



Base de Datos Avanzado II

ÍNDICE

	Pág
Presentación	7
Unidad de aprendizaje 1	
INTRODUCCIÓN A UNA BD ORACLE	
1.1 Tema 1 : INTRODUCCIÓN AL SGBD ORACLE	11
1.1.1 : Introducción a una DB relacional y a la arquitectura Oracle	11
1.1.2 : Interactuando con la base de datos: Conexión y SQL*PLUS	14
1.1.3 : Componentes de la Arquitectura de una BD Oracle	20
1.2 Tema 2 : GESTIÓN DE UNA INSTANCIA ORACLE	30
1.2.1 : Inicio de la instancia	32
1.2.2 : Apertura de la Base de Datos	33
1.2.3 : Cierre de la Base de Datos	38
1.2.4 : Configuración de un cliente Oracle	41
Unidad de aprendizaje 2	
CREACIÓN DE ESTRUCTURAS DE DATOS	
2.1 Tema 3 : CREACIÓN DE ESTRUCTURAS DE DATOS	51
2.1.1 : Creación y modificación de tablas	51
2.1.2 : Creación de restricciones	54
2.1.3 : Manejo de índices	59
2.1.4 : Manejo de secuencias	61
2.1.5 : Manejo de sinónimos	63
2.2 Tema 4 : DICCIONARIO DE DATOS	65
2.2.1 : Introducción al Diccionario de Datos	65
2.2.2 : Estructura del Diccionario de Datos	65
2.2.3 : Uso del Diccionario de Datos	66
2.2.4 : Otras tablas en el Diccionario	67
2.2.5 : La vista DBA_OBJECTS	67

Unidad de aprendizaje 3

LENGUAJE DE MANIPULACIÓN DE DATOS

3.1 Tema 5 : LENGUAJE DE MANIPULACIÓN DE DATOS	75
3.1.1 : Lenguaje SQL	75
3.1.2 : Instrucciones DML y operadores	76
3.1.3 : Consultas multitable	82
3.1.3 : Funciones Predefinidas	84

Unidad de aprendizaje 4

PROGRAMACIÓN EN ORACLE

4.1 Tema 6 : PROGRAMACIÓN PL/SQL	95
4.1.1 : Introducción a Oracle PL/SQL	95
4.1.2 : Tipos de datos en PL/SQL	97
4.1.3 : Estructuras de Bloques de PL/SQL	97
4.1.4 : Sentencias SQL en PL/SQL	105
4.2 Tema 7 : ESTRUCTURAS DE CONTROL EN PL/SQL	115
4.2.1 : Estructuras Condicionales	115
4.2.2 : Estructuras Cíclicas	118
4.3 Tema 8 : CURSORES	121
4.3.1 : Tipo de cursores	122
4.3.2 : Declaración de cursores	122
4.3.3 : Apertura de cursores	124
4.3.4 : Almacenamiento de datos de cursores	124
4.3.5 : Cierre de cursores	125
4.3.6 : Atributos de los cursores	126
4.3.7 : Uso avanzado de cursores	127
4.4 Tema 9 : EXCEPCIONES EN PL/SQL	134
4.4.1 : ¿Qué es una Excepción?	134
4.4.2 : Declaración de Excepciones	134
4.4.3 : Generación de Excepciones	137
4.4.4 : Tratamiento de Excepciones	137
4.4.5 : Propagación de Excepciones	141

Unidad de aprendizaje 5**PROGRAMACIÓN AVANZADA EN ORACLE**

5.1 Tema 10 : PROGRAMACIÓN DENTRO DE UNA BD ORACLE	147
5.1.1 : Construcción de funciones y procedimientos almacenados	147
5.1.2 : Construcción de paquetes	156
5.1.3 : Construcción de disparadores	163

Unidad de aprendizaje 6**SISTEMA DE ARCHIVOS ORACLE**

6.1 Tema 11 : SISTEMA DE ARCHIVOS ORACLE	175
6.1.1 : Archivos de Control	175
6.1.2 : Mantenimiento de los Archivos Redo logs	176
6.1.3 : Gestión de tablespaces y datafiles	177

Unidad de aprendizaje 7**MECANISMOS PARA LA GESTIÓN DE LA INFORMACIÓN**

7.1 Tema 12 : Seguridad, auditoría, respaldo y recuperación	175
7.1.1 : Seguridad y Auditoría	175
7.1.2 : Respaldo y Recuperación	176

PRESENTACIÓN

Base de Datos Avanzado II es un curso que pertenece a la línea de base de datos y se dicta en las carreras Computación e Informática, y Administración y Sistemas. Brinda un conjunto de herramientas que permite a los alumnos implementar soluciones en una BD Oracle que satisfacen necesidades de negocio, y asegura un buen rendimiento y la alta disponibilidad de los datos.

El manual para el curso ha sido diseñado bajo la modalidad de unidades de aprendizaje, las que se desarrollan durante semanas determinadas. En cada una de ellas, hallará los logros que debe alcanzar al final de la unidad; el tema tratado, el cual será ampliamente desarrollado; y los contenidos que debe desarrollar, es decir, los subtemas. Por último, encontrará las actividades que deberá desarrollar en cada sesión que le permitirán reforzar lo aprendido en la clase.

El curso es eminentemente práctico consiste en un taller de programación. En primer lugar, se inicia con una introducción a una BD Oracle, se muestra como configurar una conexión remota y se hace un recorrido por las herramientas básicas de comunicación. Luego, se hace un repaso de las principales sentencias del Lenguaje de Manipulación de Datos así como de las principales funciones predefinidas. Continúa con la creación de estructuras de datos en una BD Oracle. Luego, se presentan y desarrollan los conceptos básicos en programación PL/SQL. Continúa con la programación dentro de una BD Oracle. A continuación se desarrollan conceptos de Administración de una base de datos Oracle como la gestión del Sistema de Archivos Oracle. Se concluye con la revisión de conceptos de seguridad, auditoría, mecanismos de respaldo y recuperación en una base de datos Oracle.

**UNIDAD DE
APRENDIZAJE****1**

INTRODUCCIÓN A UNA BD ORACLE

LOGRO DE LA UNIDAD DE APRENDIZAJE

Al término de la unidad, el alumno detalla la arquitectura de la base de datos Oracle y configura la conexión de un cliente remoto a través del archivo tnsnames.ora. Asimismo manipula la información mediante la herramienta SQL*PLUS y gestiona una instancia de base de datos

TEMARIO

1.1 Tema 1 : INTRODUCCIÓN AL SGBD ORACLE

- 1.1.1 : Introducción a una DB relacional y a la arquitectura Oracle
- 1.1.2 : Interactuando con la base de datos: Conexión y SQL*PLUS
- 1.1.3 : Componentes de la Arquitectura de una BD Oracle

1.2 Tema 2 : GESTIÓN DE UNA INSTANCIA ORACLE

- 1.2.1 : Inicio de la instancia
- 1.2.2 : Apertura de la Base de Datos
- 1.2.3 : Cierre de la Base de Datos
- 1.2.4 : Configuración de un cliente Oracle

ACTIVIDADES PROPUESTAS

- Recordar el concepto de Modelo Relacional
- Identificar los principales componente de una base de datos Oracle
- Interactuar con el SQL*Plus para acceder a una base de datos Oracle
- Revisar los componentes de la Arquitectura de una base de datos Oracle

1.1 INTRODUCCIÓN AL SGBD ORACLE

1.1.1 Introducción a una DB relacional y a la arquitectura Oracle

1.1.1.1 Modelo Relacional

El modelo relacional para la gestión de una base de datos es un modelo de datos basado en la lógica de predicado y en la teoría de conjuntos. Es el modelo más utilizado en la actualidad para modelar problemas reales y administrar datos dinámicamente.

En este modelo, todos los datos son almacenados en relaciones y como cada relación es un conjunto de datos, el orden en el que estos se almacenen no tiene mayor relevancia (a diferencia de otros modelos como el jerárquico y el de red). Esto tiene la considerable ventaja de que es más fácil de entender y de utilizar por un usuario no experto. La información puede ser recuperada o almacenada por medio de «consultas» que ofrecen una amplia flexibilidad y poder para administrar la información.

Este modelo considera la base de datos como una colección de relaciones. De manera simple, una relación representa una tabla que no es más que un conjunto de filas, cada fila es un conjunto de campos y cada campo representa un valor que interpretado describe el mundo real. Cada fila también se puede denominar tupla o registro y a cada columna también se le puede llamar campo o atributo.

Una tabla es una estructura lógica que sirve para almacenar los datos de un mismo tipo (desde el punto de vista conceptual). Almacenar los datos de un mismo tipo no significa que se almacenen sólo datos numéricos, o sólo datos alfanuméricos. Desde el punto de vista conceptual, esto significa que cada entidad se almacena en estructuras separadas.

Por ejemplo, la entidad factura se almacena en estructuras diseñadas para ese tipo de entidad: la tabla FACTURA, la tabla FACTURA_COMPRA, etc. Así, cada entidad, tendrá una estructura (tabla) pensada y diseñada para ese tipo de entidad. Cada elemento almacenado dentro de la tabla recibe el nombre de registro o fila. Así, si la tabla FACTURA almacena 1000 facturas, se dice que la tabla FACTURA contiene 1000 registros o filas.

Una tabla se compone de campos o columnas, que son conjuntos de datos del mismo tipo (desde el punto de vista físico). Ahora, cuando decimos “del mismo tipo” queremos decir que los datos de una columna son todos del mismo tipo: numéricos, alfanuméricos, fechas, etc.

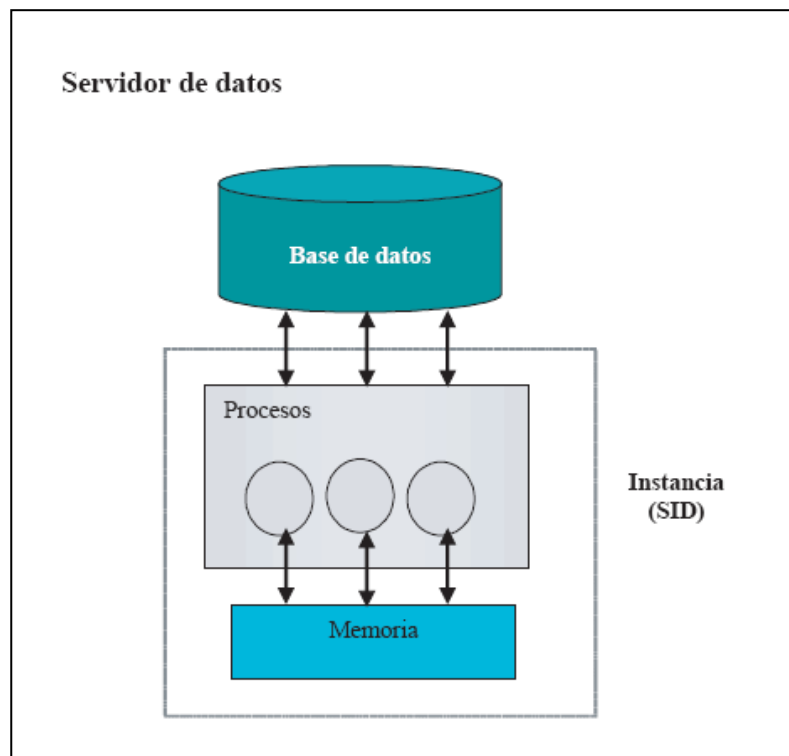
A la vez, también, puede existir la relación entre tablas que tienen un campo o atributo común. Por ejemplo, la tabla DETALLE que contiene todos los “items” correspondiente a cada factura de la tabla FACTURA.

1.1.1.2 Base de Datos Relacional

Es un programa residente en memoria que se encarga de gestionar el tratamiento de entrada, salida, protección y elaboración de la información que almacena.

Aunque, aparentemente, podamos pensar que una Base de datos es un conjunto de solo archivos donde se almacena la información, en realidad, eso no es así. El corazón de una base de datos es el motor que es el programa que debe estar ejecutándose en una máquina para gestionar los datos. Además de este programa y los archivos con datos, existen otras utilidades auxiliares, como programas para realizar copias de seguridad, administración, etc.

Oracle es una base de datos relacional para entornos cliente/servidor, es decir, que aplica las normas del álgebra relacional (conjuntos, uniones, intersecciones, etc.) y que utiliza la arquitectura cliente/servidor, donde en un lado de la red está el servidor con los datos y en el otro lado están los clientes que "interrogan" al servidor.



1.1.1.3 Funciones de la Base de Datos

- Permitir la introducción de datos por parte del usuario
- Salida de datos
- Almacenamiento de datos
- Protección de datos (seguridad e integridad)
- Elaboración de datos

1.1.1.4 Conocimientos Necesarios

Básicamente, la relación del usuario-programador con la base de datos se hace a través de un lenguaje denominado SQL: Structure Query Language (Lenguaje Estructurado de Consultas).

Para un programador de base de datos, el conocimiento mínimo debe comprender lo siguiente:

- Conocimiento básico de las estructuras internas de Oracle
- Lenguaje SQL
- Utilidades básicas (SQL*PLUS, Export, Import, etc.)
- Lenguaje de programación PL/SQL
- Tareas simples de administración
- Tunning (afinamiento) básico de sentencias SQL

Las tareas propias de un administrador de bases de datos pueden ser las siguientes:

- Los conocimientos propios de un programador de base de datos
- Conocimiento profundo de las estructuras internas de Oracle
- Conocimiento profundo de los catálogos
- Conocer utilitarios de Administración (Ejemplo: Oracle Enterprise Manager)
- Afinamiento avanzado de SQL, red, memoria, discos, CPU, etc.

1.1.1.5 Componentes básicos de una base de datos Oracle

- **Motor:** Programa ejecutable que debe estar en memoria para manejar la base de Datos. Cuando este programa está ejecutándose se dice que la Base de datos está levantada.
- **Servicio de red:** Programa que se encarga de establecer las conexiones y transmitir datos entre cliente y servidor o entre servidores. En Oracle es el protocolo Net8.
- **Listener** (Escuchador): Programa residente en memoria que se encarga de recibir las llamadas que llegan a la base de datos desde la red y de pasárselas a ésta. Una base de datos que no tenga un listener cargado, no podrá recibir llamadas remotas. El listener se comunica con el servicio de red.
- **Utilitarios:** Intérpretes de consultas, Programas de Administración de base de datos, Programas de copia de seguridad, monitores de rendimiento.

A todo este conjunto se le llama RDBMS: Relational Database Manager System – Sistema de Gestión de Base de Datos Relacionales.

1.1.1.6 LENGUAJE SQL

El lenguaje estructurado de consultas es un conjunto de sentencias u órdenes que todos los programas y usuarios deben utilizar para acceder a bases de datos Oracle. No hay otra manera de comunicarse con Oracle si no a través de SQL.

Dado que SQL es un estándar, todas las bases de datos comerciales en la actualidad utilizan SQL como puente de comunicación entre la base de datos y el usuario.

En realidad, SQL no es un lenguaje en sí, como podría ser un lenguaje de programación de 3ra generación (C, Pascal, etc.), sino que es un sublenguaje orientado a acceso y manipulación de bases de datos relacionales.

Se dice que SQL es estructurado porque trabaja con conjuntos de resultados (result set) abstractos como unidades completas.

Un conjunto de resultados es un esquema básico de una tabla: M Filas x N columnas. Este esquema se trata como un todo y es la idea principal de SQL.

Lo anterior es aplicable también cuando existe más de una tabla en un mismo SQL, el conjunto de resultados sigue siendo el resultado de una matriz de M Filas x N columnas, el cual puede ser filtrado (reducido) usando la muy conocida cláusula WHERE.

1.1.1.7 LENGUAJE PL/SQL

Ya se mencionó anteriormente que SQL es un lenguaje de comandos, no un lenguaje de programación con todas las estructuras de control típicas. Así, SQL, sólo contempla instrucciones, más o menos simples, pero no tiene ningún tipo de instrucciones de control de flujo o de otro tipo más propias de los lenguajes de programación de tercera generación (3GL).

Para subsanar esta carencia, Oracle definió un lenguaje de programación de tercera generación, que admitía sentencias SQL embebidas. Este es el PL/ SQL (Procedural Language o Programming language).

La idea básica sobre la que se sustenta el PL/SQL es aplicar las estructuras típicas de un lenguaje de programación (bifurcaciones, bucles, funciones, etc) a las sentencias SQL típicas.

1.1.2 Interactuando con la base de datos: Conexión y SQL *PLUS

SQL *PLUS, es una herramienta indispensable para un administrador de base de datos.

Es un entorno en modo texto y no contiene un motor PL/SQL local, es decir, las instrucciones o comandos se envían directamente a la base de datos.

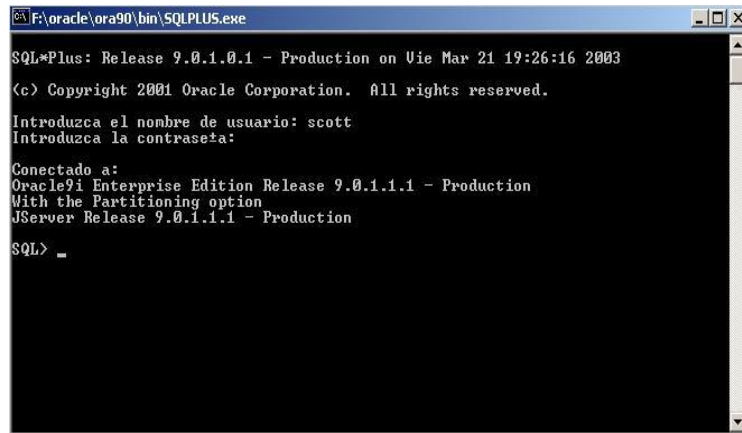
SQL *PLUS quizás sea la más sencilla de las herramientas de desarrollo de PL/SQL. Permite al usuario introducir instrucciones SQL y bloques PL/SQL de forma interactiva mediante una línea de comandos.

Generalmente, SQL *PLUS se distribuye junto con el servidor de Oracle y se encuentra disponible como parte de la instalación estándar de Oracle.

Dado que SQL *PLUS es un tema tan amplio, que no se puede estudiar en una sola sesión de clases, se van a estudiar principalmente las funciones que interesan a los DBA (database administrator).

1.1.2.1 Entorno del SQL *PLUS

El SQL *PLUS tiene un entorno orientado al carácter.



```
F:\oracle\ora90\bin\SQLPLUS.exe

SQL*Plus: Release 9.0.1.0.1 - Production on Tue Mar 21 19:26:16 2003
(c) Copyright 2001 Oracle Corporation. All rights reserved.

Introduzca el nombre de usuario: scott
Introduzca la contraseña:
Conectado a:
Oracle9i Enterprise Edition Release 9.0.1.1.1 - Production
With the Partitioning option
JServer Release 9.0.1.1.1 - Production

SQL> _
```

1.1.2.2 Conexión con la Base de Datos

Antes de realizar cualquier instrucción en la base de datos, es necesario establecer una conexión con el servidor de datos. Esta operación se puede realizar de una de las siguientes formas:

Mediante el paso de un identificador de usuario y una contraseña y/o una cadena de conexión en la línea de comandos utilizada para iniciar SQL *PLUS para el entorno orientado al carácter.

Una primera forma sería después de ejecutar el SQL *PLUS desde el explorador de Windows:

```
SQL*Plus: Release 11.2.0.1.0 Production on Lun Ago 1 19:13:09 2011
Copyright (c) 1982, 2010, Oracle. All rights reserved.

Introduzca el nombre de usuario: SCOTT
Introduzca la contraseña: TIGER
Conectado a:
Oracle11g Enterprise Edition Release 9.0.1.1.1 - Production with the partitioning
option Jserver Release 9.0.1.1.1 - Production
```

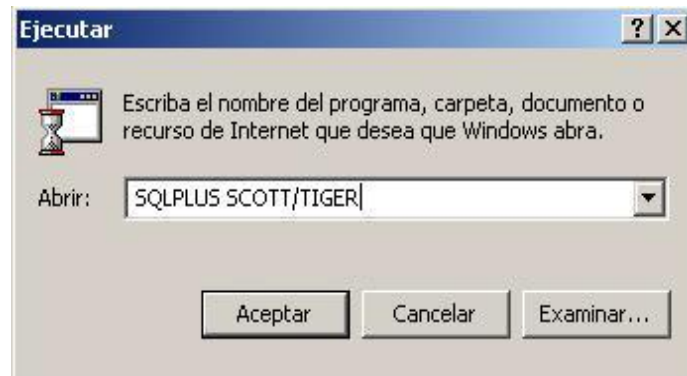
O también especificando la cadena de conexión:

```
SQL*Plus: Release 11.2.0.1.0 Production on Lun Ago 1 19:13:09 2011
Copyright (c) 1982, 2010, Oracle. All rights reserved.

Introduzca el nombre de usuario: SCOTT/TIGER@CIBERTEC
Conectado a:
Oracle11g Enterprise Edition Release 9.0.1.1.1 - Production with the
partitioning option Jserver Release 9.0.1.1.1 - Production

SQL>
```

Una segunda forma sería iniciando el SQL *PLUS desde el menú ejecutar del botón de inicio:



- Mediante el uso del comando CONNECT, una vez dentro de SQL*PLUS (en ambos entornos).

```
SQL> CONNECT
Introduzca el nombre de usuario: SCOTT
Introduzca la contraseña: *****
Conectado.
SQL>
```

1.1.2.3 Comandos de Configuración de Sesión

Existe un grupo de comandos que nos permiten configurar el SQL *PLUS mientras estamos conectados a la base de datos. Algunos de los comandos se muestran en la siguiente tabla:

COMANDO	ABREVIATURA	DESCRIPCION
SET COMPATIBILITY	SET COM V7 / V8 native	Configura la compatibilidad con la versión de oracle espec.
SET NUMFORMAT	SET NUM \$ 999.999.99	Configura el formato numérico de salida de las consultas
SET PAGESIZE	SET PAGES 250	Configura el número de líneas por paginas
SET LINESIZE	SET LINE 250	Configura el número total de caracteres por línea
SET SERVEROUTPUT {ON / OFF} [SIZE n] [FORMAT {wrapped / truncate/word_wrapped }]	SET SERVEROUT ON	Activa o desactiva la salida en pantalla provocadas por los procedimientos almacenados o bloques PL/SQL en SQL*PLUS que utilizan el paquete DBMS_OUTPUT.PUT_LINE
SET EDITFILE	SET EDITF "file.sql"	Permite cambiar el nombre del archivo que abre y crea el editor de comandos del sistema operativo

Por ejemplo, el comando LINESIZE nos permite determinar cuántos caracteres se mostrarán por cada línea de texto que devuelven las consultas a la base de datos. La cantidad por defecto es 80 caracteres. Para poner en práctica el efecto del comando, hagamos la siguiente consulta:


```
SQL> SELECT * FROM emp;
```

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	17/12/80	800		20
7499	ALLEN	SALESMAN	7698	20/02/81	1600	300	30
7521	WARD	SALESMAN	7698	22/02/81	1250	500	30

...ésta será la salida por pantalla. Ahora, si aumentamos la cantidad de caracteres por línea de texto, el resultado de la consulta se mostraría más ordenado.

```
SQL> SET LINESIZE 200
SQL> SELECT * FROM emp;
```

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	17/12/80	800		20
7499	ALLEN	SALESMAN	7698	20/02/81	1600	300	30
7521	WARD	SALESMAN	7698	22/02/81	1250	500	30
7566	JONES	MANAGER	7839	02/04/81	2975		20

.....
14 filas seleccionadas.

1.1.2.4 Ejecutando archivos de comandos desde SQL *PLUS

Para ejecutar archivos que contengan comandos válidos, se utiliza el comando START o el carácter @. Ejemplo:

```
SQL> START @unidad:\ruta\mi_archivo;
```

o también:

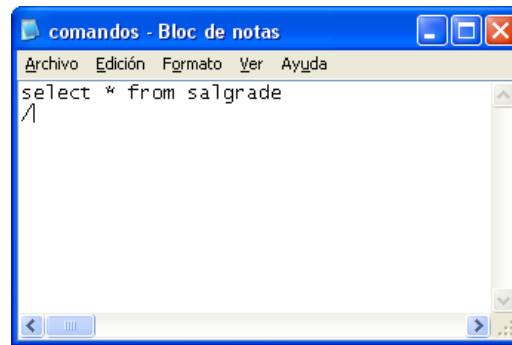
```
SQL> @unidad:\ruta\mi_archivo;
```

La extensión del archivo por defecto es .sql. Podemos utilizar archivos con extensión diferente al por defecto (cualquier archivo que pueda ser leído con un editor de texto) pero debemos especificar la extensión en el comando de ejecución. Por ejemplo, para ejecutar un archivo con extensión .txt: este sería el comando:

```
SQL> @unidad:\ruta\mi_archivo.txt;
```

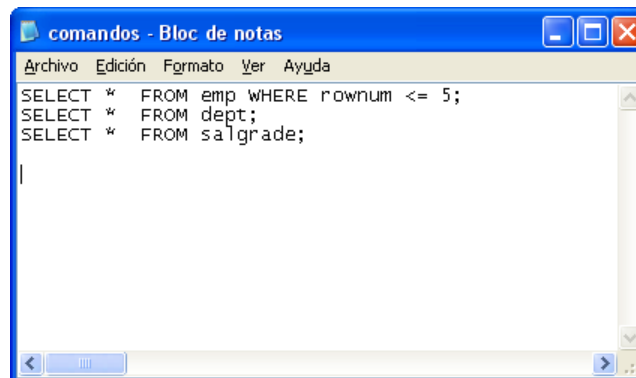
Hagamos el siguiente caso práctico. Vamos crear un archivo llamado comandos.txt utilizando el editor de Windows.

```
SQL> EDIT
```



Editamos el buffer (comandos.sql) ingresando los siguientes comandos SQL:

```
SELECT * FROM emp WHERE rownum <= 5;
SELECT * FROM dept;
SELECT * FROM salgrade;
```



Luego grabemos el archivo con el nombre C:/COMANDOS.TXT utilizando la ruta por defecto.

Ahora ejecutamos el archivo con el comando START o con el carácter @ de la siguiente forma:

```
SQL> START C:/comandos.txt o también SQL> @ C:/comandos.txt
```

Si ha creado carpetas cuyo nombre tengan espacio en blanco o el nombre del archivo también tenga espacios en blanco, debe encerrar en apóstrofes el archivo a ejecutar. Ejemplo:

```
SQL> @'c:\curso lp6\mis comandos.txt';
```

1.1.2.5 Uso comando SPOOL en SQL *PLUS

Este comando es de gran utilidad para un operador del SQL *PLUS. Pues este comando permite registrar, en un archivo especificado, los resultados de nuestras consultas y operaciones realizadas a la base de datos. No solamente registra los resultados sino que también registra los comandos utilizados. La sintaxis es la siguiente:

```
SPOOL [nombre_archivo[.extension]]
```

Ejemplo: los siguientes comandos serán registrados en el archivo sesion02.txt cuando se cierre el spool a través del comando SPOOL OFF.

```
SQL> SPOOL sesion02.txt
SQL> SET LINESIZE 150
SQL> SELECT * FROM emp;
```

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	17/12/80	800		20
7499	ALLEN	SALESMAN	7698	20/02/81	1600	300	30
7521	WARD	SALESMAN	7698	22/02/81	1250	500	30
7566	JONES	MANAGER	7839	02/04/81	2975		20

.....
14 filas seleccionadas.

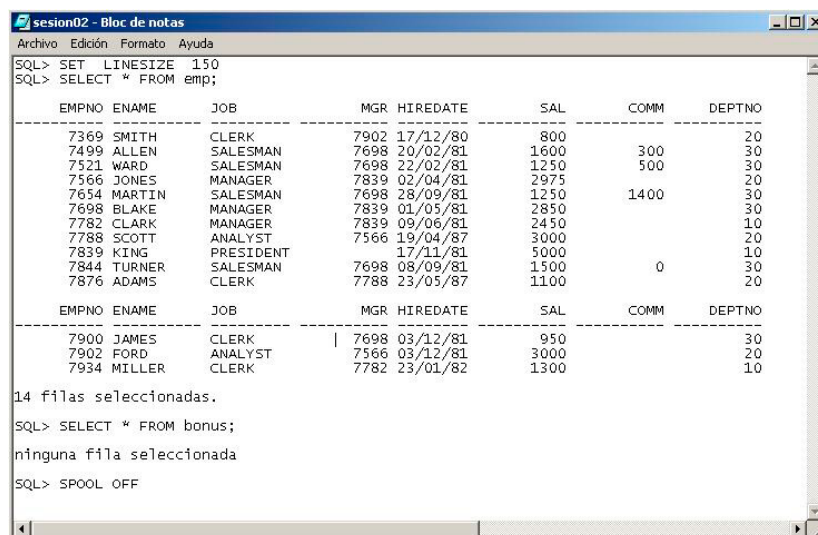
```
SQL> SELECT * FROM bonus;
Ninguna fila seleccionada

SQL> SPOOL OFF
```

Cuando se crea el archivo sesion02.txt, este archivo se mantendrá abierto y vacío, hasta que no se especifique que se cierre el spool (SPOOL OFF). Si usted no especifica la extensión al archivo a través del comando SPOOL automáticamente el archivo se le asignará la extensión LST.

Si editamos el archivo sesion02.txt creado con el comando spool encontraremos lo siguiente:

```
SQL> EDIT sesion02.txt
```



```
sesion02 - Bloc de notas
Archivo Edición Formato Ayuda

SQL> SET LINESIZE 150
SQL> SELECT * FROM emp;
```

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	17/12/80	800		20
7499	ALLEN	SALESMAN	7698	20/02/81	1600	300	30
7521	WARD	SALESMAN	7698	22/02/81	1250	500	30
7566	JONES	MANAGER	7839	02/04/81	2975		20
7654	MARTIN	SALESMAN	7698	28/09/81	1250	1400	30
7698	BLAKE	MANAGER	7839	01/05/81	2850		30
7782	CLARK	MANAGER	7839	09/06/81	2450		10
7788	SCOTT	ANALYST	7566	19/04/87	3000		20
7839	KING	PRESIDENT		17/11/81	5000		10
7844	TURNER	SALESMAN	7698	08/09/81	1500	0	30
7876	ADAMS	CLERK	7788	23/05/87	1100		20

```

EMPNO  ENAME      JOB              MGR      HIREDATE          SAL       COMM      DEPTNO
-----
7900    JAMES      CLERK            7698     03/12/81           950              30
7902    FORD       ANALYST          7566     03/12/81          3000              20
7934    MILLER     CLERK            7782     23/01/82          1300              10

14 filas seleccionadas.

SQL> SELECT * FROM bonus;
ninguna fila seleccionada

SQL> SPOOL OFF
```

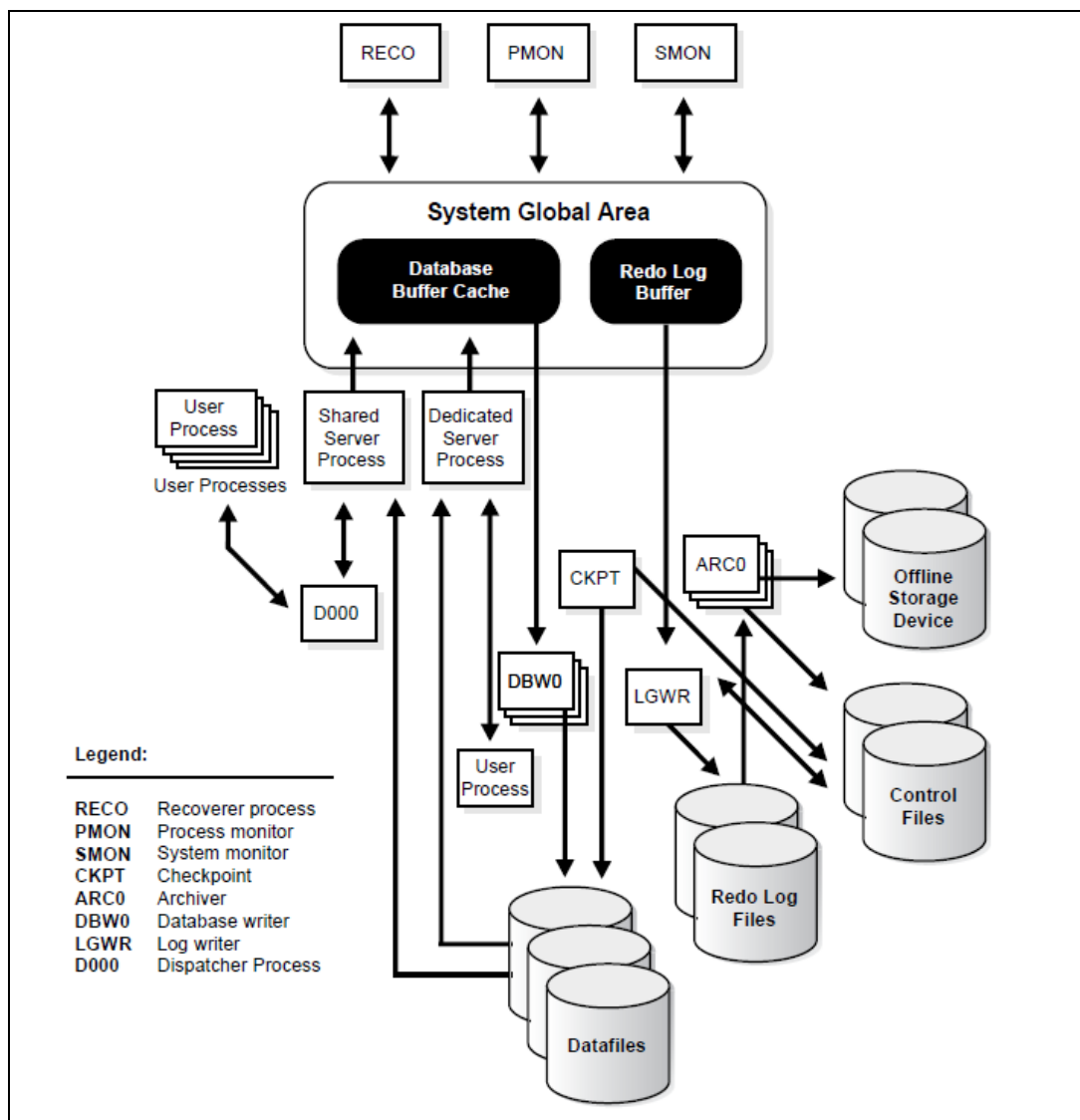
Si usted vuelve a crear el archivo sesion02.txt con el comando SPOOL, para nuestro caso el archivo ya existe, lógicamente el archivo se volverá a crear perdiéndose el contenido del mismo. Tenga cuidado al especificar el nombre del archivo. Asegúrese de que el archivo a crear no exista.

Por ejemplo, este comando volverá a crear el archivo sesion02.txt y se perderá nuestro registro de comandos realizados anteriormente.

SQL> SPOOL sesion02.txt

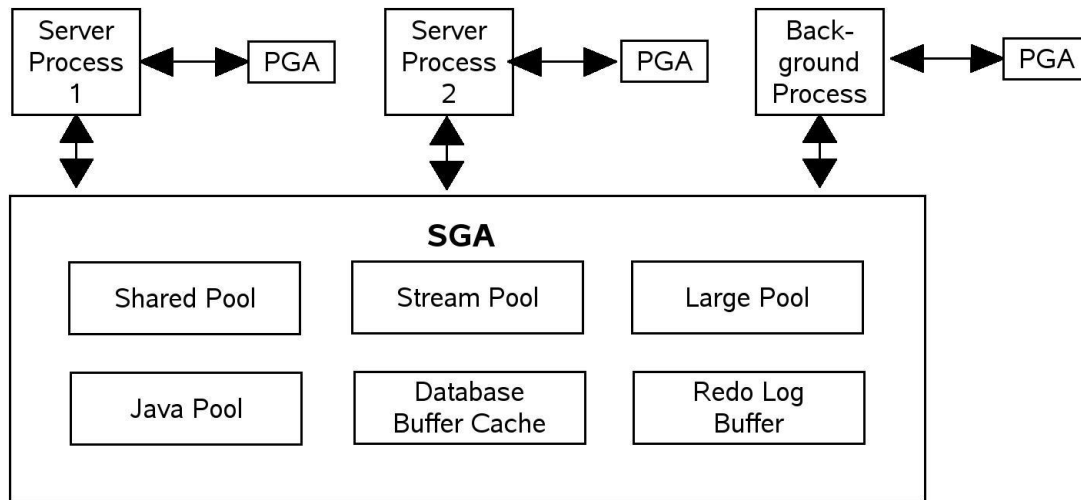
1.1.3 Componentes de la Arquitectura de una Base de Datos

La arquitectura de ORACLE tiene tres componentes básicos: las estructuras de memoria para almacenar los datos y el código ejecutable, los procesos que corren el sistema de bases de datos y las tareas de cada usuario conectado a la base de datos y los archivos que sirven para el almacenamiento físico, en disco, de la información de la base de datos.



1.1.3.1 Estructuras de memoria

Hay dos clases de memoria, una de ellas compartida por todos los usuarios conectados y otra, dedicada al trabajo de cada uno de ellos.



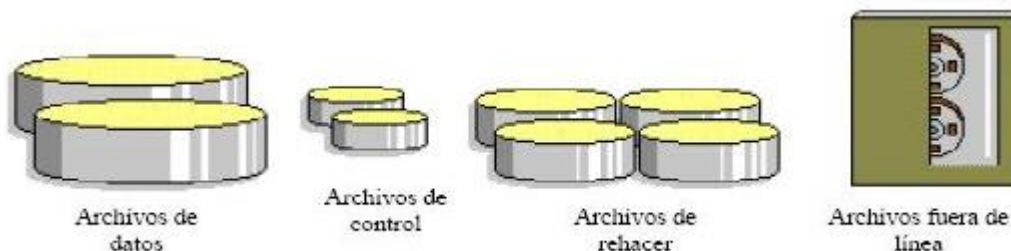
El área global del sistema SGA (system global area), es el área compartida por todos los usuarios y se divide en tres partes:

- Fondo común compartido (Shared pool), en ella mantiene el diccionario de datos y las áreas compartidas de las órdenes SQL que se solicitan para su procesamiento.
- Área de memoria rápida (Database buffer cache), donde mantiene los datos traídos por las órdenes SQL de los usuarios conectados a la base de datos.
- Área de registros de rehacer (Redo log buffer), aquí se registran los cambios hechos a la base de datos.

Por cada sesión de usuario, se crea también, en memoria, un área específica llamada área global de programa o PGA (program global area). Esta área no se comparte con las otras sesiones de usuario.

1.1.3.2 Archivos de la base de datos

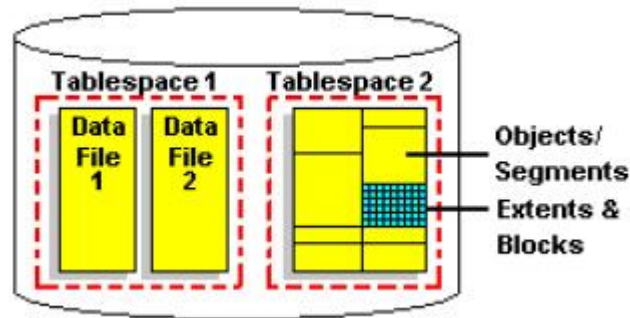
Los archivos que maneja ORACLE se clasifican en cuatro grupos:



A. Los Archivos de Datos (Datafiles)

Estos archivos sirven para el almacenamiento físico de las tablas, índices y agrupamientos (clusters), y procedimientos. Estos archivos, son los únicos que contienen los datos de los usuarios de la base de datos.

Las unidades lógicas más grandes manejadas por ORACLE, para el almacenamiento de los datos, son llamadas espacios de tablas (tablespaces) que le permiten manejar y controlar el espacio en los discos.



No es necesario que todos los espacios de tablas estén en un mismo disco. Cuando se crean en distintos discos se busca un mejor desempeño y mejor manejo del espacio de almacenamiento.

Una base de datos puede tener un solo espacio de tablas, pero, por las razones anteriores, se recomienda varios espacios de tablas. Como mínimo, se debe tener un espacio de tablas del sistema (SYSTEM), un espacio de tablas por cada aplicación, un espacio de tablas para los usuarios y otro espacio de tablas para los índices.

El espacio de tablas SYSTEM se crea automáticamente cuando se crea una base de datos. Allí se guardan los archivos de control y el diccionario de datos y toda la información de los procedimientos almacenados.

El DBA (Administrador de Base de Datos) puede crear un espacio de tablas con una orden, como la siguiente:

```
CREATE TABLESPACE indices datafile 'discod/bd/datosl.dbf' size 300m;
```

Los archivos de datos (datafiles) almacenan los datos del usuario. Se requiere como mínimo uno para una base de datos.

Cuando se agote el espacio, un DBA tiene dos alternativas:

- Adicionar un nuevo archivo de datos con la orden ALTER

```
ALTER TABLESPACE indices add datafile 'discod/bd/datos3.dbf' size 150m
```

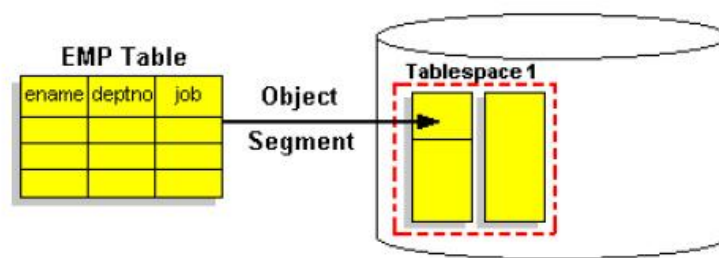
- Crear un nuevo espacio de tablas como se mostró previamente.

En el momento de la creación de una base de datos, el DBA debe planear o estimar los requerimientos de almacenamiento y, también, el nombre, tamaño y localización de

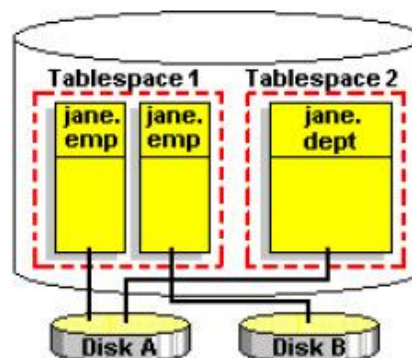
los archivos de datos, junto con el número máximo de archivos de datos permitido para la base de datos.

El DBA puede crear varios espacios de tablas (tablespaces) en discos separados para planear el crecimiento de la base de datos y hacer una mejor administración de la base de datos.

Un objeto de datos, por su parte, es una estructura lógica que puede ser una tabla, un archivo de índice, un archivo temporal, un archivo de deshacer o un cluster. Estos objetos se almacenan físicamente en segmentos que se componen de extensiones (extents).



A su vez, una extensión está hecha de bloques que, de acuerdo con el sistema operativo subyacente, puede tener un número determinado de bytes y que el DBA especifica, en el momento de la creación de la base de datos. El tamaño del bloque es dependiente del sistema operativo y nunca puede ser menor al que éste maneja.



En una base de datos, pueden existir otros objetos que no contienen datos como las vistas, los sinónimos y las secuencias. Sin embargo, todo objeto, independientemente de si contiene datos o no, debe pertenecer a un esquema.

Por eso, una colección de objetos de un usuario se denomina esquema.

Un objeto se puede crear en un esquema de tres formas:

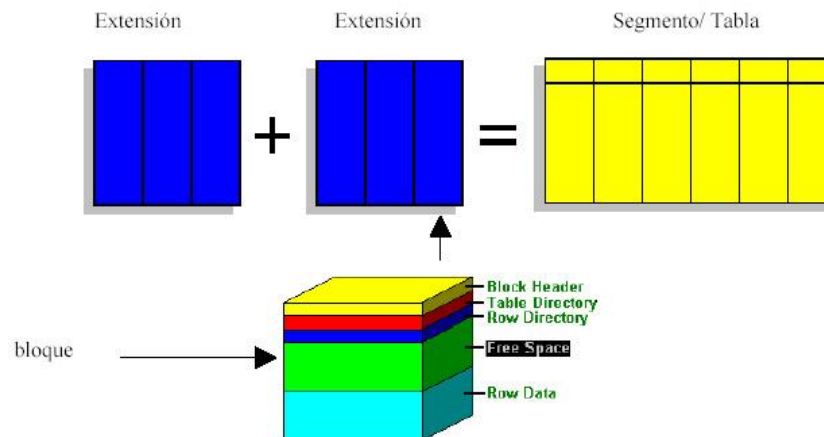
- Si un usuario da una orden de creación de un objeto, por defecto, el sistema lo crea en su propio esquema.
- Copiando el objeto de otro usuario (al nombre de un objeto siempre se le antepone el nombre del esquema, por ejemplo `juan.empleado`) con una orden como:

CREATE TABLE empleado as SELECT * from scott.emp;

- Otro usuario lo crea para uno, como en la orden:

CREATE TABLE juan.proyecto (codigo number primay key,) tablespace planeacion storage (initial 1000 next 1000 minextents 1 maxextents 6)

Reglas para el almacenamiento de objetos en la base de datos



- Un objeto puede almacenarse en uno o más archivos de datos (datafiles), pero en un solo espacio de tablas (tablespace).
- Dos objetos diferentes de un esquema pueden estar en distintos tablespaces.
- Los objetos pueden almacenarse en múltiples discos. Por ejemplo, parte de jane.emp es almacenado en el archivo de datos 1 sobre el disco A y parte en el archivo de datos 2 sobre el disco B.

B. Archivos de control (Control Files)

Tienen la descripción física y dirección de los archivos de la base de datos y de los archivos de rehacer para el arranque correcto de la base de datos. En estos archivos se especifican cuáles datafiles conforman la base de datos para poder tener acceso a los datos o para poder recuperar la base de datos, ante una falla.

Los archivos de control se crean automáticamente cuando se da una orden CREATE DATABASE y no son editables, pues también se actualizan automáticamente.

C. Archivos de rehacer (redo log files)

Tienen los cambios hechos a la base de datos para la recuperación ante fallas o para el manejo de las transacciones. Poseen los valores antes de una transacción, la orden ejecutada y, opcionalmente, el valor después de la transacción. El principal propósito de estos archivos es servir de respaldo de los datos en la memoria RAM. Este conjunto de archivos debe estar conformado por dos grupos, como mínimo, y se recomienda que cada grupo esté almacenado en discos separados. El DBMS utiliza la

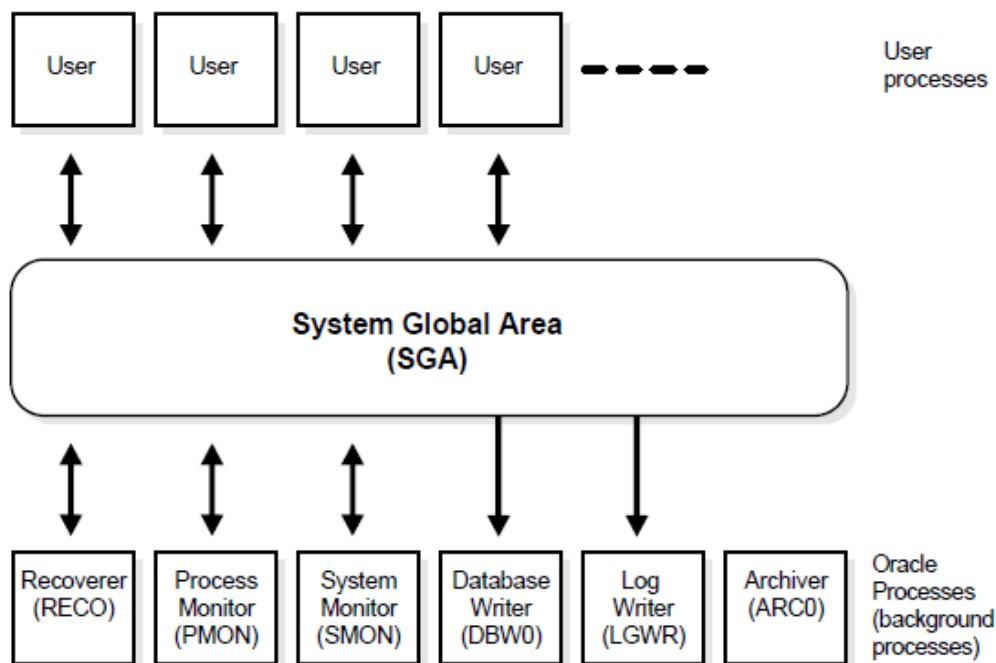
técnica de ir sobrescribiendo sobre la información más vieja, cuando se agota el espacio en estos grupos de archivos. Se puede decir que guarda las instrucciones SQL que se van realizando en toda la base de datos.

D. Archivos fuera de línea (archived files)

Son archivos opcionales donde se guarda información vieja de los archivos de rehacer, muy convenientes para los respaldos de la base de datos.

E. Procesos

Los procesos son programas que se ejecutan para permitir el acceso a los datos. Los procesos se cargan en memoria y son transparentes para los usuarios. Los procesos se clasifican en tres grupos: procesos de base, de usuario y procesos servidores.



Procesos de Base o de Soporte

Los procesos de base (background) son los que se encargan de traer datos desde y hacia la SGA, mejorando el desempeño al consolidar las tareas que son impartidas por todos los usuarios. Cada proceso de base tiene su propia área de memoria. Los procesos de base o soporte son los siguientes:

DBWR: (Database writer) se encarga de manejar los “buffers” de memoria cache para que los procesos del usuario siempre encuentren a algunos unos de ellos disponibles. Es un proceso obligatorio que, además, escribe los bloques de datos modificados por los usuarios en los archivos de datos que componen la B.D cuando el proceso LGWR le envía el mensaje de hacerlo.

LGWR: (Log writer) este proceso escribe datos desde la SGA a los archivos de rehacer (redo log files) que sirven en caso de fallas en la instancia. Este proceso es obligatorio y es el único encargado de escribir y leer en estos archivos. El proceso de llenado de estos archivos es circular, por lo tanto, antes de empezar a sobrescribir en uno de ellos, se marca un punto de verificación y LGWR envía la orden de escritura en los datafiles al proceso DBWR. Cuando se cambia de uso de grupo de redo log (archivo deshacer), se produce un SWITCH LOG

LCKn, Lock: (lock processes) El bloqueo es un proceso opcional. Efectúa los bloqueos entre instancias, en caso de ambientes con servidores paralelos (hasta con 10 servidores).

CKPT: (Check point) El punto de comprobación es un proceso opcional que ocurre cuando los usuarios conectados a la base de datos, hacen solicitudes de exámenes de datos. Uno de los eventos que dispara a este proceso es el SWITCH LOG.

SNPn: (Snapshot process) se encarga de refrescar los snapshots o réplicas de tablas que se usan, principalmente, en ambientes distribuidos.

SMON: (System monitor) recupera el sistema ante una falla de la instancia.

RECO: (Recovery) recupera ante las fallas, en una transacción en ambientes distribuidos.

ARCH: (Archive) copia los registros de rehacer de la RAM en archivos de datos (archive redo logs) que permiten la recuperación cuando se presentan fallas de los medios magnéticos.

PMON: (Process Monitor) recupera la instancia ante una falla de un proceso de usuario; libera los recursos del proceso que falló.

Procesos del Usuario

Cuando un usuario se conecta a la base de datos, se crea un proceso de usuario que se encarga de ejecutar el código de aplicación del usuario y manejar el perfil del usuario con sus variables de ambiente. Los procesos de usuario no se pueden comunicar directamente con la base de datos, únicamente, lo hacen a través de procesos servidores.

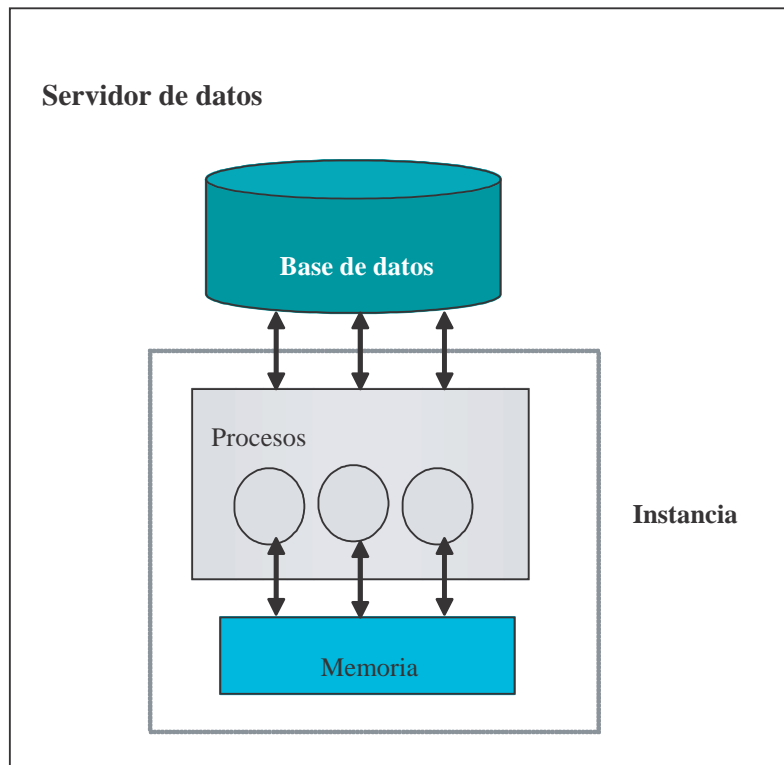
Procesos Servidores

Ejecutan las órdenes SQL de los usuarios y llevan los datos al “database buffer cache” para que los procesos del usuario puedan tener acceso a los datos. Se pueden tener distintas arquitecturas para trabajar en ORACLE, según los tipos de servidores: dedicados o multihilos.

Una configuración dedicada significa que cada conexión de un usuario de Base de Datos será atendida exclusivamente por un proceso servidor. Una configuración multihilo (multithread) o compartida es cuando existe un número limitado de procesos servidores que atienden a todas las conexiones de Bases de Datos existentes cuando haya un requerimiento de por medio. En esta última configuración, existen los despachadores (dispatchers), que son los que ante algún requerimiento de sesión asignan dicho trabajo a un proceso servidor disponible.

F. Instancia de ORACLE

Se denomina instancia al conjunto de estructuras de memoria y procesos de fondo que acceden los archivos de bases de datos. Es posible que una misma base de datos sea accedida por múltiples instancias; cada una de ellas residiendo en una máquina diferente (ésta es la opción de servidores paralelos de ORACLE).



El sistema de bases de datos ORACLE, cuando inicia, sigue los pasos que se detallan a continuación:

1. Iniciar la instancia. Para hacer este paso, ORACLE lee el archivo de parámetros y configura la instancia, con base en ellos. En ese momento, se crea la SGA y se activan los procesos de base, pero aún no se puede hacer nada.

2. Montar la base de datos. Consiste en preparar el sistema para su uso trayendo a la RAM el diccionario de datos; es como poner el sistema en primera, listo para recibir algunas órdenes del DBA.

3. Abrir la base de datos. En este momento se abren los archivos y los usuarios ya pueden tener acceso a los datos.

De acuerdo cómo se defina la instancia, ORACLE, a través de sus parámetros, puede determinarse que tan poderoso y grande es el motor. Los parámetros se definen en el archivo INIT.ORA. Entre ellos se pueden mencionar los siguientes:

db_block_buffers = número de bloques de bases de datos en la SGA. Existirá un buffer por cada bloque.

db_block_size = tamaño del bloque de la base de datos. shared_pool_size = tamaño del área compartida "shared pool", en bytes.

Además, allí se especifica el número de usuarios concurrentes, el número de transacciones concurrentes y los nombres de los archivos de control para la base de datos.

Estos parámetros se pueden ajustar durante el proceso de afinamiento porque ellos inciden en el desempeño del sistema. Algunos de los parámetros son específicos a una base de datos y, por lo tanto, deben ser cambiados antes de crear una base de datos. Se incluyen en estos:

database_name = nombre de la base de datos.

db_block_size = tamaño del bloque.

Autoevaluación

- 1.- Indique cual es la diferencia entre base de datos e instancia de base de datos.
- 2.- Mencione y describa los componentes que conforman una instancia de base de datos Oracle.
- 3.- Indique cuales son las principales diferencias entre el Lenguaje SQL y el Lenguaje PL/SQL.
- 4.- ¿Qué tipo de actividades de administración de base de datos se podrían realizar utilizando el SQL Plus?

1.2 GESTIÓN DE UNA INSTANCIA ORACLE

Para iniciar una instancia y abrir la base de datos, se debe conectar como SYSDBA e introducir el comando STARTUP. A continuación, Oracle Server leerá el archivo de parámetros de inicialización y preparará la instancia de acuerdo con los parámetros de inicialización que contiene.

Existen tres productos de Oracle que puede utilizar para crear una instancia e iniciar una base de datos:

1. SQL*PLUS.
2. Recovery Manager (RMAN).
3. Oracle Enterprise Manager (OEM).

Cualquiera que fuese la herramienta que utilizaremos para crear la instancia de Oracle los pasos y comandos son siempre los mismos. La diferencia que encontraremos entre una y otra herramienta es que mientras en el SQL*PLUS tendría que crear la instancia y levantar la base de datos a través del uso de comandos, en las otras dos herramientas, sólo tiene que hacer unos cuantos clicks para hacer lo mismo.

Ahora como el SQL*PLUS va a ser la herramienta que utilizaremos durante todo el desarrollo del curso, entonces estos son los pasos para crear una instancia desde SQL*PLUS e iniciar una base de datos:

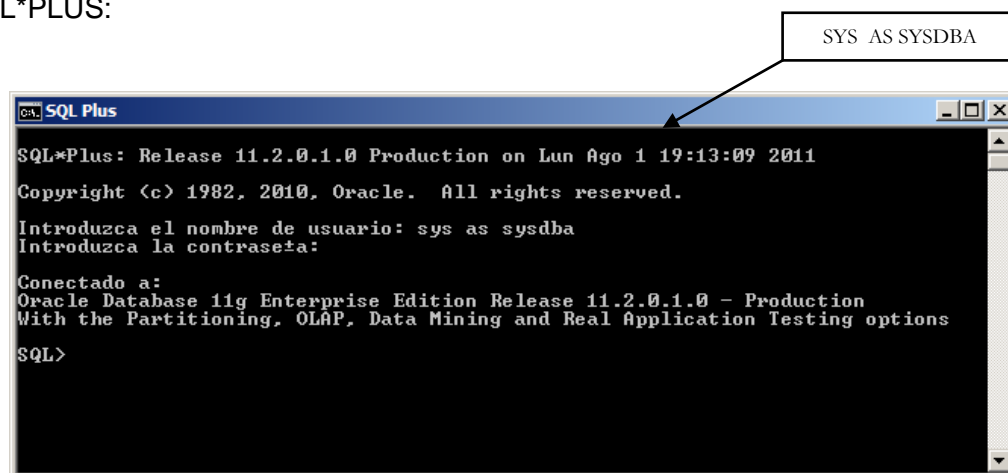
Primero: Conectarse con un usuario que tenga el privilegio SYSDBA para crear la instancia y levantar la base de datos. SYS es el usuario que inicialmente goza de este privilegio cuando la base de datos ha sido creada recientemente (*los detalles y niveles de accesos que tiene cada uno de estos usuarios son tema de administración de usuarios*). Utilizaremos al usuario SYS para poner en práctica la creación de la instancia. La clave del usuario SYS es CHANGE_ON_INSTALL

La sintaxis para conectarse como SYSDBA es la siguiente:

```
CONNECT usuario/clave[@cadena_conexión] AS SYSDBA
```

Ejemplo:

- a) Conexión a la base de datos a través de la solicitud de datos de conexión del SQL*PLUS:



b) Conexión a través del uso de comandos:

```
SQL*Plus: Release 11.2.0.1.0 Production on Lun Ago 1 19:13:09 2011  
Copyright (c) 1982, 2010, Oracle. All rights reserved.
```

```
SQL> CONNECT sys/change_on_install AS SYSDBA  
Conectado a una instancia inactiva.
```

o también utilizando la cadena de conexión

```
SQL*Plus: Release 11.2.0.1.0 Production on Lun Ago 1 19:13:09 2011  
Copyright (c) 1982, 2010, Oracle. All rights reserved.  
SQL> CONNECT sys/change_on_install@cibertec AS SYSDBA  
Conectado a una instancia inactiva.
```

Segundo: Crear la instancia e iniciar la base de datos.

```
SQL> STARTUP  
Instancia ORACLE iniciada  
Total System Global Area 118255568 bytes  
Fixed Size 282576 bytes  
Variable Size 83886080 bytes  
Database Buffers 33554432 bytes  
Redo Buffers 532480 bytes  
Base de datos montada.  
Base de datos abierta.
```

El comando STARTUP va acompañado de los siguientes parámetros:

```
STARTUP [ PFILE = init.ora ] [ NOMOUNT ] [ MOUNT ]  
[ RESTRICT ] [ FORCE ]  
[ OPEN [ RECOVER ] [ base_de_datos ] ]
```

Nota: Ésta no es la sintaxis completa.

Donde:

- OPEN: Permite a los usuarios acceder a la base de datos
- MOUNT: Monta la base de datos para ciertas actividades del DBA, aunque no permite que el usuario acceda a la base de datos
- NOMOUNT: Crea el SGA e inicia los procesos en segundo plano (background), pero no permite que el usuario tenga acceso a la base de datos
- PFILE=archivo_de_parámetros: Permite utilizar un archivo de parámetros de inicialización que no es por defecto para configurar la instancia
- FORCE: Interrumpe la instancia en ejecución antes de realizar un inicio normal.
- RESTRICT: Sólo permite que los usuarios con el privilegio RESTRICTED SESSION accedan a la base de datos
- RECOVER: Comienza la recuperación de los medios físicos cuando se inicia la base de datos

1.2.1 Inicio de la instancia

1.2.1.1 Inicio de la instancia (NOMOUNT)

Una instancia sólo se iniciaría en la etapa NOMOUNT durante la creación de la base de datos o la nueva creación de los archivos de control.

```
SQL> CONNECT sys/change_on_install@cibertec AS SYSDBA
Conectado a una instancia inactiva.
```

```
SQL> STARTUP NOMOUNT
Instancia ORACLE iniciada
Total System Global Area 118255568 bytes
Fixed Size                282576 bytes
Variable Size             83886080 bytes
Database Buffers         33554432 bytes
Redo Buffers              532480 bytes
```

El inicio de una instancia incluye las siguientes tareas:

- Lectura del archivo de inicialización desde ORACLE_HOME/database
- Asignación de SGA
- Inicio de los procesos en segundo plano
- Apertura del archivo alertSID.log y los archivos de rastreo

Estos son los comandos que necesitamos para iniciar la base de datos después de haber creado la instancia (montar y levantar la base de datos)

-- montar la base de datos (mount)

```
SQL> ALTER DATABASE MOUNT;
Base de datos montada
```

-- abrir la base de datos (open)

```
SQL> ALTER DATABASE OPEN;
Base de datos abierta
```

1.2.1.2 Montaje de la base de datos (MOUNT)

Para realizar operaciones de mantenimiento específicas, se inicia una instancia y se monta una base de datos, pero sin abrirla.

Por ejemplo, la base de datos se debe montar, pero no abrir, durante las siguientes tareas:

- Cambio del nombre de los archivos de datos
- Activación y desactivación de las opciones de archivado de archivos redo log online
- Recuperación completa de la base de datos

El montaje de una base de datos incluye las siguientes tareas:

- Asociación de una base de datos a una instancia iniciada previamente
- Ubicación y apertura de los archivos de control especificados en el archivo de parámetros
- Lectura de los archivos de control con el fin de obtener los nombres y el estado de los archivos de datos y los archivos redo log online. No obstante, no se realizan comprobaciones para verificar la existencia de los archivos de datos y los archivos redo log online en este momento.

```
SQL> CONNECT sys/change_on_install@cibertec AS SYSDBA
Conectado a una instancia inactiva.
```

```
SQL> STARTUP MOUNT
Instancia ORACLE iniciada
Total System Global Area 118255568 bytes
Fixed Size                282576 bytes
Variable Size             83886080 bytes
Database Buffers         33554432 bytes
Redo Buffers              532480 bytes
Base de datos montada.
```

Este es el comando que necesitamos para iniciar la base de datos después de haber montado la base de datos.

-- abrir la base de datos (open)

```
SQL> ALTER DATABASE OPEN;
Base de datos abierta
```

1.2.2 Apertura de la Base de Datos

1.2.2.1 Apertura de la base de datos (OPEN)

El funcionamiento normal de la base de datos significa que se inicia una instancia y la base de datos se monta y se abre. Durante el funcionamiento normal de la base de datos, cualquier usuario válido se puede conectar a la base de datos y realizar operaciones típicas de acceso a los datos.

La apertura de la base de datos incluye las siguientes tareas:

- Apertura de los archivos de datos online
- Apertura de los archivos redo log online

Si no aparece ninguno de los archivos de datos o archivos redo log online cuando se intenta abrir la base de datos, Oracle Server devuelve un error.

Durante esta etapa final, Oracle Server comprueba que todos los archivos de datos y archivos redo log online se puedan abrir y verifica la consistencia de la base de datos. Si fuera necesario, el proceso en segundo plano de SMON inicia la recuperación de la instancia.

```
SQL> CONNECT sys/change_on_install@cibertec AS SYSDBA
Conectado a una instancia inactiva.
```

```
SQL> STARTUP
```

Instancia ORACLE iniciada

Total System Global Area 118255568 bytes

Fixed Size 282576 bytes

Variable Size 83886080 bytes

Database Buffers 33554432 bytes

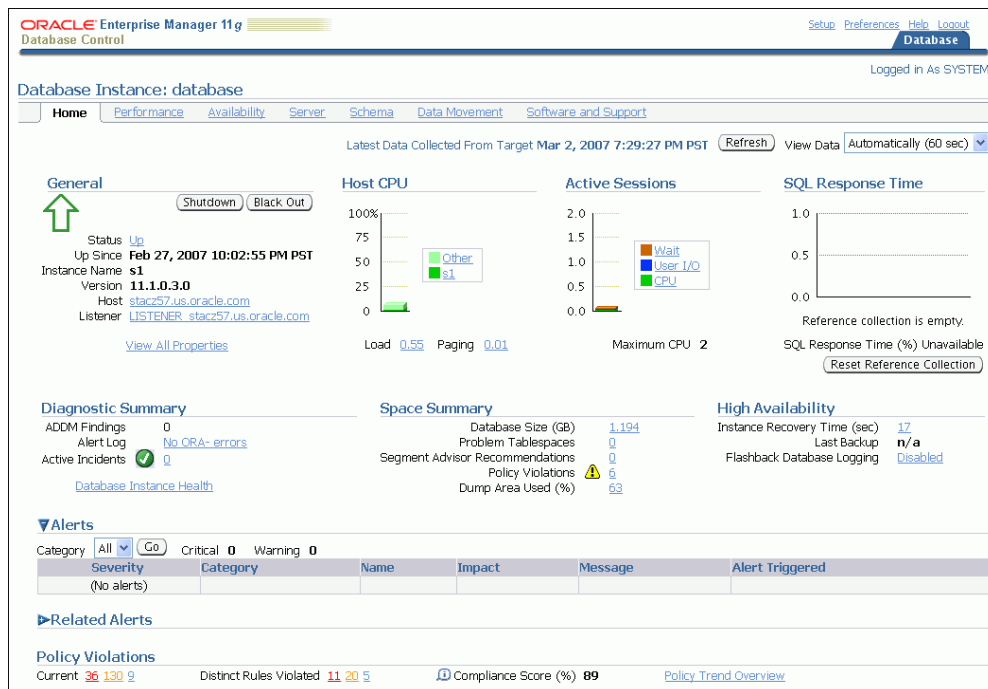
Redo Buffers 532480 bytes

Base de datos montada.

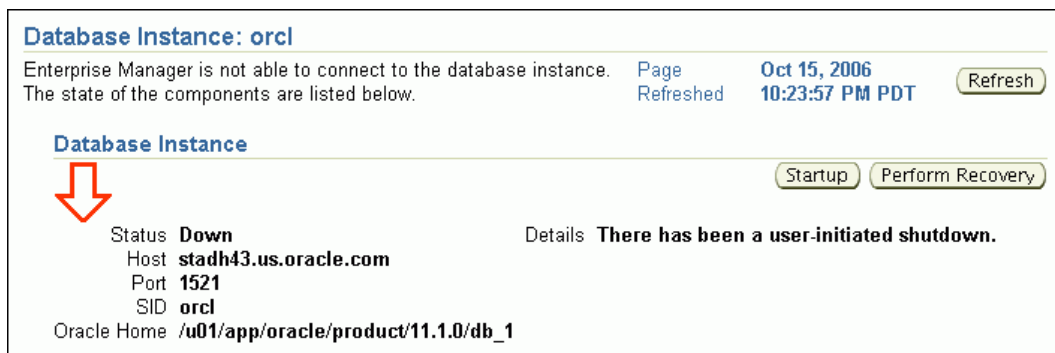
Base de datos abierta.

A. Uso de Oracle Enterprise Manager para Iniciar una Base de Datos

- Ingresar al home page para administrar la base de datos.



- El Database Home Page indicará que la base de datos se encuentra detenida ("Down")



- Dar clic en la opción Startup para instanciar la base de datos
- Aparecerá la página que solicita el ingreso de credenciales. Se tiene dos opciones:
 - Ingresar las credenciales del usuario de S.O administrador del servidor que instalo el Oracle Database.
 - Ingresar las credenciales del usuario SYS
- En la opción Connect as, seleccionar SYSDBA

Database Instance: orcl > Logged in As SYS

Startup/Shutdown: Specify Host and Target Database Credentials

Specify the following credentials in order to change the status of the database.

Host Credentials

Specify the OS user name and password to login to target database machine.

* Username

* Password

Database Credentials

Specify the credentials for the target database.
To use OS authentication, leave the user name and password fields blank.

* Username

* Password

Database orcl

* Connect As

☐ Save as Preferred Credential

i Note that you need to login to the database as SYSDBA or SYSOPER in order to change the status of the database.

- Aparecerá una pagina de confirmación, dar clic en Yes.
- Se mostrará información indicando que la base de datos ha sido instanciada.

1.2.2.2 Apertura de la base de datos en modo Restringido (RESTRICT)

Una sesión restringida es útil, por ejemplo, cuando se realiza el mantenimiento de la estructura o una importación o exportación de la base de datos. La base de datos se puede iniciar en modo restringido para que sólo esté disponible para los usuarios con el privilegio RESTRICTED SESSION.

La base de datos también se puede poner en modo restringido con el comando ALTER SYSTEM:

ALTER SYSTEM [{ENABLE|DISABLE} RESTRICTED SESSION]

Donde:

- ENABLE RESTRICTED SESSION: Permite conexiones futuras sólo para los usuarios que tienen el privilegio RESTRICTED SESSION
- DISABLE RESTRICTED SESSION: Desactiva el privilegio RESTRICTED.

```
SQL> CONNECT sys/change_on_install@cibertec AS SYSDBA
Conectado a una instancia inactiva.
SQL> STARTUP RESTRICT
Instancia ORACLE iniciada
Total System Global Area 118255568 bytes
Fixed Size                282576 bytes
Variable Size             83886080 bytes
Database Buffers          33554432 bytes
Redo Buffers              532480 bytes
Base de datos montada.
Base de datos abierta.
```

Una vez colocada una instancia en modo restringido, puede que desee finalizar todas las sesiones de usuario actuales antes de realizar las tareas administrativas. Esto es posible mediante la ejecución de:

```
ALTER SYSTEM KILL SESSION 'entero1, entero2'
```

Donde:

- entero1: Valor de la columna SID en la vista V\$SESSION
- entero2: Valor de la columna SERIAL# en la vista V\$SESSION

Nota: El identificador de sesión y el número de serie se utilizan para identificar una sesión como única. Esto garantiza que el comando ALTER SYSTEM KILL SESSION se aplique a la sesión correcta, aunque el usuario se desconecte y una nueva sesión utilice el mismo identificador de sesión.

Efectos de la Terminación de una Sesión: El comando ALTER SYSTEM KILL SESSION provoca que el proceso en segundo plano PMON haga lo siguiente cuando se ejecuta:

- Hace rollback de la transacción actual del usuario
- Libera todos los bloqueos de tabla o de fila retenidos actualmente.
- Libera todos los recursos reservados en ese momento por el usuario.

1.2.2.3 Apertura de una base de datos en modo de Sólo Lectura (READ ONLY)

Una base de datos se puede abrir en modo de sólo lectura, si no se ha abierto ya en modo de lectura y escritura. Esta característica es especialmente útil para que una base de datos en espera descargue de la base de datos de producción el procesamiento de la consulta.

```
SQL> CONNECT sys/change_on_install@cibertec AS SYSDBA
Conectado a una instancia inactiva.
```

```
SQL> STARTUP OPEN READ ONLY
```

```
Instancia ORACLE iniciada
Total System Global Area 118255568 bytes
Fixed Size                282576 bytes
Variable Size             83886080 bytes
Database Buffers         33554432 bytes
Redo Buffers              532480 bytes
Base de datos montada.
Base de datos abierta.
```

1.2.2.4 Apertura de una base de datos especificando archivo parámetros (PFILE)

Para iniciar una instancia, Oracle Server lee el archivo de parámetros de inicialización. Existen dos tipos de archivos de parámetros de inicialización:

- El archivo de parámetros estático, PFILE, que normalmente se denomina initSID.ora.
- El archivo de parámetros de servidor persistente, SPFILE, que normalmente se denomina spfileSID.ora.

A. Contenido del Archivo de Parámetros de Inicialización

- Una lista de parámetros de instancia
- El nombre de la base de datos a la que se ha asociado la instancia
- Asignaciones para estructuras de memoria del SGA (Área Global del Sistema)
- Instrucciones sobre qué hacer con los archivos redo log online llenos
- Los nombres y las ubicaciones de los archivos de control
- Información sobre segmentos de deshacer

```
SQL> CONNECT sys/change_on_install@cibertec AS SYSDBA
Conectado a una instancia inactiva.
```

```
SQL> STARTUP PFILE='d:\oracle\admin\pfile\mi_init.ora';
```

```
Instancia ORACLE iniciada
Total System Global Area 118255568 bytes
Fixed Size                282576 bytes
Variable Size             83886080 bytes
Database Buffers         33554432 bytes
Redo Buffers              532480 bytes
Base de datos montada.
Base de datos abierta.
```

Pueden existir varios archivos de parámetros de inicialización para una instancia con el fin de optimizar el rendimiento en distintas situaciones.

Cuando se ejecuta el comando STARTUP, el Orden de prioridad del uso del archivo de parámetros es el siguiente:

- Cuando se utiliza el comando STARTUP, se usa el archivo spfileSID.ora del servidor para iniciar la instancia.

- Si no se encuentra el archivo spfileSID.ora, se utilizará el archivo SPFILE por defecto del servidor para iniciar la instancia.
- Si no se encuentra el archivo SPFILE por defecto, se utilizará el archivo initSID.ora del servidor para iniciar la instancia.

Si se especifica el parámetro PFILE con STARTUP, se sustituye el comportamiento por defecto.

1.2.3 Cierre de la Base de Datos

Cierre la base de datos para realizar copias de seguridad offline de sistema operativo de todas las estructuras físicas y para que, al reiniciar, entren en vigor los parámetros de inicialización estáticos modificados.

Para cerrar una instancia, se debe conectar como SYSOPER o SYSDBA y utilizar el siguiente comando:

```
SHUTDOWN [NORMAL | TRANSACTIONAL | IMMEDIATE | ABORT ]
```

1.2.3.1 SHUTDOWN Normal

El modo normal es el modo de cierre por defecto. El cierre normal de la base de datos se lleva a cabo en las siguientes condiciones:

- No se pueden realizar conexiones nuevas.
- Oracle Server espera a que se desconecten todos los usuarios antes de finalizar el cierre.
- Los buffers de la base de datos y de redo se escriben en el disco.
- Los procesos en segundo plano han terminado y se ha eliminado el SGA de la memoria.
- Oracle cierra y desmonta la base de datos antes de cerrar la instancia.
- El siguiente inicio no requiere una recuperación de la instancia.

```
SQL> CONNECT sys/change_on_install@cibertec AS SYSDBA
Conectado a una instancia inactiva.
```

```
SQL> SHUTDOWN
Base de datos cerrada.
Base de datos desmontada.
Instancia ORACLE cerrada.
```

1.2.3.2 SHUTDOWN TRANSACTIONAL

Un cierre transaccional evita que los clientes pierdan su trabajo. Un cierre transaccional de la base de datos se lleva a cabo en las siguientes condiciones:

- Ningún cliente puede iniciar una nueva transacción en esta instancia particular.
- Un cliente se desconecta cuando el cliente finaliza la transacción en curso.
- Una vez finalizadas todas las transacciones, se produce un cierre inmediato.
- El siguiente inicio no requiere una recuperación de la instancia.

```
SQL> CONNECT sys/change_on_install@cibertec AS SYSDBA
Conectado a una instancia inactiva.
```

```
SQL> SHUTDOWN TRANSACTIONAL
```

```
Base de datos cerrada.
Base de datos desmontada.
Instancia ORACLE cerrada.
```

1.2.3.3 SHUTDOWN IMMEDIATE

El cierre inmediato de la base de datos se lleva a cabo en las siguientes condiciones:

- No se finalizan las sentencias SQL actuales que está procesando Oracle.
- Oracle Server no espera a que se desconecten los usuarios conectados en ese momento a la base de datos.
- Oracle hace rollback de las transacciones activas y desconecta a todos los usuarios conectados.
- Oracle cierra y desmonta la base de datos antes de cerrar la instancia.
- El siguiente inicio no requiere una recuperación de la instancia.

```
SQL> CONNECT sys/change_on_install@cibertec AS SYSDBA
Conectado a una instancia inactiva.
```

```
SQL> SHUTDOWN IMMEDIATE
```

```
Base de datos cerrada.
Base de datos desmontada.
Instancia ORACLE cerrada.
```

1.2.3.4 SHUTDOWN ABORT

Si las opciones de cierre NORMAL e IMMEDIATE no funcionan, se puede abortar la instancia de base de datos actual. La interrupción de una instancia se lleva a cabo en las siguientes condiciones:

- Se terminan inmediatamente las sentencias SQL actuales que Oracle Server está procesando.
- Oracle no espera a que se desconecten los usuarios conectados actualmente a la base de datos.
- Los buffers de la base de datos y de redo no se escriben en el disco.
- No se hace rollback de las transacciones no validadas
- La instancia se termina sin cerrar los archivos.
- La base de datos no se cierra ni se desmonta.
- El siguiente inicio requiere la recuperación de la instancia, que se produce automáticamente.

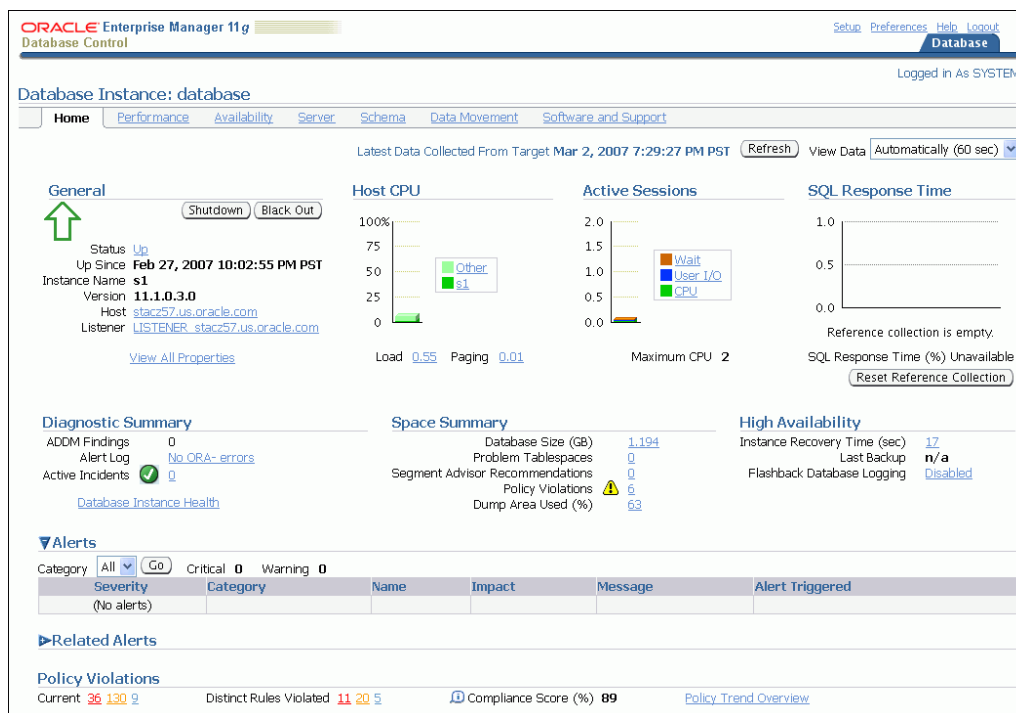
Nota: No se recomienda realizar una copia de seguridad de una base de datos que tiene un estado inconsistente.

```
SQL> CONNECT sys/change_on_install@cibertec AS SYSDBA
Conectado a una instancia inactiva.
```

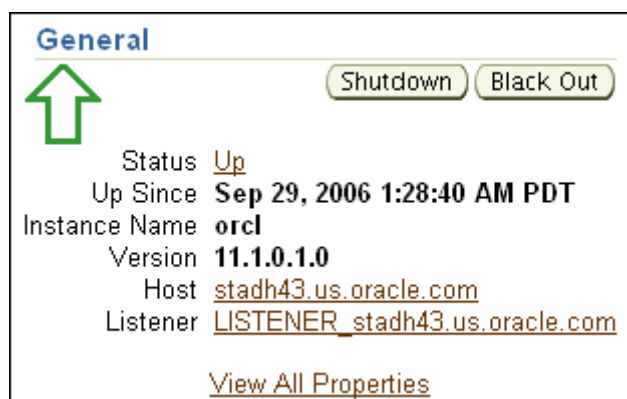
```
SQL> SHUTDOWN ABORT
Instancia ORACLE cerrada.
```

A. Uso de Oracle Enterprise Manager para Cerrar una Base de Datos

- Ingresar al home page para administrar la base de datos.



- En la sección General, dar clic en Shutdown



- Aparecerá la página que solicita el ingreso de credenciales. Ingresar con el usuario SYS.
- Aparecerá una página de confirmación, dar clic en Yes.

Database Instance: orcl

Enterprise Manager is not able to connect to the database instance. Page Refreshed Oct 15, 2006 10:23:57 PM PDT [Refresh](#)

The state of the components are listed below.

Database Instance

↓

Status **Down** Details **There has been a user-initiated shutdown.**

Host **stadh43.us.oracle.com**

Port **1521**

SID **orcl**

Oracle Home **/u01/app/oracle/product/11.1.0/db_1**

[Startup](#) [Perform Recovery](#)

- Se mostrará información indicando que la base de datos está siendo detenida.

1.2.4 Configuración de un cliente Oracle

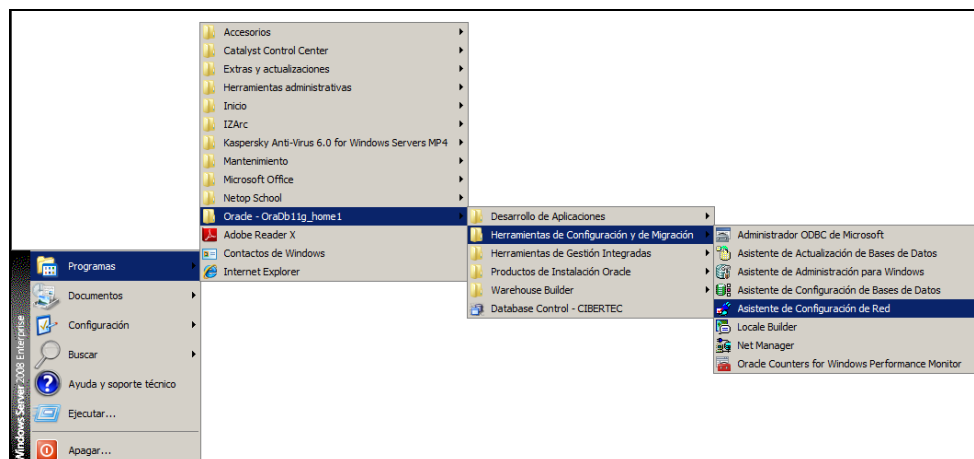
En Oracle, esto se llama creación de un nombre de servicio de red o una cadena de conexión o un 'connect string' o 'host string'. Esto se hace después de haber instalado el software Oracle Cliente. Si se ha instalado el Oracle Server, implícitamente, se tiene instalado también el Cliente Oracle.

Existen dos formas para crear un nombre de servicio de red en Oracle: manual o mediante el uso de un asistente. En este curso, sólo nos centraremos en crear un nuevo nombre de servicio de red local mediante el uso del asistente (asistente de configuración de red), para ver más detalles respecto a la conexión y configuración de redes consulte el manual del administrador – Networking.

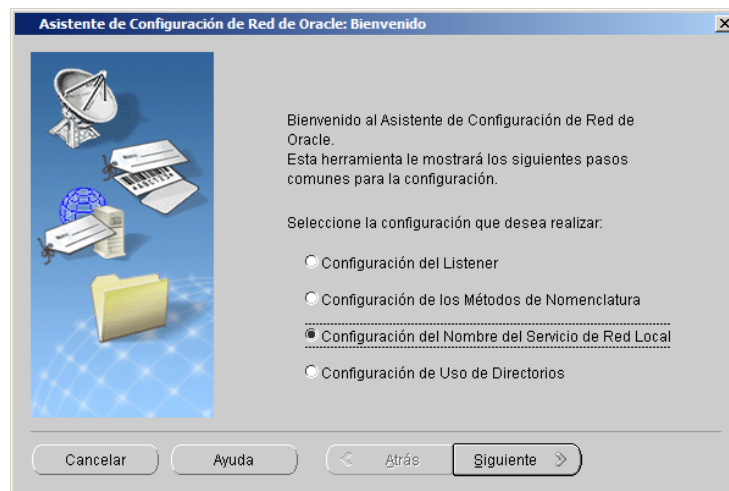
A. Asistente de Configuración de Red (NET Configuration Assistant)

Estos son los pasos que debe seguir para establecer una conexión a la base de datos a través del asistente:

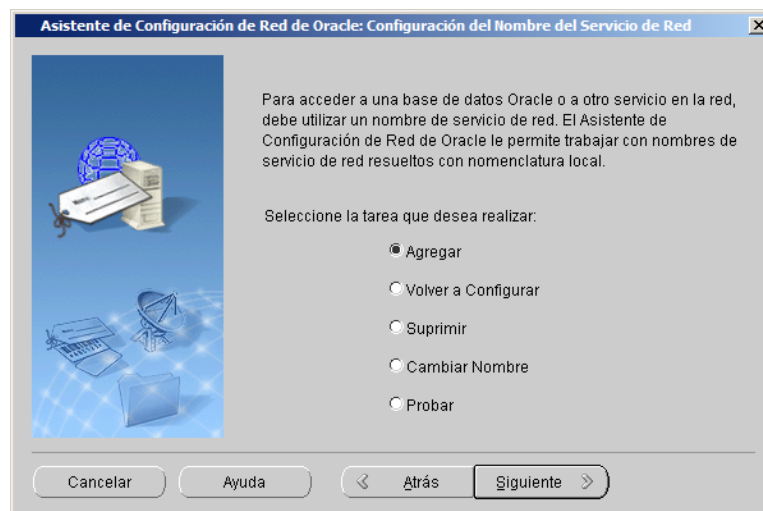
Ejecutando el asistente: En el gráfico, se muestra la ubicación del acceso directo al asistente.



Pantalla principal o bienvenida: En esta pantalla, elegimos el tipo de configuración de red que vamos a realizar. Para nuestro caso, elegimos la tercera opción (configuración del nombre del servicio de red local). En el gráfico, se muestra la ubicación del acceso directo al asistente.



Configuración del Nombre del Servicio de Red: Cuando se instala el software de Oracle, se crea el archivo TNSNAMES.ORA con algunos nombres de servicio de red de ejemplo que pueden ser utilizados como plantillas. El asistente realiza los cambios en el archivo TNSNAMES.ORA como agregar un nuevo nombre de servicio de red, eliminar uno existente, renombrar un servicio de red o modificar los parámetros de un descriptivo. Los cambios hechos en el archivo TNSNAMES.ORA son realizados si el test de conexión es satisfactorio. Para nuestro caso, elegimos agregar para insertar un nuevo nombre de servicio en el archivo TNSNAMES.ORA. Usted podrá encontrar el archivo TNSNAMES.ORA dentro del directorio del gestor de la base de datos: `../11.2.0/dbhome_1/Network/admin`.

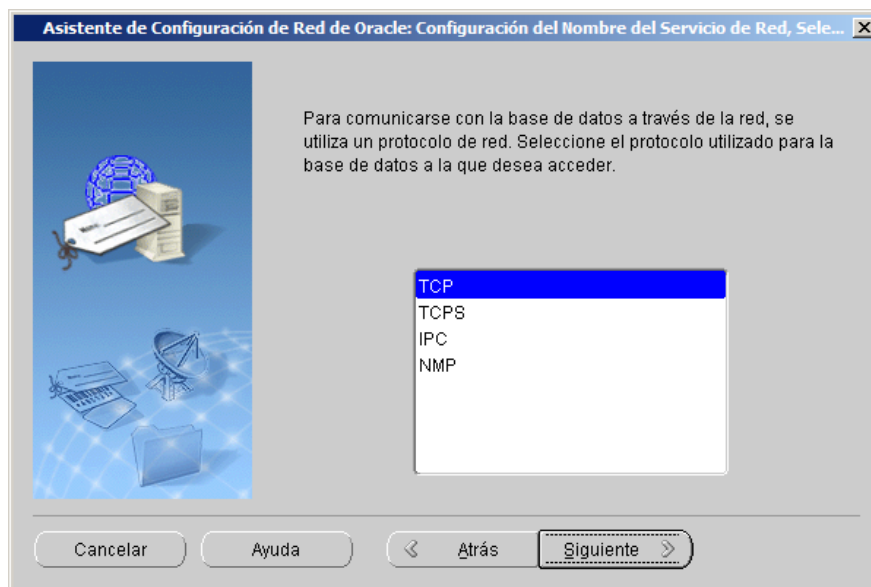


Seleccionar versión de la base de datos a acceder:

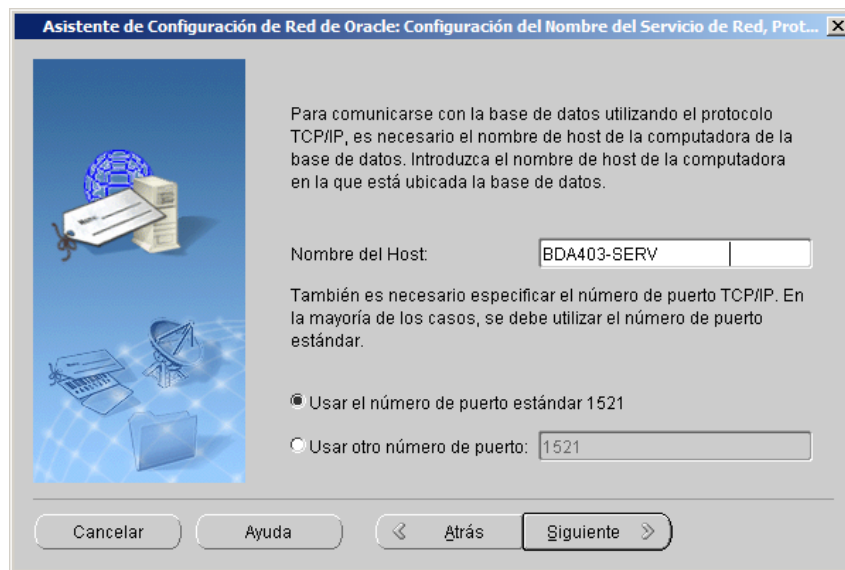
Ingreso de nombre de la base de datos a acceder: Aquí se ingresa el nombre de la base de datos remota a la cual se desea acceder.



Seleccionar el protocolo de red



Parámetro de conexión a través del Protocolo TCP: Cada protocolo de red necesita de ciertos parámetros de conexión. En la pantalla anterior, seleccionamos el protocolo TCP como el protocolo de comunicación. Este protocolo necesita del nombre de Host y de un puerto de comunicación que, por estándar, es el 1521.



Asistente de Configuración de Red de Oracle: Configuración del Nombre del Servicio de Red, Prot...

Para comunicarse con la base de datos utilizando el protocolo TCP/IP, es necesario el nombre de host de la computadora de la base de datos. Introduzca el nombre de host de la computadora en la que está ubicada la base de datos.

Nombre del Host:

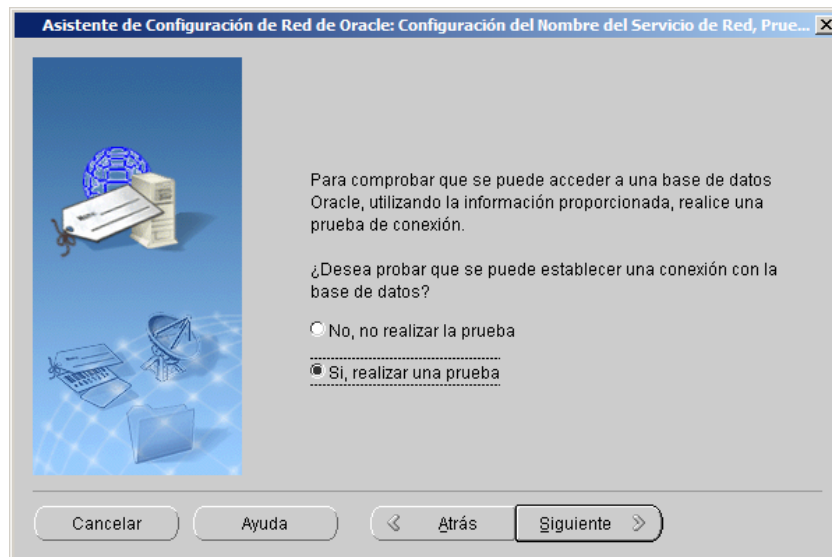
También es necesario especificar el número de puerto TCP/IP. En la mayoría de los casos, se debe utilizar el número de puerto estándar.

☒ Usar el número de puerto estándar 1521

☐ Usar otro número de puerto:

Cancelar Ayuda < Atrás Siguiete >

Test de conexión: Antes de registrar los datos al archivo TNSNAMESORA a través del asistente, debemos realizar una prueba de conexión para así probar si los parámetros ingresados son los correctos o si hay algún otro problema de conexión.



Asistente de Configuración de Red de Oracle: Configuración del Nombre del Servicio de Red, Prue...

Para comprobar que se puede acceder a una base de datos Oracle, utilizando la información proporcionada, realice una prueba de conexión.

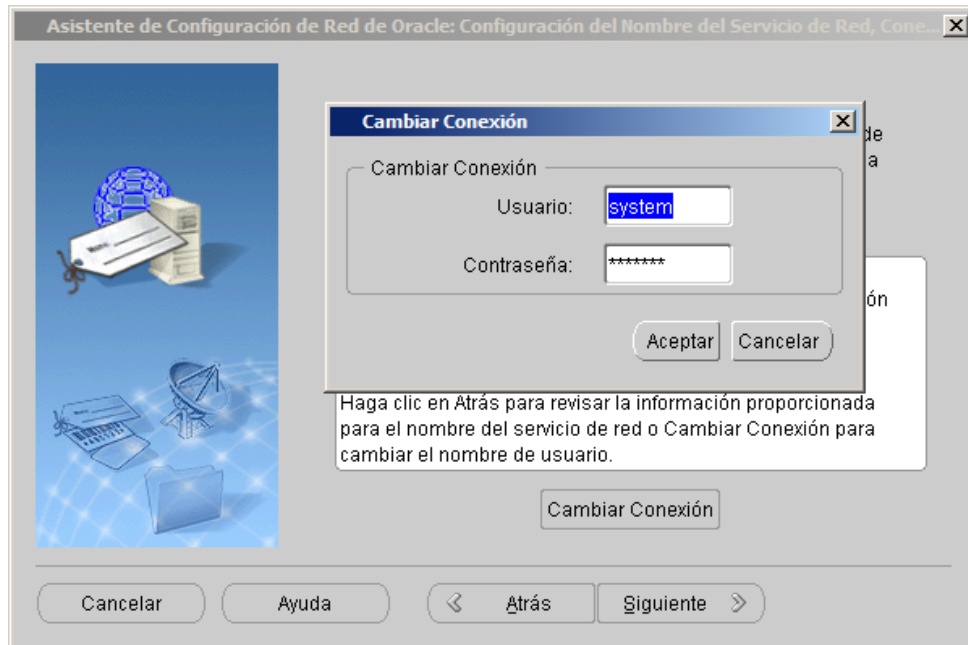
¿Desea probar que se puede establecer una conexión con la base de datos?

☐ No, no realizar la prueba

☒ Si, realizar una prueba

Cancelar Ayuda < Atrás Siguiete >

El asistente siempre utiliza al usuario SYSTEM para realizar las pruebas de conexión. Si SYSTEM no existiese, elija a otro usuario para realizar las pruebas de conexión, haga un clic en el botón Cambiar conexión para cambiar de usuario.



Ésta sería la vista del archivo TNSNAMES.ORA, después de agregar el nuevo nombre de servicio "CIBERTEC":

```

tnsnames - Bloc de notas
Archivo Edición Formato Ver Ayuda
# tnsnames.ora Network Configuration File: C:\app\Administrador\product\11.2.0\dbhome_1
# network\admin\tnsnames.ora
# Generated by Oracle configuration tools.

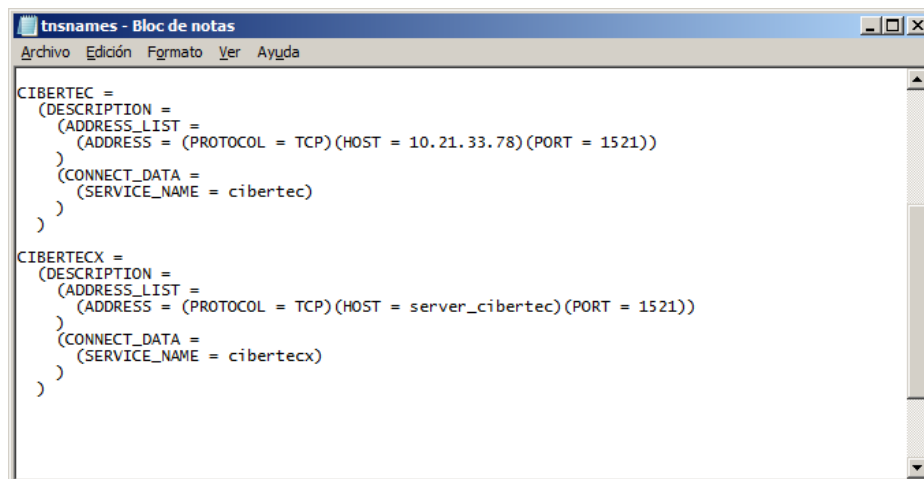
ORACLR_CONNECTION_DATA =
  (DESCRIPTION =
    (ADDRESS_LIST =
      (ADDRESS = (PROTOCOL = IPC)(KEY = EXTPROC1521))
    )
    (CONNECT_DATA =
      (SID = CLRExtProc)
      (PRESENTATION = RO)
    )
  )

CIBERTEC =
  (DESCRIPTION =
    (ADDRESS_LIST =
      (ADDRESS = (PROTOCOL = TCP)(HOST = 10.21.33.78)(PORT = 1521))
    )
    (CONNECT_DATA =
      (SERVICE_NAME = cibertec)
    )
  )

```

Muchas veces, si se desea crear otro Servicio de Red, se acostumbra ir directamente al archivo TNSNAMES.ORA, copiar una de las entradas, renombrarla y cambiar los datos necesarios.

Por ejemplo, si se desea agregar un nuevo servicio de red, para que se pueda conectar a una base de datos llamada cibertecx (en el campo SERVICE_NAME), que reside en el servidor server_cibertec (en el campo HOST), se configuraría de la siguiente manera:



```
tnsnames - Bloc de notas
Archivo  Edición  Formato  Ver  Ayuda

CIBERTEC =
  (DESCRIPTION =
    (ADDRESS_LIST =
      (ADDRESS = (PROTOCOL = TCP)(HOST = 10.21.33.78)(PORT = 1521))
    )
    (CONNECT_DATA =
      (SERVICE_NAME = cibertec)
    )
  )

CIBERTECX =
  (DESCRIPTION =
    (ADDRESS_LIST =
      (ADDRESS = (PROTOCOL = TCP)(HOST = server_cibertec)(PORT = 1521))
    )
    (CONNECT_DATA =
      (SERVICE_NAME = cibertecx)
    )
  )
```

Autoevaluación

1.- Si se ha perdido los control-files, y se trata de “levantar” la Base de Datos, ésta se detiene (o se queda) en el estado:

- a) Open
- b) Open restrict
- c) Mount
- d) Nomount
- e) N.A.

2.- Si se ha perdido uno de los datafiles, y se trata de “levaentar” la Base de Datos, ésta se detiene (o se queda) en el estado:

- a) Open
- b) Open restrict
- c) Mount
- d) Nomount
- e) N.A.

3.- Explique en qué caso usted abrirá la Base de Datos en modo RESTRICT.

4.- ¿En qué estado un usuario de Base de Datos NORMAL, se puede conectar a la misma?

- a) Open
- b) Open restrict
- c) Mount
- d) Nomount
- e) N.A.

5.- Usted acaba de ingresar a un forum de DBAs, y por ahí observa un comentario de un DBA “junior” que dice que el SHUTDOWN ABORT es mucho más rápido que los demás tipos de SHUTDOWN. ¿Qué opinaría usted al respecto ?.

6.- ¿Cuál es la secuencia lógica de estados cuando se “levanta” una Base de Datos?

- a) open, shutdown,mount, nomount.
- b) shutdown, nomount, mount, open.
- c) shutdown, mount, nomount, open.
- d) shutdown, mount, open, nomount.
- e) shutdown, open, mount, nomount.

7.- Explique la diferencia entre los diferentes tipos de SHUTDOWN.

UNIDAD DE
APRENDIZAJE

2

CREACIÓN DE ESTRUCTURAS DE DATOS

LOGRO DE LA UNIDAD DE APRENDIZAJE

Al término de la unidad, el alumno diseña e implementa modelos de datos que incorporen reglas o restricciones mediante la definición de objetos tales como tablas, secuencias y sinónimos.

TEMARIO

2.1 Tema 3 : CREACIÓN DE ESTRUCTURAS DE DATOS

- 2.1.1 : Creación y modificación de tablas
- 2.1.2 : Creación de restricciones
- 2.1.3 : Manejo de índices
- 2.1.4 : Manejo de secuencias
- 2.1.5 : Manejo de sinónimos

2.2 Tema 4 : DICCIONARIO DE DATOS

- 2.2.1 : Introducción al Diccionario de Datos
- 2.2.2 : Estructura del Diccionario de Datos
- 2.2.3 : Uso del Diccionario de Datos
- 2.2.4 : Otras tablas en el Diccionario
- 2.2.5 : La vista DBA_OBJECTS

ACTIVIDADES PROPUESTAS

- Los alumnos crearán y modificarán tablas en una BD Oracle.
- Los alumnos crearán restricciones en una BD Oracle.
- Los alumnos crearán secuencias en una BD Oracle.
- Los alumnos crearán sinónimos en una BD Oracle.

2.1 Creación de estructuras de datos

2.1.1 Creación y modificación de tablas

Para que un usuario de Base de Datos pueda crear una tabla debe tener los siguientes privilegios:

- Para crear una nueva tabla en el esquema del usuario, se debe contar con el privilegio de sistema CREATE TABLE.
- Para crear una nueva tabla en otro esquema de usuario, se debe contar con el privilegio de sistema CREATE ANY TABLE.

2.1.1.1 Creación de Tablas

Antes de crear una tabla debemos de determinar a que esquema de la base de datos pertenecerá. Para poner esto en práctica, vamos a crear en la base de datos un espacio para las tablas (Tablespace) llamado TS_DATA y, también, vamos a crear un usuario llamado DESARROLLO que utilizaremos para crear todos los objetos a ver en esta sesión.

-- Conéctese con una cuenta que tenga los privilegios suficientes para crear tablespaces y nuevos usuarios.

```
SQL>CONN system/cibertec@cibertec
```

-- Cree el espacio de tablas ts_data, para almacenar en esta parte o porción de la base de datos la data de las tablas a crear.

```
SQL> CREATE TABLESPACE TS_DATA
        Datafile 'C:\oracle\oradata\df_data01.dbf' SIZE 50M reuse
        Default Storage (    initial 8k
                           Next 8k
                           Minextents 1
                           Maxextents unlimited
                           Pctincrease 0)
```

-- Cree el usuario o esquema Desarrollo, especificando qué tablespace va a utilizar y cuánta información podrá almacenar. (Consultar la vista user_tablespaces para ver la información de los tablespaces creados en la base de datos).

```
SQL> CREATE USER DESARROLLO
        IDENTIFIED BY DESARROLLO
        DEFAULT TABLESPACE TS_DATA
        TEMPORARY TABLESPACE TEMP
        QUOTA UNLIMITED ON TS_DATA
        QUOTA UNLIMITED ON TEMP;
```

-- Otorgue los privilegios necesarios para que el usuario Desarrollo pueda conectarse y pueda crear tablas en su esquema o en el esquema de otros usuarios de la base de datos.

```
SQL> GRANT CONNECT TO DESARROLLO;
```

```
SQL> GRANT CREATE ANY TABLE TO DESARROLLO;
```

Luego de crear el tablespace TS_DATA y el esquema DESARROLLO, crearemos las tablas EMP y DEPT en el esquema DESARROLLO dentro del tablespace TS_DATA.

-- Conéctese con el usuario Desarrollo.

```
SQL>CONN desarrollo/desarrollo@cibertec
```

-- Cree las tablas Emp y Dep. para el esquema Desarrollo.

```
SQL> CREATE TABLE Emp (  
        Empno      NUMBER(4) not null,  
        Ename      VARCHAR2(20) not null,  
        Job        VARCHAR2(15),  
        Mgr        NUMBER(4),  
        Hiredate   DATE default sysdate not Null,  
        Sal        NUMBER(7,2) check(sal>=0),  
        Comm       NUMBER(7,2) check(comm>=0),  
        Deptno     NUMBER(2) not null)  
    Tablespace TS_DATA;
```

```
SQL> CREATE TABLE Dept (  
        Deptno     NUMBER(2) not null)  
        Dname      VARCHAR2(14) not null,  
        Local      VARCHAR2(13))  
    Tablespace TS_DATA;
```

Para consultar la información de las tablas que hemos creado se puede acceder a la vista del diccionario de datos USER_TABLES.

2.1.1.2 Comentando Tablas y Columnas

Usted puede agregar comentarios a las tablas y a sus columnas utilizando el comando COMMENT ON.

Por ejemplo, el siguiente comando comenta la tabla EMP:

```
SQL> COMMENT ON TABLE Emp IS 'Tabla de Empleados';
```

Por ejemplo, los siguientes comandos comentan algunas columnas de la tabla EMP:

```
SQL> COMMENT ON COLUMN Emp.Empno IS 'Código de Empleado';
```

```
SQL> COMMENT ON COLUMN Emp.Ename IS 'Nombre de Empleado';
```

```
SQL> COMMENT ON COLUMN Emp.Hiredate IS 'Fecha de Ingreso';
```

2.1.1.3 Alterando Tablas

El comando utilizado para alterar una tabla es ALTER TABLE. Para alterar una tabla, ésta debe estar contenida en el esquema del usuario o debe tener el privilegio de sistema ALTER ANY TABLE.

Las siguientes son las principales razones para alterar o modificar una tabla:

- Para adicionar, eliminar o renombrar columnas, o modificar la definición de una columna existente (tipo de dato, longitud, valores por defecto y constraints).
- Para modificar los atributos lógicos o físicos de una tabla.
- Para adicionar, modificar o eliminar constraints asociados con la tabla.
- Habilitar o deshabilitar los constraints de integridad o disparadores asociados con la tabla.
- Para renombrar la tabla.
- Para renombrar las columnas de la tabla.

A. Adicionando Columnas

Para adicionar una columna a una tabla, utilice el comando ALTER TABLE ADD.

Por ejemplo, el siguiente comando adiciona una nueva columna llamada Bonus a la tabla EMP:

```
SQL> ALTER TABLE Emp ADD Bonus NUMBER(12,2) CHECK (Bonus>=0);
```

El siguiente ejemplo adiciona más de una columna:

```
SQL> ALTER TABLE Emp ADD (Dirección Date, Telefono Char(9));
```

B. Modificando Columnas

Para modificar la definición de una columna existente, utilice el comando ALTER TABLE MODIFY. Usted puede modificar un tipo de dato de la columna, un valor por defecto, o un constraints. También puede aumentar o disminuir la longitud de la columna.

Por ejemplo, el siguiente comando disminuye la longitud de la columna Bonus de la tabla EMP:

```
SQL> ALTER TABLE Emp MODIFY Bonus NUMBER(9,2);
```

C. Renombrando Columnas

Para renombrar una columna existente, utilice el comando ALTER TABLE RENAME COLUMN. El nuevo nombre no debe tener conflicto con el nombre de una columna existente.

Por ejemplo, el siguiente comando renombra la columna Bonus de la tabla EMP:

```
SQL> ALTER TABLE Emp RENAME COLUMN Bonus TO Bono_Emp;
```

El comando puede ser cancelado si la columna tiene objetos dependientes.

D. Eliminando Columnas

Para eliminar una columna de una tabla, utilice el comando ALTER TABLE DROP COLUMN.

Por ejemplo, el siguiente comando elimina la columna Teléfono de la tabla EMP:

```
SQL> ALTER TABLE Emp DROP COLUMN Telefono;
```

2.1.1.4 Eliminando Tablas

El comando utilizado para eliminar una tabla es DROP TABLE. Para eliminar una tabla de cualquier esquema se debe tener el privilegio de sistema DROP ANY TABLE.

Por ejemplo, el siguiente comando elimina la tabla Emp del esquema de desarrollo:

```
SQL> DROP TABLE Emp;
```

Si la tabla a eliminar está referenciada por otra tabla a través de una llave foránea, usted no podrá eliminarla con el comando anterior, usted tiene que adicionar la cláusula CASCADE CONSTRAINT al comando DROP TABLE para forzar la eliminación de la tabla.

Por ejemplo, el siguiente comando elimina la tabla EMP del esquema SCOTT junto con la llave primaria y las foráneas que hacen referencia (constraints).

```
SQL> DROP TABLE scott.Emp CASCADE CONSTRAINTS;
```

2.1.2 Creación de restricciones

Los constraints de integridad son reglas que restringen los valores de uno o más columnas en una tabla. La cláusula CONSTRAINTS puede utilizarse en los comandos CREATE TABLE o ALTER TABLE.

2.1.2.1 Creando llaves primarias

Usted puede crear la llave primaria de una tabla durante la creación con el comando CREATE TABLE o adicionarla después de haber creado la tabla con el comando ALTER TABLE.

Los siguientes ejemplos crean una llave primaria a través del comando CREATE TABLE:

-- Ejemplo 1: Compuesta de una columna y sin indicar el nombre explícitamente.

```
SQL> CREATE TABLE Ord (  
        Ordid      NUMBER(4) primary key,  
        Orderdate   DATE,  
        Commplan    CHAR(1),  
        Custid      NUMBER(6) not null,  
        Shipdate    DATE,  
        Total       NUMBER(8,2))
```

```
Tablespace TS_DATA;
```

-- Ejemplo 2: Compuesta de una columna e indicando el nombre explícitamente.

```
SQL> CREATE TABLE Ord (
        Ordid          NUMBER(4) constraints pk_ord primary key,
        Orderdate       DATE,
        Commplan        CHAR(1),
        Custid          NUMBER(6) not null,
        Shipdate        DATE,
        Total            NUMBER(8,2))
Tablespace TS_DATA;
```

Los siguientes ejemplos crean una llave primaria a través del comando ALTER TABLE:

-- Ejemplo 1: Compuesta de una columna sin nombre explícito.

```
SQL> ALTER TABLE ORD ADD PRIMARY KEY (ordid);
```

-- Ejemplo 2: Compuesta de una columna con nombre explícito.

```
SQL> ALTER TABLE ORD ADD constraint pk_ord PRIMARY KEY (ordid);
```

-- Ejemplo 3: Compuesta por más de una columna con nombre explícito.

```
SQL> ALTER TABLE ITEM ADD constraint pk_item PRIMARY KEY (ordid, itemid);
```

2.1.2.2 Creando llaves foráneas

Puede crear llaves foráneas haciendo referencia a las tablas de su esquema. Para poder referenciar tablas de otros esquemas de usuarios, debe contar con el privilegio REFERENCES sobre la tabla a referenciar.

Usted puede crear la llave foránea de una tabla durante la creación con el comando CREATE TABLE o adicionarla después de haber creado la tabla con el comando ALTER TABLE.

Los siguientes ejemplos crean una llave foránea a través del comando CREATE TABLE:

-- Ejemplo 1: Sin nombre explícito.

```
SQL> CREATE TABLE Item (
        Ordid          NUMBER(4) REFERENCES Ord,
        Itemid         NUMBER(4) not null,
        Prodid         NUMBER(6),
        Actualprice     NUMBER(8,2),
        Cantidad        NUMBER(8),
        Itemtot         NUMBER(8,2),
        CONSTRAINTS pk_item primary key (ordid, itemid))
Tablespace TS_DATA;
```

-- Ejemplo 2: Con nombre explícito.

```
SQL> CREATE TABLE Item (
        Ordid          NUMBER(4),
        Itemid         NUMBER(4) not null,
        Prodid         NUMBER(6),
        Actualprice    NUMBER(8,2),
        Cantidad       NUMBER(8),
        Itemtot        NUMBER(8,2),
        CONSTRAINTS pk_item primary key (ordid, itemid),
        CONSTRAINTS fk_item foreign key (ordid) references ord(ordid))
        Tablespace TS_DATA;
```

Los siguientes ejemplos crean una llave foránea a través del comando ALTER TABLE:

-- Ejemplo 1: Sin nombre explícito.

```
SQL> ALTER TABLE ITEM ADD FOREIGN KEY (ordid) REFERENCES Ord(ordid);
```

-- Ejemplo 2: Con nombre explícito.

```
SQL> ALTER TABLE ITEM
      ADD CONSTRAINTS fk_item
      FOREIGN KEY (ordid) REFERENCES Ord(ordid);
```

2.1.2.3 Creando Check

Las restricciones CHECK son reglas de validación a nivel de columna. Esta restricción puede crearse durante la creación de la tabla con el comando CREATE TABLE o añadirla después de haber creado la tabla con el comando ALTER TABLE.

Los siguientes ejemplos crean una restricción CHECK a través del comando CREATE TABLE:

-- Ejemplo 1:

```
SQL> CREATE TABLE Item (
        Ordid          NUMBER(4),
        Itemid         NUMBER(4) not null,
        Prodid         NUMBER(6),
        Actualprice    NUMBER(8,2) CONSTRAINTS ck_item_01
                        CHECK (actualprice >= 0),
        Cantidad       NUMBER(8)  CONSTRAINTS ck_item_02
                        CHECK (cantidad >= 0),
        Itemtot        NUMBER(8,2) CONSTRAINTS ck_item_02
                        CHECK (itemtot >= 0),
        CONSTRAINTS pk_item primary key (ordid, itemid),
        CONSTRAINTS fk_item foreign key (ordid) references ord(ordid))
        Tablespace TS_DATA;
```


Los siguientes ejemplos crean una restricción CHECK a través del comando ALTER TABLE:

-- Ejemplo 1:

```
SQL> ALTER TABLE ITEM
      ADD CONSTRAINTS ck_item_01 CHECK (actualprice >=0);
```

-- Ejemplo 2:

```
SQL> ALTER TABLE ITEM
      ADD (CONSTRAINTS ck_item_02 CHECK (cantidad >=0),
          CONSTRAINTS ck_item_03 CHECK (itemtot >=0));
```

2.1.2.4 Estableciendo valores por defecto

Utilice el comando DEFAULT con el comando CREATE TABLE o con el comando ALTER TABLE para especificar un valor por defecto.

Los siguientes ejemplos crean esta restricción a través del comando CREATE TABLE:

-- Ejemplo 1:

```
SQL> CREATE TABLE Errores (
      Id_error      NUMBER(4),
      Descrip       VARCHAR2(50),
      Usuario       CHAR(10) DEFAULT user,
      Fecha         DATE   DEFAULT sysdate,
      Hora          CHAR(8) DEFAULT to_char(sysdate, hh24:mi:ss),
      Estacion      VARCHAR2(20) DEFAULT USERENV('terminal')
      Tablespace TS_DATA;
```

Los siguientes ejemplos crean esta restricción a través del comando ALTER TABLE:

-- Ejemplo 1:

```
SQL> ALTER TABLE errores MODIFY (Usuario DEFAULT user);
```

-- Ejemplo 2:

```
SQL> ALTER TABLE errores MODIFY (Fecha DEFAULT sysdate,
      Hora DEFAULT to_char(sysdate, hh24:mi:ss),
      estacion DEFAULT USERENV('terminal'));
```

2.1.2.5 Habilitando y Deshabilitando Constraints

Usted puede habilitar o deshabilitar cualquier constraint de su esquema. Para modificar un constraint de otro esquema de usuario debe tener el privilegio ALTER sobre el objeto tabla.

Por ejemplo, el siguiente comando deshabilita la llave primaria de la tabla ORD:

-- Ejemplo 1:

```
SQL> ALTER TABLE Scott.Ord DISABLE PRIMARY KEY;
```

-- Ejemplo 2:

```
SQL> ALTER TABLE Scott.Ord ENABLE CONSTRAINTS pk_ord;
```

Por ejemplo, el siguiente comando deshabilita la llave foránea de la tabla ITEM:

-- Ejemplo 1:

```
SQL> ALTER TABLE Desarrollo.Item DISABLE FOREIGN KEY;
```

-- Ejemplo 2:

```
SQL> ALTER TABLE Desarrollo.Item DISABLE CONSTRAINTS fk_item;
```

2.1.2.6 Eliminando Constraints

Usted puede un constraint utilizando el comando ALTER TABLE con una de las siguientes cláusulas:

- DROP PRIMARY KEY
- DROP UNIQUE
- DROP CONSTRAINT

Por ejemplo, los siguientes dos comandos eliminan un constraint de integridad:

-- Ejemplo 1:

```
SQL> ALTER TABLE Dept DROP PRIMARY KEY;
```

-- Ejemplo 2:

```
SQL> ALTER TABLE Item DROP CONSTRAINTS fk_item;
```

2.1.3 Manejando Índices

Un índice es una estructura diseñada para obtener un acceso más rápido a los datos contenidos dentro de una tabla.

Un índice es independiente de los datos almacenados en la tabla y cuando se encuentra bien definido, es decir, cuando se forma atendiendo a la gran mayoría de las consultas que se harán sobre una tabla, reduce significativamente la búsqueda, aumentando el rendimiento.

Inmediatamente luego de creado el índice, la base de datos Oracle comienza a mantenerlo de acuerdo a las inserciones, actualizaciones y eliminaciones de registros de la tabla en la cual se ha implementado.

2.1.3.1 Tipos de índices

Existen tres tipos de índices cuya naturaleza depende de la forma en que haya sido creado. Estos tipos son:

Un índice único es aquél que tiene la restricción adicional de que el grupo de columnas indexadas define una única fila. Sin embargo, si no van a existir más grupos de columnas con estas características, dentro de una misma tabla, se recomienda crear el conjunto como una clave primaria ya que de todas formas Oracle asociará un índice único a esta restricción (la clave primaria).

Un índice no único, que es aquél que no impone la restricción de que las filas no deban repetirse.

Un índice compuesto es aquél que agrupa varias columnas de la tabla. Este tipo es muy útil cuando las sentencias de selección (SELECT) efectúan búsquedas por varios criterios (columnas) en una misma tabla. Es importante el orden en que se ponen las columnas al crear el índice; la columna más referenciada debería ser puesta en primer lugar y así sucesivamente.

2.1.3.2 Consideraciones en el diseño de índices

Un índice sólo es efectivo cuando es utilizado. Es por eso que debe asegurarse que la frecuencia de uso sea muy alta y que su implementación redunde en mejoras de rendimiento de las consultas efectuadas a la tabla donde reside el índice. Sin embargo, no debe explotarse el uso de los índices dentro de una misma tabla porque con cada operación de inserción, actualización o eliminación que se lleva a cabo sobre una tabla, sus índices se deben recrear, con el consiguiente uso de recursos que se produce. A menudo, es conveniente eliminar o desactivar temporalmente un índice cuando sabemos que se va a efectuar una operación de carga/actualización/eliminación masiva en la tabla y más tarde volver a crearlo, cuando la operación haya finalizado.

Considere las siguientes reglas de indexación para cuando se enfrente a la tarea de decidir qué tablas indexar:

- Indexe solamente las tablas cuando las consultas (queries) no accedan a una gran cantidad de filas de la tabla. Use índices cuando una consulta acceda a un porcentaje menor al 5% de las filas de una tabla.
- No indexe tablas que son actualizadas con mucha frecuencia.
- Indexe aquellas tablas que no tengan muchos valores repetidos en las columnas escogidas. Recuerde que, finalmente, el índice hace una búsqueda secuencial dentro de un conjunto de filas objetivo.
- Las consultas muy complejas (en la cláusula WHERE), por lo general, no toman mucha ventaja de los índices. Cuando posea más experiencia podrá corroborar esta afirmación y estará preparado para arreglar estas situaciones.

También es importante decidir qué columnas indexar. Siga las siguientes reglas cuando tenga que tomar esta decisión:

- Escoja las columnas que se utilizan con mayor frecuencia en las cláusulas WHERE de las consultas.
- No indexe aquellas columnas que tengan demasiados valores repetidos en ellas.
- Las columnas que toman valores únicos son excelentes candidatas para indexar. Oracle, automáticamente, indexa las claves primarias de las tablas.
- Indexe las columnas que sirven para unir una tabla con otras (join en consultas).
- Si hay columnas que no tienen valores únicos por sí solas, pero que en conjunto con otra columna forman una dupla única o con pocas repeticiones (menos que las columnas individualmente), entonces conviene indexarlas (siempre y cuando existan consultas que las utilicen en conjunto). Estos índices reciben el nombre de índices compuestos.

Otra consideración importante a tomar en cuenta es que los índices deben residir en un tablespace diferente de donde residen las tablas.

Por ejemplo, los siguientes dos comandos crean un índice:

-- Ejemplo 1:

```
SQL> CREATE INDEX emp_ename ON Emp(ename) TABLESPACE indx;
```

Los índices también pueden ser únicos o no únicos. Un índice único garantiza que dos filas de una tabla no tengan valores duplicados en la columna o columnas que conforman el índice.

-- Ejemplo 2:

```
SQL> CREATE UNIQUE INDEX dept_unique_index ON Dept(dname)
TABLESPACE indx;
```

2.1.3.3 Creando índices basados en funciones

Los índices basados en funciones facilitan las consultas que evalúan un valor retornado por una función o expresión. El valor de la función o expresión es recalculado y guardado por el índice.

Para la creación de un índice basado en funciones en su esquema debe tener el privilegio de sistema QUERY REWRITE. Para crearlos en otros esquemas debe tener los privilegios CREATE ANY INDEX y GLOBAL QUERY REWRITE.

-- Otorgando Privilegios.

```
SQL> GRANT query rewrite TO Scott;
```

```
SQL> GRANT global query rewrite TO Scott;
```

Por ejemplo, considere la expresión en la cláusula WHERE de abajo:

```
SQL> CREATE INDEX idx ON desarrollo.emp (comm + sal);
```

```
SQL> SELECT * FROM emp WHERE (comm + sal) < 5000;
```

Por ejemplo, considere la función en la cláusula WHERE de abajo:

```
SQL> CREATE INDEX idx_emp_name ON desarrollo.emp (Upper(ename));
```

```
SQL> SELECT * FROM emp WHERE Upper(ename) = `JONES`;
```

2.1.3.4 Eliminando índices

Para eliminar un índice, debe estar contenido en su esquema o debe tener el privilegio de sistema DROP ANY INDEX.

El siguiente comando elimina un índice:

```
SQL> DROP INDEX idx_emp_name;
```

2.1.4 Manejo de secuencias

A menudo, es preciso generar números en forma ordenada para implementar, por ejemplo, valores para una llave primaria en una tabla o garantizar que esos números no se repiten y van siempre en un orden predefinido por el desarrollador (no necesariamente secuenciales).

La forma tradicional de efectuar lo anterior sería almacenar el último número utilizado en un registro especial, bloquearlo, obtener el próximo valor, actualizar el registro, desbloquearlo y utilizar el número. Sin embargo, para eso, Oracle implementa los objetos denominadas secuencias que permiten hacer lo anterior de manera transparente para el usuario.

Las secuencias son objetos de la base de datos en los que, múltiples usuarios, pueden generar series de números enteros diferentes.

2.1.4.1 Creando Secuencias

Para crear una secuencia en su esquema, debe tener el privilegio de sistema CREATE SEQUENCE. Para crear una secuencia en otro esquema de usuario, debe tener el privilegio CREATE ANY SEQUENCE.

Por ejemplo, la siguiente sentencia crea una secuencia para generar números que servirán para dar valores a la columna empno de la tabla Emp:

```
SQL> CREATE SEQUENCE seq_empleado
      INCREMENT BY 1
      START WITH 1
      NOMAXVALUE
      NOCYCLE;
```

Por ejemplo, la siguiente sentencia crea una secuencia para generar números que servirán para dar valores a la columna empno de la tabla Emp:

Los parámetros significan lo siguiente:

- Increment by: Indica la cantidad de incremento de la secuencia.
- Start with: Es el valor de partida de la secuencia.
- Minvalue: Indica cuál será el valor mínimo de la secuencia.
- Maxvalue: Corresponde al valor máximo que puede tomar la secuencia.
- Nocycle: Es el valor por defecto para establecer si la secuencia deberá comenzar nuevamente a generar valores una vez que ha alcanzado el máximo.

Para acceder a la secuencia creada, utilice las pseudocolumnas CURRVAL y NEXTVAL.

La pseudocolumna NEXTVAL genera un nuevo número diferente a los anteriormente generados. La pseudocolumna CURRVAL devuelve el último número generado.

```
SQL> SELECT seq_empleado.NEXTVAL FROM DUAL;
NEXTVAL
```

1

```
SQL> SELECT seq_empleado.CURRVAL FROM DUAL;
CURRVAL
```

1

2.1.4.2 Modificando Secuencias

Puede alterar una secuencia para cambiar cualquiera de los parámetros que definió, excepto el número de secuencia inicial. Para cambiar este parámetro es necesario eliminar la secuencia y volverla a crear.

Por ejemplo, la siguiente sentencia modifica una secuencia:

```
SQL> ALTER SEQUENCE seq_empleado
      INCREMENT BY 10
      MAXVALUE 10000
      CYCLE;
```

2.1.4.3 Eliminando Secuencias

Usted puede eliminar cualquier secuencia de su esquema. Para eliminar una secuencia de otro esquema, debe tener el privilegio `DROP ANY SEQUENCE`.

Por ejemplo, la siguiente sentencia elimina una secuencia:

```
SQL> DROP SEQUENCE seq_empleado;
```

2.1.5 Manejo de sinónimos

Los sinónimos son objetos del sistema que apuntan a otros objetos. Implementan alias de tablas, vistas, secuencias o unidades de programas. Por lo general se utilizan para esconder ciertos detalles del objeto que representan al usuario final.

Los sinónimos pueden ser públicos o privados. Los primeros son aquéllos que caen dentro del esquema PUBLIC y son vistos por todos los usuarios de la misma base de datos. Los sinónimos privados se crean dentro del esquema de un usuario en particular y sólo estará visible para quienes él estime conveniente.

Los sinónimos proporcionan un nivel de seguridad ocultando el nombre y el propietario de un objeto y permitiendo su localización transparente para objetos remotos de una base de datos distribuida. También, reduce la complejidad de los comandos SQL para los usuarios de la base de datos.

2.1.5.1 Creando Sinónimos

Para crear un sinónimo privado en su esquema, debe tener el privilegio CREATE SYNONYM. Para crear un sinónimo público, debe tener el privilegio de Sistema CREATE PUBLIC SYNONYM.

Por ejemplo, las siguientes sentencias crean sinónimos:

-- Ejemplo 1: Sinónimo privado

```
SQL> CREATE SYNONYM priv_dept FOR Desarrollo.Dept;
```

-- Ejemplo 1: Sinónimo público

```
SQL> CREATE PUBLIC SYNONYM public_emp FOR Desarrollo.Emp;
```

2.1.5.2 Eliminando Sinónimos

Para eliminar un sinónimo privado en cualquier esquema, debe tener el privilegio DROP ANY SYNONYM. Para eliminar un sinónimo público, debe tener el privilegio de Sistema DROP PUBLIC SYNONYM.

Por ejemplo, las siguientes sentencias eliminan sinónimos:

-- Ejemplo 1: Sinónimo privado

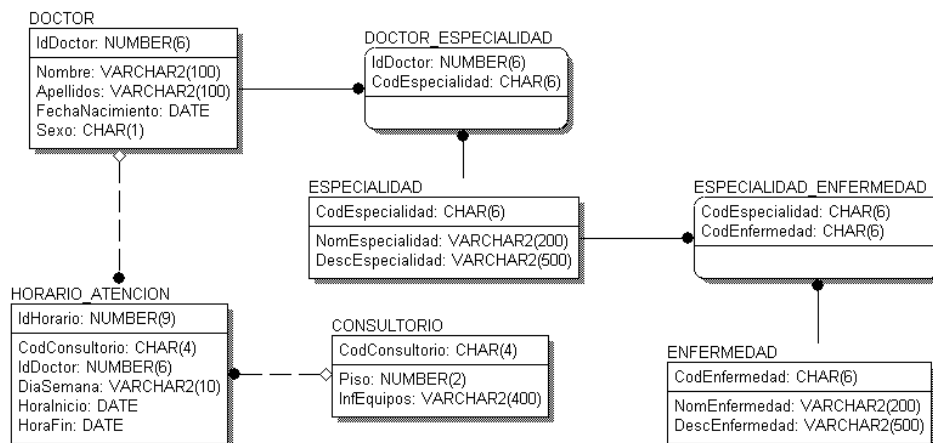
```
SQL> DROP SYNONYM priv_dept;
```

-- Ejemplo 1: Sinónimo público

```
SQL> DROP PUBLIC SYNONYM public_emp;
```

Autoevaluación

El siguiente gráfico nos muestra un modelo de una base de datos que almacena información de los horarios de atención de una clínica.



Realice usted las modificaciones que crea conveniente para cumplir los siguientes requerimientos.

- 1.- Se han presentado solicitudes de los pacientes para saber los síntomas de las enfermedades, por tal motivo se requiere almacenar información de los síntomas que pueda presentar una determinada enfermedad. Una enfermedad puede tener varios síntomas y un síntoma siempre pertenece a una única enfermedad.
- 2.- Todo síntoma debe tener un id numérico y una descripción. Asegurarse que la generación de valores para los id de los síntomas sea eficiente.
- 3.- Asimismo, se han registrado muchos problemas al momento de registrar el sexo de un doctor, por tal motivo se solicita que en ese campo solo se puedan registrar lo siguientes valores: M o F.
- 4.- No deben existir dos enfermedades con el mismo nombre.
- 5.- Se debe poder almacenar por especialidad la información de la universidad y el año en que el docente culminó sus estudios.

2.2 Creación de estructuras de datos

2.2.1 Introducción al Diccionario de Datos

El diccionario de datos de Oracle es una estructura de tablas y vistas, de sólo lectura, que contiene información de la base de datos, tal como:

- las definiciones de todos los esquemas en la BD (tablas, índices, vistas, clusters, sinónimos, secuencias, procedimientos, funciones, paquetes, triggers, etc)
- el espacio asignado y actualmente ocupado por un esquema
- los usuarios de la base de datos
- los privilegios y roles que cada usuario tiene
- información sobre quienes han accedido y actualizado esquemas
- información general de la base de datos

El diccionario de datos es creado cuando la base de datos es creada. Para reflejar con exactitud el estado de la BD en todo momento, el diccionario de datos es automáticamente actualizado por Oracle en respuesta a acciones específicas (tales como cuando la estructura de la BD es modificada). La importancia del diccionario de datos radica en que Oracle cuenta con éste para operar la base de datos.

2.2.2 Estructura del Diccionario de Datos

2.2.2.1 Tablas base

Estas tablas que almacenan información acerca de la base de datos. Sólo Oracle debe escribir y leer esas tablas. Los usuarios rara vez accesan a ellas directamente porque no están normalizadas y la mayoría de los datos están encriptados.

2.2.2.2 Vistas accesibles por el usuario

Estas vistas resumen la información almacenada en las tablas base del diccionario de datos.

Todas las tablas y vistas del diccionario de datos son almacenadas en el tablespace SYSTEM.

El usuario SYS es el propietario de todas las tablas base y las vistas accesibles por el usuario del diccionario de datos.

2.2.3 Uso del Diccionario de Datos

Principalmente se usa de tres maneras:

- Oracle accesa el diccionario de datos para hallar la información acerca de los usuarios, esquemas y estructuras almacenadas.
- Oracle modifica el diccionario de datos cada vez que se usa una instrucción DDL.
- Cualquier usuario puede usar el diccionario de datos

Hay tres grupos de vistas con información similar y que se distinguen por su prefijo:

Prefijo	Descripción
<i>USER_</i>	Vistas con información del esquema del usuario en sesión
<i>ALL_</i>	Estas vistas presentan al usuario una perspectiva global de la base de datos. Le da al usuario información sobre todos los objetos en la base de datos a los que tiene acceso.
<i>DBA_</i>	Vistas con la información de lo existe en todos los esquemas

Ejemplos de algunas de estas vistas son:

Información	Vistas correspondientes
Errores de compilación	<u>all_errors</u> , <u>dba_errors</u> , <u>user_errors</u>
Trabajos en la BD	<u>all_jobs</u> , <u>dba_jobs</u> , <u>user_jobs</u>
Usuarios	<u>all_users</u> , <u>dba_users</u> , <u>user_users</u>
Objetos de la BD: tablas, programas, vistas, secuencias, índices, etc.	<u>all_objects</u> , <u>dba_objects</u> , <u>user_objects</u>
Dependencias entre los objetos (PROCEDURE, FUNCTION, PACKAGE, PACKAGE BODY)	<u>all_dependencies</u> , <u>dba_dependencies</u> , <u>user_dependencies</u>
Tablas de la base de datos	<u>all_tables</u> , <u>dba_tables</u> , <u>user_tables</u>
Vistas en la BD	<u>all_views</u> , <u>dba_views</u> , <u>user_views</u>
Secuencias	<u>all_sequences</u> , <u>dba_sequences</u> , <u>user_sequences</u>
Nombres de tablas, vistas, secuencias, sinónimos	<u>all_catalog</u> , <u>dba_catalog</u> , <u>user_catalog</u>
Columnas de las tablas y vistas	<u>all_tab_columns</u> , <u>dba_tab_columns</u> , <u>user_tab_columns</u>
Comentarios sobre tablas o columnas	<u>all_tab_comments</u> , <u>dba_tab_comments</u> , <u>user_tab_comments</u> , <u>all_col_comments</u> , <u>dba_col_comments</u> , <u>user_col_comments</u>
Restricciones de integridad	<u>all_constraints</u> , <u>dba_constraints</u> , <u>user_constraints</u> , <u>all_cons_columns</u> , <u>dba_cons_columns</u> , <u>user_cons_columns</u>
Triggers en la BD	<u>all_triggers</u> , <u>dba_triggers</u> , <u>user_triggers</u>
Uso de las columnas en un trigger	<u>all_trigger_cols</u> , <u>dba_trigger_cols</u> , <u>user_trigger_cols</u>

2.2.4 Otras tablas en el diccionario

DUAL es una tabla del diccionario de datos que tiene una sola columna, DUMMY, y un único registro, 'X'. Es útil para calcular el resultado de una expresión o ejecutar una función usando instrucción SELECT. Por ejemplo, la consulta

```
Select To_Date('16/05/2000', 'DD/MM/YYYY')+7 from dual;
```

devuelve '23/05/00'.

DICTIONARY y DICT_COLUMNS proporcionan información sobre las tablas y vistas del diccionario de datos de Oracle.

TAB proporciona información sobre las tablas, vistas y sinónimos del usuario. En su lugar debería de usarse USER_TABLES o TABS.

2.2.5 La vista DBA_OBJECTS

Esta vista contiene información de TODOS los objetos de la Base de Datos, tal como se observa en el siguiente cuadro:

SQL> desc DBA_OBJECTS		
Nombre	¿Nulo?	Tipo
OWNER		VARCHAR2(30)
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)

Entre los principales campos tenemos:

- Owner: propietario del objeto
- Object_name: Nombre del Objeto
- Object_type: Tipo del Objeto (tabla, índice, etc.)
- Created: Fecha y hora exacta de creación del objeto
- Status: Estado del objeto. Por ejemplo, si un índice está inválido, entonces es INVALID.

Importante

“SQL y PL/SQL no distinguen entre mayúsculas y minúsculas por lo que todos los datos de los objetos se convierten a mayúsculas antes de ser almacenados en las tablas del diccionario de datos. Por tanto, deben emplearse mayúsculas cuando se consulte el diccionario de datos. “

A continuación se muestran las principales vistas del diccionario de datos:

Tipo de Objeto	Vista del Diccionario de Datos	Contiene Información sobre
Tablas y vistas	ALL_ALL_TABLES	Tablas accesibles (relaciones y de objetos).
	DBA_ALL_TABLES	Todas las tablas (relaciones y de objetos) en la base de datos.
	USER_ALL_TABLES	Las tablas del usuario actual (relaciones y de objetos).
	ALL_COL_COMMENTS	Comentarios sobre columnas de tablas y vistas accesibles
	DBA_COL_COMMENTS	Comentarios sobre todas las columnas de tablas y vistas.

Tipo de Objeto	Vista del Diccionario de Datos	Contiene Información sobre
	USER_COL_COMMENTS	Comentarios sobre columnas de tablas y vistas propiedad del usuario actual.
	ALL_PARTIAL_DROP_TABS	Tablas accesibles que han sido parcialmente borradas.
	DBA_PARTIAL_DROP_TABS	Tablas accesibles que han sido parcialmente borradas.
	USER_PARTIAL_DROP_TABLES	Tablas de propiedad del usuario actual que han sido parcialmente borradas.
	ALL_REFS	Columnas REF y atributos de columnas accesibles de tipos de objeto.
	DBA_REFS	Todas las columnas REF y atributos de la base de datos.
	USER_REFS	Columnas REF y atributos de las columnas de tipo de objeto propiedad del usuario actual.
	ALL_TAB_COLUMNS	Columnas de tablas accesibles
	DBA_TAB_COLUMNS	Todas las columnas de tablas
	USER_TAB_COLUMNS	Columnas de tablas propiedad del usuario actual.
	ALL_TAB_COMMENTS	Comentarios sobre tablas accesibles
	DBA_TAB_COMMENTS	Comentarios sobre todas las tablas de la base de datos.
	USER_TAB_COMMENTS	Comentarios sobre las tablas propiedad del usuario actual.
	ALL_TABLES	Tablas relacionales accesibles.
	DBA_TABLES	Todas las tablas relacionales
	USER_TABLES	Tablas relacionales propiedad del usuario actual
	ALL_UNUSED_COL_TABS	Tablas accesibles que contienen columnas no utilizadas.
	DBA_UNUSED_COL_TABS	Todas las tablas que contienen columnas no utilizadas
	USER_UNUSED_COL_TABS	Tablas propiedad del usuario actual que contienen columnas no utilizadas
	ALL_UPDATABLE_COLUMNS	Columnas actualizables en las vistas de combinación accesibles
	DBA_UPDATABLE_COLUMNS	Columnas actualizables en todas las vistas de combinación
	USER_UPDATABLE_COLUMNS	Columnas actualizables en las vistas de combinación propiedad del usuario actual
	ALL_VIEWS	Vistas accesibles
	DBA_VIEWS	Todas las vistas de la base de datos.
	USER_VIEWS	Vistas propiedad del usuario actual

Tipo de Objeto	Vista del Diccionario de Datos	Contiene Información sobre
Secuencias	ALL_SEQUENCES	Secuencias accesibles
	DBA_SEQUENCES	Todas las secuencias
	USER_SEQUENCES	Secuencias de propiedad del usuario actual
Sinónimos	ALL_SYNONYMS	Sinónimos accesibles
	ALL_SYNONYMS	Todas las sinónimos de la base de datos
	USER_SYNONYMS	Sinónimos de propiedad del usuario actual
Índices	ALL_IND_COLUMNS	Columnas de índice de las tablas accesibles
	DBA_IND_COLUMNS	Columnas de índice de todas las tablas
	USER_IND_COLUMNS	Columnas de índice de las tablas del usuario actual
	ALL_IND_EXPRESSIONS	Expresiones funcionales de índice para los índices de las tablas accesibles.
	DBA_IND_EXPRESSIONS	Expresiones funcionales de índice para los índices de todas las tablas
	USER_IND_EXPRESSIONS	Expresiones funcionales de índice para los índices de las tablas propiedad del usuario actual
	ALL_INDEXES	Índice de las tablas accesibles
	DBA_INDEXES	Índice de todas las tablas
	USER_INDEXES	Índice de las tablas del usuario actual
	ALL_INDEXTYPES	Tipos de índice accesibles
	DBA_INDEXTYPES	Todos los tipos de índice
	USER_INDEXTYPES	Tipos de índice del usuario actual
	ALL_INDEXTYPE_OPERATORS	Operadores soportados por los tipos de índices.
	DBA_INDEXTYPE_OPERATORS	Operadores soportados por todos los tipos de índices.
	USER_INDEXTYPE_OPERATORS	Operadores soportados por los tipos de índices del usuario actual
Subprogramas	ALL_ARGUMENTS	Los parámetros de subprogramas almacenados accesibles (empaquetados y de nivel de esquema)
	USER_ARGUMENTS	Los parámetros de subprogramas almacenados de l usuario actual (empaquetados y de nivel de esquema)

Tipo de Objeto	Vista del Diccionario de Datos	Contiene Información sobre
Disparadores	ALL_INTERNAL_TRIGGERS	Disparadores internos asociados a tablas accesibles
	DBA_INTERNAL_TRIGGERS	Disparadores internos asociados a todas las tablas accesibles
	USER_INTERNAL_TRIGGERS	Disparadores internos asociados a tablas pertenecientes al usuario actual
	ALL_TRIGGERS	Disparadores de base de datos accesibles
	DBA_TRIGGERS	Todos los disparadores de base de datos
	USER_TRIGGERS	Disparadores de base de datos propiedad del usuario actual
	ALL_TRIGGER_COLS	Columnas utilizadas en disparadores accesibles
	DBA_TRIGGER_COLS	Columnas utilizadas en todos los disparadores
	USER_TRIGGER_COLS	Columnas utilizadas en disparadores propiedad del usuario actual
Código fuente y errores de compilación	ALL_ERRORS	Errores de compilación para las vistas, paquetes, cuerpos de paquetes, funciones, procedimientos, tipos de objeto y disparadores accesibles.
	DBA_ERRORS	Errores de compilación para todas vistas, paquetes, cuerpos de paquetes, funciones, procedimientos.
	USER_ERRORS	Errores de compilación para las vistas, paquetes, cuerpos de paquetes, funciones, procedimientos del usuario actual.
	ALL_SOURCE	Código fuente para paquetes, cuerpos de paquetes, tipos de objeto, cuerpos de tipos de objeto, funciones y procedimientos accesibles
	DBA_SOURCE	Código fuente para todos los paquetes, cuerpos de paquetes, tipos de objeto, cuerpos de tipos de objeto, funciones y procedimientos
	USER_SOURCE	Código fuente para paquetes, cuerpos de paquetes, tipos de objeto, cuerpos de tipos de objeto, funciones y procedimientos del usuario actual.
Restricciones	ALL_CONS_COLUMNS	Columnas en las restricciones de las tablas accesibles.
	DBA_CONS_COLUMNS	Columnas en las restricciones de todas las tablas accesibles.
	USER_CONS_COLUMNS	Columnas en las restricciones de las tablas del usuario actual.
	ALL_CONSTRAINTS	Restricciones en las tablas accesibles
	DBA_CONSTRAINTS	Restricciones en todas las tablas
	USER_CONSTRAINTS	Restricciones en las tablas del usuario actual

Autoevaluación

- 1.- Averigue consultando el Diccionario de Datos, ¿cuántos y cuáles son los tipos de objetos que existen actualmente en la Base de Datos?
- 2.- Lista todas las tablas que ha creado el usuario SYSTEM durante todo el año 2004 y 2005.
- 3.- Encuentre cuál es el usuario de Base de Datos que ha creado más tablas?.
- 4.- Encuentre cuál es el usuario de Base de Datos que ha creado más objetos?
- 5.- Encuentre cuál es el usuario de Base de Datos que ha creado más índices?
- 6.- Liste los campos (nombre y tipo), que tiene la tabla EMP del usuario SCOTT.
- 7.- Liste los constraints de la tabla SCOTT.EMP.

**UNIDAD DE
APRENDIZAJE****3**

LENGUAJE DE MANIPULACIÓN DE DATOS Y FUNCIONES PREDEFINIDAS

LOGRO DE LA UNIDAD DE APRENDIZAJE

Al término de la unidad, el alumno manipula la información mediante el uso de la herramienta SQL*PLUS, sentencias SQL y funciones predefinidas en la base de datos Oracle.

TEMARIO

- Introducción al SQL
- Manipulación de datos y uso de las funciones predefinidas de una BD Oracle

ACTIVIDADES PROPUESTAS

- Identificar las principales categorías de sentencias SQL
- Ejecutar cada caso de sentencias DML
- Utilizar las principales funciones predefinidas de una BD Oracle

3.1 Lenguaje de Manipulación de Datos

3.1.1 Lenguaje SQL

El lenguaje estructurado de consulta (SQL) define cómo manipular los datos en Oracle. En este curso, no se pretende enseñar a detalle cada instrucción SQL de Oracle. Se sobre entiende que el alumno conoce este lenguaje. Lo que sí es importante conocer, son algunas de las funcionalidades o diferencias propias que Oracle ha adicionado a las instrucciones SQL para facilitar las operaciones de acceso y manipulación de datos. El SQL de Oracle está basado sobre los estándares de ANSI (American National Standards Institute y la ISO (International Standards Organization). Las instrucciones SQL pueden dividirse en cinco categorías que se muestran a continuación:

- Las instrucciones del lenguaje de manipulación de datos (Data Manipulation Language, DML) permiten cambiar o consultar los datos contenidos en una tabla de la base de datos, pero no permiten cambiar la estructura de una tabla u otro objeto.
- Las instrucciones del lenguaje de definición de datos (Data Definition Language, DDL) permiten crear, borrar o modificar la estructura de un objeto del esquema. Las instrucciones que cambian los permisos relativos a los objetos del esquema también pertenecen al DDL.
- Las instrucciones de control de transacciones garantizan la consistencia de los datos, organizando las instrucciones SQL en transacciones lógicas, que se completan o fallan como una unidad.
- Las instrucciones de control de sesión cambian las opciones de una conexión determinada con la base de datos, por ejemplo para activar las trazas de SQL o activar un suceso.
- Las instrucciones de control del sistema cambian las opciones que afectan a la base de datos completa, por ejemplo para activar o desactivar el archivado definitivo.

SELECT	Data retrieval
INSERT UPDATE DELETE	Data manipulation language (DML)
CREATE ALTER DROP RENAME TRUNCATE	Data definition language (DDL)
COMMIT ROLLBACK SAVEPOINT	Transaction control
GRANT REVOKE	Data control language (DCL)

3.1.2 INSTRUCCIONES DML Y OPERADORES

Las instrucciones de manipulación de datos (DML) permitidas son SELECT, INSERT, UPDATE y DELETE:

- SELECT devuelve filas de la base de datos que cumplan los criterios definidos en su cláusula WHERE.
- INSERT añade filas a una tabla de La base de datos.
- UPDATE modifica las filas de una tabla que cumplan la cláusula WHERE.
- DELETE borra las filas identificadas por la cláusula WHERE.

Además de WHERE estas instrucciones pueden tener otras cláusulas que se describen más adelante en esta sección. Cuando se ejecutan instrucciones SQL desde SQL *PLUS, los resultados se presentan en la pantalla. En el caso de las instrucciones UPDATE, INSERT o DELETE. SQL *PLUS devuelve el número de filas procesadas.

3.1.2.1 Instrucción SELECT

Una instrucción SELECT extrae datos de la base de datos. Permite consultar los valores que contienen los campos de tablas y/o vistas de la base de datos. La forma general de una instrucción SELECT es:

```
SELECT [DISTINCT | ALL] {*, columna [alias], expresión, ...}
FROM listado de tablas
[WHERE condición(es)]
[GROUP BY expresión [, expresión] ... ]
[HAVING condición]
[ORDER BY {columna, expresión, alias} {ASC | DESC}]
```

En SQL * PLUS, una instrucción SELECT, muestra en pantalla las filas que satisfacen las condiciones de la consulta. Toda búsqueda de caracteres es “case sensitive”, si los datos se guardaron en mayúscula. El dato a buscar deberá escribirse en mayúscula. Para efecto de la manipulación del Diccionario de datos, se debe tener en cuenta que Oracle guarda dicha información SIEMPRE en mayúsculas. Por ejemplo, al crear una tabla:

```
CREATE TABLE emp (EMPNO NUMBER(4), ENAME ...);
```

Si se desea buscar información acerca de esta tabla, muchas veces se hace lo siguiente:

```
SQL> SELECT owner, table_name, tablespace_name FROM dba_tables WHERE
table_name = 'emp';
```

No rows selected

Lo correcto debería ser de la siguiente manera:

```
SQL> SELECT owner, table_name, tablespace_name FROM dba_tables WHERE
table_name = 'EMP';
```

OWNER	TABLE_NAME	TABLESPACE_NAME
SCOTT	EMP	SYSTEM

Es importante conocer la estructura del objeto que se pretende acceder. Para ello, se utiliza el comando DESC[CRIBE] de SQL *PLUS. Este comando no es un comando de SQL estándar. Es sólo para el intérprete. Por ejemplo, para mostrar la estructura de la tabla EMP hagamos lo siguiente:

```
SQL>DESC emp
```

O también:

```
SQL>DESCRIBE emp
```

A. Cláusula WHERE

Para mostrar o seleccionar solo algunos registros, es decir, aplicar un 'filtro', se usa la cláusula WHERE seguido de una condición:

```
SELECT ename, job, deptno, hiredate, sal
FROM emp
WHERE job = 'MANAGER'
AND TO_CHAR(hiredate,'DD/MM/YYYY') = '02/04/81';
```

B. Cláusula ORDER BY

Para ordenar la salida de una sentencia SELECT se usa la cláusula ORDER BY. A continuación se muestra un ejemplo:

```
SELECT empno, ename, sal * 12 salario
FROM emp
ORDER BY salario desc;
```

Para ordenar por múltiples columnas:

```
SELECT empno, ename, sal * 12 salario
FROM emp
ORDER BY salario, ename desc;
```

En este caso al poner la palabra reservada DESC, significa que la salida del SELECT será retornado en orden DESCENDENTE de acuerdo a las columnas mencionadas. Para el caso contrario (ordenación ascendente), se utiliza la palabra ASC.

C. Uso de Alias

Los alias son útiles cuando se requiere resolver algún tipo de ambigüedad, por ejemplo cuando haya columnas en dos tablas con el mismo nombre.

Por ejemplo, esta consulta utiliza alias para las dos tablas para diferenciar, la columna DEPTNO, cuyo nombre se repite en ambas tablas:

```
SELECT empno, ename, A.deptno
FROM emp A, dept B
WHERE A.deptno = B.deptno;
```

Cuando necesitamos definir un alias cuyo nombre esta compuesto por más de una palabra, como el ejemplo anterior, este tipo de alias debe encerrarse entre comillas (""). Por ejemplo:

```
SELECT ename AS nombre, sal salario, sal * 12 "salario anual"
FROM emp;
```

D. Subconsultas

Es una sentencia SELECT que esta dentro de una cláusula de otra sentencia SELECT. La cláusula puede ser Where, Having o From.

Sintaxis:

```
SELECT select_list
FROM table
WHERE expr operator (SELECT select_list FROM table)
```

Donde operator puede ser >, =, >=, <, <>, <= o IN, ANY, ALL.

E. Tipos de Subconsultas

Subconsultas de una fila

Se utilizarán los operadores: >, =, >=, <, <>, <= .

```
SQL> SELECT ename FROM emp WHERE sal > (select sal from emp where
empno=7566)
```

Subconsultas de múltiples filas

Se utilizarán los operadores: IN, ANY, ALL.

```
SQL> SELECT ename, sal, deptno FROM emp WHERE sal IN (SELECT MIN(sal)
FROM emp group by deptno)
```

```
SQL> SELECT empno,ename,job FROM emp WHERE sal < ANY (SELECT sal FROM emp
where job='CLERK') AND job = 'CLERK';
```

Subconsultas de múltiples columnas

Si se desea comparar dos o más columnas se debe escribir una cláusula WHERE compuesta utilizando operadores locales. Esto permite combinar condiciones de where duplicados dentro de una cláusula where simple.

```
SQL> SELECT * FROM emp WHERE (empno,deptno) IN (SELECT empno,deptno
FROM emp where job='MANAGER')
```

F. Rownum

Devuelve, en una consulta, el número de fila actual. Resulta útil para limitar número total de filas, y se usa principalmente en la cláusula WHERE de las consultas y en la cláusula SET de las instrucciones UPDATE. ROWNUM devuelve un valor de tipo NUMBER.

El ejemplo siguiente devuelve sólo las primeras dos filas de la tabla EMP:

```
SQL> SELECT * FROM emp WHERE ROWNUM < 3;
```

El valor de ROWNUM para la primera fila es 1, para la segunda es 2, y así sucesivamente. El valor de ROWNUM se asigna a las filas antes de que se realice una ordenación.

3.1.2.2 Instrucción INSERT

La sintaxis de la orden INSERT es la siguiente:

```
INSERT INTO tabla de referencia  
(nombre de columnas)  
VALUES  
(expresiones);
```

Donde tabla_de_referencia, es la tabla en la cual se ingresará un nuevo registro, nombres_de_columnas es la lista de campos que será considerada en la inserción. Esta lista va entre paréntesis. Expresiones son los valores que serán insertados correspondiendo uno a uno a los nombres de columnas anteriormente mencionados. La expresión puede ser una sentencia SELECT cuya lista de campos deben ser correspondientes a los nombres de columnas. En este caso, no se debe incluir la cláusula VALUES.

Convencional

```
INSERT INTO emp  
(empno, ename, job, manager, hiredate, sal, comm)  
VALUES  
(2296,  
'JPEREZ', 'SALESMAN', 7782, TO_DATE('25/09/2008', 'DD/MM/YYYY'),  
3000, 100);
```

Con variables de sustitución

Las variables de sustitución son usadas en el entorno de SQL*PLUS, para ellos se antepone el símbolo '&', lo que hará que el SQL*Plus solicite los valores para aquellas variables que tienen antepuestas el mencionado símbolo.

```
INSERT INTO dept (deptno, dname, local)  
VALUES ('&Departamento', '&Nombre', '&Localizacion');
```

```
SQL> INSERT INTO dept (deptno, dname, local)  
VALUES ('&Departamento', '&Nombre', '&Localizacion');  
Enter value for Departamento: 80  
Enter value for Nombre: SISTEMAS  
Enter value for Localizacion: LIMA
```

Con prompts personalizados

```
ACCEPT depto_d PROMPT 'Por favor ingrese el número del depto'
ACCEPT name_d PROMPT 'Por favor ingrese el nombre del depto'
INSERT INTO dept (deptno, dname)
VALUES ('&depto_d', '&name_d');
```

Copiando filas de otra tabla

```
INSERT INTO managers
(id, name, salary, hiredate)
SELECT empno, ename, sal, hiredate FROM emp WHERE job =
'MANAGER';
```

3.1.2.3 Instrucción UPDATE

La sintaxis de la orden UPDATE es la siguiente:

```
UPDATE tabla_de_referencia SET
Nombre_columna1 = expresion1,
Nombre_columna2 = expresion2,
Nombre_columna3 = (sentencia_select),...
WHERE cláusula_where;
```

Donde, tabla_de_referencia es la tabla que se actualizará, nombre_columnaN son las columnas a las que se le asigna un valor, expresionN son los valores correspondientes para cada nombre de columna. El valor que se asigne a una columna puede venir del resultado de una sentencia SELECT. La condiciones que el SELECT retorne sólo una fila, y los valores de los campos del SELECT sean correspondientes a las columnas que están actualizando en el UPDATE. La cláusula_where es la condición que deben cumplir los registros para ser actualizados.

Convencional

```
UPDATE emp SET deptno = 20 WHERE empno = 7782;
```

Con una subconsulta de múltiples columnas

```
UPDATE emp SET (job, deptno) = (SELECT job, deptno FROM emp
WHERE empno = 7499) WHERE empno = 7698;
```

Con múltiples subconsultas

```
UPDATE emp SET deptno = (SELECT deptno FROM emp WHERE empno
= 7788) WHERE job = (SELECT job FROM emp WHERE empno = 7788);
```

3.1.2.4 Instrucción DELETE

La orden DELETE elimina los registros que cumplen con la condición definida en el WHERE. La sintaxis es la siguiente:

```
DELETE FROM tabla_de_referencia WHERE cláusula_where;
```


Donde, `tabla_de_referencia` es la tabla en la cual se eliminarán los registros correspondientes, `cláusula_where` es la condición que cumplirán los registros que serán eliminados. 'El CURRENT OF cursor' es para ser usado cuando se esta 'barriendo' un cursor y se desea eliminar de la tabla el registro actual del cursor.

Convencional

```
DELETE FROM dept WHERE dname = 'ACCOUNTING';
```

Con una subconsulta

```
DELETE FROM emp WHERE deptno = (SELECT deptno FROM dept WHERE  
dname = 'SALES');
```

3.1.2.5 Operadores

A. Operadores de comparación

=, >, >=, <, <=, <>

```
SELECT ename, sal, comm FROM emp WHERE sal <= comm;
```

BETWEEN ... AND ..

```
SELECT ename, sal FROM emp WHERE sal BETWEEN 1000 AND 1500;
```

IN (List)

Este operador de comparación sirve para verificar que un valor (campo) este en una lista de valores, donde esta lista puede ser un conjunto de elementos enumerados, como también puede ser un SELECT.

```
SELECT empno, ename, sal, mgr FROM emp WHERE mgr IN (7902,  
7566, 7788);
```

LIKE

Este operador de comparación sirve para comparar parcialmente los valores de un campo. Por ejemplo:

- Registros cuyos valores en el campo Nombre empiece con la palabra 'JUAN'. condición en cláusula: Nombre LIKE 'JUAN%'
- Registros cuyos valores en el campo Nombre contengan la palabra 'PEREZ'. condición en cláusula: Nombre LIKE '%PEREZ%'

A continuación se muestra otro caso:

Se utilizará %_

Para mostrar los nombres de empleados que tengan como segunda letra de su nombre la "A"

```
SELECT ename FROM emp WHERE ename like '_A%';
```

IS NULL

Mostrar los empleados que no ganan comisión.

```
SELECT empno, ename, sal, job, comm FROM emp WHERE comm IS NULL;
```

La columna null indica si el campo es obligatorio o no. Si está como NOT NULL, la columna o también llamado campo, es obligatorio ya que no acepta la constante NULL como contenido y NULL la representa la ausencia de valor o vacío.

B. Operadores lógicos

AND

```
SELECT empno, ename, job, sal FROM emp WHERE sal >= 1100 AND job = 'CLERK';
```

OR

```
SELECT empno, ename, job, sal FROM emp WHERE sal >= 1100 OR job = 'CLERK';
```

NOT, NOT IN, NOT BETWEEN, NOT LIKE

```
SELECT empno, ename, job FROM emp WHERE job NOT IN ('CLERK', 'MANAGER', 'ANALYST');
```

3.1.3 Consultas Multitabla

3.1.3.1 JOIN

Se utiliza para consultar datos de más de una tabla. Se escribe en la parte de la sentencia WHERE. Se pone como prefijo de la columna el nombre de la tabla cuando la misma columna aparece en más de una tabla.

3.1.3.2 Tipos de Join

- Equijoin
- Non-equijoin.
- Outer-join
- Self-join.

A. Equijoin

```
SELECT emp.ename, emp.job, emp.sal, dept.dname FROM emp, dept  
WHERE emp.deptno = dept.deptno;
```

Lo que hace un Equijoin es igualar los valores de los campos que son comunes entre las diferentes tablas. Por ejemplo, en el caso anterior, para cada empleado se está mostrando el campo dept.dname (nombre del departamento), el cual no está en la tabla EMP. El dato común que se tiene es que la tabla EMP tiene el campo

emp.deptno (código del departamento). Con ese dato se va a buscar el nombre del departamento que está en la tabla DEPT. Por eso, se hace el JOIN emp.deptno=dept.deptno. Otra manera de expresar el select anterior sería:

```
SELECT emp.ename, emp.job, emp.sal, dept.dname FROM emp JOIN
dept ON emp.deptno = dept.deptno;
```

B. Non-Equijoin

```
SELECT e.ename, e.job, e.sal, s.grade FROM emp e, salgrade s
WHERE e.sal BETWEEN s.losal AND s.hisal;
```

C. Outer-join

Si un registro no satisface un join, el registro no aparecerá en los resultados de la consulta. Por ejemplo, en la condición equijoin de EMP y DEPT, el departamento OPERATIONS no aparecerá porque nadie trabaja en ese departamento.

El operador de outer join es el signo(+) que se usa para ver los registros que no cumplen en el join. En el ejemplo aparecerán los nombres de departamentos.

```
SELECT e.empno, e.ename, e.job, e.sal, d.dname
FROM emp e, dept d
WHERE e.deptno(+) = d.deptno;
```

```
SELECT e.empno, e.ename, e.job, e.sal, d.dname
FROM emp e LEFT JOIN dept d
ON e.deptno = d.deptno;
```

D. Self-join

Este tipo de JOIN se utiliza para consultar datos que se encuentran dentro de una misma tabla (join consigo mismo). Se requiere consultar el nombre del manager del empleado con código empno igual a 7369 (MGR).

```
SELECT e.ename EMPLEADO, e.sal SALARIO, e.job OCUPACION,
m.ename MANAGER FROM emp e, emp m WHERE
e.mgr=m.empno AND e.empno=7369;
```

3.1.4 Funciones Predefinidas

Aceptan uno o más argumentos y retornan un valor por cada fila retornada por la consulta. Las funciones estándar del Oracle 11g permiten realizar diversas operaciones útiles para los desarrolladores y administradores de base de datos. Éstas se agregan al Server en el momento de la instalación de una Base de datos por defecto.

Las funciones se clasifican en los siguientes grupos:

- Funciones de carácter
- Funciones numéricas
- Funciones de fecha
- Funciones de conversión
- Funciones generales
- Funciones de grupo

3.1.4.1 Funciones de carácter

Se dividen en dos grupos:

A. Funciones de conversión:

Función	Resultado
LOWER('SQL Course')	sql course
UPPER('SQL Course')	SQL COURSE
INITCAP('SQL Course')	Sql Course

B. Funciones de manipulación de caracteres:

Función	Resultado
CONCAT('Good', 'String')	GoodString
SUBSTR('Hola', 1, 3)	Hol
LENGTH('Hola')	4
INSTR('Hola','o')	2
LPAD(120,10,'*')	*****120
RPAD(120,10,'*')	120*****
Trim(' esto es una prueba ')	esto es una prueba

3.1.4.2 Funciones Numéricas

Función	Acción
ROUND	Redondea el valor a un número específico de decimales.
TRUNC	Trunca el valor a un número específico de decimales.
MOD	Retorna el residuo de la división

Función	Resultado
ROUND (45.936, 2)	45.94
TRUNC (45.936, 2)	45.93
MOD (1600, 300)	100

3.1.4.3 Funciones de fecha

Oracle guarda datos en un formato numérico interno: por centuria, año, mes, día, hora, minuto y segundo. El formato por defecto depende de la instalación del Oracle software. SYSDATE es una función que retorna fecha y hora. DUAL es una tabla pública que se utiliza para ejecutar funciones predefinidas.

Se puede realizar cálculos utilizando operaciones como adición y sustracción:

date + number = date
 date – number = date
 date – date = número de días
 date + number/24 = date (suma horas)

Por ejemplo: asumiendo que sysdate = 26/12/2004

Función	Resultado
MONTHS_BETWEEN(sysdate + 40, sysdate)	1.29032258
ADD_MONTHS(sysdate, 6)	26/06/2005
NEXT_DAYS(sysdate, 'VIERNES')	31/12/2004
LAST_DAY(sysdate)	31/12/2004
ROUND(sysdate, 'MONTH')	01/01/2005
ROUND(sysdate, 'YEAR')	01/01/2005
TRUNC(sysdate, 'MONTH')	01/12/2004
TRUNC(sysdate, 'YEAR')	01/01/2004

```
SELECT SYSDATE, ADD_MONTHS (sysdate,4) Agregados,
NEXT_DAY(sysdate, 'VIERNES') Viernes, LAST_DAY(sysdate)
ultimo FROM DUAL;
```

3.1.4.4 Funciones de conversión

A. Conversión de tipos de datos implícitos

El servidor Oracle convierte automáticamente lo siguiente:

Función	Resultado
VARCHAR2 or CHAR	NUMBER
VARCHAR2 or CHAR	DATE
NUMBER	VARCHAR2
DATE	VARCHAR2

B. Conversión de tipos de datos explícitos

SQL provee tres funciones para convertir un tipo de dato a otro:

a) TO_CHAR():

Funciones TO_CHAR con fecha: TO_CHAR (fecha, 'formato')

Elementos de Formato de fecha

Función	Resultado
YYYY	Año completo en número
YEAR	Año deletreado
MM	Valor de dos dígitos para el mes
MONTH	Nombre completo del mes
DD	Número del día en el mes
DAY	Nombre del día
D	Número del día en la semana
HH24	Horas en formato 24
MI	Minutos
SS	Segundos

A continuación algunos ejemplos:

```
SELECT ename, TO_CHAR(hiredate, 'MM/YYYY') "Mes de Ingreso"
FROM emp;
```

```
SELECT TO_CHAR(sysdate, 'DD/MM/YYYY HH2:MI:SS') fecha
FROM dual;
```

Funciones TO_CHAR con números: TO_CHAR (número, 'formato')

Elementos de Formato Numérico

Función	Resultado
9	Representa un dígito
0	Fuerza que se muestre un 0
\$	Coloca el signo dólar
L	Utiliza el símbolo de la moneda
.	Imprime un punto decimal
,	Imprime un indicador de miles

A continuación algunos ejemplos:

```
SELECT TO_CHAR(sal, '$ 99,999') Salario FROM emp;
```

```
SELECT TO_CHAR(comm, 'L 0999.99') Comision FROM emp;
```

b) TO_NUMBER():

Convierte una cadena de caracteres que contiene dígitos a un número con formato especificado por el modelo de formato.

c) TO_DATE():

Convierte una cadena de caracteres representando una fecha a un valor de fecha con formato especificado. Si se omite, el formato es DD-MON-YY.

d) NVL():

Convierte un valor nulo a un valor que puede ser un número, un carácter o una fecha.

```
NVL(comm, 0)
NVL(hiredate, sysdate)
NVL(job, 'no job')
```

e) DECODE():

```
DECODE(mivariable, valor1, retorno1, valor2, retorno2, ...,
retorno_defecto)
```

Donde si “mivariable” es igual a “valor1”, se retorna “retorno1”

Donde si “mivariable” es igual a “valor2”, se retorna “retorno2”

...

Donde si “mivariable” no es igual a ninguno de los valores anteriores, se retorna “retorno_defecto”

```
SQL> SELECT DECODE(USER, 'SCOTT', 'usted es Scott', 'SYSTEM', 'Es el
administrador', 'No es conocido') resultado FROM DUAL;
```

RESULTADO

usted es Scott

3.1.4.5 Funciones de grupo

Estas funciones operan en grupos de registros para entregar un resultado por grupo.

A. Tipo de Funciones

AVG, COUNT, MAX, MIN, STDDEV, SUM, VARIANCE

B. Creando grupo de datos

Divide las filas de una tabla en datos agrupados.

```
SQL> SELECT deptno, AVG(sal) FROM emp GROUP BY deptno;
```

```
SQL> SELECT deptno, job, SUM(sal) FROM emp GROUP BY deptno, job;
```

C. Operadores Cube y Rollup

Especifique los operadores CUBE y ROLLUP en la cláusula GROUP BY de una consulta.

El agrupamiento ROLLUP produce un juego de resultados que contiene las filas agrupadas normales y los subtotales. La operación CUBE en la cláusula GROUP BY agrupa las filas seleccionadas basadas en los valores de todas las combinaciones posibles de expresiones de la especificación y devuelve una sola fila de información de resumen para cada grupo. Puede utilizar el operador CUBE para producir filas de tabulación cruzada.

Nota: Al trabajar con ROLLUP y CUBE, asegúrese de que las columnas que siguen a la cláusula GROUP BY tengan entre sí una relación significativa y real; de lo contrario, los operadores devolverían información irrelevante.

Los operadores CUBE y ROLLUP están disponibles solamente en Oracle8i y en versiones posteriores.

a) Operador ROLLUP

El operador ROLLUP entrega agregados y superagregados para expresiones dentro de una sentencia GROUP BY. Los escritores de informes pueden utilizar el operador ROLLUP para extraer estadísticas e información de resumen de los juegos de resultados. Los agregados acumulativos se pueden utilizar en informes, diagramas y gráficos.

```
SELECT [column,] group_function(column). . .  
FROM table  
[WHERE condition]  
[GROUP BY [ROLLUP] group_by_expression]  
[HAVING having_expression]  
[ORDER BY column];
```

El operador ROLLUP crea agrupamientos moviéndose en una dirección, de derecha a izquierda, a lo largo de la lista de columnas especificada en la cláusula GROUP BY. A continuación, aplica la función agregada a estos agrupamientos.

Nota: Para producir subtotales en n dimensiones (es decir, n columnas en la cláusula GROUP BY) sin un operador ROLLUP, n+1 las sentencias SELECT deben estar enlazadas con UNION ALL. Esto hace que la ejecución de la consulta sea ineficaz, ya que cada sentencia SELECT provoca acceso de tabla. El operador ROLLUP recoge los resultados de esta con solo un acceso de tabla. El operador ROLLUP es útil si hay muchas columnas implicadas en la producción de subtotales.

Ejemplo:

```
SQL> SELECT deptno as department_id,  
       Job as Job_id,  
       SUM(sal) salary  
       FROM emp  
       WHERE deptno < 60  
       GROUP BY ROLLUP(deptno, job);
```


DEPARTMENT_ID	JOB_ID	SALARY
10	CLERK	1300
10	MANAGER	2450
10	PRESIDENT	5000
10		8750
20	CLERK	1900
20	ANALYST	3000
20	MANAGER	2975
20		7875
30	CLERK	950
30	MANAGER	2850
30	SALESMAN	5600
30		9400
50	ANALYST	3000
50		3000
		29025

15 filas seleccionadas.

b) Operador CUBE

El operador CUBE es una opción adicional de la cláusula GROUP BY en una sentencia SELECT. El operador CUBE se puede aplicar a todas las funciones agregadas, incluidas AVG, SUM, MAX, MIN y COUNT. Se utiliza para producir juegos de resultados que, normalmente, se utilizan para informes de tabulación cruzada. Mientras ROLLUP produce sólo una fracción de las posibles combinaciones subtotales, CUBE produce subtotales para todas las posibles combinaciones de agrupamientos especificados en la cláusula GROUP BY y una suma total.

```
SELECT [column,] group_function(column). . .
FROM table
[WHERE condition]
[GROUP BY [CUBE] group_by_expression]
[HAVING having_expression]
[ORDER BY column];
```

El operador CUBE se utiliza con una función agregada para generar filas adicionales en un juego de resultados. Las columnas incluidas en la cláusula GROUP BY tienen referencias cruzadas para producir un superjuego de grupos. La función agregada especificada en la lista de selección se aplica a dichos grupos para producir valores de resumen para las filas superagregadas adicionales. El número de grupos adicionales del juego de resultados está determinado por el número de columnas incluidas en la cláusula GROUP BY.

De hecho, cada posible combinación de las columnas o expresiones de la cláusula GROUP BY se utiliza para producir superagregados. Si tiene n columnas o expresiones en la cláusula GROUP BY, habrá 2^n posibles combinaciones superagregadas. Matemáticamente, estas combinaciones forman un cubo de n -dimensiones que es la forma con la que el operador obtuvo su nombre.

Mediante herramientas de programación o aplicaciones, estos valores superagregados se pueden introducir en diagramas y gráficos que transmiten resultados y relaciones de forma visual y efectiva.

Ejemplo:

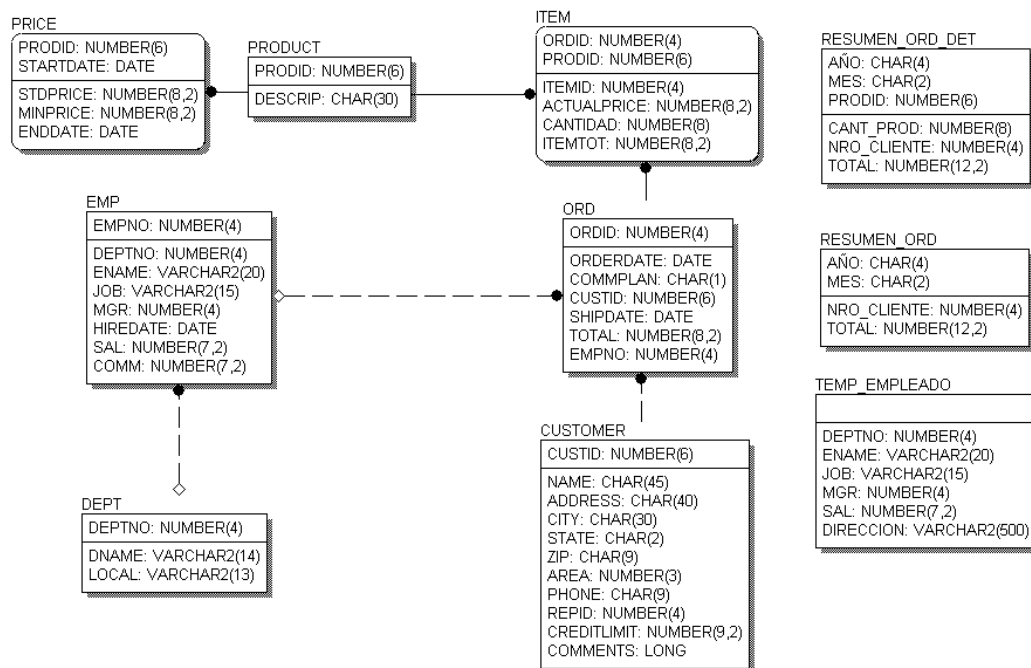
```
SQL> SELECT deptno as department_id,
Job as Job_id,
SUM(sal) salary
FROM emp
WHERE deptno < 60
GROUP BY CUBE(deptno, job);
```

DEPARTMENT_ID	JOB_ID	SALARY
10	CLERK	1300
10	MANAGER	2450
10	PRESIDENT	5000
10		8750
20	CLERK	1900
20	ANALYST	3000
20	MANAGER	2975
20		7875
30	CLERK	950
30	MANAGER	2850
30	SALESMAN	5600
30		9400
50	ANALYST	3000
50		3000
	CLERK	4150
	MANAGER	8275
	PRESIDENT	5000
	ANALYST	6000
	SALESMAN	5600
		29025

15 filas seleccionadas.

Autoevaluación

El siguiente gráfico nos muestra un modelo de una base de datos que almacena información de las ventas de una empresa.



Elabore las siguientes sentencias SQL de acuerdo a lo solicitado

- 1.- Listado de número de empleados por departamento. La información a mostrar es código y nombre de departamento, número de empleados. Debe mostrarse la información de todos los departamentos así no tengan empleados registrados.
- 2.- Listado de ventas anual por producto. El listado debe mostrar los siguientes datos: año, código de producto, nombre de producto, total de ventas.
- 3.- Listado de empleados que han realizado al menos 5 ventas en los últimos 3 meses. Mostrar por cada empleado, el código, el nombre, el número de ventas y el monto vendido.
- 4.- Implemente un update que le incremente el salario en 5% a todos los empleados que en el último año vendieron más que el promedio de ventas.
- 5.- Actualice todas las ordenes de compra de tal forma se asegure que el total de la orden es igual a la suma de los subtotales de sus índices.

**UNIDAD DE
APRENDIZAJE****4**

PROGRAMACIÓN EN ORACLE

LOGRO DE LA UNIDAD DE APRENDIZAJE

Al término de la unidad, el alumno construye programas estructurados utilizando el lenguaje PL/SQL dentro del manejador de base de datos Oracle. Incorpora cursores para procesar grandes volúmenes de información y gestiona los posibles errores de ejecución con el uso de excepciones.

TEMARIO

- Introducción a Oracle PL/SQL
- Creación de bloques PL/SQL
- Sentencias DML en PL/SQL
- Control de Transacciones

ACTIVIDADES PROPUESTAS

- Los alumnos implementarán un bloque PL/SQL.
- Los alumnos resolverán casos prácticos para revisar las características de cada una de las secciones de un bloque PL/SQL.
- Los alumnos implementarán un bloque PL/SQL con acceso a datos haciendo uso de sentencias DML.
- Los alumnos implementarán un bloque PL/SQL con control de transacciones.

4.1 PROGRAMACIÓN PL/SQL

4.1.1 Introducción a Oracle PL/SQL

El lenguaje utilizado para acceder a las bases de datos relacionales es el llamado Lenguaje Estructurado de Consulta (SQL, Structured Query Language), que es muy flexible y transparente, es decir, sencillo y poderoso. SQL es un lenguaje de cuarta generación, lo cual quiere decir que describe lo que quiere hacerse.

Los lenguajes de tercera generación, por ejemplo C y COBOL, son de naturaleza más procedimental. Un programa escrito en lenguaje de tercera generación (3GL) requiere de un algoritmo paso a paso para resolver el problema.

Cada lenguaje tiene sus ventajas y desventajas. Por ejemplos los lenguajes 4GL, aíslan al usuario de los algoritmos y estructuras de datos. Sin embargo, en algunos casos, las estructuras procedimentales de los lenguajes 3GL resultan muy útiles para expresar un determinado programa. Es en este caso que PL/SQL combina la potencialidad de un lenguaje procedimental (PL) con la flexibilidad de un lenguaje de 4GL que es SQL.

PL/SQL son las siglas de Procedural Language/SQL, es decir, amplía la funcionalidad de SQL añadiendo estructuras y funcionalidades de las que pueden encontrarse en otros lenguajes procedimentales, como los que se mencionan a continuación:

4.1.1.1 Modularización en el desarrollo de programas

- Agrupar sentencias lógicamente relacionadas dentro de bloques
- Anidar sub-bloques en bloques mayores para construir programas poderosos
- Particionar un problema complejo en un conjunto de módulos lógicos manejables y bien definidos, e implementarlos mediante bloques.
- Declaración de identificadores
- Declarar variables, constantes, cursores y excepciones y luego utilizarlos en SQL y en sentencias procedimentales.
- Declarar variables pertenecientes a tipos de datos simples y compuestos
- Declarar variables basadas dinámicamente en la estructura de datos de alguna tabla o columna de la base de datos

4.1.1.2 Programación con estructuras de control de lenguajes procedimentales

- Ejecutar una serie de sentencias condicionalmente
- Ejecutar una serie de sentencias en forma iterativa dentro de un loop (ciclo)
- Procesar en forma individual cada una de las filas retornadas por una consulta utilizando un cursor explícito
- Combinar PL/SQL con herramientas de Oracle, tales como Forms de Developer/2000, para agrupar comandos asociados que controlen la ejecución.
- Manejo de errores
- Procesar los errores reportados por el Servidor Oracle mediante rutinas de manejo de excepciones
- Declarar condiciones de error definidas por el usuario y procesarlas con rutinas de manejo de excepciones

4.1.1.3 Portabilidad

- Dado que PL/SQL es propio de Oracle, puede mover programas a cualquier ambiente que soporte Oracle y PL/SQL.

4.1.1.4 Integración

- PL/SQL juega un rol central tanto en el Servidor Oracle (a través de procedimientos almacenados, triggers de la base de datos y paquetes) como en las herramientas de desarrollo de Oracle (a través de los triggers de componentes de Developer/2000).
- Variables y tipos de datos en PL/SQL y SQL son compatibles. Por esto, PL/SQL combina el acceso conveniente a la tecnología de bases de datos con la necesidad de contar con capacidad de programación procedimental.
- Mejora del rendimiento
- PL/SQL puede mejorar el rendimiento de una aplicación. Los beneficios varían dependiendo del ambiente de ejecución.
- PL/SQL agrupa sentencias SQL en un único bloque y envía el bloque completo al servidor en una única invocación, reduciendo así el tráfico en la red.

A continuación se muestra un ejemplo para ver la potencialidad de un PL/SQL:

La siguiente consulta retornará un error:

```
SELECT TO_DATE ('19/06/2002', 'mmddyyyy') FROM DUAL
```

Pero si la hacemos con un bloque PL/SQL podemos capturar el error y procesarlo:

```
DECLARE
    Fecha DATE;
BEGIN
    SELECT TO_DATE('19/06/2012', 'mmddyyyy')
    INTO Fecha
    FROM DUAL;
EXCEPTION
    WHEN OTHERS THEN
        Dbms_output.put_line ('Error en la sentencia');
END;
```


4.1.2 Tipos de datos

La siguiente tabla muestra algunos tipos de datos que se pueden manejar en PL/SQL:

Tipos Escalares	Tipos Compuestos
BINARY_INTEGER	RECORD
DECIMAL (DEC)	TABLE
FLOAT	VARRAY
INTEGER (INT)	
NUMBER	Tipos LOB
REAL	CLOB
CHAR	BLOB
VARCHAR	BFILE
VARCHAR2	
LONG	
LONG RAW	
BOOLEAN	
DATE	

Es posible, también, crear tipos de datos propios. Para ello, se usa la palabra reservada TYPE, de la siguiente forma:

```
TYPE nombre_tipo IS tipo_dato_plsql;
```

Por ejemplo se muestra:

```
TYPE arreglo_char IS TABLE OF VARCHAR2(30);

TYPE registro IS RECORD (campo1 VARCHAR2, campo2 NUMBER NOT NULL);
```

4.1.3 Estructura de Bloques de PL/SQL

Un bloque de código PL/SQL básico puede estar formado a lo sumo por tres partes. A continuación, muestra el orden en que deben escribirse las secciones de los bloques.

Sección	Descripción	Inclusión
Declarativa	Contiene todas las variables, constantes, cursores y excepciones definidas por el usuario que serán referenciadas dentro de la sección Ejecutable.	Opcional
Ejecutable	Contienen sentencias SQL para manipular datos en la base de datos y sentencias PL/SQL para manipular datos en el bloque.	Obligatoria
Manejo de excepciones	Especifica las acciones que se deben tomar cuando ocurre un error y se producen condiciones de ejecución anormales en la sección Ejecutable.	Opcional

Nota: En PL/SQL, un error o advertencia se denomina **excepción (exception)**.

Las palabras clave indicativas de sección DECLARE, BEGIN y EXCEPTION no llevan punto y coma a continuación. Sin embargo, END y todas las demás sentencias PL/SQL requieren un punto y coma para terminar la sentencia.

```

DECLARE
    Variables, constantes, cursores, excepciones definidas por el
    usuario.
BEGIN
    Sentencias SQL.
    Sentencias de control PL/SQL.
EXCEPTION
    Acciones que se llevarán a cabo al producirse algún error.
END;
```

Toda unidad de PL/SQL comprende uno o más bloques. Estos bloques pueden estar completamente separados o anidados uno dentro de otro. Así, un bloque puede representar una pequeña parte de otro bloque que, a su vez, puede ser parte de la unidad de programa completa.

4.1.3.1 Construcciones de Programa PL/SQL

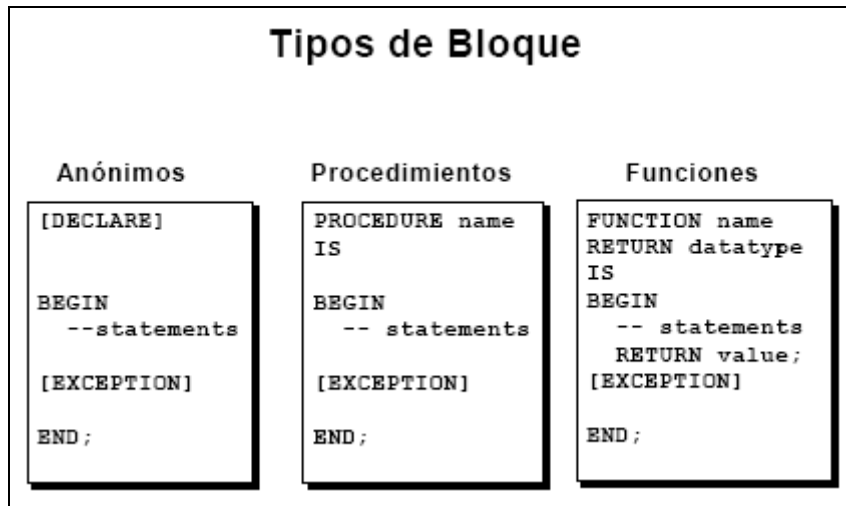
La siguiente lista presenta distintas construcciones de programa PL/SQL que utilizan el bloque de programa básico PL/SQL. Estas construcciones están disponibles de acuerdo al ambiente de ejecución en el que son ejecutadas.

Construcción de programa	Descripción	Disponibilidad
Bloque anónimo	Bloque PL/SQL sin nombre incluida en una aplicación o emitida en forma interactiva.	Todos los ambientes PL/SQL.
Función o Procedimiento almacenado	Bloque PL/SQL con nombre que puede aceptar parámetros y puede ser invocado repetidamente.	Servidor de Base de Datos Oracle 7.X y superior
Función o procedimiento de aplicación	Bloque PL/SQL con nombre que puede aceptar parámetros y puede ser invocado repetidamente.	Componentes de Developer/2000
Paquete (Package)	Módulos PL/SQL con nombre que agrupan procedimientos, funciones e identificadores relacionados.	Servidor de Base de Datos Oracle 7.X y superior.
Trigger de la base de datos	Bloque PL/SQL asociado con una tabla de la base de datos y que es ejecutado automáticamente.	Servidor de Base de Datos Oracle 7.X y superior.
Trigger de aplicación	Bloque PL/SQL asociado con un evento de la aplicación y que es ejecutado automáticamente.	Componentes de Developer / 2000.

Nota: Una función es similar a un procedimiento, con la diferencia de que la función debe devolver un valor.

A. Bloques anónimos

Los bloques anónimos no poseen nombre. Son declarados en el punto de la aplicación en el que deben ser ejecutados y son pasados al motor de PL/SQL para su ejecución en el momento en que la misma se requiere. Puede incluir bloques anónimos dentro de un programa precompilado o dentro de SQLPlus. Los triggers de los componentes de Developer/2000 están formados por estos bloques.



B. Subprogramas

Los subprogramas son bloques PL/SQL con nombre. Puede declararlos como funciones o como procedimientos. Los procedimientos ejecutan acciones sin retornar valores y las funciones retornan valores.

Los componentes de Developer permiten declarar procedimientos y funciones como parte de la aplicación (una forma o reporte) e invocarlos desde otros procedimientos, funciones o triggers.

4.1.3.2 Desarrollo de un Bloque Simple de PL/SQL

Un bloque PL/SQL se compone de hasta tres secciones: declarativa, ejecutable y control de excepciones. Sólo las palabras clave BEGIN y END son obligatorias. Puede almacenar y cambiar valores dentro de un bloque PL/SQL declarando y haciendo referencia a variables y a otros identificadores.

A. Manejo de Variables

- Se declaran e inicializan (si se desea) dentro de la sección de declaración.
- Se asignan valores dentro de la sección ejecutable.

Nota: La palabra clave END puede estar seguida por el nombre del subprograma para lograr mayor claridad.

B. Declarando Variables y Constantes

Es necesario que declare todos los identificadores dentro de la sección de declaración antes de hacer referencia a ellos dentro del bloque PL/SQL. La sintaxis usada es:

`identificador [CONSTANT] tipo de dato [NOT NULL] [:= | DEFAULT expresión]`

Donde:

Identificador: es el nombre del identificador

CONSTANT: restringe el identificador en forma tal que su valor no puede cambiar, las constantes deben ser inicializadas.

Tipo de dato: un tipo de datos escalar o compuesto.

NOT NULL: restringe la variable en forma tal que debe contener siempre un valor; las variables NOT NULL deben ser inicializadas.

Expresión: cualquier expresión PL/SQL tal como un literal, otra variable, o una expresión que incorpore operadores y funciones.

Para asignar valores a una variable se usa el símbolo “:=”.

Adicionalmente existen dos modificadores para declarar variables: %TYPE y %ROWTYPE.

%TYPE es usado para que una variable tome el tipo de otra variable de una columna de una tabla. Por ejemplo:

```
v_pi NUMBER(7,3);  
v_pi2 v_pi%TYPE;  
v_ename emp.ename%TYPE;
```

Este tipo de declaración tiene dos ventajas. La primera es que no se necesita conocer el tipo de dato de la variable o de la columna. La segunda es que el tipo varía de acuerdo con los cambios de la variable origen o de la columna de la tabla en tiempo de ejecución.

%ROWTYPE se usa para definir variables que contendrán registros de una tabla o de un Cursor. Por ejemplo:

```
v_emp_reg emp%ROWTYPE;  
  
CURSOR c_emp IS  
    SELECT empno, sal from EMP;  
  
V_emp_reg1 c_emp%ROWTYPE;
```

Recomendaciones

- Nombre los identificadores con las mismas reglas de nombres utilizadas para los objetos SQL.
- Puede usar convenciones para los nombres, por ejemplo v_nombre para representar una variable y c_nombre para representar una constante.

C. Declarando variables escalares

PL/SQL soporta tres clases de tipos de datos: escalares, compuestos, referencias (apuntadores) y LOB (Large Object), que puede utilizar para declarar variables y constantes.

a) Tipos de datos Escalares

Un tipo de datos escalar mantiene un valor único y no posee componentes internos. Los tipos de datos escalares pueden clasificarse en cuatro categorías: numéricos, caracteres, fecha y hora y booleanos. Los tipos de datos de caracteres y numéricos poseen subtipos que asocian una restricción al tipo de datos. Por ejemplo, INTEGER (entero) y POSITIVE (positivo) son subtipos del tipo de datos básico NUMBER.

Tipo de datos	Descripción
BINARY_INTEGER	Tipo base para los enteros entre -2147483647 y +2147483647.
NUMBER [(precisión, escala)]	Tipo base para los números de punto fijo y flotante.
CHAR [(longitud máxima)]	Tipo base para los datos de caracteres de longitud fija, hasta 2000 bytes. Si no especifica una longitud máxima, la longitud será 1 por defecto.
LONG	Tipo base para los datos de caracteres de longitud variable, hasta 2 Gigabytes.
LONG RAW	Tipo base para los datos binarios, hasta 2 Gigabytes.
VARCHAR2(longitud máxima)	Tipo base para los datos de caracteres de longitud variable, hasta 4000 bytes.
DATE	Tipo base para fechas y horas
BOOLEAN	Tipo base que almacena uno de los tres posibles valores que se obtienen de un cálculo lógico: TRUE, FALSE y NULL.
RAW (size)	Tipo de dato binario RAW, hasta 2000 bytes
LONG RAW	Lo mismo que lo anterior pero hasta 2 Gigabytes.
CLOB	Tipo que almacena simples caracteres, hasta 4 Gigabytes. Significa 'Character Large Object'
BLOB	Data binaria pero hasta 4 Gigabytes. Significa 'Binary Large Object'.
BFILE	Contiene un puntero a un archivo externo a la base de datos. Hasta 4 Gigabytes.

Ejemplos:

```
v_genero CHAR(1);  
v_count BINARY_INTEGER := 0;  
v_total_sal NUMBER(9,2) := 0;  
v_fecha_orden DATE := SYSDATE + 7;  
C_pi CONSTANT NUMBER(3,2) := 3.15;  
v_valid BOOLEAN NOT NULL := TRUE;
```

b) Reglas de Sintaxis de los bloques PL/SQL

- Las palabras reservadas no pueden ser usadas como identificadores.
- No elegir el mismo nombre para un identificador que el nombre de alguna columna de una tabla usada en el bloque. Si los identificadores PL/SQL se encuentran en la misma sentencia SQL y tienen el mismo nombre que una columna, Oracle asume que se intenta referenciar a la columna.
- Los literales de caracteres y fechas deben encerrarse entre comillas simples.
- Los literales numéricos pueden expresarse como un valor simple (por ejemplo, -32.5) o mediante notación científica (por ejemplo, 2E5, que significa 2×10 elevado a la 5 = 200000).
- Los comentarios multilínea pueden estar encerrados entre los símbolos `/*` y `*/`. Un comentario de una línea comienza con `--` y finaliza con la marca de fin de línea.

4.1.3.3 BLOQUES ANIDADOS

Una de las ventajas que brinda PL/SQL sobre SQL, es la posibilidad de anidar sentencias. Puede anidar bloques en cualquier lugar en el que se permita una sentencia ejecutable, convirtiendo de esta forma el bloque anidado en una sentencia. Sin embargo, puede particionar la sección de código de un bloque en bloques más pequeños. La sección de excepciones, también, puede contener bloques anidados.

A. Alcance de una Variable

El alcance de un objeto es la porción del programa en la que se puede hacer referencia a dicho objeto. En la sección de código se puede referenciar a una variable declarada.

Un identificador es visible en el bloque en el que ha sido declarado y en todos los sub-bloques, procedimientos y funciones anidados. Si un bloque no encuentra el identificador declarado lógicamente, comienza a buscarlo en la sección de declaración de los bloques superiores (o padres). Un bloque nunca busca en los bloques que contiene (o hijos) o en bloques del mismo nivel.

El alcance se aplica a todos los objetos declarados, incluyendo variables, cursores, excepciones definidas por el usuario y constantes.

Bloques Anidados y Alcance de las Variables: Ejemplo

```

...
x BINARY_INTEGER;
BEGIN
...
DECLARE
  y NUMBER;
  BEGIN
    ...
    END;
  ...
END;

```

Diagram illustrating variable scope (Alcance) for variables `x` and `y` in PL/SQL blocks:

- `x` is declared in the outer block and its scope (Alcance de x) extends to the end of the outer block.
- `y` is declared in the inner block and its scope (Alcance de y) extends to the end of the inner block.

4.1.3.4 OPERADORES Y FUNCIONES

Los operadores lógicos, aritméticos y de concatenación usados en PL/SQL son los mismos que se utilizan en SQL. Existe además un operador exponencial (**).

La siguiente tabla muestra el orden de precedencia de los operadores:

Operador	Operación
** , NOT	Exponenciación, negación lógica
+ , -	Identidad, negación
* , /	Multiplicación, división
+ , -,	Suma, resta, concatenación
=, <, >, <=, >=, <>, !=, ~=, ^=, IS NULL, LIKE, BETWEEN, IN	Comparación
AND	Conjunción
OR	Inclusión

Los operadores con prioridad más alta son aplicados primero. En los siguientes ejemplos, ambas expresiones producen 8:

$$\begin{aligned}
 5 + 12 / 4 &= 8 \\
 12 / 4 + 5 &= 8
 \end{aligned}$$

Otros ejemplos:

```

V_count := v_count + 1;
V_igual := (v_n1 = v_n2);

```

Los operadores de comparación son >, <, <=, >=, <>, <=>, <=>, <=>, y adicionalmente existen los siguientes:

IS [NOT] NULL: evalúa a verdadero si la variable es NULA.

```
IF X IS NULL THEN Y
...
END IF;
```

[NOT] LIKE: compara patrones de caracteres

```
Ename LIKE 'JON%'
```

[NOT] BETWEEN: compara contra un rango de valores

```
X BETWEEN 1 AND 10
```

IN: verifica si un valor hace parte de una lista

```
X NOT IN (1,2,3,4,5,6,7,8,9,10)
```

NOTA: PL/SQL soporta la concatenación de cadenas de caracteres con el carácter '||' (barras verticales – pipe).

```
'Juan' || 'Carlos' => 'Juan Carlos'
```

La mayoría de las funciones válidas en SQL se pueden utilizar en las expresiones de PL/SQL:

- Funciones numéricas de una fila
- Funciones de caracteres de una fila
- Funciones de conversión de tipos de datos
- Funciones de fecha y hora
- Funciones misceláneas

Las funciones no disponibles en sentencias procedimentales son las siguientes:

- GREATEST y LEAST
- Funciones de grupo: AVG, MIN, MAX, COUNT, SUM, STDDEV y VARIANCE

Las funciones de grupo se aplican a grupos de filas de una tabla y por eso no pueden usarse en bloques PL/SQL, sino sólo dentro de sentencias SQL.

```
v_nombre := UPPER (v_nombre);
v_hoy := TO_DATE(SYSDATE, 'day');
v_limite := ROUND(5,46);
```


Para convertir datos dentro de una expresión, debe asegurarse que los tipos son los mismos. Oracle sólo realiza conversiones automáticas de VARCHAR2 a DATE y de VARCHAR2 a NUMBER. Si se mezclan tipos de datos en una misma expresión, deberá usar las funciones de conversión de tipos de datos que correspondan para realizar la conversión. Utilice las funciones de la lista que se muestra a continuación.

TO_CHAR (valor, formato)
TO_DATE (valor, formato)
TO_NUMBER (valor, formato)

4.1.4 Sentencias SQL en PL/SQL

Las órdenes SQL pueden clasificarse en 6 grupos:

- Las órdenes del lenguaje de manipulación de datos (Data Manipulation Language, DML) permiten modificar o consultar los datos contenidos en una tabla, pero no permiten cambiar la estructura de la tabla u otro objeto en general.
- Las órdenes del lenguaje de definición de datos (Data Definition Language, DDL) permiten crear, eliminar o modificar la estructura de un objeto de un esquema. Las órdenes que tienen que ver con asignación de permisos también caen en este grupo.
- Las órdenes de control de transacciones garantizan la consistencia de los datos, organizando las órdenes SQL en transacciones lógicas que terminan satisfactoriamente o fallan como una sola unidad.
- Las órdenes de control de sesión cambian las opciones de una conexión de la Base de Datos. Tenemos por ejemplo, habilitar un trace SQL.
- Las órdenes de control del sistema cambian las opciones que afectan a la base de datos completa. Por ejemplo, para activar o desactivar el modo Archive.
- Las órdenes SQL empotradas (embedded) se emplean en los programas de precompilación de Oracle.

4.1.4.1 Utilización de SQL en PL/SQL

Las únicas ordenes SQL permitidas en un programa PL/SQL, son las DML y las de control de transacciones. Clásicamente, las órdenes DDL son ilegales, al igual que el EXPLAIN PLAN. Existe, sin embargo, una alternativa, gracias al paquete predefinido DBMS_SQL. Este paquete permite crear, dinámicamente, (en tiempo de ejecución) órdenes SQL, para después analizarlas y luego ejecutarlas. Otra alternativa que ya se tiene en las últimas versiones de Oracle es la cláusula EXECUTE IMMEDIATE.

4.1.4.2 DML en PL/SQL

Las órdenes DML permitidas son SELECT, INSERT, UPDATE y DELETE. SELECT retorna registros de la Base de Datos que cumplan con cierta condición dada en la cláusula WHERE. INSERT agrega registros a una tabla de Base de Datos. UPDATE modifica los registros de una tabla que cumplan con una condición dada en la cláusula WHERE. DELETE elimina los registros de una tabla que cumplan con la condición que se da en el WHERE.

A. Select

La sentencia u oren SELECT extrae datos de la base de datos y los almacena en variables PL/SQL. La forma general de un SELECT es:

```
SELECT elemento_lista_eleccion INTO
  Registro PL/SQL o variable FROM
  Tabla_de_referencia
  WHERE cláusula_where
```

Donde:

- Elemento_lista_elección, es la columna o expresión que se desea seleccionar. Cada elemento de la lista de elección va separada de una coma y se puede identificar mediante un alias. Al conjunto de todos los elementos, también, se le llama lista de selección. La sintaxis * significa todas las columnas de la fila, en el orden de creación de las mismas.
- Variable, es una variable PL/SQL, en que se almacenará un elemento de la lista de selección. Cada variable debe ser compatible con su par (elemento), y es obligatorio que haya el mismo número de elementos que de variables de salida.
- Registro_PL/SQL, se puede usar en lugar de una lista de variables. El registro debe ser compatible con la lista de selección. La ventaja es que permite manejar de manera más sencilla los datos devueltos. Los registros combinan los campos relacionados en una única unidad sintáctica, de modo que se les puede manejar tanto individualmente como en grupo. Esto sería aplicable si la lista de selección es simplemente *, definiendo esta variable 'registro' con el uso de tabla_de_referencia%ROWTYPE.
- Tabla_de_referencia, es la tabla o son las tablas de donde se extraerán los datos. Se puede usar sinónimos y/o vistas, también. Es más, puede ser un sinónimo que mediante 'enlace de base de datos' acceda a una tabla que está en otra Base de datos.
- Cláusula where, condición que deben cumplir los registros para ser mostrados.

Realmente, cualquier tipo de SELECT que se pueda ejecutar desde el SQL*PLUS puede estar incluido en un bloque PL/SQL. Las cláusulas ORDER BY y GROUP BY, también, están incluidas.

Nota.- La forma anterior descrita de la sentencia SELECT no debería devolver más de una fila. La cláusula WHERE deberá filtrar solo una fila, sino se obtendrá un error Oracle.

A continuación se muestra un ejemplo:

```
DECLARE
  v_ename varchar2(100);
BEGIN
  SELECT ename into v_ename FROM emp WHERE
  job='MANAGER';
  dbms_output.put_line('empleado: '||v_ename);
END;
```

Se mostrará un error como el que se muestra:

```
DECLARE
*
ERROR at line 1:
ORA-01422: exact fetch returns more than requested number of rows
ORA-06512: at line 6
```

La explicación es que existen muchos empleados de la tabla EMP, que tienen el job='MANAGER'.

Ahora se prueba un bloque PL/SQL donde se retorna un solo valor:

```
DECLARE
    v_ename varchar2(100);
BEGIN
    SELECT ename into v_ename FROM emp WHERE
        empno=7369;
    dbms_output.put_line('Empleado: '||v_ename);
END;
```

Puede suceder el otro caso, en el que el SELECT no retorne filas. Este es un caso que Oracle lo considera un error:

```
DECLARE
    v_ename varchar2(100);
BEGIN
    SELECT ename into v_ename FROM emp WHERE empno=9999;
    dbms_output.put_line('Empleado: '||v_ename);
END;
```

En este caso, no existe ningún empleado con código 9999. Entonces se mostrará el siguiente error:

```
DECLARE
*
ERROR at line 1:
ORA-01403: no data found
ORA-06512: at line 6
```

B. Insert

La sintaxis de la orden INSERT es la siguiente:

```
INSERT INTO tabla_de_referencia
(nombres_de_columnas)
VALUES (expresiones)
```

Donde tabla_de_referencia, es la tabla en la cual se ingresará un nuevo registro, nombres_de_columnas es la lista de campos que será considerados en la inserción. Esta lista va entre paréntesis. Expresiones son los valores que serán insertados correspondiendo uno a uno a los nombres de columnas anteriormente mencionadas. La expresión, puede ser una sentencia SELECT cuya lista de campos debe ser correspondiente a los nombres de columnas. En este caso, no se debe incluir la cláusula VALUES.

A continuación, se muestra un ejemplo, en el cual se inserta un registro en la tabla DEPT:

```
DECLARE
    v_deptno dept.deptno%TYPE;
    v_dname dept.dname%TYPE;
    v_loc dept.loc%TYPE;
BEGIN
    v_deptno:=50;
    v_dname:='TECNOLOGIA';
    v_loc :='CALIFORNIA';

    INSERT INTO dept (deptno,dname,loc) VALUES
    (v_deptno,v_dname,v_loc);
END;
```

También se puede obviar la parte de la lista de las columnas, siempre y cuando se pongan todos los valores de todos los campos. Se muestra el mismo ejemplo con esta opción:

```
DECLARE
    v_deptno dept.deptno%TYPE;
    v_dname dept.dname%TYPE;
    v_loc dept.loc%TYPE;
BEGIN
    v_deptno:=50;
    v_dname:='TECNOLOGIA';
    v_loc :='CALIFORNIA';

    INSERT INTO dept VALUES (v_deptno,v_dname,v_loc);
END;
```

C. Update

La sintaxis de la orden UPDATE es la siguiente:

```
UPDATE tabla_de_referencia SET
    Nombre_columna1 = expresion1,
    Nombre_columna2 = expresion2,
    Nombre_columna3 = (sentencia_select), ...
WHERE cláusula_where [CURRENT OF cursor]
```

Donde, tabla_de_referencia es la tabla que se actualizará, nombre_ columnaN son las columnas a las que se le asigna un valor, expresionN son los valores correspondientes para cada nombre de columna. El valor que se asigne a una columna puede venir del resultado de una sentencia SELECT. Las condiciones son que el SELECT retorne solo una fila, y los valores de los campos del SELECT sean correspondientes a las columnas que están actualizando en el UPDATE. La cláusula_where es la condición que deben cumplir los registros para ser actualizados. El 'CURRENT OF cursor' es para que cuando dentro de un loop que navega dentro de un cursor se indique que se está actualizando el registro actual donde esta el cursor.

A continuación se muestra un ejemplo:

```
DECLARE
    v_empno number(4):=7934;
BEGIN
    UPDATE emp SET sal=2000 WHERE empno=v_empno;
    dbms_output.put_line('empleado con codigo: '||v_empno||' ha sido
    actualizado ');
END;
```

En el caso de UPDATE, no hay problema si el WHERE actualiza más de un registro o no actualiza nada. En la asignación de valor se puede utilizar como se mencionó antes, una orden SELECT. Por ejemplo, se quiere actualizar el salario del empleado con código 7934, poniéndole 500 soles más de lo que gana el mejor empleado pagado.

```
DECLARE
    v_empno number(4):=7934;
BEGIN
    UPDATE emp SET sal=(SELECT MAX(sal) FROM emp
    WHERE empno=v_empno;
    dbms_output.put_line('empleado con codigo: '||v_empno||' ha sido
    actualizado ');
END;
```

D. Delete

La orden DELETE elimina los registros que cumplen con la condición definida en el WHERE. La sintaxis es la siguiente:

```
DELETE FROM tabla_de_referencia
WHERE cláusula_where
[ CURRENT OF cursor]
```

Donde, tabla_de_referencia es la tabla en la cual se eliminarán los registros correspondientes, cláusula_where es la condición que cumplirán los registros que serán eliminados. 'El CURRENT OF cursor' es para ser usado cuando se está 'barriendo' un cursor y se desea eliminar de la tabla el registro actual del cursor.

A continuación se muestra un ejemplo:

```
DECLARE
    v_empno number(4):=7934;
BEGIN
    DELETE FROM emp WHERE empno=v_empno;
    dbms_output.put_line('empleado con codigo: '||v_empno||' ha sido
    eliminado');
END;
```

En general, para los UPDATE y DELETE se puede saber si al menos un registro cumplió la condición del WHERE. Esto se hace usando SQL%NOTFOUND. A continuación, se tratará de eliminar un empleado que no existe (código 9999):

```

DECLARE
    v_empno number(4):=9999;
BEGIN
    DELETE FROM emp WHERE empno=v_empno;
    IF SQL%NOTFOUND THEN
        dbms_output.put_line('empleado con codigo: `'||v_empno||` no
        existe `');
    ELSE
        dbms_output.put_line('empleado con codigo: `'||v_empno||` ha
        sido eliminado `');
    END IF;
END;

```

E. Uso de Rowid

Rowid es una pseudocolumna que se utiliza en la lista de selección de una consulta. Es un identificador único de cualquier registro en TODA la base de datos. Está formado por el identificador del objeto, el datafile donde está el registro, el bloque Oracle y la fila dentro del bloque. Resumiendo, es la ubicación física de cada registro. A continuación se muestra un ejemplo:

```

SQL> SELECT rowid,dname FROM dept;
ROWID                                DNAME
-----                                -
AAAHW5AABAAAMUSAAA    ACCOUNTING
AAAHW5AABAAAMUSAAAB   RESEARCH
AAAHW5AABAAAMUSAAAC   SALES
AAAHW5AABAAAMUSAAD    OPERATIONS
AAAHW5AABAAAMUSAAE    TECNOLOGIA

```

Como el rowid es la ubicación física de cada registro, es la manera más rápida de buscar un registro de una tabla. Por eso, muchos trabajos de procesos masivos (por ejemplo actualizaciones y eliminaciones masivas) se hacen usando el rowid. Por ejemplo, se puede eliminar el registro del departamento de SALES mediante:

```

DELETE FROM dept WHERE rowid='AAAHW5AABAAAMUSAAAC';

```

4.1.4.3 Control de Transacciones

Una transacción es una serie de órdenes SQL que se completa o falla como una unidad. Las transacciones sirven para evitar la inconsistencia de los datos.

Un ejemplo clásico es el de una transacción bancaria. Por ejemplo, en una transferencia se tendría:

```
...  
UPDATE tabla_cuenta SET saldo= saldo - monto_transferencia  
WHERE cuenta = v_cuenta_bancaria_origen;
```

```
-- (*) Hay problemas si falla en este punto!!!!!!  
UPDATE tabla_cuenta SET saldo= saldo + monto_transferencia  
WHERE cuenta = v_cuenta_bancaria_destino; ...
```

Si por un motivo hay una falla de base de datos, hardware, red o cualquier otra causa en (*), y hasta allí SI se aplicaran los cambios hechos, la información quedaría inconsistente, pues se habría descontado de la cuenta_origen pero no cargado en la cuenta_destino. Para resolver esta inconsistencia la solución es que las dos sentencias UPDATEs pertenezcan a una misma transacción.

A. Commit y rollback

Cuando se da el comando COMMIT a la base de datos, termina la transacción y:

- Se hace permanente todo el trabajo realizado por la transacción.
- Otras sesiones pueden ver los cambios realizados por esta transacción
- Se liberan todos los bloques establecidos por la transacción.

Cuando se da a la base de datos una orden ROLLBACK, la transacción termina y:

- Todo el trabajo hecho por la transacción se deshace, como si no se hubieran emitido las órdenes correspondientes.
- Se liberan todos los bloques establecidos por la transacción.

A continuación se muestra un ejemplo:

```
DECLARE  
    v_empno number(4):=7934;  
BEGIN  
    DELETE FROM emp WHERE empno=v_empno;  
    COMMIT;  
END;
```

La sentencia ROLLBACK explícita se usa cuando se detecta un error en el programa que impide continuar con el trabajo. Si una sesión se desconecta de la base de datos, sin terminar la transacción actual con COMMIT o ROLLBACK, la transacción es, automáticamente, cancelada por la base de datos.

A continuación se muestra un ejemplo:

```
DECLARE  
    v_empno number(4):=7934;  
BEGIN  
    DELETE FROM emp WHERE empno=v_empno;  
    ROLLBACK;  
END;
```

B. Puntos de salvaguarda

La orden ROLLBACK deshace toda la transacción, como hemos visto. Con el comando SAVEPOINT, se puede deshacer parte de la transacción.

```
SAVEPOINT nombre;
```

Donde nombre es el nombre del punto de salvaguarda. Los nombres de los puntos de salvaguarda se basan en las reglas para los identificadores SQL. Los puntos de salvaguarda no se declaran en la sección declarativa, puesto que son globales para la transacción y la transacción puede continuar más allá del final del bloque. Una vez que se define el punto de salvaguarda, el programa puede deshacer la transacción hasta el punto de salvaguarda, con la siguiente sintaxis:

```
ROLLBACK TO SAVEPOINT nombre;
```

Cuando se realiza un ROLLBACK TO SAVEPOINT, sucede lo siguiente:

- Cualquier trabajo realizado desde el punto de salvaguarda se deshace. El punto de salvaguarda permanece activo y se puede volver a él de nuevo, si se desea.
- Se libera cualquier bloqueo establecido y recursos adquiridos por las órdenes SQL desde el punto de salvaguarda.
- La transacción NO finaliza, dado que hay órdenes SQL todavía pendientes.

A continuación se muestra un ejemplo:

```
SQL> select * from dept;
DEPTNO  DNAME          LOC
-----  -
10      ACCOUNTING     NEW YORK
20      RESEARCH       DALLAS
30      SALES          CHICAGO
40      OPERATIONS     BOSTON

DECLARE
v_deptno number(2):=80;
BEGIN
  INSERT INTO DEPT values (v_deptno,'CONTABILIDAD','LIMA');
  SAVEPOINT A;
  INSERT INTO DEPT values (v_deptno+1,'CAJA','TACNA');
  SAVEPOINT B;
  INSERT INTO DEPT values (v_deptno+2,'GERENCIA','LIMA');
  SAVEPOINT C;
  INSERT INTO DEPT values (v_deptno+3,'RRHH','PIURA');
  ROLLBACK TO B;
  -- no se inserta ni el depto de GERENCIA ni de RRHH, CAJA y
  CONTABILIDAD si
  COMMIT;
END;
```



```
SQL> select * from dept;
```

```
DEPTNO DNAME      LOC
```

```
-----
```

```
10 ACCOUNTING     NEW YORK
```

```
20 RESEARCH       DALLAS
```

```
30 SALES          CHICAGO
```

```
40 OPERATIONS     BOSTON
```

```
80      CONTABILIDAD  LIMA
```

```
81      CAJA         TACNA
```

Autoevaluación

1.- Implemente un programa PL/SQL que reciba los datos de un nuevo grado salarial y los inserte en la tabla SALGRADE. Debe verificar que este nuevo rango no coincida con ninguno de los previamente ingresados.

2.- Implemente un programa PL/SQL que reciba como dato el id de una orden y actualice el total de la orden a partir de la suma de los subtotales de sus ítems asociados.

3.- Implemente un programa PL/SQL que reciba como data el código de un empleado y le asigne como comisión el 5% de sus ventas directas así como el 1% de las ventas totales del área a la que pertenece.

4.2 ESTRUCTURAS DE CONTROL EN PL/SQL

Puede cambiar el flujo lógico de las sentencias dentro de un bloque PL/SQL con un conjunto de estructuras de control. Existen dos tipos de estructuras de control:

- Estructuras de control condicionales
- Estructuras de control de ciclos o repeticiones

4.2.1 Estructuras Condicionales

4.2.1.1 La sentencia IF-THEN-ELSE

La estructura de la sentencia IF de PL/SQL es similar a la estructura de la sentencia IF de otros lenguajes. Le permite a PL/SQL realizar acciones en forma selectiva, dependiendo de las condiciones que se cumplen. La sintaxis es:

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

Las cláusulas ELSIF y ELSE son opcionales y pueden haber tantas cláusulas ELSIF como se desee.

A continuación se muestran varios ejemplos:

```
IF v_last_name = 'Dumas' THEN  
    v_job := 'Representante de ventas';  
    v_region := 35;  
END IF;  
... ..  
IF v_date_shipped - v_date_ordered < 5 THEN  
    v_ship_flag := 'Acceptable';  
ELSE  
    v_ship_flag := 'Inacceptable';  
END IF;  
... ..  
IF v_start > 100 THEN  
    RETURN (2 * v_start);  
ELSIF v_start >= 50 THEN  
    RETURN (.5 * v_start);  
ELSE  
    RETURN (.1 * v_start);  
END IF;
```

Recomendaciones

- Cualquier expresión que contenga un valor nulo retorna NULL como resultado, con la excepción de la expresión de concatenación que trata los valores nulos como cadenas vacías.
- Cualquier comparación simple que contenga un valor nulo retorna NULL.
- Una comparación IS NULL retorna como resultado TRUE o FALSE.
- La negación de NULL (NOT NULL) retorna NULL como resultado, puesto que los valores nulos son indeterminados.

En el siguiente ejemplo se esperaría ejecutar las sentencias debido a que X e Y son diferentes. Pero NULL es indeterminado, así que no se sabe si X es o no igual a Y. De allí que la condición retorne NULL y la secuencia de sentencias no se ejecute.

```
x := 5;
y := NULL;
...
IF x != y THEN -- retorna NULL, no TRUE
    secuencia_de_sentencias; -- no ejecutado
END IF;
```

En el siguiente ejemplo se esperaría ejecutar las sentencias debido a que z y b parecen iguales, pero de nuevo esto no se conoce, así que la condición retorna NULL.

```
a := NULL;
b := NULL;
...
IF a = b THEN -- retorna NULL, no TRUE
    secuencia_de_sentencias; -- no ejecutado
END IF;
```

En lo posible use ELSIF en vez de sentencias IF anidadas:

IF Anidado	ELSIF
IF condición1 THEN sentencias1; ELSE IF condición2 THEN sentencias2; ELSE sentencias3; END IF; END IF;	IF condición1 THEN sentencias1; ELSIF condición2 THEN sentencias2; ELSE sentencias3; END IF;

El uso del ELSIF ayuda a reducir código, así como también un mejor entendimiento (intuitivo) del mismo.

4.2.1.2 La sentencia CASE

La sintaxis es la siguiente:

```
CASE
    WHEN expresion1 THEN
        secuencia_de_instrucciones;
    WHEN expresion2 THEN
        ...
    WHEN expresionN THEN
        secuencia_de_instrucciones;
    [ELSE
        secuencia_de_instrucciones;]
END CASE;
```

Nota.- Si no se utiliza el “ELSE” puede salir un error CASE_NOT_FOUND sino se cumple ninguna condición.

A continuación se muestra un ejemplo:

DECLARE

v_comision NUMBER;

BEGIN

CASE

```

WHEN v_comision is null THEN          -- si comision es nula
    DBMS_OUTPUT.PUT_LINE('Comision '||v_comision||' es nula');
WHEN v_comision > 0 THEN      -- si comision es mayor a cero
    DBMS_OUTPUT.PUT_LINE('Comision '||v_comision||' es positiva');
WHEN v_comision < 0 THEN      -- si comision es menor a cero
    DBMS_OUTPUT.PUT_LINE('Comision '||v_comision||' es negativa');
ELSE          -- Otros casos
    DBMS_OUTPUT.PUT_LINE('Comision '||v_comision||' es cero');

```

END CASE;

END;

Este caso de CASE, mostrado, se basa en el analisis de condiciones o expresiones en general. Una expresión se traduce en operaciones BOOLEANAS que pueden retornar TRUE o FALSE.

Pero existe otro tipo de CASE donde, directamente, se ‘pregunta’ por el valor de una variable. A continuación se muestra dicha sintaxis:

CASE [variable]

```

WHEN valor1 THEN
    secuencia_de_instrucciones;
WHEN valor2 THEN
    ...
WHEN valorN THEN
    secuencia_de_instrucciones;
[ELSE
    secuencia_de_instrucciones;]

```

END CASE;

A continuación se muestra un ejemplo:

DECLARE

v_comision NUMBER :=20;

BEGIN

CASE v_comision

```

WHEN null THEN
    DBMS_OUTPUT.PUT_LINE('Comision es nula');
WHEN 10 THEN
    DBMS_OUTPUT.PUT_LINE('Comision es diez');
WHEN 20 THEN
    DBMS_OUTPUT.PUT_LINE('Comision es veinte');
ELSE
    DBMS_OUTPUT.PUT_LINE('Comision no cumple ninguna condicion!!');

```

END CASE;

END;

4.2.2 Estructuras Cíclicas

4.2.2.1 La sentencia LOOP

PL/SQL provee varias formas de estructurar ciclos para repetir una sentencia o un conjunto de ellas varias veces. Muchos lenguajes proveen una cláusula que permite escoger el incremento del contador en cada iteración (Por ejemplo STEP), pero PL/SQL no la posee.

El contador del ciclo no necesita ser explícitamente declarado porque PL/SQL lo declara de tipo INTEGER.

El loop más simple consiste en una secuencia de sentencias encerradas entre los delimitadores LOOP y END LOOP. Cada vez que el flujo de ejecución alcanza la sentencia END LOOP, el control es retornado a la correspondiente sentencia LOOP anterior a la misma. Este loop sin control es infinito y debe ser evitado. Para evitar un loop infinito, se agrega una sentencia EXIT.

La sentencia EXIT

Puede terminar un loop usando la sentencia EXIT. El control pasara a la sentencia que se encuentra a continuación de la sentencia END LOOP. Puede ejecutar EXIT como la acción de una sentencia IF, o como una sentencia independiente dentro del loop. En el último caso, puede asignar una cláusula WHEN para permitir la terminación condicional del loop. La sintaxis es como sigue:

```
LOOP
    sentencia1;
    sentencia2;
EXIT [WHEN condicion]
END LOOP;
```

Ejemplo:

```
... ..
v_ord_id s_item.ord_id%TYPE := 101;
v_counter NUMBER (2) := 1;

BEGIN
... ..
LOOP
    Je,INSERT INTO s_item (ord_id, item_id)
    VALUES (v_ord_id, v_counter);
    v_counter := v_counter + 1;
    EXIT WHEN v_counter > 10;
END LOOP;
```

4.2.2.2 La sentencia FOR (Bucle)

El loop FOR tiene la misma estructura general que los loops vistos hasta aquí. Tienen, además, una sentencia de control delante de la palabra clave LOOP que determina la cantidad de iteraciones que realizará PL/SQL. La sintaxis es como sigue:

```
FOR index IN [REVERSE] limite inferior..limite superior
LOOP
    sentencia1;
    sentencia2;
    ... ..
END LOOP;
```

Ejemplo:

```
FOR i IN v_lower..v_upper LOOP
    v_counter := v_counter + 1;
    v_output := i;
END LOOP;

DBMS_OUTPUT.PUT_LINE ('El ultimo valor es ' || to_char (v_output) || '.
Total loop = ' || to_char(v_counter));
```

4.2.2.3 La sentencia WHILE (Bucle)

Puede utilizar el loop WHILE para repetir un conjunto de sentencias hasta que la condición de control no sea válida (TRUE). La condición será evaluada al comienzo de cada iteración. El loop termina cuando la condición es FALSE. Si la condición es FALSE al comienzo del loop, entonces no se iterará nuevamente. La sintaxis es la siguiente:

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

Ejemplo:

```
WHILE v_counter <=50 LOOP
    INSERT INTO temp_table VALUES (v_counter,'Loop Index');
    v_counter:=v_counter+1;
END LOOP;
```

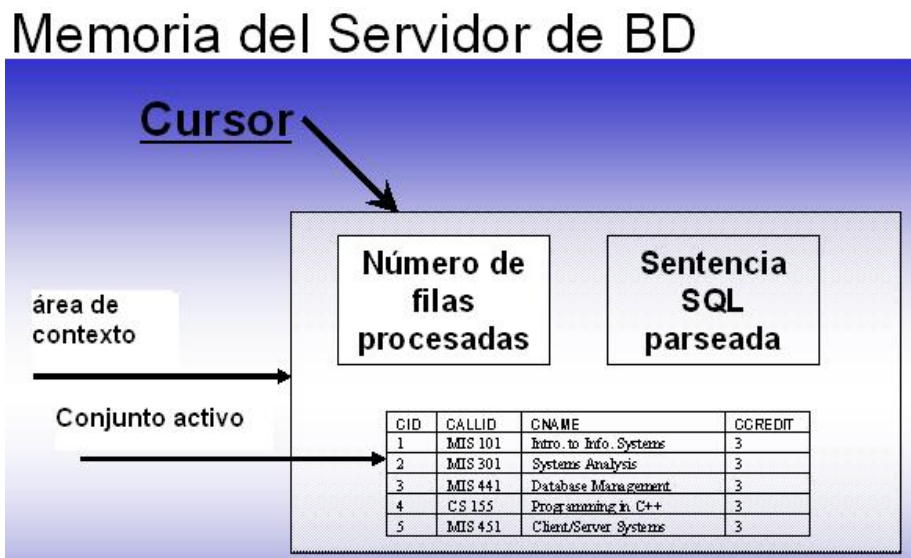
Autoevaluación

- 1.- Implemente un bloque PL/SQL que imprima en pantalla los 20 primeros números de la serie de FIBONACCI.
- 2.- Implemente un bloque PL/SQL que imprima el calendario del mes actual. Debe imprimir la siguiente información: día de semana, número de día, nombre del mes y año.
- 3.- Implemente un program que imprima en pantalla los primeros 50 números capicúas mayores a 1000.

4.3 CURSORES

Para poder procesar una orden SQL, Oracle asignan un área de memoria que recibe el nombre de área de contexto. Esta área contiene informaciones necesarias para completar el procesamiento, incluyendo el número de filas procesadas por la orden, un puntero a la versión analizada de la orden 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 el cursor, un programa PL/SQL puede controlar el área del contexto y en lo que ella sucede a medida que se procesa la orden. El siguiente bloque PL/SQL ilustra un bucle, en el cual a cada empleado se le aumenta el sueldo (salary) en 10%:



A continuación se muestra un ejemplo sencillo de manejo de cursores.

```

DECLARE

    CURSOR sal_cursor IS
        SELECT employee_id, salary
        FROM employees;

BEGIN

    FOR emp_record IN sal_cursor LOOP
        UPDATE employees
        SET salary = emp_record.salary * 1.10
        WHERE employee_id = emp_record.employee_id;
    END LOOP;
    COMMIT;

END;
```

Este ejemplo práctico demuestra el uso de cursores explícitos. Las sentencias simples de SQL manejan cursores implícitos. El procesamiento de un cursor explícito consta de 4 fases que se explica a continuación.

4.3.1 Tipos de Cursores

Los cursores en PL/SQL se clasifican en 2: implícitos y explícitos. Los cursores implícitos son declarados por el motor de base de datos para ejecutar sentencias SQL de manipulación de datos, incluyendo consultas que retornen una sola fila. Para consultas que retornan más de una fila se deben usar cursores explícitos y ciclos FOR LOOP para recorrerlos.

4.3.1.1 Cursores explícitos

Los 4 pasos necesarios para el manejo de un cursor explícito son los siguientes:

- Declaración del cursor
- Activación del cursor para una consulta
- Almacenamiento de los resultados en variables PL/SQL
- Cierre del cursor

La declaración del cursor es el único paso que se lleva en la sección de declaración de un bloque PL/SQL. Los otros pasos adicionales se encuentran en las secciones ejecutables o de manejo de excepciones.

4.3.2 Declaración de cursores

La declaración de un cursor se hace utilizando una sentencia SELECT. La sintaxis es como sigue:

```
CURSOR nombre_de_cursor IS sentencia_select
```

Donde nombre_del_cursor es el nombre que se le define al cursor y sentencia_SELECT es la consulta que el cursor ejecutará. En la sentencia SELECT no va la cláusula INTO, ésta es parte de la cláusula FETCH.

Como el cursor se define a partir de una sentencia SELECT, ésta puede ser de cualquier complejidad, conteniendo joins, subconsultas, acceso a otra base de datos con database links, entre otros. A continuación se muestra algunos ejemplos:

```
DECLARE  
CURSOR sal_cursor IS  
  SELECT employee_id, salary  
  FROM employees;
```

```
DECLARE  
CURSOR emp_adm IS  
  SELECT *  
  FROM Employees  
  WHERE department_id IN (SELECT department_id  
                        FROM departments  
                        WHERE department_name='Administration')
```

En el ejemplo 2, se muestra la definición de un cursor que manipula la información de todos los empleados que están en el departamento de **'Administration'**. Para esto se ha usado sentencias anidadas (subconsulta).

4.3.2.1 USO DE %TYPE y %ROWTYPE

Muchas veces es más práctico declarar el tipo de dato de una variable sólo haciendo referencia indirecta al tipo de dato de una columna de una tabla o al conjunto de registros de una tabla o de un cursor

A continuación se muestran algunos ejemplos:

```
DECLARE
CURSOR emp_adm IS
  SELECT employee_id, salary
  FROM employees
  WHERE department_id in (SELECT department_id
                        FROM departments
                        WHERE department_name='Administration');

v_emp employees.employee_id%TYPE;
v_sal employees.salary%TYPE;

BEGIN
  OPEN emp_adm;
  FETCH emp_adm INTO v_emp, v_sal;
  UPDATE employees set salary=v_sal*1.10
  WHERE employee_id=v_emp;
END;
```

```
DECLARE
CURSOR emp_adm IS
  SELECT employee_id, salary
  FROM employees
  WHERE department_id in (SELECT department_id
                        FROM departments
                        WHERE department_name='Administration');

r_emp employees%ROWTYPE;

BEGIN
  OPEN emp_adm;
  FETCH emp_adm INTO r_emp;
  UPDATE employees
  SET salary=r_emp.salary*1.05
  WHERE employee_id=r_emp.employee_id;
END;
```

4.3.3 Apertura de cursores

La sintaxis para abrir un cursor es la siguiente:

```
OPEN nombre_de_cursor;
```

Donde *nombre_del_cursor* identifica un cursor que se ha definido en la sección de declaración. Cuando se abre un cursor, suceden tres cosas:

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

4.3.4 Almacenamiento (extracción) de datos de cursores

La cláusula INTO de la consulta es parte de la orden FETCH. Tiene dos modalidades:

```
FECTH nombre_de_cursor INTO lista_variables;
```

y

```
FECTH nombre_de_cursor INTO registro_PL/SQL;
```

Donde *nombre_del_cursor* identifica un cursor previamente definido en la parte declarativa, y además debe estar abierto. *Lista de variables* es una lista de variable PL/SQL donde se almacenarán cada fila recorrida del cursor. *Registro_PL/SQL* es una estructura que almacenará los datos de cada fila del cursor. En ambos casos, éstos deben haber sido declarados en la sección declarativa, además de ser compatible con los tipos de datos que existe en cada fila.

Después de cada FETCH, el puntero del conjunto activo es incrementado para que apunte a la fila siguiente.

A continuación se muestran algunos ejemplos:

```
DECLARE
  CURSOR emp_adm IS
    SELECT mployee_id, salary
    FROM Employees
    WHERE department_id IN (SELECT department_id
                           FROM departments
                           WHERE department_name='Administration');

  v_emp number(6);
  v_sal number(8,2);

BEGIN
  OPEN emp_adm;
  FETCH emp_adm INTO v_emp, v_sal;
  UPDATE employees
  SET salary=v_sal*1.10
  WHERE employee_id=v_emp;
END;
```

```
DECLARE
  CURSOR emp_adm IS
    SELECT *
    FROM Employees
    WHERE department_id IN (SELECT department_id
                           FROM departments
                           WHERE department_name='Administration');

  v_emp number(6);
  v_sal number(8,2);

BEGIN
  OPEN emp_adm;
  FETCH emp_adm INTO v_emp, v_sal;
  UPDATE employees
  SET salary=v_sal*1.10
  WHERE employee_id=v_emp;
END;
```

El segundo ejemplo mostrará un error, debido a que existe incompatibilidad entre el número de campos que se definen en el SELECT del cursor, y el número de variables de almacenamiento usadas en el FETCH. La tabla EMPLOYEES tiene 11 campos. El error será como el que se muestra:

```
fetch emp_adm into v_emp, v_sal;
*
ERROR at line 11:
ORA-06550: line 11, column 3:
PLS-00394: wrong number of values in the INTO list of a FETCH statement
ORA-06550: line 11, column 3:
PL/SQL: SQL Statement ignored
```

4.3.5 Cierre de cursores

Cuando se ha terminado de retornar todas las filas del conjunto activo, debe cerrarse el cursor. Con esto se libera los recursos que fueron necesarios para abrir el cursor. La sintaxis de cierre de un cursor es como sigue:

```
CLOSE nombre_de_cursor;
```

Donde, nombre_del_cursor, identifica un cursor anteriormente abierto.

Una vez que el cursor ha sido cerrado, no podrá ejecutarse un FETCH sobre el mismo. Le aparecería el siguiente error:

```
DECLARE
*
ERROR at line 1:
ORA-01001: invalid cursor
ORA-06512: at line 15
```

4.3.6 Atributos de los cursores

Adicionalmente, los cursores cuentan con ciertos atributos que permiten obtener información acerca de la manipulación de datos. Estos atributos son los siguientes:

4.3.6.1 %FOUND

Retorna TRUE cuando una sentencia FETCH ha retornado una fila. No se puede usar antes del FETCH porque retorna NULL. Ejemplo:

```
LOOP
FETCH c1 INTO mi_ename, mi_sal, mi_hiredate;
IF c1%FOUND THEN
...
ELSE
...
EXIT;
END IF;
END LOOP;
```

4.3.6.2 %ISOPEN

Retorna TRUE si un cursor está abierto, de lo contrario retorna FALSE. Ejemplo:

```
IF c1%ISOPEN THEN -- cursor está abierto
...
ELSE -- cursor está cerrado, luego se abre
OPEN c1;
END IF;
```

4.3.6.3 %NOTFOUND

Es el opuesto lógico a %FOUND. Retorna TRUE si el último FETCH no retorna ninguna fila. Se usa comúnmente para controlar el recorrido del cursor. Ejemplo:

```
LOOP
FETCH c1 INTO mi_ename, mi_sal, mi_hiredate;
EXIT WHEN c1%NOTFOUND;
...
END LOOP;
```

4.3.6.4 %ROWCOUNT

Cuando un cursor es abierto %ROWCOUNT retorna cero (0). Inmediatamente después del primer FETCH también retorna cero (0). De allí en adelante, retorna el número de filas obtenidas hasta el momento. Ejemplo:

```
LOOP
FETCH c1 INTO mi_ename, mi_deptno;
IF c1%ROWCOUNT > 10 THEN
...
END IF;
...
END LOOP;
```

Si un cursor no está abierto y se referencia usando %ROWCOUNT, se dispara la excepción INVALID_CURSOR.

Estos mismos atributos se usan para los cursores implícitos agregando antes del símbolo porcentaje (%) la palabra SQL. Por ejemplo:

```
DELETE FROM emp
WHERE empno = my_empno;
IF SQL%FOUND THEN -- eliminación exitosa
INSERT INTO new_emp VALUES (my_empno, my_ename, ...);
```

4.3.7 Uso avanzado de cursores

4.3.7.1 Cursores con parámetros

Hasta el momento, hemos visto que la declaración del cursor en la sección DECLARE, se define con un SELECT 'estático', es decir, siempre que se abra el cursor, éste ejecutará la misma sentencia con las mismas condiciones. Existen casos en que sí se necesita pasar parámetros al cursor.

```
DECLARE
CURSOR emp_cursor (v_employee_id varchar2) IS
  SELECT first_name, last_name
  FROM employees
  WHERE employee_id=v_employee_id;

  v_first_name employees.first_name%TYPE;
  v_last_name employees.last_name%TYPE;

BEGIN
  OPEN emp_cursor(199);
  FETCH emp_cursor INTO v_first_name, v_last_name;
  dbms_output.put_line ('Seleccionó a: ' || v_first_name || ' ' || v_last_name);
END;
```

4.3.7.2 Cursores en estructuras cíclicas

G. LOOP ... EXIT WHEN

Sintaxis:

```
OPEN cursor_name;
LOOP
    FETCH cursor_name INTO variable_name(s);
    EXIT WHEN cursor_name%NOTFOUND;
...
END LOOP;
CLOSE cursor_name;
```

A continuación se muestra un ejemplo:

```
DECLARE
    CURSOR c_dpto IS
        SELECT department_id, department_name
        FROM departments;

    v_department_id departments.department_id%TYPE;
    v_department_name departments.department_name%TYPE;

BEGIN
    OPEN c_dpto;
    LOOP
        FETCH c_dpto INTO v_department_id, v_department_name;
        EXIT WHEN c_dpto%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE (v_department_id||' - '|| v_department_name);
    END LOOP;
    CLOSE c_dpto;
END;
```

H. FOR LOOP

En este caso, no se necesita abrir el cursor ni cerrarlo. Eso se realiza implícitamente. Existirá una variable tipo 'registro' que se declarará automáticamente al momento de iniciar el FOR.

```
FOR variable_name(s) in cursor_name LOOP
    Procesamiento_adicional__de_sentencias;
END LOOP;
```

A continuación se muestra un ejemplo:


```
DECLARE
  CURSOR c_dpto IS
    SELECT department_id, department_name
    FROM departments;

BEGIN
  FOR v_department IN c_dpto LOOP
    DBMS_OUTPUT.PUT_LINE (v_department.v_department_id || ' - ' ||
                          v_department.v_department_name);
  END LOOP;
END;
```

I. WHILE Loop

Es similar a los dos anteriores, pero tiene algunas particulares. Se muestra con un ejemplo su funcionamiento:

```
DECLARE
  CURSOR c_dpto IS
    SELECT department_id, department_name
    FROM departments;

  v_department_id departments.department_id%TYPE;
  v_department_name departments.department_name%TYPE;

BEGIN
  OPEN c_dpto;
  FETCH c_dpto INTO v_department_id, v_department_name;
  WHILE c_dpto%FOUND LOOP
    DBMS_OUTPUT.PUT_LINE (v_department_id || ' - ' || v_department_name);
    FETCH c_dpto INTO v_department_id, v_department_name;
  END LOOP;
END;
```

4.3.7.3 Cláusula FOR UPDATE

Muchas veces, el proceso que se lleva a cabo en un bucle de extracción modifica las filas extraídas por el cursor. PL/SQL proporciona una sintaxis conveniente para estas situaciones. Las alternativas son las cláusula FOR UPDATE y WHERE CURRENT OF (se explicarán en el siguiente punto).

FOR UPDATE es usado para bloquear registros de un cursor que van a ser actualizados o eliminados.

La sintaxis es como sigue:

```
SELECT .. FROM ... FOR UPDATE [OF columna_de_referencia] [NOWAIT]
```

Donde, *columna_de_referencia* es una columna o lista de columnas, de la tabla sobre la que se realiza la consulta y se desea bloquear. A cláusula NOWAIT hace que, si las

filas están bloqueadas por otra sesión, la orden OPEN termine, inmediatamente, devolviendo el error:

ORA – 00054: resource busy an acquire with NOWAIT specified

A continuación se muestra un ejemplo:

```
DECLARE
  CURSOR emp_cursor IS
    SELECT empno, ename, sal
    FROM emp
    WHERE deptno = 30
    FOR UPDATE NOWAIT;
```

4.3.7.4 Cláusula WHERE CURRENT OF

Si se declara el cursor con la cláusula FOR UPDATE, puede emplearse la cláusula WHERE CURRENT OF en una orden UPDATE o DELETE.

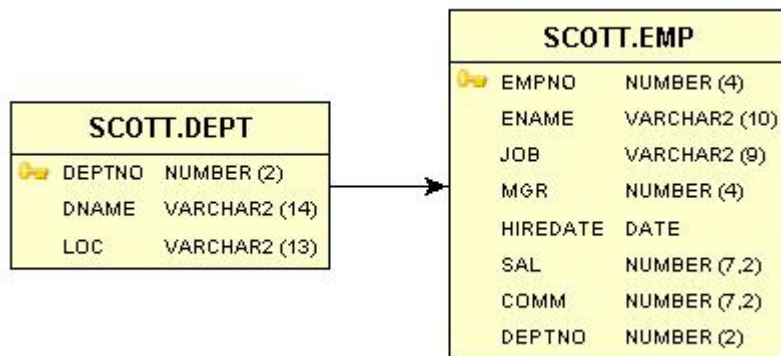
La cláusula WHERE CURRENT OF hace referencia a la fila recién extraída por el cursor

A continuación se muestra un ejemplo:

```
DECLARE
  CURSOR sal_cursor IS
    SELECT    sal
    FROM      emp
    WHERE     deptno = 30
    FOR UPDATE NOWAIT;

BEGIN
  FOR emp_record IN sal_cursor LOOP
    UPDATE    emp
    SET       sal = emp_record.sal * 1.10
    WHERE CURRENT OF sal_cursor;
  END LOOP;
  COMMIT;
END;
```

Autoevaluación



1.- Imprima por pantalla los empleados que tienen el mismo trabajo que JONES

```
DECLARE
  CURSOR e IS
  SELECT ename
  FROM emp
  WHERE job = (SELECT job FROM emp WHERE ename = 'JONES');

BEGIN
  FOR vr_reg IN e LOOP
    Dbms_output.put_line(vr_reg.ename);
  END LOOP;
END;
```

2.- Liste los empleados que tienen un salario mayor que el de algún vendedor (SALESMAN)

```
DECLARE
  CURSOR e IS
  SELECT ename FROM emp
  WHERE sal > ALL (SELECT sal FROM emp WHERE job = 'SALESMAN');

BEGIN
  FOR vr_reg IN e LOOP
    Dbms_output.put_line(vr_reg.ename);
  END LOOP;
END;
```

3.- Seleccione el nombre, departamento, empleo y salario del empleado mejor pagado de cada departamento

```
DECLARE
  CURSOR e IS
  SELECT deptno, ename, job, sal
  FROM emp
  WHERE sal in (SELECT MAX(sal) FROM emp GROUP BY deptno);

BEGIN
  FOR vr_reg IN e LOOP
    Dbms_output.put_line(vr_reg.ename);
    Dbms_output.put_line(vr_reg.deptno);
    Dbms_output.put_line(vr_reg.job);
    Dbms_output.put_line(vr_reg.sal);
  END LOOP;
END;
```

4.- Liste los departamentos que tengan al menos dos administrativos (CLERK)

```
DECLARE
  CURSOR c IS
  SELECT dname, count (job)
  FROM dept, emp
  WHERE dept.deptno = emp.deptno AND job = 'CLERK'
  GROUP BY dname
  HAVING count(job) >= 2;

BEGIN
  FOR vr_reg IN c LOOP
    Dbms_output.put_line(vr_reg.dname);
  END LOOP;
END;
```

5.- Muestre el salario medio por departamento, excluyendo a los MANAGERS y al PRESIDENT, ordenado descendientemente por salario medio

```
DECLARE
  CURSOR c IS
  SELECT AVG(sal) promedio, dname FROM dept, emp
  WHERE dept.deptno = emp.deptno AND
  job NOT IN ('PRESIDENT', 'MANAGER')
  GROUP BY dname
  ORDER BY 1 DESC;

BEGIN
  FOR vr_reg IN c LOOP
    Dbms_output.put_line(vr_reg.promedio);
    Dbms_output.put_line(vr_reg.dname);
  END LOOP;
END;
```

6.- Listar todos los empleados con el mismo trabajo y salario que otro empleado X. Tip: use un cursor con parámetros

```
DECLARE
  CURSOR c (p_nombre VARCHAR2) IS
    SELECT ename FROM emp
    WHERE job = (SELECT job FROM emp WHERE ename = p_nombre) AND
    sal = (SELECT sal FROM emp WHERE ename = p_nombre);

BEGIN
  FOR vr_reg IN c ('FORD') LOOP
    Dbms_output.put_line(vr_reg.ename);
  END LOOP;
END;
```

7.- Listar los empleados que trabajan en una locación X. Tip: use un cursor con parámetros.

```
DECLARE
  CURSOR c (p_loc VARCHAR2) IS
    SELECT ename FROM emp, dept
    WHERE emp.deptno = dept.deptno AND
    loc = p_loc;

BEGIN
  FOR vr_reg IN c ('NEW YORK') LOOP
    Dbms_output.put_line(vr_reg.ename);
  END LOOP;
END;
```

4.4 EXCEPCIONES EN PL/SQL

4.4.1 ¿Qué es una Excepción?

Los errores en la ejecución de código PL/SQL, provenientes de fallos en el diseño, errores de codificación, fallos de hardware y otros, no son totalmente identificables, de ahí que sea necesario anticipar todos los posibles errores que se puedan presentar.

Como muchos otros lenguajes de programación, PL/SQL tiene su propio método de manipulación de errores a través de una característica llamada EXCEPCIÓN.

En PL/SQL una advertencia o error es llamada EXCEPCIÓN. Las excepciones pueden ser definidas internamente o definidas por el usuario. Cuando un error ocurre una excepción se dispara, ya sea que esté o no controlada en el código. Si es controlada en el código, se ejecutarán las sentencias indicadas en la rutina de tratamiento de excepciones, sino se detendrá la ejecución.

Recordemos la sección de excepciones en un bloque PL/SQL:

```
DECLARE
...
BEGIN
...
EXCEPTION
...
END;
```

Cuando se produce un error, se genera una excepción. Cuando esto sucede, el control pasa al gestor de excepciones, que es una sección independiente del programa. Esto permite separar la gestión de errores del resto de programa, lo que hace que sea más fácil de entender la lógica de éste y, también, asegura que todos los errores serán interceptados.

4.4.2 Declaración de Excepciones

Las excepciones se declaran en la sección declarativa de un bloque, se generan en la sección ejecutable y se tratan en la sección de excepciones. Existen dos clases de excepciones: definidas por el usuario y predefinidas.

4.4.2.1 Excepciones definidas por el usuario

PL/SQL permite al usuario declarar sus propias excepciones con el fin de dar un mejor manejo a la ejecución de un bloque.

Para declarar una excepción se usa la siguiente estructura:

```
DECLARE
    Error_excp EXCEPTION;
```

El siguiente ejemplo muestra cómo usar una excepción definida por el usuario dentro de un bloque:

```
DECLARE
    fuera_de_inventario EXCEPTION;
    numero_existencias NUMBER (4);

BEGIN
    ...
    IF numero_de_existencias < 1 THEN
        RAISE fuera_de_inventario;
    END IF;

    EXCEPTION
        WHEN fuera_de_inventario THEN
            ... manejar el error
END;
```

4.4.2.2 Excepciones predefinidas

Oracle ha predefinido diversas excepciones que se corresponden con los errores clásicos de Oracle.

EXCEPCIÓN ORACLE	CODIGO DE ERROR
TIMEOUT_ON_RESOURCE	ORA-00051
INVALID_CURSOR	ORA-01001
NOT_LOGGED_ON	ORA-01012
LOGIN_DENIED	ORA-01017
NO_DATA_FOUND	ORA-01403
TOO_MANY_ROWS	ORA-01422
ZERO_DIVIDE	ORA-01476
INVALID_NUMBER	ORA-01722
STORAGE_ERROR	ORA-06500
PROGRAM_ERROR	ORA-06501
VALUE_ERROR	ORA-06502
CURSOR_ALREADY_OPEN	ORA-06511
ACCESS_INTO_NULL	ORA-06530
COLLECTION_IS_NULL	ORA-06531
SUBSCRIPT_OUTSIDE_LIMIT	ORA-06532
SUBSCRIPT_BEYOND_COUNT	ORA-06533

A continuación se detalla la explicación de cada una de las excepciones predefinidas:

ACCESS_INT0_NULL

Su programa trata de asignar valores a los atributos de un objeto no inicializado.

COLLECTION_IS_NULL

Su programa trata de aplicar métodos de colección a una colección no inicializada

CURSOR_ALREADY_OPEN

Su programa trata de abrir un cursor que ya está abierto.

DUP_VAL_ON_INDEX

Su programa trata de almacenar valores duplicados en una columna de base de datos que debe ser única.

INVALID_CURSOR

Su programa trata de hacer una operación inválida sobre un cursor.

INVALID_NUMBER

Una conversión de carácter a número ha fallado en una sentencia.

LOGIN_DENIED

Su programa trató de conectarse a Oracle con un nombre de usuario o clave inválidos.

NO_DATA_FOUND

Una sentencia SELECT INTO no retornó filas o su programa referencia un elemento no existente en una colección.

NOT_LOGGED_ON

Su programa envía una llamado de BD sin estar conectado a la misma.

PROGRAM_ERROR

PL/SQL tiene un problema interno.

STORAGE_ERROR

PL/SQL se está ejecutando fuera de memoria o la memoria está corrupta.

SUBSCRIPT_BEYOND_COUNT

Su programa referencia un elemento de una colección usando un índice mayor que el número de elementos de la misma.

SUBSCRIPT_OUTSIDE_LIMIT

Su programa referencia una colección usando un índice fuera del rango legal (INTEGER).

TIMEOUT_ON_RESOURCE

Ocurrió un time-out mientras se esperaba por un recurso.

TOO_MANY_ROWS

Una sentencia SELECT INTO retorna más de una fila.

VALUE_ERROR

Un error de conversión aritmética ocurrió. En procedimientos se retorna esta misma excepción, pero en sentencias SQL se retorna INVALID_NUMBER.

ZERO_DIVIDE

Su programa trató de efectuar una división por cero.

4.4.3 Generación de EXCEPCIONES

Cuando se produce el error asociado con una excepción, dicha excepción es generada. Las excepciones definidas por el usuario se generan implícitamente cuando ocurre su error Oracle asociado. Las excepciones predefinidas pueden ser generadas también explícitamente mediante la orden RAISE, si así lo desea.

A continuación de muestra un ejemplo:

```
DECLARE
  no_multiplo_10 EXCEPTION;
  numero number(2);

BEGIN
  numero:=12;

  IF MOD(numero,10)!=0 THEN
    RAISE no_multiplo_10;
  END IF;

  DBMS_OUTPUT.PUT_LINE ('el número es múltiplo de 10');

  EXCEPTION
    WHEN no_multiplo_10 THEN
      dbms_output.put_line ('el número NO es múltiplo de 10');
END;
```

4.4.4 Tratamiento de EXCEPCIONES

Cuando se genera una excepción, el control pasa a la sección de excepciones del bloque. Esta sección está compuesta por gestores para las diferentes excepciones. Un gestor de excepciones contiene el código que se ejecutará cuando ocurra el error asociado con la excepción y ésta sea generada.

```
DECLARE
  A EXCEPTION; --Declaración de la excepción A
BEGIN
  ....
  RAISE A; --Generación de la excepción A
  ... ----- Cualquier código aquí no se ejecuta
  EXCEPTION
    WHEN A THEN --el control pasa al gestor de excepciones
      ... ----- Este código sera ejecutado
END;
```

Cada gestor de excepciones está formado por la cláusula WHEN y las órdenes que se ejecutarán cuando la excepción sea generada. La cláusula WHEN identifica la excepción correspondiente a cada gestor.

```

DECLARE
  A EXCEPTION; --Declaración de la excepción A
  B EXCEPTION; --Declaración de la excepción B
  ...
BEGIN
  ....
  RAISE A; --Generación de la excepción A
  ...
  RAISE B --Generación de la excepción B
  ...
EXCEPTION
  WHEN A THEN --el control pasa al gestor de excepciones
    ... ----- Este código sera ejecutado

  WHEN B HEN --el control pasa al gestor de excepciones
    ... ----- Este código sera ejecutado

  WHEN OTHERS THEN
    ... ----- Este código sera ejecutado
END;

```

4.4.4.1 Gestor de excepciones WHEN OTHERS

Lo observamos en el ejemplo anterior. Debe ser siempre el último gestor del bloque. Es una buena práctica de programación el definir un gestor OTHERS en el nivel superior del programa (el bloque más exterior), para asegurarse que ningún error quede sin detectar.

A menudo, es útil saber dentro de un gestor OTHERS, qué error Oracle dio lugar a la excepción. Una razón podría ser el registrar el error que tuvo lugar, más que el hecho de que tuvo lugar un error. O bien, puede desearse tomar acciones diferentes, dependiendo del error que suceda. PL/SQL proporciona dos funciones predefinidas, SQLCODE y SQLERRM.

En el manejo de errores se pueden usar las funciones SQLERRM y SQLCODE, las cuales retornan el mensaje de error y el código de error Oracle para cada advertencia o error que se presente en la ejecución del código, respectivamente.

Por ejemplo:

```

DECLARE
  err_num NUMBER;
  err_msg VARCHAR2(100);

BEGIN
  ...
EXCEPTION
  ...
  WHEN OTHERS THEN
    err_num := SQLCODE;
    err_msg := SUBSTR(SQLERRM, 1, 100);
    INSERT INTO errors VALUES (err_num, err_msg);
END;

```

4.4.4.2 Uso de Pragma EXCEPTION_INIT

Para manipular excepciones internas no especificadas en el código, se puede usar OTHERS como se vio en la sección anterior o EXCEPTION_INIT. Un pragma es una directiva del compilador procesada en tiempo de compilación.

En PL/SQL, la directiva EXCEPTION_INIT dice al compilador que asocie un nombre de excepción con un número de error de Oracle. Esto permite referirse a un error de Oracle específico con un nombre común y asociarle un conjunto de sentencias para manipularlo.

La siguiente es la sintaxis de esta directiva:

```
PRAGMA EXCEPTION_INIT(nombre_excepción, error_Oracle);
```

Donde *nombre_excepción* es el nombre de una excepción declarada con anterioridad al pragma, y *error_Oracle* es el código de error que se desea asociar con esta excepción nominada. Este pragma debe incluirse en la sección declarativa

A continuación se muestra un ejemplo:

```
DECLARE
    deadlock_detected EXCEPTION;
    PRAGMA EXCEPTION_INIT(deadlock_detected, -60);
BEGIN
    ...
EXCEPTION
    WHEN deadlock_detected THEN
        -- handle the error
    ...
END;
```

En este caso, debemos saber que el error ORA-00060 se produce cuando ocurre un deadlock (bloqueo mutuo).

4.4.4.3 Utilización de RAISE_APPLICATION_ERROR

Para utilizar la función predefinida RAISE_APPLICATION_ERROR para crear sus propios mensajes de error, que pueden ser más descriptivos que las opciones ya existentes que tiene Oracle. La sintaxis es la siguiente:

```
RAISE_APPLICATION_ERROR (nro_de_error, mensaje_error,[preservar_error]);
```

Donde *número_de_error* es un parámetro comprendido entre -20000 y -20999, *mensaje_error* es el texto asociado con este error, y *preservar_errores* es un valor booleano. El parámetro *mensaje_error* debe ser menor de 512 caracteres. El parámetro booleano *preservar_errores* es opcional. Si dicho parámetro toma el valor TRUE, el nuevo error se añade a la lista de errores que ya han sido generados (si es que existe); si toma el valor FALSE, como hace de forma predeterminada, el nuevo error reemplazará a la lista actual de errores.

A continuación se muestra una comparación:

```
DECLARE
  numero NUMBER(2);
BEGIN
  numero:=12;
  numero:=12/0;
END;
```

```
DECLARE
*
ERROR at line 1:
ORA-01476: divisor is equal to zero
ORA-06512: at line 5
```

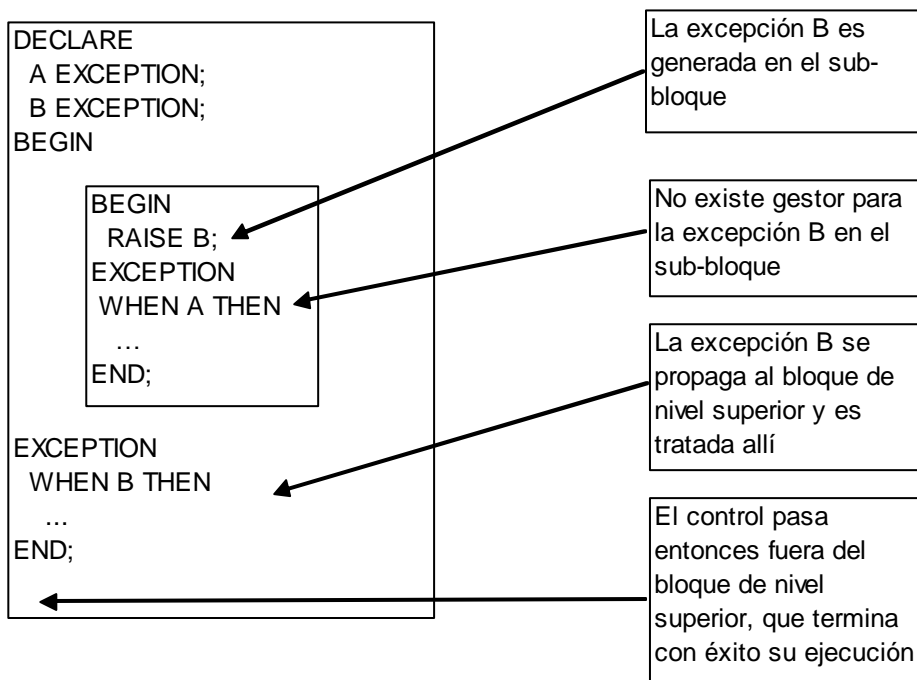
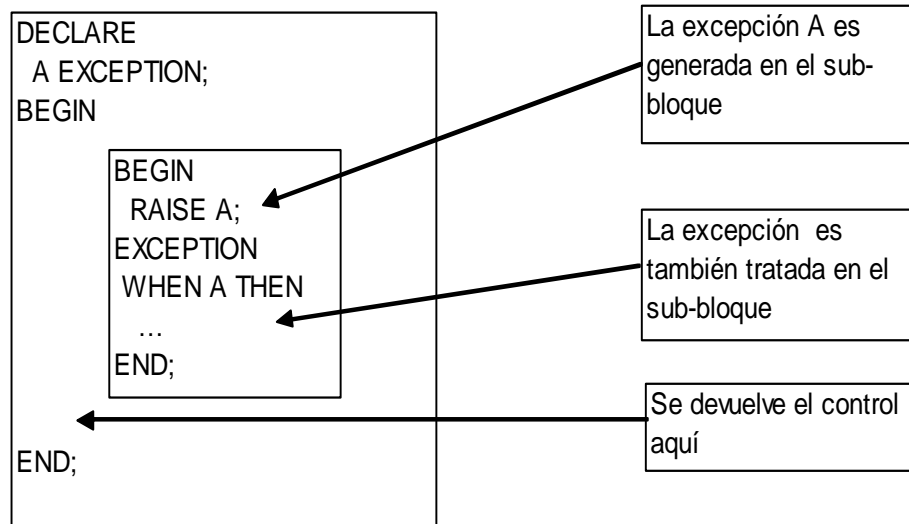
Este ejemplo es clásico error de división entre cero. En el siguiente ejemplo se observa la personalización del error:

```
DECLARE
  numero NUMBER(2);
BEGIN
  numero:=12;
  numero:=12/0;
EXCEPTION
  WHEN ZERO_DIVIDE THEN
    RAISE_APPLICATION_ERROR (-20000,'Usted hizo una división por cero');
END;
```

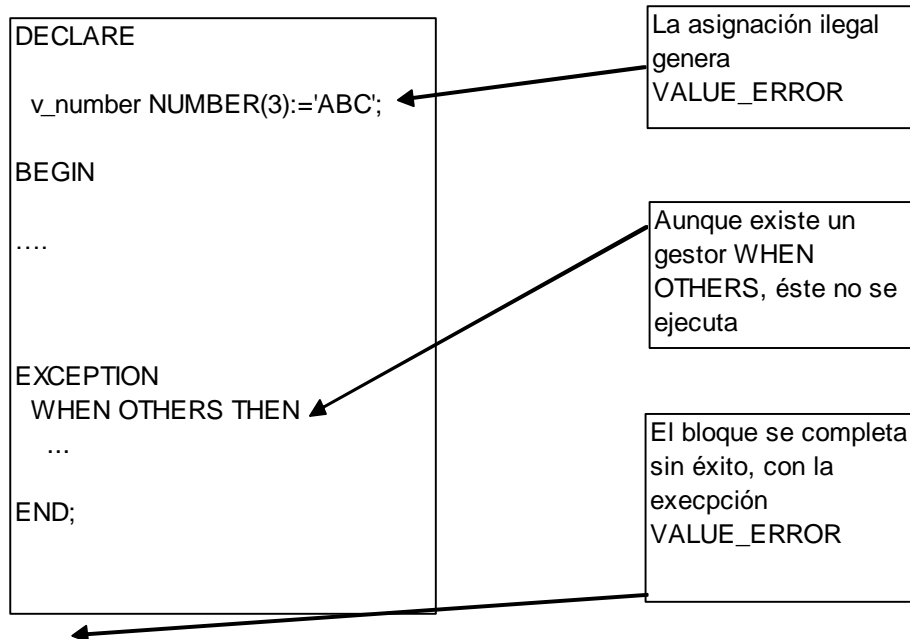
```
DECLARE
*
ERROR at line 1:
ORA-20000: usted hizo una división por cero
ORA-06512: at line 8
```

4.4.5 Propagación de EXCEPCIONES

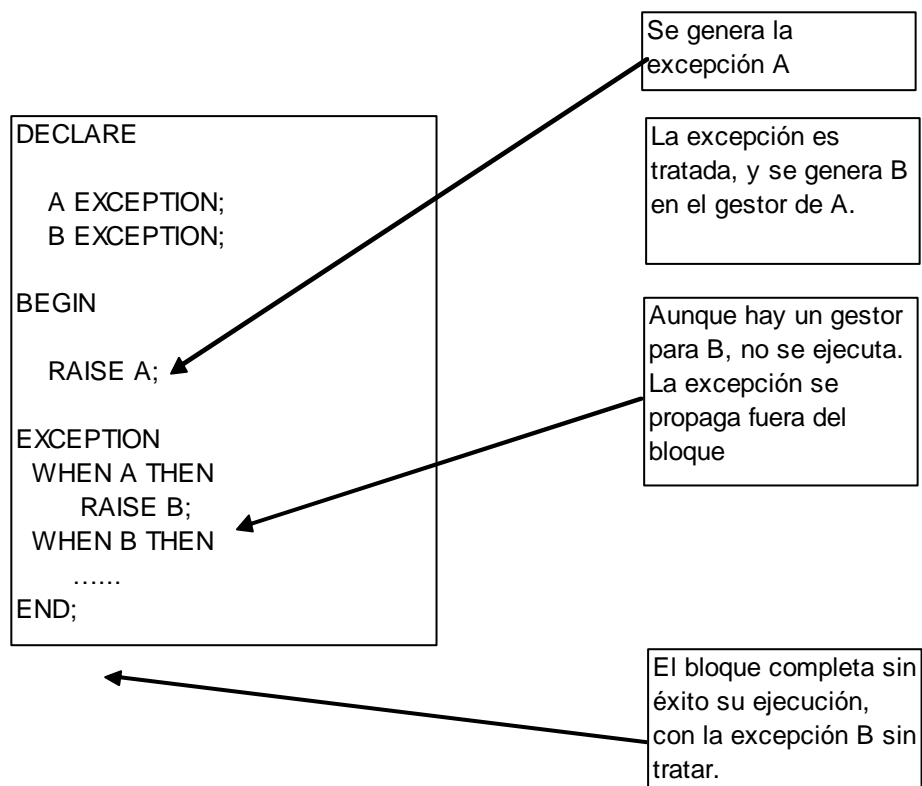
4.4.5.1 Excepciones generadas en la sección ejecutable



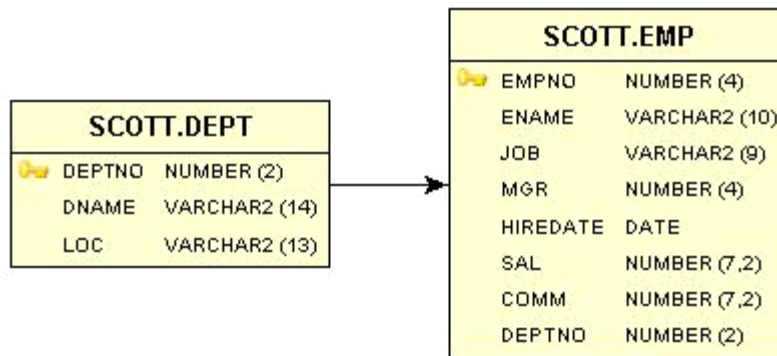
4.4.5.2 Excepciones generadas en la sección declarativa



4.4.5.3 Excepciones generadas en la sección de excepciones



Autoevaluación



- 1.- En un bloque PL/SQL, sin usar un cursor explícito, realice la búsqueda simple (select, que es un cursor implícito) de aquel empleado que tiene un sueldo de 1000. En caso no existiese, usando excepciones envíe un mensaje 'empleado no existe' mediante el gestor WHEN OTHERS.
- 2.- En la pregunta anterior, use el gestor WHEN NO_DATA_FOUND.
- 3.- Actualice la tabla EMP, de tal forma que al menos 2 empleados tengan el sueldo de 1000. Realice la búsqueda de la pregunta y capture el tipo de error que se genera en el gestor WHEN OTHERS. Use SQLCODE y SQLERRM.
- 4.- Defina la excepción VIOLACION_FK, para detectar el error ORA-02292. Para eso en la sección de ejecución hacer un DELETE FROM DEPT. Use pragma EXCEPTION_INIT.
- 5.- En la pregunta anterior, agregue la utilización del RAISE_APPLICATION_ERROR para personalizar el mensaje a 'Usted ha violado una llave foránea'. Asígnele el valor ORA-20000.

**UNIDAD DE
APRENDIZAJE****5**

PROGRAMACIÓN AVANZADA EN ORACLE

LOGRO DE LA UNIDAD DE APRENDIZAJE

Al término de la unidad, el alumno construye programas avanzados utilizando el lenguaje PL/SQL que soporten la lógica de negocio mediante la incorporación de funciones y procedimientos almacenados organizados en paquetes y a través de disparadores almacenados en la base de datos.

TEMARIO

5.1 Tema 10 : PROGRAMACIÓN DENTRO DE UNA BD ORACLE

- 5.1.1 : Construcción de funciones y procedimientos almacenados
- 5.1.2 : Construcción de paquetes almacenados
- 5.1.3 : Construcción de disparadores

ACTIVIDADES PROPUESTAS

- Implementar e invocar procedimientos almacenados
- Implementar e invocar funciones almacenadas
- Implementar e invocar subprogramas locales

5.1 Programación dentro de una base de datos Oracle

5.1.1 Construcción de funciones y procedimientos almacenados

Los procedimientos y funciones de PL/SQL se comportan, igualmente, como los procedimientos y funciones en otros lenguajes de tercera generación, compartiendo muchas de sus propiedades. A los procedimientos y funciones, también, se les denominan, en conjunto, SUBPROGRAMAS.

```
CREATE OR REPLACE PROCEDURE imprime (mensaje
varchar2)
as
BEGIN
  dbms_output.put_line (mensaje);
END;
```

Para invocar a este subprograma se hace de la siguiente forma:

```
BEGIN
  imprime ('esto es mi primer procedimiento');
END;
```

Este ejemplo muestra varios puntos importantes:

- En primer lugar, se crea el procedimiento **Imprime**, con la orden CREATE OR REPLACE PROCEDURE. Cuando se crea un procedimiento, éste se compila en primer lugar y luego se le almacena en la base de datos en forma compilada. Este código compilado puede ser posteriormente ejecutado desde otro bloque PL/SQL.
- Cuando se llama al procedimiento, pueden pasárseles parámetros. En el ejemplo anterior, se pasa al procedimiento en tiempo de ejecución un 'mensaje'. Dentro del procedimiento el parámetro **mensaje** tendrá el valor de 'esto es mi primer procedimiento'.
- Una llamada a un procedimiento es una orden PL/SQL por sí misma. La llamada no se produce como parte de una expresión. Cuando se llama a un procedimiento, el control pasa a la primera orden ejecutable dentro de él. Cuando el procedimiento termina, se devuelve el control a la orden que sigue a la llamada al procedimiento.
- Un procedimiento es un bloque PL/SQL, con una sección declarativa, una sección ejecutable y una sección de manejo de excepciones. Al igual que con los bloques anónimos, la única sección obligatoria es la sección ejecutable.

5.1.1.1 Creación de un procedimiento

La sintaxis es como sigue:

```
CREATE [OR REPLACE] PROCEDURE nombre_procedimiento
[(argumento [IN|OUT |INOUT ]tipo)
.....
[(argumento [IN|OUT |INOUT ]tipo) ] {IS |AS}
Cuerpo_procedimiento
```

donde *nombre_procedimiento* es el nombre del procedimiento que se quiere crear, *argumento* es el nombre de un parámetro del procedimiento, *tipo* es el tipo del parámetro asociado y *cuerpo_procedimiento* es un bloque PL/SQL que contiene el código del procedimiento.

El uso de OR REPLACE permite sobrescribir un procedimiento existente. Si se omite y el procedimiento ya existe se producirá un error. Los modificadores IN, OUT, IN OUT indican si el parámetro es de entrada, salida o ambos. Esto último se verá en el siguiente punto.

5.1.1.2 Eliminación de un procedimiento

Para eliminar un procedimiento se usa la cláusula DROP como se muestra:

```
DROP PROCEDURE nombre_procedimiento;
```

5.1.1.3 Creación de funciones

Una función es bastante similar a un procedimiento. Ambos aceptan argumentos, y éstos pueden ser de cualquier tipo. Ambos son formas diferentes de bloque PL/SQL, con sus secciones declarativa, ejecutable y de excepciones. Ambos pueden ser almacenados en la base de datos o ser declarados dentro de un bloque. Sin embargo, una llamada a un procedimiento es una orden PL/SQL en sí misma, mientras que una llamada a función se realiza como parte de una expresión.

La sintaxis es como sigue:

```
CREATE [OR REPLACE] FUNCTION nombre_función
[(argumento [IN|OUT |INOUT ]tipo)
.....
[(argumento [IN|OUT |INOUT ]tipo) ]
RETURN tipo_retorno [IS |AS ] Cuerpo_función
```

Donde *nombre_función* es el nombre de la función, *argumento* y *tipo* son iguales que un procedimiento, *tipo_retorno* es el tipo del valor que devuelve la función y *cuerpo_función* es un bloque PL/SQL que contiene el código de la función.

A continuación se muestra un ejemplo:

```
CREATE OR REPLACE FUNCTION formatea_mayuscula (mensaje VARCHAR2)
RETURN VARCHAR2 is
BEGIN
    RETURN UPPER(mensaje);
END;
```

Para invocar, se podría hacer lo siguiente:

```
DECLARE
    men_mayuscula varchar2(100);
BEGIN
    men_mayuscula:=formatea_mayuscula('mi primera función');
    dbms_output.put_line (men_mayuscula);
END;
```

A. La orden RETURN

Dentro del cuerpo de la función, la orden RETURN se emplea para devolver el control y un valor al entorno que realizó la llamada. La sintaxis es como sigue:

```
RETURN expresión;
```

Donde *expresión* es el valor que la función devuelve, el cual se convierte al tipo especificado en la cláusula RETURN de la definición de la función, si es que no es ya de dicho tipo. Cuando se ejecuta la orden RETURN, se devuelve el control, inmediatamente, al entorno que hizo la invocación.

Puede haber más de una orden RETURN en una función, aunque sólo se ejecutará una de ellas. Es un error que una función termine sin que se ejecute una orden RETURN. Por ejemplo, esta función es creada satisfactoriamente, a pesar que no 'retorna' nada:

```
CREATE OR REPLACE FUNCTION formatea_mayuscula (mensaje VARCHAR2)
RETURN VARCHAR2 is
BEGIN
    dbms_output.put_line (UPPER(mensaje));
END;
```

Al ejecutarse:

```
DECLARE
    men_mayuscula varchar2(100);
BEGIN
    men_mayuscula:=formatea_mayuscula('mi primera función');
    dbms_output.put_line (men_mayuscula);
END;
```

Se muestra el error:

```
DECLARE
*
ERROR at line 1:
ORA-06503: PL/SQL: Function returned without value
ORA-06512: at "HR.FORMATEA_MAYUSCULA", line 4
ORA-06512: at line 4
```

A continuación se muestra un ejemplo donde una función puede tener varios RETURN, pero sólo se ejecutará una:

```
CREATE OR REPLACE FUNCTION es_par
(numero NUMBER) RETURN VARCHAR2 is
BEGIN
  IF MOD(numero,2) = 0 then
    RETURN 'ES PAR';
  ELSE
    RETURN 'ES IMPAR';
  END IF;
END;
```

Para una correcta utilización de las funciones deben tenerse en cuenta estas sencillas reglas:

- Cuando se invoca desde una sentencia SELECT la función no puede modificar las tablas de la base de datos.
- Cuando se invoca desde una sentencia DML la función no puede consultar o modificar las tablas consideradas en la misma sentencia.
- Cuando se invoca desde una sentencia DML la función no puede ejecutar operaciones de control de transacciones (como COMMIT), sentencias de control de sesión (ALTER SESSION o SET ROLE) ni sentencias de control del sistema (como ALTER SYSTEM). Tampoco puede ejecutar sentencias DDL (Data Definition Language como CREATE) porque éstas van seguidas de un COMMIT implícito.

5.1.1.4 Eliminación de una función

Para eliminar una función también se usa la cláusula DROP como se muestra a continuación:

```
DROP FUNCTION nombre_procedimiento;
```

A. Modos de paso de parámetros

Los modos de paso de parámetros definen el comportamiento de los parámetros formales. Existen tres modos para el paso de parámetros: IN, OUT e IN OUT.

Sin embargo, debe evitarse el uso de parámetros OUT e IN OUT con funciones, dado que su propósito no es retornar múltiples valores. La siguiente tabla muestra el uso de los modos de paso de parámetros.

IN	OUT	IN OUT
Pasa un valor al subprograma	Retorna un valor a quien invoca el subprograma	Pasa un valor al subprograma y retorna el valor actualizado
Puede ser una constante o variable	Debe ser una variable	Debe ser una variable

5.1.1.5 SUBPROGRAMAS LOCALES.

El siguiente es un ejemplo de un subprograma local declarado en la sección declarativa de un bloque PL/SQL:

```

DECLARE
  CURSOR c_Emps IS
    SELECT ename, job FROM emp;
  v_salida VARCHAR2(50);

  /* Función que devuelve el nombre y job separados por espacio */
  FUNCTION Formateador (p_ename IN VARCHAR2, p_job IN VARCHAR2)
    RETURN p_ename||' '||p_job;
  END Formateador;

-- Inicio del bloque principal
BEGIN
  FOR v_emp_record IN c_Emps LOOP
    v_salida := Formateador(v_emp_record.ename, v_emp_record.ename.job);
    DBMS_OUTPUT.PUT_LINE(v_salida);
  END LOOP;
END;
```

Nota: Los programas locales deben ser declarados al final de la sección declarativa. Si situamos Formateador antes de c_Emps, obtendríamos un error de compilación.

5.1.1.6 SUBPROGRAMAS Y EL DICCIONARIO DE DATOS.

El código de un procedimiento o función se puede verificar en la vista ALL_SOURCE.

Por ejemplo, si se desea ver el código fuente de la función ES_PAR (creada anteriormente), se hace lo siguiente:

```
SQL> select text from all_source where name='ES_PAR';
TEXT
-----
FUNCTION es_par (numero NUMBER) RETURN VARCHAR2 is
BEGIN
  IF MOD(numero,2) = 0 then
    RETURN 'ES PAR';
  ELSE
    RETURN 'ES IMPAR';
  END IF;
END;

9 filas seleccionadas.
```

La vista ALL_SOURCE tiene 5 campos:

CAMPOS	DESCRIPCIÓN
OWNER	Propietario del procedimiento o función.
NAME	Nombre del procedimiento o función.
TYPE	Los diferentes tipos de subprogramas. Realmente, aparte de funciones y procedimientos, se puede observar el código de los diferentes tipos: PACKAGE, PACKAGE BODY, PROCEDURE, TRIGGER, TYPE, TYPE BODY
LINE	Número de la línea del código
TEXT	Contenido de código que está en la presente línea.

Autoevaluación

1.- Cree una función para convertir una cadena a número teniendo en cuenta que la entrada puede ser cualquier cadena.

```
FUNCTION mi_to_number (p_num IN VARCHAR2)
  RETURN NUMBER IS
  V_num NUMBER;
BEGIN
  V_num := TO_NUMBER(p_num);
  RETURN V_num;
EXCEPTION
  WHEN OTHERS THEN
  RETURN -1;
END;
```

2.- Obtener información de un empleado dado el ENAME o el JOB.

```
PROCEDURE empleado (p_nom IN VARCHAR2, p_job IN VARCHAR2) IS
  TYPE r_cursor IS REF CURSOR RETURN emp%ROWTYPE;
  V_cursor r_cursor;
  V_registro v_cursor%ROWTYPE;
BEGIN
  IF p_nom IS NOT NULL THEN
    OPEN v_cursor FOR SELECT * FROM emp WHERE ENAME = p_nom;
  ELSIF p_job IS NOT NULL THEN
    OPEN v_cursor FOR SELECT * FROM emp WHERE JOB = p_job;
  END IF;
  LOOP
    FETCH v_cursor INTO v_registro;
    EXIT WHEN v_cursor%NOTFOUND;
    Dbms_output.put_line(v_registro.ename);
    Dbms_output.put_line(v_registro.sal);
    Dbms_output.put_line(v_registro.job);
  END LOOP;
  IF v_cursor%ISOPEN THEN
    Close v_cursor;
  END IF;
END;
```

3.- Crear un procedimiento que permita ingresar un empleado. Manejar explícitamente el error si ya existe el empleado en la tabla.

```
PROCEDURE ins_empleado (p_cod IN NUMBER, p_nom IN VARCHAR2,
                        P_job IN VARCHAR2, p_mgr IN NUMBER,
                        P_fecha IN VARCHAR2, p_sal IN NUMBER,
                        P_comm IN NUMBER, p_dept IN NUMBER) IS
BEGIN
  INSERT INTO EMP (empno, ename, job, mgr, hiredate, sal, comm, deptno)
  VALUES (p_cod, p_nom, p_job, p_mgr, p_fecha, p_sal, p_comm, p_dept);
  COMMIT;
  Dbms_output.put_line('Empleado creado con éxito');

EXCEPTION
  WHEN DUP_VAL_ON_INDEX THEN
    Dbms_output.put_line('Ya existe un empleado con ese código');
  WHEN OTHERS THEN
    Dbms_output.put_line('Ha ocurrido un error: ' ||
    SUBSTR(SQLERRM, 1,110));
END;
```

4.- Actualizar el sueldo en X valor a los empleados que ganan más de Y valor.

```
PROCEDURE aumento (p_aumento IN NUMBER, p_base IN NUMBER)
IS
BEGIN
  UPDATE EMP SET sal = sal + p_aumento WHERE sal > p_base;
  COMMIT;

EXCEPTION
  WHEN OTHERS THEN
    Dbms_output.put_line('Ha ocurrido un error: ' || SUBSTR(SQLERRM, 1,110));
END;
```

5.- Dar una bonificación de X valor a los empleados que ganan menos de Y valor.

```
PROCEDURE bonificación (p_bono IN NUMBER, p_base IN NUMBER) IS
CURSOR c_bon IS
  SELECT ename, job, sal FROM EMP WHERE sal < p_base;
  v_bon c_bon%ROWTYPE;

BEGIN
  OPEN c_bon;
  LOOP
    FETCH c_bon INTO v_bon;
    EXIT WHEN c_bon%NOTFOUND;
    BEGIN
      INSERT INTO bonus VALUES (v_bon.ename, v_bon.job, v_bon.sal, v_bon.sal/2);
    EXCEPTION
      WHEN OTHERS THEN
        Dbms_output.put_line('Ha ocurrido un error: ' || SUBSTR(SQLERRM, 1, 110));
    END;
  END LOOP;
  CLOSE c_bon;
END;
```

6.- Obtener un reporte de nómina teniendo en cuenta las bonificaciones.

```
PROCEDURE nomina IS
  CURSOR c_emp IS
    SELECT empno, ename, job, sal, nvl(comm, 0) comm FROM EMP;
  v_bono number := 0;
BEGIN
  FOR v_emp IN c_emp LOOP
    BEGIN
      SELECT comm. INTO v_bono FROM BONUS
        WHERE ename = v_emp.ename;
    EXCEPTION
      WHEN NO_DATA_FOUND THEN
        V_bono := 0;
    END;
    Dbms_output.put_line(v_emp.empno || CHR(9) || v_emp.ename || CHR(9) ||
      v_emp.job || CHR(9) || to_char(v_emp.sal + v_emp.comm + v_bono,
        '$99,999.99'));
  END LOOP;
END;
```

7.- Buscar todos los empleados que tienen un salario + comisión superior a X y asignarles como nuevo salario esta suma. Sólo para los que tienen comisión.

```
PROCEDURE ACTUALIZAR(p_base IN NUMBER) IS
  CURSOR C1 IS
    SELECT empno, ename, sal, comm FROM EMP WHERE sal + comm > p_base;
  REGISTRO C1%ROWTYPE;
BEGIN
  OPEN C1;
  LOOP
    FETCH C1 INTO REGISTRO;
    EXIT WHEN C1%NOTFOUND;
    UPDATE EMP SET sal = sal + comm
      WHERE empno = C1.empno;
  END LOOP;
  CLOSE C1;
END ACTUALIZAR;
```

5.1.2 Construcción de Paquetes

5.1.2.1 Definición de Paquete

En el mundo de Oracle, Paquete se conoce como PACKAGE. Es una estructura PL/SQL que permite almacenar, juntos, una serie de objetos relacionados. Un paquete tiene dos partes diferenciadas, la especificación y el cuerpo del paquete. Cada una de ellas se almacena por separado en el diccionario de datos. A diferencia de los procedimientos y funciones, que pueden estar contenidos localmente en un bloque o ser almacenados en la base de datos, un paquete no puede ser local; sólo puede almacenarse. También presentan una serie de ventajas en lo que se refiere al rendimiento.

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 en PL/SQL.

5.1.2.2 ESPECIFICACIÓN DE UN PAQUETE

La especificación (o cabecera) del paquete contiene información acerca del contenido del paquete. Sin embargo, no contiene el código de los procedimientos.

Sintaxis:

```
CREATE [OR REPLACE] PACKAGE Nombre_paquete {IS|AS}
    Especificación_procedimiento |
    Especificacion_funcion |
    Especificacion_variable |
    Definicion_tipo |
    Declaración_excepcion |
    Declaración_cursor
END[nombre_paquete];
```

Donde:

- *Nombre_paquete*, es el nombre del paquete.
- Los *elementos* del paquete (especificaciones de procedimientos y funciones, variables, etc) son los mismos que los que se podrían encontrar en la sección declarativa de un bloque anónimo.

Las reglas sintácticas para una cabecera de paquete son las mismas que para una sección declarativa, excepto en lo que se refiere a la declaración de procedimientos y funciones. Estas reglas son las siguientes:

- Los elementos del paquete pueden aparecer en cualquier orden. Sin embargo, al igual que sucede en una sección declarativa, un objeto debe ser declarado antes de poderlo referenciar. Sin un cursor contiene una variable como parte de la cláusula WHERE, por ejemplo, la variable debe ser declarada antes que el cursor.

- No es necesario que estén presentes todos los tipos de elementos. Un paquete puede contener sólo especificaciones de procedimientos y funciones, por ejemplo, sin declarar ninguna excepción ni ningún tipo.
- Las declaraciones de procedimientos y funciones deben ser declaraciones formales, a diferencia de la sección declarativa de un bloque, que puede contener tanto declaraciones formales como el código de los procedimientos y funciones. El código que implementa los procedimientos y funciones del paquete se encuentra en el cuerpo de éste.

Por ejemplo:

```
CREATE OR REPLACE PACKAGE paquetecliente AS
  PROCEDURE insertarcliente(p_cliente IN cliente%ROWTYPE);
  PROCEDURE borrarcliente(p_DNI IN cliente.DNI%TYPE);
END paquetecliente;
```

5.1.2.3 Cuerpo del Paquete

El cuerpo del paquete es un objeto del diccionario de datos distinto de la cabecera. El cuerpo no puede ser compilado a menos que se haya, previamente, compilado la cabecera correspondiente. El cuerpo contiene el código de las declaraciones formales de subprogramas incluidos en la cabecera.

Por ejemplo:

```
CREATE OR REPLACE PACKAGE BODY paquetecliente AS
  PROCEDURE insertarcliente (p_cliente IN cliente%ROWTYPE) IS
  BEGIN
    INSERT INTO cliente VALUES (p_cliente.DNI,
                                p_cliente.nombre,
                                p_cliente.apellidos,
                                p_cliente.fecha,
                                p_cliente.tipodtrabajo,
                                p_cliente.sueldo);

    COMMIT
  END insertarcliente;
  PROCEDURE borrarcliente (p_DNI IN cliente%TYPE) IS
  BEGIN
    DELETE FROM cliente WHERE DNI= p_DNI;
  END borrarcliente;
END paquetecliente;
```

5.1.2.4 Llamada a funciones y procedimientos de un paquete.

Por ejemplo:

```
DECLARE
r_cliente cliente%ROWTYPE;
BEGIN
    r_cliente.DNI :=2245432;
    r_cliente.Nombre := 'Pepito';
    r_CLIENTE.Apellidos:='Grillo';
    r_cliente.fecha :=sysdate;
    r_cliente.Tipotrabajo : 'Completo';
    r_cliente.sueldo :=222222;
    paquetecliente.Insertarcliente (r_cliente);
END;
```

5.1.2.5 Ámbito para los paquetes

Cualquier objeto declarado en la cabecera de un paquete está dentro de ámbito y es visible fuera del paquete, sin más que invocar el objeto con el nombre del paquete. Por ejemplo, podemos efectuar una llamada a `ClassPackage.RemoveStudent` desde el siguiente bloque Oracle:

```
BEGIN
    ClassPackage.RemoveStudent(1006,'HIS',101);
END;
```

Donde el paquete se llama **ClassPackage** y el procedimiento se llama `RemoveStudent`. La llamada al procedimiento es igual que si fuera un procedimiento independiente. La única diferencia es que hay que incluir como prefijo el nombre del paquete.

5.1.2.6 Sobrecarga de los subprogramas de un paquete

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. Esta característica es muy útil, dado que permite aplicar la misma operación a objetos de tipos diferentes. Resumiendo, pueden diferir en el número, orden o familia de tipo de datos de los parámetros.

Por ejemplo, dentro de un paquete se puede definir:

```
PROCEDURE uno (p1 IN NUMBER, p2 IN VARCHAR2) IS
...
BEGIN
...
END;
PROCEDURE uno (p1 IN NUMBER) IS
...
BEGIN
...
END;
```

La sobrecarga puede ser una técnica muy útil cuando se pueda hacer la misma operación con argumentos de tipos diferentes. Sin embargo, la sobrecarga está sujeta a diversas restricciones:

- No se pueden sobrecargar dos subprogramas si sus parámetros sólo difieren en el nombre o en el modo. Por ejemplo, los siguientes dos procedimientos no pueden ser sobrecargados:

```
PROCEDURE Sobrecargado (p_Param IN NUMBER);
PROCEDURE Sobrecargado (p_Param OUT NUMBER);
```

- No pueden sobrecargarse dos funciones basándose sólo en su tipo de retorno. Por ejemplo, las siguientes funciones no pueden ser sobrecargadas:

```
FUNCTION Sobrecargada RETURN DATE;
FUNCTION Sobrecargada RETURN NUMBER;
```

- Finalmente, los parámetros de las funciones sobrecargadas deben diferir también en cuanto a familia de tipos, no pudiendo realizarse sobrecargas dentro de la misma familia. Por ejemplo, puesto que CHAR y VARCHAR2 pertenecen a la misma familia de tipos, no podrían sobrecargarse los siguientes procedimientos:

```
PROCEDURE SobrecargaChar (p_Param IN CHAR);
PROCEDURE SobrecargaChar (p_Param IN VARCHAR2);
```

5.1.2.7 Paquetes de ORACLE

El motor de Base de Datos Oracle contiene un conjunto variado de paquetes preestablecidos que pueden ser usados para ejecutar algunas tareas. A continuación veremos algunos de estos paquetes y su funcionalidad:

Paquete	Descripción
DBMS_OUTPUT	<p>Permite desplegar la salida de bloques PL/SQL en la consola de SQLPlus. Se usa comúnmente con propósitos de pruebas y verificación de errores. El procedimiento put_line escribe información a un buffer en la SGA. Para desplegar la información del buffer, se debe establecer, antes, la variable de ambiente SERVEROUTPUT ON en SQLPlus.</p> <p>Ejemplo:</p> <pre>CREATE PROCEDURE calc_nomina (pago IN OUT REAL) AS CURSOR c1 IS SELECT sal,comm. FROM emp; BEGIN pago := 0; FOR c1rec IN c1 LOOP c1rec.comm := NVL(c1rec.comm, 0); payroll := payroll + c1rec.sal + c1rec.comm; END LOOP; /* Mostrar debug info. */ dbms_output.put_line('pago: ' TO_CHAR(pago)); END calc_nomina;</pre> <pre>SQL> SET SERVEROUTPUT ON SQL> VARIABLE num NUMBER SQL> EXECUTE calc_nomina(:num)</pre>
UTL_FILE	Este paquete permite a los programas PL/SQL leer y escribir archivos de texto en el Sistema Operativo.
DBMS_SQL	Permite ejecutar operaciones SQL (DML o DDL) dinámicamente en tiempo de ejecución.
DBMS_APPLICATION_INFO	Permite que las sesiones informen a la vista V\$SESSION.
DBMS_METADATA	Permite extraer la definición (scripts) de los objetos de la Base de Datos.
DBMS_REPCAT	Paquete que contiene todos los procedimientos y funciones que soportan la replicación.
DBMS_JOB	Paquete que sirve para programar procesos PL/SQL para que se ejecuten en la Base de Datos.
DBMS_LOGMNR	Permite visualizar los archivos redo logs
DBMS_FLASHBACK	Paquete que maneja la funcionalidad de Flash Back Query.

Existen muchos más paquetes. Para ver el listado de los paquetes en la Base de Datos se hace la siguiente consulta en la Base de Datos:

```
SELECT object_name FROM dba_objects WHERE object_type='PACKAGE';
```

Para ver el contenido de un paquete se usa la opción DESCRIBE (DESC):

```
DESC DBMS_APPLICATION_INFO
```

Autoevaluación

1.- Construya un paquete para manipulación de empleados. El paquete debe tener las siguientes funcionalidades:

- Insertar un nuevo empleado
- Actualizar el salario de un empleado
- Listar el reporte de ventas de todos los empleados en un determinado año
- Función que muestre las ventas realizadas por un empleado en un determinado año

```
CREATE OR REPLACE PACKAGE CRUD_EMP AS
  PROCEDURE ins_empleado
    (p_nro IN VARCHAR2, p_nom IN VARCHAR2 IS, p_job IN VARCHAR2,
     p_fing IN VARCHAR2, p_sal IN NUMBER, p_deptno IN NUMBER);
END;
/
CREATE OR REPLACE PACKAGE BODY CRUD_EMP A
  PROCEDURE ins_empleado
  ....
END;
/
```

2.- Aplique la funcionalidad de 'SOBRECARGA' de un paquete. Éste debe contener un procedimiento MUESTRA_EMPLEADO que puede recibir como parámetro el dato EMPNO (código de empleado) o puede recibir el ENAME (nombre del empleado). El procedimiento debe imprimir el salario de dicho empleado.

5.1.3 Construcción de Disparadores

Los disparadores se asemejan a los procedimientos y funciones en que son bloques PL/SQL nominados con las secciones declarativa, ejecutable y de manejo de excepciones. Al igual que los paquetes, los disparadores se guardan en la Base de datos y no pueden ser locales a un bloque. Un disparador se ejecuta de manera implícita (automático), cada vez que se realiza un evento que origina el disparo. Al disparador no se le puede pasar parámetros directamente. Los clásicos eventos de un disparador son las operaciones DML: INSERT, DELETE y UPDATE.

Entre las utilidades de los disparadores se tienen las siguientes:

- El mantenimiento de restricciones de integridad complejas, que no sean posibles con las restricciones declarativas definidas en el momento de crear la tabla. En resumen, para restricciones muy complejas.
- La auditoría de la información contenida en una tabla, registrando las acciones realizadas en la misma, identificando al usuario de Base de datos quien hizo la acción.
- El aviso automático a otros módulos de PL/SQL de que hay que llevar a cabo una determinada acción, cuando se realiza un cambio en una tabla.

La sintaxis es la siguiente:

```
CREATE [OR REPLACE] TRIGGER nombre_disparador
{BEFORE|AFTER} {DELETE|INSERT|UPDATE [OF col1, col2, . . . , colN]
[OR {DELETE|INSERT|UPDATE [OF col1, col2, . . . , colN]. . .}]
ON tabla_referencia
[REFERENCING OLD AS oldname, NEW as newname]
[FOR EACH ROW [WHEN (condición)]]
cuerpo_disparador
```

Donde *nombre_disparador* es el nombre del disparador, *condición* específica cuando se activa el disparador, *tabla_referencia* es la tabla para la cual se define el disparador y *cuerpo_disparador* es el código principal del disparador. Antes se evalúa la *condición* incluida en la cláusula WHEN, si es que está presente.

Por ejemplo, tenemos las siguientes tablas de órdenes y su respectiva tabla de auditoría:

```
CREATE TABLE orders (
    order_id          number(5),
    qty               number(4),
    cost_per_item     number(6,2),
    total_cost        number(8,2));

CREATE TABLE orders_aud (
    order_id          number(5),
    qty               number(4),
    cost_per_item     number(6,2),
    total_cost        number(8,2),
    user_audit        varchar2(20));
```

Y se construye un trigger que capture qué usuario ha insertado nuevas órdenes:

```
CREATE OR REPLACE TRIGGER orders_after_insert
AFTER INSERT
ON orders
FOR EACH ROW
DECLARE
    v_usr varchar2(10);
BEGIN
    -- Se obtiene el usuario que está haciendo el insert
    SELECT user INTO v_usr FROM dual;

    -- Registra la acción en la tabla auditoría
    INSERT INTO orders_aud(order_id,qty,cost_per_item,total_cost,username)
    VALUES(:new.order_id,:new.qty,:new.cost_per_item,:new.total_cost,v_usr);
END;
```

5.1.3.1 Eliminación y deshabilitación de los disparadores.

Para eliminar, la sintaxis es:

```
DROP TRIGGER nombre_disparador;
```

Donde *nombre_disparador* es el nombre del disparador que se desea eliminar. Esta orden elimina el disparador definitivamente de la base de datos. También, puede incluir la cláusula OR REPLACE en el momento de crear un disparador.

Existe la posibilidad de deshabilitar un disparador sin necesidad de eliminarlo, a diferencia de los procedimientos y paquetes. Cuando se deshabilita un disparador, éste continúa existiendo en el diccionario de datos, pero no se disparará. La sintaxis para deshabilitar un disparador es la siguiente:

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

Donde *nombre_disparador* es el nombre del disparador. Todos los disparadores al momento de su creación (exitosa) están habilitados.

5.1.3.2 Componentes de un disparador

A. Nombres de un disparador

El espacio de nombres para los disparadores es diferente del de otros subprogramas. El espacio de nombres es el conjunto de identificadores válidos que pueden ser usados como nombres de objetos. Las tablas, procedimientos y paquetes usan el mismo espacio de nombres, por lo que es 'error' dar el mismo nombre a un procedimiento y a un paquete.

Los disparadores, sin embargo, existen en un espacio de nombres separado, por lo que un disparador puede tener el mismo nombre que una tabla o un procedimiento.

5.1.3.3 Tipos de disparadores

El suceso de disparo determina el tipo de disparador. Los disparadores pueden definirse para las operaciones INSERT, UPDATE o DELETE, y pueden dispararse antes o después de la operación. Además, se puede decir que el nivel de los disparadores puede ser por FILA o por SENTENCIA.

Categoría	Valores	Comentarios
Orden	INSERT, DELETE y UPDATE	Define qué tipo de orden DML provoca la activación del disparador
Temporización	BEFORE o INSERT	Define si el disparador se activa antes o después de que se ejecute la orden (disparador previo o posterior)
Nivel	Fila o sentencia	Los disparadores con nivel de fila (FOR EACH ROW) se activan una vez por cada fila afectada por la orden que provocó el disparo. Los disparadores con nivel de orden (STATEMENT) se activan sólo una vez, antes o después de la sentencia que activa el disparo.

5.1.3.4 Restricciones de los disparadores

Cualquier orden que sea legal en un bloque PL/SQL, es legal en el cuerpo de un disparador, con las siguientes restricciones:

- Un disparador no puede emitir ninguna orden de control de transacciones: COMMIT, ROLLBACK o SAVEPOINT. La razón es que el disparador pertenece a la transacción de la sentencia que provocó la ejecución del disparador. Si la sentencia es cancelada, el disparador también es cancelado.
- En base a la restricción anterior, se concluye, también, que ningún procedimiento o función que es llamado desde un disparador puede emitir una orden de control de transacciones.
- El cuerpo del disparador no puede contener ninguna declaración de variables LONG o LONG RAW.
- No puede modificar tablas mutantes. Tema que se verá más adelante: 2.8.

5.1.3.5 Los disparadores y el diccionario de datos

A. Vistas del Diccionario de Datos

Cuando se crea un disparador, el código fuente se puede observar mediante la vista USER_TRIGGERS.

Por ejemplo, para buscar datos del trigger UPDATE_JOB_HISTORY se hace lo siguiente:

```
SQL> select trigger_type,table_name,triggering_event from user_triggers
2 where trigger_name='UPDATE_JOB_HISTORY';
```

TRIGGER_TYPE	TABLE_NAME	TRIGGERING_EVENT
-----	-----	-----
AFTER EACH ROW	EMPLOYEES	UPDATE

Si se desea ver el código del trigger, habría que mostrar el campo TRIGGER_BODY.

5.1.3.6 Utilización de :old y :new en los disparadores con nivel de fila

Cuando un disparador es 'disparado', puede accederse a la fila que está siendo actualmente procesada utilizando, para ello, dos pseudo-registros, :old y :new.

A continuación se muestra una tabla de comparación:

Orden de disparo	:old	:new
INSERT	No definido: todos los campos toman el valor NULL.	Valores que serán insertados cuando se complete la orden.
UPDATE	Valores originales de la fila, antes de la actualización	Nuevos valores que serán escritos cuando se complete la orden.
DELETE	Valores originales, antes del borrado de la fila	No definido: todos los campos toman el valor NULL.

A continuación se muestra un ejemplo, en el cual un disparador almacena los valores de los empleados que están siendo actualizados. Para ello, guarda los valores antiguos.

```
CREATE OR REPLACE TRIGGER "HR"."UPDATE_JOB_HISTORY"
AFTER UPDATE OF "DEPARTMENT_ID", "JOB_ID"
ON "HR"."EMPLOYEES"
FOR EACH ROW
BEGIN
add_job_history(:old.employee_id,:old.hire_date,sysdate,:old.job_id,:old.department_id);
END;
```

5.1.3.7 La cláusula WHEN

La cláusula WHEN sólo es válida para disparadores con nivel de fila. Si está presente, el cuerpo del disparador sólo se ejecutará para las filas que cumplan la condición especificada en la cláusula. Esta cláusula se define de la forma:

WHEN *condición*

Donde *condición* es una expresión booleana que será evaluada para cada fila. Se puede hacer referencia a las variables :new y :old dentro de la condición, pero no se utilizan los dos puntos (:)

A continuación se muestra un ejemplo de un disparador que sólo se ejecutará para aquellos empleados que han ingresado a la empresa hace un mes (30 días).

```
CREATE OR REPLACE TRIGGER "HR"."UPDATE_JOB_HISTORY"
  AFTER UPDATE
  OF "DEPARTMENT_ID", "JOB_ID" ON "HR"."EMPLOYEES"
  FOR EACH ROW
  WHEN (old.hire_date > sysdate - 30)
BEGIN
  add_job_history(:old.employee_id,:old.hire_date,sysdate,:old.job_id,:old.department_id);
END;
```

5.1.3.8 Utilización de predicados: INSERTING, UPDATING y DELETING

Un sólo disparador se puede 'ejecutar' por una inserción, por una actualización o por una eliminación de una fila de una tabla. El problema radica en diferenciar, dentro del cuerpo del disparador, cuál fue el tipo de acción que 'ejecutó' el disparador. Para esto se utilizan estos predicados que se muestran en una tabla de comparación:

Predicado	Comportamiento
INSERTING	TRUE si la orden de disparo es INSERT; FALSE en caso contrario
UPDATING	TRUE si la orden de disparo es UPDATING; FALSE en caso contrario
DELETING	TRUE si la orden de disparo es DELETE; FALSE en caso contrario

A continuación se muestra un ejemplo:

```
CREATE OR REPLACE TRIGGER "HR"."JOB_HISTORY" AFTER
  UPDATE OR INSERT OR DELETE
  OF "DEPARTMENT_ID", "JOB_ID" ON "HR"."EMPLOYEES"
  FOR EACH ROW
BEGIN
  IF UPDATING OR DELETING THEN
    add_job_history(:old.employee_id,:old.hire_date,sysdate,:old.job_id,
    :old.department_id);
  END IF;
  IF INSERTING THEN
    add_job_history(:new.employee_id,:new.hire_date,sysdate,:new.job_id,:new.d
    epartment_id);
  END IF;
END;
```

5.1.3.9 Tablas mutantes

Una tabla mutante es una tabla que está modificándose actualmente por una orden DML. Para un disparador, ésta es la tabla sobre la que está definido.

Las órdenes SQL, en el cuerpo de un disparador, no pueden leer o modificar ninguna tabla mutante de la orden que provoca el disparo. Esto incluye a la propia tabla del disparador. A continuación se muestra un ejemplo:

```
CREATE OR REPLACE TRIGGER "HR"."MUTANDO"
  AFTER
  UPDATE OR INSERT OR DELETE
  OF "DEPARTMENT_ID", "JOB_ID" ON
  "HR"."EMPLOYEES"
  FOR EACH ROW

BEGIN
  UPDATE "HR"."EMPLOYEES" set department_id=2;
END;
```

Luego, al ejecutar una actualización aparecerá el siguiente error:

```
SQL> update "HR"."EMPLOYEES" set department_id=3;
update "HR"."EMPLOYEES" set department_id=3
      *
ERROR en línea 1:
ORA-04091: table HR.EMPLOYEES is mutating, trigger/function may
not see it
ORA-06512: at "HR.MUTANDO", line 2
ORA-04088: error during execution of trigger 'HR.MUTANDO'
```

5.1.3.10 DISPARADORES DE SISTEMA

Lo clásico en todo Manejador de Base de Datos, son los disparadores anteriormente desarrollado, es decir, aquéllos que se ‘disparan’ ante una operación DML.

En las últimas versiones, han aparecido otros tipos de disparadores. Éstos son los disparadores de Sistema, los cuales se clasifican en dos: Disparadores ante una operación DDL y Disparadores ante un evento de Base de Datos.

3.1.- Disparadores ante una operación DDL

General	
Vistas relacionadas a los disparadores DDL	trigger\$ user_triggers all_triggers dba_triggers
Privilegios de sistema necesarios	create trigger create any trigger administer database trigger -- required for ON DATABASE alter any trigger drop any trigger

Disparadores DDL y las operaciones que lo 'disparan'	BEFORE / AFTER ALTER BEFORE / AFTER CREATE BEFORE / AFTER DROP BEFORE / AFTER RENAME BEFORE / AFTER ANALYZE BEFORE / AFTER ASSOCIATE STATISTICS BEFORE / AFTER DISASSOCIATE STATISTICS BEFORE / AFTER AUDIT BEFORE / AFTER NOAUDIT BEFORE / AFTER COMMENT BEFORE / AFTER DDL BEFORE / AFTER GRANT BEFORE / AFTER REVOKE BEFORE / AFTER TRUNCATE AFTER SUSPEND
Un ejemplo	
Usuario Demo	CREATE USER nwoug IDENTIFIED BY nwoug DEFAULT TABLESPACE users TEMPORARY tablespace temp QUOTA 0 ON system QUOTA 10M ON data_sml; GRANT create session TO nwoug; GRANT create procedure TO nwoug; GRANT create sequence TO nwoug; GRANT create table TO nwoug; GRANT create trigger TO nwoug; GRANT create view TO nwoug; GRANT select ON v_\$open_cursor TO nwoug; CONN nwoug/nwoug
Tabla donde se capturará la operación DDL	CREATE TABLE ddl_log (operation VARCHAR2(30), obj_owner VARCHAR2(30), object_name VARCHAR2(30), sql_text VARCHAR2(60), attempt_by VARCHAR2(30), attempt_dt DATE) TABLESPACE <tablespace_name>;

Disparadores DDL

Disparador que captura Actividad de creación en un esquema	CREATE OR REPLACE TRIGGER <trigger_name> <BEFORE AFTER> <triggering_action> ON <SCHEMA DATABASE> DECLARE -- variable declarations BEGIN -- trigger code EXCEPTION -- exception handler END <trigger_name>; /
--	--

```
CREATE OR REPLACE TRIGGER bcs_trigger
BEFORE CREATE
ON SCHEMA
DECLARE
oper ddl_log.operation%TYPE;
BEGIN
INSERT INTO ddl_log
SELECT ora_sysevent, ora_dict_obj_owner,
ora_dict_obj_name, NULL, USER, SYSDATE
FROM dual;
END bcs_trigger;
/
col object_name format a20
SELECT object_name, object_type FROM user_objects;

col triggering_event format a20
SELECT trigger_name, trigger_type,
triggering_event, base_object_type
FROM user_triggers;

SELECT * FROM ddl_log;

CREATE SEQUENCE s_test;

CREATE TABLE t_test (
testcol VARCHAR2(20));

CREATE OR REPLACE VIEW v_test AS
SELECT * FROM t_test;

set linesize 150

SELECT operation, obj_owner, object_name
FROM ddl_log;

TRUNCATE TABLE ddl_log;

conn system/manager

CREATE TABLE nwoug.xyz (
testcol VARCHAR2(20));

conn nwoug/nwoug

SELECT operation, obj_owner, object_name
FROM ddl_log;

DROP TRIGGER bcs_trigger;
DROP SEQUENCE s_test;
DROP TABLE t_test;
DROP VIEW v_test;
DROP TABLE xyz;
TRUNCATE TABLE ddl_log;
```

<p>Disparador que captura un evento de creación en una Base de Datos</p>	<pre> CREATE OR REPLACE TRIGGER <trigger_name> <BEFORE AFTER> <triggering_action> ON <SCHEMA DATABASE> DECLARE -- variable declarations BEGIN -- trigger code EXCEPTION -- exception handler END <trigger_name>; / CONN system/manager GRANT administer database trigger TO nwoug; CONN nwoug/nwoug CREATE OR REPLACE TRIGGER bcd_trigger BEFORE CREATE ON DATABASE DECLARE oper ddl_log.operation%TYPE; BEGIN INSERT INTO ddl_log SELECT ora_sysevent, ora_dict_obj_owner, ora_dict_obj_name, NULL, USER, SYSDATE FROM dual; END bcd_trigger; / col object_name format a20 SELECT object_name, object_type FROM user_objects; col triggering_event format a20 SELECT trigger_name, trigger_type, triggering_event, base_object_type FROM user_triggers; SELECT * FROM ddl_log; CREATE SEQUENCE s_test; CREATE TABLE t_test (testcol VARCHAR2(20)); CREATE OR REPLACE VIEW v_test AS SELECT * FROM t_test; set linesize 150 SELECT operation, obj_owner, object_name FROM ddl_log; </pre>
--	---

	<pre> TRUNCATE TABLE ddl_log; conn system/manager CREATE TABLE nwoug.xyz (testcol VARCHAR2(20)); conn nwoug/nwoug SELECT operation, obj_owner, object_name FROM ddl_log; DROP TRIGGER bcd_trigger; TRUNCATE TABLE ddl_log; </pre>
	<pre> CREATE OR REPLACE TRIGGER <trigger_name> <BEFORE AFTER> <triggering_action> OR <trigger_action> ON SCHEMA DECLARE -- variable declarations BEGIN -- trigger code EXCEPTION -- exception handlers END <trigger_name>; / </pre>
Disparador que captura múltiples operaciones DDL	<pre> CREATE OR REPLACE TRIGGER ddl_trigger BEFORE CREATE OR ALTER OR DROP ON SCHEMA DECLARE oper ddl_log.operation%TYPE; BEGIN SELECT ora_sysevent INTO oper FROM dual; IF oper IN ('CREATE', 'DROP') THEN INSERT INTO ddl_log SELECT ora_sysevent, ora_dict_obj_owner, ora_dict_obj_name, NULL, USER, SYSDATE FROM dual; ELSIF oper = 'ALTER' THEN INSERT INTO ddl_log SELECT ora_sysevent, ora_dict_obj_owner, ora_dict_obj_name, sql_text, USER, SYSDATE FROM v\$open_cursor WHERE UPPER(sql_text) LIKE 'ALTER%'; END IF; END ddl_trigger; / col operation format a20 col obj_owner format a10 SELECT * FROM ddl_log; </pre>

	<pre> CREATE TABLE new_table (charcol VARCHAR(20)); SELECT * FROM ddl_log; ALTER TABLE new_table ADD (numbcol NUMBER(10)); SELECT * FROM ddl_log; DROP TABLE new_table; SELECT * FROM ddl_log; TRUNCATE TABLE ddl_log; </pre>
<p>Disparador que captura y evita los intentos de hacer un DROP o TRUNCATE</p> <p>Requiere que sea creado el procedimiento que está a continuación (log_proc)</p>	<pre> CREATE OR REPLACE TRIGGER save_our_db BEFORE DROP OR TRUNCATE ON SCHEMA DECLARE oper ddl_log.operation%TYPE; BEGIN SELECT ora_sysevent INTO oper FROM dual; log_proc(ora_sysevent, ora_dict_obj_owner, ora_dict_obj_name); IF oper = 'DROP' THEN RAISE_APPLICATION_ERROR(-20998, 'Attempt To Drop In Production Has Been Logged'); ELSIF oper = 'TRUNCATE' THEN RAISE_APPLICATION_ERROR(-20999, 'Attempt To Truncate A Production Table Has Been Logged'); END IF; END save_our_db; / -- be sure log_proc is built first (see below) SELECT * FROM ddl_log; DROP TRIGGER ddl_trigger; SELECT * FROM ddl_log; ALTER TRIGGER save_our_db DISABLE; DROP TRIGGER ddl_trigger; SELECT * FROM ddl_log; TRUNCATE TABLE ddl_log; SELECT * FROM ddl_log; </pre>

	<pre> ALTER TRIGGER save_our_db ENABLE; DROP VIEW v_test; SELECT * FROM ddl_log; SELECT object_name FROM user_objects; DROP SEQUENCE v_test; SELECT * FROM ddl_log; SELECT object_name FROM user_objects; DROP TRIGGER save_our_db; TRUNCATE TABLE ddl_log; </pre>
Procedimiento de Log	<pre> CREATE OR REPLACE PROCEDURE log_proc (ose ddl_log.operation%TYPE, odoo ddl_log.obj_owner%TYPE, odon ddl_log.object_name%TYPE) IS PRAGMA AUTONOMOUS_TRANSACTION; BEGIN INSERT INTO ddl_log SELECT ora_sysevent, ora_dict_obj_owner, ora_dict_obj_name, 'Not Allowed', USER, SYSDATE FROM dual; COMMIT; END log_proc; / </pre>
Disparador que previene la creación de objetos cuyos nombres empiezan con 'X'	<pre> CREATE OR REPLACE TRIGGER no_xtabs BEFORE CREATE ON SCHEMA DECLARE x user_tables.table_name%TYPE; BEGIN SELECT ora_dict_obj_name INTO x FROM dual; IF SUBSTR(x, 1, 1) = 'X' THEN RAISE_APPLICATION_ERROR(-20099, 'Table Names Can Not Start With The Letter X'); END IF; END no_xtabs; / CREATE TABLE ztest (testcol VARCHAR2(20)); CREATE TABLE xtest (testcol VARCHAR2(20)); </pre>

Ejemplo de una aplicación que identifica y registra TODAS las operaciones DDL	CONN / AS SYSDBA CREATE TABLESPACE logging DATAFILE 'c:\temp\logtsp01.dbf' SIZE 100M EXTENT MANAGEMENT LOCAL UNIFORM SIZE 64K; CREATE USER loguser IDENTIFIED BY loguser DEFAULT TABLESPACE logging TEMPORARY TABLESPACE temp QUOTA 0 ON SYSTEM QUOTA UNLIMITED ON logging; CREATE TABLE loguser.ddl_log (user_name VARCHAR2(30), ddl_date DATE, ddl_type VARCHAR2(30), object_type VARCHAR2(18), owner VARCHAR2(30), object_name VARCHAR2(128)) TABLESPACE logging; CREATE OR REPLACE TRIGGER ddl_trig AFTER DDL ON DATABASE BEGIN INSERT INTO loguser.ddl_log (user_name, ddl_date, ddl_type, object_type, owner, object_name) VALUES (ora_login_user, SYSDATE, ora_sysevent, ora_dict_obj_type, ora_dict_obj_owner, ora_dict_obj_name); END ddl_trig; /

Eventos de Sistema – Funciones de eventos			
Atributo	Tipo	Descripción	Ejemplo
ora_client_ip_address	VARCHAR2	Dirección Client IP	IF (ora_sysevent='LOGON') THEN addr := ora_client_ip_address; END IF;
ora_database_name	VARCHAR2 (50)	Nombre de Base de datos	Similar al anterior
ora_dict_obj_name	VARCHAR2 (30)	Objeto en el cual el DDL ocurre	Similar al anterior
ora_dict_obj_owner	VARCHAR2 (30)	Dueño del objeto	Similar al anterior
ora_dict_obj_type	VARCHAR2 (20)	Tipo de Objeto	INSERT INTO event_table (object_type) VALUES (ora_dict_obj_type);
ora_is_alter_column (column_name IN VARCHAR2)	BOOLEAN	Devuelve TRUE si la columna especificada es alterada	IF (ora_sysevent = 'ALTER') AND ora_dict_obj_type='TABLE' THEN alter_column := ora_is_alter_column('FOO'); END IF;

ora_is_drop_column (column_name IN VARCHAR2)	BOOLEAN	Devuelve TRUE si la columna especificada es eliminada	Similar al anterior
ora_login_user	VARCHAR2 (30)	Login user name	SELECT ora_login_user FROM dual;
ora_sysevent	VARCHAR2 (20)	System event firing the trigger	INSERT INTO event_table VALUES (ora_sysevent);

5.1.3.11 Disparadores ante un evento de Base de Datos

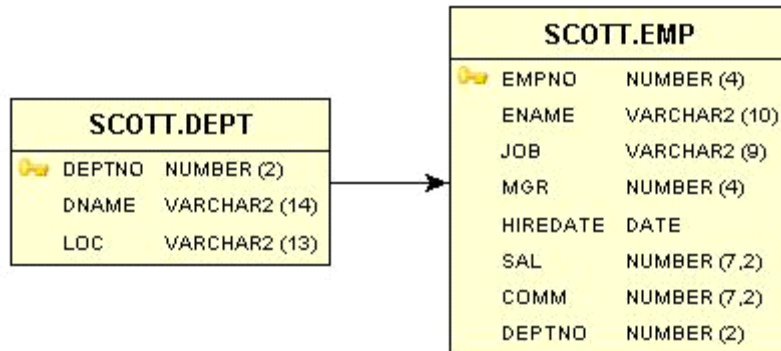
General	
Vistas del Diccionario de Datos	trigger\$ user_triggers all_triggers dba_triggers
Privilegios necesarios	create trigger create any trigger administer database trigger alter any trigger drop any trigger
Disparadores de eventos de Base de Datos	AFTER LOGON AFTER STARTUP BEFORE LOGOFF BEFORE SHUTDOWN AFTER SERVERERROR
Disparador después del LOGON	
Tabla de ejemplo para capturar los eventos	CREATE TABLE log_logons (sqltext VARCHAR2(25) NOT NULL) TABLESPACE <tablespace_name>;
Procedimiento ejecutado por el disparador	CREATE OR REPLACE PROCEDURE logproc IS BEGIN INSERT INTO log_logons (sqltext) VALUES (user ' - ' TO_CHAR(SYSDATE)); END logproc; /
Ejemplo de disparador de evento de Base de Datos. La tabla de ejemplo y el procedimiento logproc deben ser creados antes que el disparador sea compilado	CREATE OR REPLACE TRIGGER <trigger_name> <trigger_type> ON DATABASE CALL <procedure_name> /
	CREATE OR REPLACE TRIGGER logintrig AFTER LOGON ON DATABASE CALL logproc /
Ejemplo para capturar los LOGON exitosos y no	CREATE TABLE connection_audit (login_date DATE,

exitosos	<pre>username VARCHAR2(30)); -- trigger to trap successful logons CREATE OR REPLACE TRIGGER logon_audit AFTER LOGON ON DATABASE BEGIN INSERT INTO connection_audit (login_date, username) VALUES (SYSDATE, USER); END logon_audit; / -- trigger to trap unsuccessful logons CREATE OR REPLACE TRIGGER logon_failures AFTER SERVERERROR ON DATABASE BEGIN IF (IS_SERVERERROR(1017)) THEN INSERT INTO connection_audit (login_date, username) VALUES (SYSDATE, 'ORA-1017'); END IF; END logon_failures; /</pre>
----------	---

Autoevaluación

1.- Explique los diferentes usos de los disparadores DML, DDL y de eventos de Base de Datos.

A continuación, se propone ejercicios prácticos basado en el modelo mostrado.



2.- Cree un trigger que valide que cualquier actualización del salario de un empleado, no sobrepase los 10,000 nuevos soles. En caso, se esté actualizando el salario de un empleado con un monto mayor que 10,000 soles, el trigger deberá asignar automáticamente a 10,000 soles (como tope) al salario del empleado involucrado.

3.- Cree un trigger que capture información de los empleados a los cuales se le está modificando el salario. Solamente para aquéllos que son del área de contabilidad (ACCOUNTING).

4.- Simule una eliminación en cascada, de tal forma que cuando se elimine un registro de la tabla DEPT, automáticamente, se eliminen a los empleados que pertenecen a ese departamento.

5.- Genere una estadística de cuántos usuarios se han conectado a la Base de Datos (y los que han intentado pero no han podido hacerlo). Tip: use disparador de eventos de Base de Datos (LOGON) en un día.

6.- Genere una auditoría que capture información de los usuarios que han creado objetos en la base de datos durante el mes actual. Tip: Use disparador DDL.