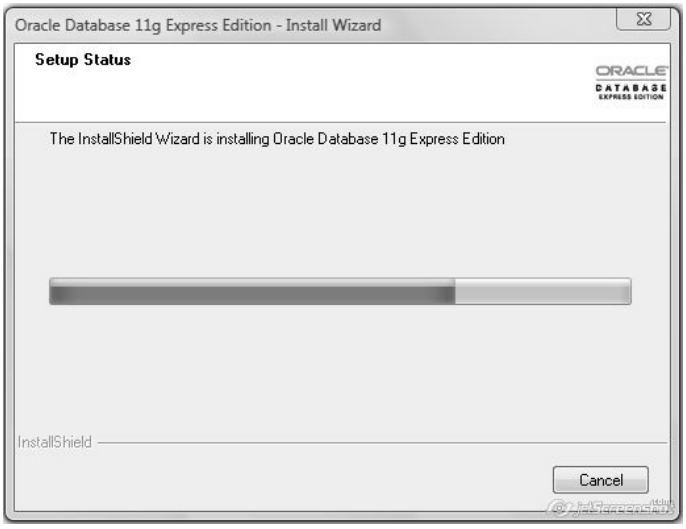
Paso 9: Resumen previo a la instalación

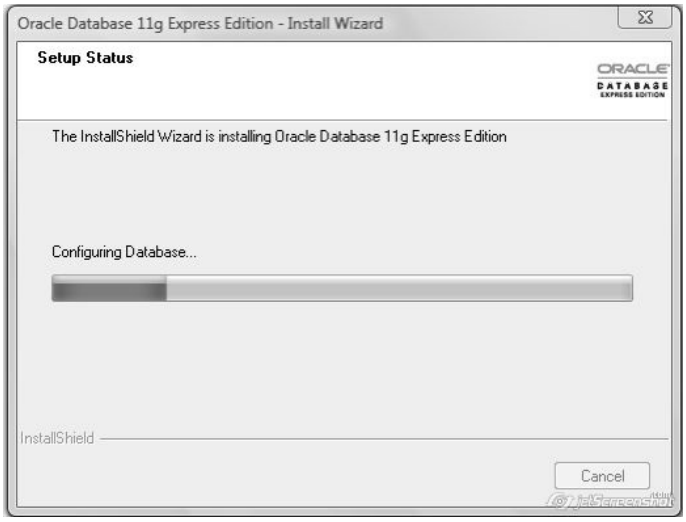
Justo antes de comenzar la instalación física de acuerdo a los parámetros introducidos en las pantallas previas, se presenta una pantalla resumen con las opciones que se van a instalar, así como los puertos que se habilitarán después de la instalación, para poder acceder a la base de datos. Pulse el botón *Install*.



Paso 10: Progreso de la instalación

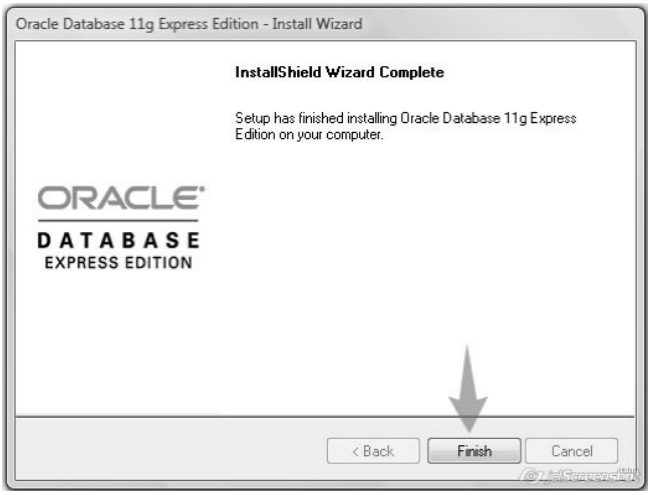
En esta fase el usuario recibirá en pantalla una serie de imágenes con el progreso de la instalación en su equipo. No debe pulsar ningún botón hasta que concluya la instalación.





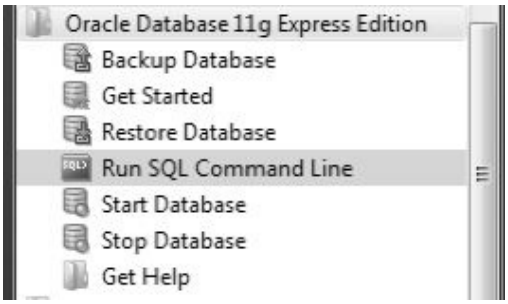
Paso 11: Finalización de la instalación

La instalación finalizará cuando se presente una pantalla como la que se muestra a continuación, con un botón *Finish*, que deberá pulsar para concluirla.



Paso 12: Familiarizándose con los elementos instalados

Finalizada la instalación se encontrará con el siguiente grupo de elementos instalado en su equipo.



E igualmente le aparecerá el siguiente icono en su escritorio.

El funcionamiento básico de estos elementos es el siguiente:

backup database

Permite realizar un backup o salvaguarda de la base de datos.

GET STARTED

Abre el sistema de gestión administrativa de la base de datos, a través del explorador de su equipo.

RESTORE database

Permite realizar un restore o recuperación de una salvaguarda que se haya realizado de la base de datos, con la utilidad backup comentada anteriormente.

RUN SQL COMMAND LINE

Permite ejecutar sentencias y scripts contra la base de datos. Es en esta herramienta donde deberá probar la ejecución de las sentencias SQL que se han explicado durante este curso, al igual que la usará para la resolución de los supuestos prácticos que se le han sugerido en el libro.

START DATABASE

Permite arrancar la base de datos para poderse conectar con ella. Normalmente esta utilidad queda configurada en el equipo para que se ejecute automáticamente cuando arranque el ordenador.

STOP DATABASE

Permite parar la base de datos.

[1] El motivo de haber incluido en este capítulo preguntas, así como su resolución en el anexo II, en lengua inglesa, se debe a que así se encuentran en el examen real para la obtención de cada certificación. Por tanto, es aconsejable tener conocimientos a nivel medio de inglés o básicos con amplio dominio de vocabulario técnico para presentarse a los exámenes de certificación de Oracle.