ADMINISTRACION SISTEMAS GESTORES DE BASES DE DATOS (ASGBD)

Apuntes

y

Ejercicios

Administración de sistemas gestores de Bases de datos.

Introducción al lenguaje PL/SQL (Cap. 8)

Oracle incorpora un gestor PL/SQL (basado en lenguaje ADA), incorpora todas las características de lenguajes de tercera generación:

- Manejo de variables.
- Estructura modular (procedimientos y funciones).
- Estructuras de control (bifurcaciones, bucles y demás estructuras).
- Control de excepciones.
- Almacenamiento en la B.D. (para su posterior ejecución).
- Soporte para la programación orientada a objetos (POD).

La unidad de trabajo es el **bloque**, que es un conjunto de declaraciones, instrucciones y mecanismos de gestión de errores y excepciones. Es la estructura básica característica de todos los programas PL/SQL, tiene tres zonas claramente definidas:

- Zona de declaraciones — declaración de variables, constantes, precedido por la clausula DECLARE, es opcional.
- Conjunto de instrucciones — precedido por la clausula BEGIN.
- Zona de excepciones — precedido por la clausula EXCEPTION, es opcional también.
- Ejemplo:

```
[ DECLARE
< declaraciones > ]
BEGIN
< ordenes >
[ EXCEPTION
< gestión de excepciones > ]
END ;
```

La utilización de bloques supone una notable mejora de rendimiento, ya que se envían los bloques completos al servidor para que sean procesados, en lugar de cada sentencia SQL. También se pueden anidar:

```
DECLARE
    ….
BEGIN
    …...
    DECLARE    comienzo bloque interior (un DECLARE dentro de otro)
    …..
    BEGIN
    …..
    EXCEPTION
    …..
    END ;
    ….
EXCEPTION
    …...
END
```

Definición de datos compatibles con SQL

PL/SQL dispone de tipos de datos compatibles con los tipos utilizados para las columnas de las tablas: NUMBER, VARCHAR2, DATE.

Las declaraciones de los datos deben hacerse en la sección de declaraciones:

```
        Contador        NUMBER2 ;
        Nombre          CHAR (20) ;
        Nuevo           VARCHAR2 (15) ;
BEGIN
…..
```

PL/SQL permite declarar una variable del mismo tipo que otra variable o que una columna de una tabla mediante el atributo **%TYPE**.

También se puede declarar una variable para guardar una fila completa de una tabla mediante el atributo **%ROWTYPE**.

Estructuras de control alternativas.

- **Alternativa simple** (IF)
  Si la condición se cumple, se ejecutan las instrucciones que siguen a la clausula THEN

```
IF      <condición> THEN
        instrucciones;
END IF;
```

- **Alternativa doble** (IF ELSE)
  Si la condición se cumple, se ejecutan las instrucciones que siguen a la clausula THEN, en caso contrario, se ejecutaran las instrucciones que siguen a la clausula ELSE

```
IF      <condición> THEN
        instrucciones;
ELSE
        instrucciones;
END IF;
```

- **Alternativa múltiple** (ELSIF)
  Evalúa comenzando desde el principio cada condición, hasta encontrar alguna condición que se cumpla, en cuyo caso se ejecutara las instrucciones que siguen al THEN correspondiente, la clausula ELSE es opcional y se ejecuta si no se ha cumplido ninguna de las anteriores.

```
IF      <condición> THEN
        instrucciones;
ELSIF <condicion2> THEN
        instrucciones;
ELSIF <condicion3> THEN
        instrucciones;
…..
ELSE
        instrucciones;
END IF;
```

- **Alternativa múltiple** (CASE de comprobación)

  Calcula el resultado de la expresión que sigue a la clausula CASE, a continuación comprueba si el valor obtenido coincide con alguno d ellos valores especificados detrás de las clausulas WHEN, en cuyo caso ejecutara la instrucción/es correspondientes, ELSE es opcional.

```
CASE [<expresión>]
WHEN <test1> THEN
        <instrucciones1>;
WHEN <test2> THEN
        <instrucciones2>;
WHEN <test3> THEN
        <instrucciones3>;
[ELSE
        <otras-instrucciones>;]
END CASE;
```

- **Alternativa múltiple** (CASE de búsqueda)

  Evalúa, comenzando desde el principio, cada condición, hasta encontrar alguna condición que se cumpla, en cuyo caso ejecutara las instrucciones que siguen al THEN, el ELSE es opcional.

```
CASE
WHEN <condicion1> THEN
        <instrucciones1>;
WHEN <condicion2> THEN
        <instrucciones2>;
WHEN <condicion3> THEN
        <instrucciones3>;
[ELSE
        <otras-instrucciones>;]
END CASE;
```

Estructuras de control repetitivas.

- **Mientras** (WHILE)

  Se evalúa la condición y si se cumple, se ejecutaran las instrucciones del bucle. El bucle se seguirá ejecutando mientras se cumpla la condición, es una condición de continuación.

```
WHILE <condición>
LOOP
        instrucciones;
END LOOP;
```

- **Para** (FOR)

  <variablecontrol> es la variable de control del bucle que se declara (BINARY_INTEGER), esta variable toma el valor especificado en valorinicio y se repite incrementándose en 1 hasta el valorfinal.

```
FOR <variablecontrol> IN <valorinicio> … <valorfinal>
LOOP
        instrucciones;
END LOOP;
```

  Se utiliza cuando se conoce o se puede conocer el numero de veces que se debe ejecutar el bucle.

- **Para** (FOR) en incrementos negativos.

```
FOR <variablecontrol> IN REVERSE <valorinicio> … <valorfinal>
LOOP
        instrucciones;
END LOOP;
```

Donde <variablecontro> es la variable de control del bucle que se declara (tipo BINARY_INTEGER), esta variable tomara el valor especificado en valorfinal y se repetirá decrementandose en 1 hasta el valor inicial.

- Respecto de la variable de control en cualquiera de los FOR:
  - No hay que declararla
  - Es local al bucle y no accesible desde el exterior
  - Se puede usar dentro del bucle en una expresión pero no se le pueden asignar valores.

- **Iterar... fin iterar salir si...**(LOOP)
  Se trata de un bucle que se repetirá indefinidamente hasta que se encuentre una instrucción EXIT sin condición o hasta que se cumpla la condición asociada a la clausula EXIT WHEN. Es una condición de salida.

```
LOOP
        instrucciones;
EXIT WHEN <condición>;
        instrucciones;
END LOOP;
```

Desde PL/SQL se puede ejecutar cualquier orden de manipulación de datos.

Ejemplos:

- DELETE FROM clientes WHERE nif = v_nif;

- UPDATE productos SET stock_disponible := stock_disponible – unidades_vendidas WHERE producto_no = v_producto;
  := se utiliza cuando se va a hacer una operación.

PL/SQL permite ejecutar cualquier consulta admitida por la base de datos. Pero cuando se ejecuta la consulta, el resultado no se muestra automáticamente en el terminal del usuario, sino que se queda en un área de memoria denominada cursor, a la que accederemos utilizando variables.

Utilizaremos una o mas variables (declaradas previamente) junto con la clausula INTO para poder acceder a los datos devueltos por nuestra consulta (la consulta solo debe devolver una fila o dará error).
SELECT <columna/s> INTO <variable/s> FROM <tabla> WHERE …..];

Gestión de excepciones.

La excepciones sirven para tratar errores y mensajes de aviso.

- NO_DATA_FOUND una orden de tipo SELECT INTO no ha devuelto ningún valor.
- TOO_MANY_ROWS una orden SELECT INTO ha devuelto mas de una fila.

La sección **EXCEPTION** es la encargada de gestionar mediante los manejadores (WHEN) las excepciones que puedan producirse durante la ejecución de un programa.

Cuando PL/SQL detecta una excepción, automáticamente pasa el control del programa a la sección EXCEPTION del bloque PL/SQL.

PL/SQL permite que el programador defina sus propias excepciones.

Caso practico: Visualiza el apellido y el oficio del empleado con numero 7900

```
DECLARE
        v_ape VARCHAR2 (10);
        v_oficio VARCHAR2(10);
BEGIN
        SELECT apellido, oficio INTO v_ape, v_oficio FROM EMPLEADOS WHERE EMP_NO = 7900 ;
        DBMS_OUTPUT.PUT_LINE (v_ape || '*' || v_oficio) ;
EXCEPTION
        WHEN NO_DATA_FOUND THEN
                RAISE_APPLICATION_ERROR (-20000,'ERROR no hay datos');
        WHEN TOO_MANY_ROWS THEN
                RAISE_APPLICATION_ERROR (-20000,'ERROR demasiados datos');
        WHEN OTHERS THEN
                RAISE_APPLICATION_ERROR (-20000,'ERROR en la aplicación');
END ;
/       si se pone al final el bloque se almacena en el bufer y luego se ejecuta
.       si se pone al final el bloque se almacena en el bufer pero no se ejecuta hasta poner / o RUN.
```

El resultado es:

```
JIMENEZ*EMPLEADO

Statement processed.
```

Estructura modular, tipos de programas que se ejecutan en PL/SQL:

- **Bloques anónimos** (los ejemplos vistos)
  - no tiene nombre, se ejecutan pero no se guardan
  - La zona declaraciones comienza con DECLARE
  - Son las estructuras de ejemplo vistas, pero sin utilización real.

- **Subprogramas**: (Procedimientos y funciones)
  - Son bloques PL/SQL que tienen nombre por el que se invocan desde otros programas.
  - Se compilan, almacén y ejecutan en la base de datos.
  - Tienen una cabecera que incluye el nombre del subprograma.
    - ✔ Se indica si es función o o procedimiento, así como los parámetros.
  - La zona de declaraciones y el bloque del programa empiezan con IS o AS.
  - Pueden ser de dos tipos **Procedimientos** y **Funciones**.

Procedimiento:

```
PROCEDURE <nombreprocedimiento>
        [( <lista de parámetros> )]
IS
        [<declaración objetos locales>;]
BEGIN
        <instrucciones>;
[EXCEPTION
        <excepciones>;]
END [<nombreprocedimiento>] ;
```

Función:

```
FUNCTION <nombrefuncion>
        [( <lista de parámetros> )]
        RETURN <tipo valor devuelto>
IS
        [<declaración objetos locales>;]
BEGIN
        <instrucciones>;
        RETURN <expresión>;
[EXCEPTION
        <excepciones>;]
END [<nombrefuncion>] ;
```

Disparadores de bases de datos / Trigger.

- Son programas almacenados en la base de datos que se asocian a un evento.
- Se ejecutan automáticamente al producirse determinados cambios en la tabla asociada.
- Son muy útiles para controlar los cambios que suceden en la base de datos
  - ✗ Ejemplo:

```
CREATE OR REPLACE TRIGGER audit_borrado_emple
                BEFORE DELETE
                ON emple
            FOR EACH ROW
BEGIN
                DBMS_OUTPUT.PUT_LINE ('BORRADO EMPLEADO'
                        || '*' || :old.emp_no
                        || '*' || :old.apellido);
END;
```

Oracle utiliza el paquete DBMS_OUTPUT con fines de depuración. Este incluye entre otros el procedimiento **PUT_LINE** que permite visualizar textos en la pantalla (para que funcione correctamente la variable de entorno **SERVEROUTPUT** deberá estar en **ON**).

```
SQL> SET  SERVEROUTPUT ON
```

- Podemos guardar el bloque del buffer con la orden **SAVE**:
  - ✔ SAVE nombrefichero [REPLACE] REPLACE se usa si ya existía el fichero.
- Para cargar un bloque de un fichero en el buffer SQL se hace con **GET**:
  - ✔ GET nombrefichero una vez cargado se ejecuta con **RUN** o **/**
- También se puede cargar y ejecutar con una sola orden con **START** o **@**:
  - ✔ START nombre fichero
  - ✔ @nombrefichero

Ejemplo:

```
CREATE OR REPLACE
PROCEDURE ver_depart [numdepart NUMBER]
AS
        v_dnombre VARCHAR2(14);
        v_localidad VARCHAR2(14);
BEGIN
        SELECT dnombre, loc INTO v_dnombre, v_localidad
                FROM depart
                WHERE dept_no = numdepart;
        DBMS_OUTPUT.PUT_LINE ('Num depart:' | | numdepart | | ' * Nombre dep:' | | v_dnombre | |
                                        ' * Localidad: ' | | v_localidad) ;
EXCEPTION
        WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE ('No encontrado departamento ') ;
END ver_depart ;
/
```

Para ejecutar este procedimiento creado se hace con EXECUTE nombre(dato), en modo terminal, ejemplo:
EXECUTE ver_depart(20); o **BEGIN** ver_depart(20) **END** ; en modo gráfico.

Colocar el símbolo **&** delante de una variable indica que se pedirá la introducción del dato especificado.

Fundamentos del lenguaje PL/SQL (Cap. 9)

Pl/SQL dispone de los mismos datos que SQL ademas de otros propios, estos datos se pueden clasificar en:

- Escalares Almacenan valores simples
  - ✔ **Carácter/cadena**:
    - CHAR (L) Cadenas de caracteres de longitud fija, posiciones no usadas son blancos
    - NCHAR (L) similar al anterior
    - VARCHAR2 (L) Cadenas de caracteres de longitud variable
    - NVARCHAR2 (L)
    - LONG (L) Almacena cadenas de longitud variable, se debe especificar el máximo
    - RAW (L) Almacena datos binarios de longitud fija
    - LONG RAW (L) Almacena datos binarios de longitud variable
    - ROWID Almacena identificadores de direcciones de fila
    - UROWID (L) Almacena identificadores de direcciones de fila, especificando nº bytes
  - ✔ **Numérico**
    - NUMBER (P, E) Numérico, P numero total de dígitos, E numero decimales
    - BINARY_INTEGER Numérico entero, se almacena en memoria en formato binario
    - PLS_INTEGER Similar al anterior pero mas rápido/da error si desbordamiento
    - BINARY_DOUBLE Para el ámbito científico con cálculos muy precisos y complejos
    - BINARY_FLOAT Para el ámbito científico con cálculos muy precisos y complejos
  - ✔ **Boolean** Almacena valores lógicos, TRUE, FALSE, NULL
  - ✔ **Fecha/hora**
    - DATE Almacena fechas incluyendo la hora
    - TIMESTAMP Almacena fechas incluyendo la hora y fracciones de segundo
    - INTERVAL Intervalo de tiempo entre fechas
  - ✔ **Otros**
    - ROWID
    - UROWIN

- Compuestos compuestos de otros simples
  - ✔ Tablas indexadas
  - ✔ Tablas anidadas
  - ✔ Varrays
  - ✔ Objetos

- Referencias difieren de los anteriores por características de manejo y almacenamiento.
  - ✔ REF CURSOR Son referencias a cursores
  - ✔ REF Son punteros a objetos

- LOB almacenan objetos de grandes dimensiones (de hasta 128 terabytes, sustituyen a LONG RAW)

Los **Identificadores** se utilizan para nombrar los objetos que intervienen en un programa:
- Variables, constantes, cursores, excepciones, procedimientos, funciones, etiquetas....
- Pueden tener entre 1 y 30 caracteres de longitud
- El primer carácter ha de ser una letra
- El resto de caracteres han de ser alfanuméricos
- No pueden incluir signos de puntuación, espacios, etc.
- No se diferencia entre mayúsculas o minúsculas.

La opción **DEFAULT** o **:=** sirve para asignar valores por defecto a la variable tras su creación.

La opción **NOT NULL** fuerza a que la variable tenga siempre un valor.

Atributo **%TYPE** declara una variable del mismo tipo que otra (nombre_variable nombre_objeto%TYPE;)

Atributo **%ROWTYPE** declara una variable de registro cuyos campos se corresponden con las columnas de una tabla (nombre_variable nombre_objeto%ROWTYPE;).

Las **variables** se crean al comienzo del bloque y dejan de existir una vez finalizada la ejecución del bloque en que han sido declaradas, el ámbito de una variable incluye el bloque en el que se declara y sus bloques hijos.

Las variables declaradas en los bloques hijos no son accesibles desde el bloque padre.

Al declarar **constantes** se deberá asignar un valor: <nombre_constante> CONSTANT <tipo> := <valor> ;

Los **literales** representan valores constantes directamente, se utilizan para visualizar valores.
- ◆ Carácter 'A', 'a', '*'
- ◆ Cadena 'Cliente Nº:'
- ◆ Numérico 567, 2.23, -8.75
- ◆ Booleano v_cobrado := TRUE;
- ◆ Fecha/hora DATE '2005-11-09'

Los **operadores** se utilizan para asignar valores y formar expresiones:
- ➔ Asignación **:=** Asigna un valor a una variable Edad := 19
- ➔ Concatenación **||** Une dos o mas cadenas 'Buenos' || 'dias'
- ➔ Comparación = , **!=** , < , > , <= , >= , **IN** , **IS NULL** , **LIKE** , **BETWEEN**
- ➔ Aritméticos + , **-** , * , / , **
  - f1 – f2 Devuelve el numero de días que hay entre las fechas f1 y f2.
  - f + n Devuelve una fecha que es el resultado de sumar n días a la fecha f.
  - f - n Devuelve una fecha que es el resultado de restar n días a la fecha f.

➔ Lógicos **AND** , **OR** , **NOT**

➔ Otros indicadores

- **( )** Delimitador de expresiones
- **' '** Delimitador de literales de cadena
- **“ “** Delimitador de identificadores
- **<< >>** Etiquetas
- **/* */** Delimitador comentarios de varias lineas
- **--** Indicador de comentario de una linea
- **%** Indicador de atributo (TYPE, ROWTYPE, FOUND, …)
- **:** Indicador de variables de transferencia
- **,** Separador de item de lista
- **;** Terminador de instrucción
- **@** Indicador de enlace de bases de datos

Administración de Oracle (Cap. 12)

Las tareas del Administrador de la Base de datos son:

- ✔ Instalar Oracle.
- ✔ Diseñar y crear una base de datos.
- ✔ Arrancar y detener la Base de Datos.
- ✔ Crear y controlar usuarios.
- ✔ Conceder privilegios.
- ✔ Gestionar el espacio.
- ✔ Hacer copias de seguridad y recuperar la Base de datos.

Componentes de la Base de Datos:

- ✔ Archivos de Datos (Database files). Contienen la información de la base de datos (datos de usuario y datos de sistema), es necesario crear un espacio para las tablas (tablespace) y dentro de el crear las tablas, tablespaces por defecto que se crean al instalar Oracle:
  - SYSTEM. Contiene la información que necesita Oracle para gestionarse a si misma.
  - USERS. Contiene información personal de los usuarios.
  - TEMP. Contiene las tablas temporales.
  - UNDOTBS1. Es donde Oracle guarda la información de deshacer.
- ✔ Registros de rehacer o Redo log (el registro de las transacciones). Son archivos de datos el los que Oracle registra los cambios que se efectúan sobre los datos (INSERT, UPDATE y DELETE) de la base de datos dentro de la cache de buffers de la base de datos, estos archivos se utilizan en situaciones de fallo para recuperar datos.
- ✔ Archivos de control (Control files). Contienen información sobre los archivos asociados con una base de datos Oracle. Todas las modificaciones importantes que se hagan en la estructura de la base de datos se registran en el archivo de control. Estos archivos mantienen la integridad de la base de datos.

La seguridad de la base de datos se clasifica en dos categorías:

- ✔ Seguridad del sistema. Controla el acceso y uso de la base de datos a nivel del sistema.
- ✔ Seguridad de los datos. Controlan el acceso y uso de la base de datos a nivel de objetos (tabla, vista, etc.).

Un **usuario** es un nombre definido en la base de datos que se puede conectar a ella y acceder a determinados objetos según ciertas condiciones que define el administrador. Asociado a cada usuario de la base de datos hay un esquema con el mismo nombre.

- ✔ Esquema es una colección lógica de objetos (tablas, vistas, secuencias, sinónimos, indices, cluster, procedures, funciones, paquetes, etc.), por defecto cada usuario accede a los objetos de su esquema y puede acceder a los objetos de otro usuario si este le concede privilegios.

Creación de usuarios. Al instalar la base de datos se crean dos usuarios con el privilegio de administrador de la base de datos (**DBA**), SYS y SYSTEM.

- ✔ **SYS** es el propietario de las tablas del diccionario de datos y solo nos conectaremos con este usuario cuando las instrucciones de Oracle lo exijan.
- ✔ **SYSTEM** se crea para las tareas administrativas de la base de datos. No se suelen crear tablas de usuario en el esquema SYSTEM. Para crear otros usuarios es preciso conectarse como usuario SYSTEM.

El **comando para crear usuarios** es:

```
CREATE USER nombre_usuario IDENTIFIED BY clave
DEFAULT TABLESPACE espacio_tabla
TEMPORARY TABLESPACE espacio_tabla
QUOTA (K/M) | UNLIMITED ON espacio_tabla
PROFILE perfil ;
```

Vistas con información de usuarios:

- ✔ USER_USERS información usuario actual.
- ✔ ALL_USERS información de todos los usuarios.

Modificación de usuarios:

```
ALTER USER nombre_usuario
IDENTIFIED BY clave
DEFAULT TABLESPACE espacio_tabla
TEMPORARY TABLESPACE espacio_tabla
QUOTA (K/M) | UNLIMITED ON espacio_tabla
PROFILE perfil ;
```

Borrado de usuarios:

```
DROP USER usuario [CASCADE] ;
```

- la opción CASCADE suprime todos los objetos del usuario antes de borrarlo.

Un **Privilegio** es la capacidad de un usuario dentro de la base de datos para realizar determinadas operaciones o acceder a determinados objetos de otros usuarios.

- Roles (funciones)
  - CONNECT ALTER SESSION, CREATE CLUSTER, CREATE DATABASE LINK, CREATE SEQUENCE, CREATE SESSION, CREATE SYNONYM, CREATE TABLE y CREATE VIEW.
  - RESOURCE CREATE CLUSTER, CREATE INDEXTYPE, CREATE OPERATOR, CREATE PROCEDURE, CREATE TABLE, CREATE SEQUENCE, CREATE TRIGGER y CREATE TYPE.
  - DBA Posee todos los privilegios de sistema.

- Privilegios sobre los objetos.
  - ALTER (Tabla y Secuencia)
  - DELETE (Tabla, Vista)
  - EXECUTE (Procedure)
  - INDEX (Tabla)
  - INSERT (Tabla, Vista)
  - REFERENCES (Tabla)
  - SELECT (Tabla, Vista, Secuencia)
  - UPDATE (Tabla, Vista)
  -
- Privilegios del sistema
  - CREATE PROCEDURE Crear procedimientos y funciones.
  - CREATE PROFILE Crear perfil de usuario.
  - ALTER PROFILE Modificar perfil.
  - DROP PROFILE Borrar perfil.
  - CREATE SESSION Conectarse a la base de datos.
  - ALTER SESSION
  - CREATE TABLE Crear tablas en nuestro esquema.
  - CREATE TABLESPACE
  - ALTER TABLESPACE
  - CREATE USER
  - ALTER USER
  - DROP USER
  - CREATE VIEW
  - Y muchos mas.....

La **orden para dar privilegios sobre los objetos** es GRANT:

```
GRANT priv.objeto, priv.objeto …. | ALL PRIVILEGES
        columna, columna …..
ON usuario.objeto
TO usuario | rol | PUBLIC , usuario | rol | PUBLIC
WITH GRANT OPTION ;
```

La **orden para dar privilegios del sistema** es GRANT de la siguiente manera:

```
GRANT privilegio | rol,  privilegio | rol
TO usuario | rol | PUBLIC , usuario | rol | PUBLIC
WITH ADMIN OPTION ;
```

Retirada de privilegios sobre objetos.

```
REVOKE priv.objeto, priv.objeto …. | ALL PRIVILEGES
ON usuario.objeto
FROM usuario | rol | PUBLIC , usuario | rol | PUBLIC ;
```

Retirada de privilegios de sistema.

```
REVOKE priv_sistema | rol , priv_sistema | rol ….
FROM usuario | rol | PUBLIC , usuario | rol | PUBLIC …. ;
```

Vistas con información de los privilegios (SELECT * FROM …..)

✔ SESSION_PRIVS	Privilegios del usuario activo.
✔ USER_SYS_PRIVS	Privilegios de sistema asignado al usuario.
✔ DBA_SYS_PRIVS	Privilegios de sistema asignados a los usuarios o a los roles.
✔ USER_TAB_PRIVS	Concesiones sobre objetos que son propiedad del usuario, concedidos o recibidos.
✔ USER_TAB_PRIVS_MADE	Concesiones sobre objetos que son propiedad del usuario (asignadas).
✔ USER_TAB_PRIVS_RECD	Concesiones sobre objetos que recibe el usuario.
✔ USER_COL_PRIVS	Concesiones sobre columnas en las que el usuario es el propietario, asigna el privilegio o lo recibe.
✔ USER_COL_PRIVS_MADE	Todas las concesiones sobre columnas de objetos que son propiedad del usuario.
✔ USER_COL_PRIVS_RECD	Concesiones sobre columnas recibidas por el usuario.

Roles, un rol engloba un conjunto de privilegios y este puede asignarse a uno o varios usuarios. El formato para crear un rol es:

```
CREATE ROLE nombreROL ;
```

Conceder privilegios a los roles.

```
GRANT SELECT, INSERT ON EMPLE TO ACCESO ;
GRANT INSERT ON DEPART TO ACCESO ;
GRANT CREATE SESSION TO ACCESO ;
```

Conceder el rol a un usuario:

```
GRANT ACCESO TO usuario ;
```

Supresión de un rol.

```
DROP ROLE nombreROL ;
```

Establecer un rol por defecto.

```
ALTER USER usuario DEFAULT ROLE nombreROL ;
```

Información sobre roles en el diccionario de datos.

- ✔ ROLE_SYS_PRIVS Privilegios del sistema asignados a roles.
- ✔ ROLE_TAB_PRIVS Privilegios sobre tablas aplicados a roles.
- ✔ ROLE_ROLE_PRIVS Roles asignados a otros roles.
- ✔ SESSION_ROLES Roles activos para el usuario.
- ✔ USER_ROLE_PRIVS Roles asignados al usuario.

Perfiles, es un conjunto de limites a los recursos de la base de datos, si no se crean perfiles se utiliza el perfil por defecto que especifica recursos ilimitados.
CREATE PROFILE nombre_perfil LIMIT parámetros_recursos ;

Los recursos son:

- ✔ SESSIONS_PER_USER Numero de sesiones múltiples concurrentes permitidas por nombre de usuario.
- ✔ CONNECT_TIME Indica el numero de minutos que puede estar una sesión conectada.
- ✔ IDLE_TIME Indica el numero de minutos que puede estar una sesión conectada sin ser utilizada de forma activa.
- ✔ FAILED_LOGIN_ATTEMPTS Numero de intentos de acceso sin éxito consecutivos que producirá el bloqueo de la cuenta.
- ✔ Y otros tantos mas …...

Información sobre los perfiles:
DBA_PROFILES

Borrado de un perfil.
DROP PROFILE nombre_perfil [CASCADE] ;

Administración de Oracle (Cap. 13)

Un **tablespace** es una unidad lógica de almacenamiento de datos representada físicamente por uno o mas archivos de datos.

Para crear un tablespace se usa la orden CREATE TABLESPACE.

```
CREATE TABLESPACE tablespace1
DATAFILE ‘arch1’ [SIZE entero [K|M]
[REUSE]
[,‘arch2’ [SIZE entero [K|M]
[REUSE]…
[DEFAULT STORAGE
(
INITIAL tamaño
NEXT tamaño
MINEXTENTS tamaño
MEXEXTENTS tamaño
PCTINCREASE valor
)]
[ONLINE | OFLINE] ;
```

Para borrar un tablespace se usa la orden
DROP TABLESPACE nombre_tablespace

Desconexion de un tablespace.
ALTER TABLESPACE tablespace1
{ONLINE | OFFLINE [NORMAL | TEMPORARY | INMEDIATE]} ;

Ejercicio. Gimnasio

Supongamos que tenemos una Base de Datos denominada GIMNASIO formada por las siguientes tablas:

PAGOS:
(CODIGO_USUARIO VARCHAR2 (7) NO NULO
NUMERO_MES NUMBER (2) NO NULO
CUOTA NUMBER (7)
OBSERVACIONES VARCHAR2 (500)
CLAVE PRIMARIA (CODIGO_USUARIO, NUMERO_MES)
CLAVE EXTERNA (CODIGO_USUARIO) que referencia a USUARIOS (NUM_SOCIO)

USUARIOS:
(NUM_SOCIO VARCHAR2 (7) NO NULO
DNI VARCHAR2 (9) NO NULO
NOMBRE VARCHAR2 (20)
APELLIDOS VARCHAR2 (30)
FOTOGRAFIA LONG RAW
DOMICILIO VARCHAR2 (40)
LOCALIDAD VARCHAR2 (50)
CP VARCHAR2 (5)
FECHA_NACIMIENTO DATE
TELEFONO VARCHAR2 (20)
TAQUILLA VARCHAR2 (15)
HORARIO VARCHAR2 (15)
FECHA_ALTA DATE
FECHA_BAJA DATE
CUOTA_SOCIO NUMBER (7)
CUOTA_FAMILIAR NUMBER (7)
PAGA_BANCO CHAR NOT NULL
CODIGO_BANCO NUMBER (8)
CUENTA NUMBER (10)
DIGITO_CONTROL NUMBER (2)
OBSERVACIONES VARCHAR2 (500)
CLAVE PRIMARIA (NUM_SOCIO)
CLAVE ALTERNATIVA (DNI)
CLAVE EXTERNA (CODIGO_BANCO) que referencia a BANCOS (ENT_SUC)

ACTIVIDADES
(CODIGO_ACTIVIDAD VARCHAR2 (7) NO NULO
DESCRIPCION VARCHAR2 (50)
CUOTA NUMBER (7)
CLAVE PRIMARIA (CODIGO_ACTIVIDAD)

BANCOS:
(ENT_SUC NUMBER (8) NO NULO
NOMBRE VARCHAR2 (50)
DIRECCION VARCHAR2 (50)
LOCALIDAD VARCHAR2 (30)
TELEFONOS VARCHAR2 (30)
CLAVE PRIMARIA (ENT_SUC)

ACTIVIDADES_USUARIOS:
(CODIGO_ACTIVIDAD VARCHAR2 (7) NO NULO
CODIGO_USUARIO VARCHAR2 (7) NO NULO
FECHA_ALTA DATE,
FECHA_BAJA DATE,
CLAVE PRIMARIA (CODIGO_ACTIVIDAD, CODIGO_USUARIO),
CLAVE EXTERNA (CODIGO_ACTIVIDAD) que referencia a ACTIVIDADADES (CODIGO_ACTIVIDAD)
CLAVE AJENA (CODIGO_USUARIO) que referencia a USUARIOS (NUM_SOCIO)

1º Crear las tablas:

```
CREATE TABLE BANCOS
(ENT_SUC NUMBER (8) NOT NULL,
NOMBRE VARCHAR2 (50),
DIRECCION VARCHAR2 (50),
LOCALIDAD VARCHAR2 (30),
TELEFONOS VARCHAR2 (30),
CONSTRAINT BANCOS_PK PRIMARY KEY(ENT_SUC)) ;

CREATE TABLE USUARIOS
(NUM_SOCIO VARCHAR2 (7) NOT NULL,
DNI VARCHAR2 (9) NOT NULL,
NOMBRE VARCHAR2 (20),
APELLIDOS VARCHAR2 (30),
DOMICILIO VARCHAR2 (40),
LOCALIDAD VARCHAR2 (50),
CP VARCHAR2 (5),
FECHA_NACIMIENTO DATE,
TELEFONO VARCHAR2 (20),
TAQUILLA VARCHAR2 (15),
HORARIO VARCHAR2 (15),
FECHA_ALTA DATE,
FECHA_BAJA DATE,
CUOTA_SOCIO NUMBER (7),
CUOTA_FAMILIAR NUMBER (7),
PAGA_BANCO CHAR NOT NULL,
CODIGO_BANCO NUMBER (8),
CUENTA NUMBER (10),
DIGITO_CONTROL NUMBER (2),
OBSERVACIONES VARCHAR2 (500),
CONSTRAINT USUARIOS_PK PRIMARY KEY (NUM_SOCIO),
CONSTRAINT USUARIOS_FK FOREIGN KEY (CODIGO_BANCO) REFERENCES BANCOS
(ENT_SUC) ON DELETE CASCADE);

CREATE TABLE ACTIVIDADES
( CODIGO_ACTIVIDAD VARCHAR2 (7) NOT NULL,
DESCRIPCION VARCHAR2 (50),
CUOTA NUMBER (7),
CONSTRAINT ACTIVIDADES_PK PRIMARY KEY (CODIGO_ACTIVIDAD)) ;
```

```
CREATE TABLE ACTIVIDADES_USUARIOS
(CODIGO_ACTIVIDAD      VARCHAR2 (7) NOT NULL,
CODIGO_USUARIO         VARCHAR2 (7) NOT NULL,
FECHA_ALTA             DATE,
FECHA_BAJA             DATE,
CONSTRAINT ACTIVIDADADES_USUARIOS_PK PRIMARY KEY
(CODIGO_ACTIVIDAD,CODIGO_USUARIO),
CONSTRAINT ACTIVIDADADES_USUARIOS_FK_USUARIOS  FOREIGN KEY
(CODIGO_USUARIO) REFERENCES USUARIOS (NUM_SOCIO) ON DELETE CASCADE,
CONSTRAINT ACTIVIDADADES_USUARIOS_FK  FOREIGN KEY (CODIGO_ACTIVIDAD)
REFERENCES ACTIVIDADES (CODIGO_ACTIVIDAD) ON DELETE CASCADE) ;
```

```
CREATE TABLE PAGOS
(CODIGO_USUARIO            VARCHAR2 (7) NOT NULL,
NUMERO_MES             NUMBER (2) NOT NULL,
CUOTA                  NUMBER (7),
OBSERVACIONES          VARCHAR2 (500),
CONSTRAINT PAGOS_PK PRIMARY KEY(CODIGO_USUARIO, NUMERO_MES),
CONSTRAINT PAGOS_FK FOREIGN KEY(CODIGO_USUARIO) REFERENCES USUARIOS
(NUM_SOCIO) ON DELETE CASCADE) ;
```

2º Visualizar los Bancos con sus correspondientes usuarios ordenados por nombre de Banco.

```
SELECT INITCAP(CONCAT(U.NOMBRE||' ', U.APELLIDOS)) "Usuario", B.NOMBRE "Banco"
FROM USUARIOS U, BANCOS B
WHERE U.CODIGO_BANCO = B.ENT_SUC
ORDER BY B.NOMBRE, U.NOMBRE;
```

Usuario	Banco
Jose Alvarez Castro	ATLANTICO
Luis Bulnes Balbin	ATLANTICO
Marta Arrien Gonzalez	ATLANTICO
Pablo Rodriguez Arias	ATLANTICO
Juan Luis Arias Alvarez	BANESTO
Dolores Moreno Rodriguez	BBV
Manuel Alonso Ovies	BBV
Maria Alonso Gutierrez	BBV
Ines Perez Diaz	CAJA DE ASTURIAS
Marta Arias Santolaya	CAJA DE ASTURIAS
Maria Egia Santamarina	HERRERO
Ana Gutierrez Alonso	SANTANDER
Laura Fernandez Alonso	SANTANDER
Ramon Arboleya Garcia	SANTANDER

3º Selecciona las actividades cuya cuota sea inferior a la media de las cuotas de los usuarios que tienen cuota familiar.

```
SELECT DESCRIPCION, CUOTA FROM ACTIVIDADES
WHERE CUOTA < (SELECT TRUNC(AVG(CUOTA), 2)
FROM PAGOS
WHERE CODIGO_USUARIO IN
(SELECT NUM_SOCIO FROM USUARIOS WHERE CUOTA_FAMILIAR IS NOT NULL));
```

DESCRIPCION	CUOTA
GIMNASIA DE MANTENIMIENTO	1000
GIMNASIA RITMICA	800
JUDO	1200
KARATE	1100
NATACION 1	900
NATACION 2	1000
MUSCULACION	1300

4º Visualizar nombres y apellidos de aquellos usuarios cuyo primer y segundo apellido empiece por a.

```
SELECT NOMBRE, SUBSTR(APELLIDOS,1,(INSTR (APELLIDOS,' ')-1)) "Primer apellido",
SUBSTR(APELLIDOS,(INSTR (APELLIDOS,' ')+1)) "Segundo apellido"
FROM USUARIOS
WHERE UPPER(SUBSTR(APELLIDOS,1,(INSTR (APELLIDOS,' ')-1))) LIKE 'A%'
AND UPPER(SUBSTR(APELLIDOS,(INSTR (APELLIDOS,' ')+1))) LIKE 'A%'
ORDER BY NOMBRE;
```

saco el primer apellido:

```
SELECT APELLIDOS, SUBSTR(APELLIDOS,1,(INSTR (APELLIDOS,' ')-1))
FROM USUARIOS ;
```

saco el segundo apellido:

```
SELECT APELLIDOS, SUBSTR(APELLIDOS,(INSTR (APELLIDOS,' ')+1))
FROM USUARIOS ;
```

NOMBRE	Primer Apellido	Segundo Apellido
JUAN LUIS	ARIAS	ALVAREZ

5º Visualizar los nombres de usuarios, la actividad que realizan y el nombre del banco a través del que pagan la cuota.

```
SELECT INITCAP(CONCAT(U.NOMBRE||' ', U.APELLIDOS)) "Usuario", A.DESCRIPCION,
B.NOMBRE
FROM USUARIOS U, ACTIVIDADES A, BANCOS B, ACTIVIDADES_USUARIOS AU
WHERE U.NUM_SOCIO=AU.CODIGO_USUARIO
AND AU.CODIGO_ACTIVIDAD=A.CODIGO_ACTIVIDAD
AND B.ENT_SUC=U.CODIGO_BANCO;
```

Usuario	DESCRIPCION	NOMBRE
Juan Luis Arias Alvarez	KARATE	BANESTO
Juan Luis Arias Alvarez	GIMNASIA DE MANTENIMIENTO	BANESTO
Ines Perez Diaz	NATACION 1	CAJA DE ASTURIAS
Ines Perez Diaz	JUDO	CAJA DE ASTURIAS
Marta Arias Santolaya	NATACION 2	CAJA DE ASTURIAS
Marta Arias Santolaya	KARATE	CAJA DE ASTURIAS

6º Visualizar nombre de usuario, numero y apellido de los usuarios de la siguiente forma:
"El usuario con numero xxxx se llama Juan Luis Arias Alvarez**"**

```
SELECT CONCAT('El usuario con numero ', NUM_SOCIO) || CONCAT(' se llama ',
CONCAT(INITCAP(NOMBRE) || ' ', INITCAP(APELLIDOS))) "Numero usuario y Nombre"
FROM USUARIOS
ORDER BY NUM_SOCIO;
```

Usuario	DESCRIPCION	NOMBRE
Juan Luis Arias Alvarez	KARATE	BANESTO
Juan Luis Arias Alvarez	GIMNASIA DE MANTENIMIENTO	BANESTO
Ines Perez Diaz	NATACION 1	CAJA DE ASTURIAS
Ines Perez Diaz	JUDO	CAJA DE ASTURIAS
Marta Arias Santolaya	NATACION 2	CAJA DE ASTURIAS
Marta Arias Santolaya	KARATE	CAJA DE ASTURIAS

7º Visualizar el nombre de todos los bancos junto con la localidad donde se encuentra :
"Banesto esta en Algorta"

```
SELECT CONCAT(NOMBRE, ' esta en ') || INITCAP(LOCALIDAD) "Banco y localidad"
FROM BANCOS
ORDER BY NOMBRE;
```

Banco Y Localidad
ATLANTICO esta en Luanco
BANESTO esta en Mieres
BBV esta en Luanco
CAJA DE ASTURIAS esta en Mieres
HERRERO esta en Mieres
SANTANDER esta en Aviles

8º Cuenta el nº de usuarios que pagan a través del Banco Santander

```
SELECT COUNT (*) "Usuarios pagan Santander" FROM USUARIOS
WHERE CODIGO_BANCO=(SELECT ENT_SUC FROM BANCOS WHERE UPPER(NOMBRE)
LIKE 'SANTANDER')
GROUP BY CODIGO_BANCO;
```

Usuarios Pagan Santander
3

9º Obtener los nombres y fechas de alta de todos los usuarios de la siguiente forma:
"Juan Luis se dio de alta el día 1 de Enero de 1970"

```
SELECT CONCAT(INITCAP (NOMBRE), ' se dio de alta el dia ')|| TO_CHAR (FECHA_ALTA, 'dd  "
de " Month " de " yyyy') "Nombre y Fecha de alta" FROM USUARIOS;
```

Nombre Y Fecha De Alta
Juan Luis se dio de alta el dia 10 de Enero de 1996
Ines se dio de alta el dia 10 de Enero de 1996
Jose se dio de alta el dia 21 de Mayo de 1996
Maria se dio de alta el dia 10 de Noviembre de 1998
Marta se dio de alta el dia 12 de Diciembre de 1997
Ana se dio de alta el dia 02 de Marzo de 1998
Laura se dio de alta el dia 11 de Diciembre de 1997
Maria se dio de alta el dia 10 de Noviembre de 1998
Manuel se dio de alta el dia 05 de Junio de 1996
Ramon se dio de alta el dia 03 de Abril de 1995
Dolores se dio de alta el dia 03 de Marzo de 1998
Pablo se dio de alta el dia 06 de Septiembre de 1997
Marta se dio de alta el dia 06 de Septiembre de 1997
Luis se dio de alta el dia 14 de Noviembre de 1996
Jose se dio de alta el dia 01 de Abril de 1995
Pelayo se dio de alta el dia 01 de Mayo de 1997

La pregunta original: **Obtener nombres y fecha nacimiento de todos los usuarios de la siguiente forma:**
"Juan Luis nació el día 1 de Enero de 1970" No se puede hacer, falta el dato: fecha nacimiento.

Datos a meter en las tablas (para las consultas)**:**

```
Insert into bancos (ent_suc, nombre, direccion, localidad) values (
30700018,'BANESTO','MANUEL LLANEZA, 33','MIERES');

Insert into bancos (ent_suc, nombre, direccion, localidad) values (
20480070,'CAJA DE ASTURIAS','MANUEL LLANEZA, 17','MIERES');

Insert into bancos (ent_suc, nombre, direccion, localidad) values (
43000250,'HERRERO','MANUEL LLANEZA, 22','MIERES');

Insert into bancos (ent_suc, nombre, direccion, localidad) values (
85002222,'SANTANDER','LA CAMARA, 13','AVILES');

Insert into bancos (ent_suc, nombre, direccion, localidad) values (
22223333,'BBV','LA RIBERA, 17','LUANCO');

Insert into bancos (ent_suc, nombre, direccion, localidad) values (
33334444,'ATLANTICO','GIJON, 56','LUANCO');

Insert into usuarios (num_socio, dni, nombre, apellidos, domicilio,
    localidad, cp, fecha_alta, cuota_socio ,cuota_familiar, paga_banco, codigo_banco,
    cuenta, digito_control)
    values ('A1111','01111111','JUAN LUIS','ARIAS ALVAREZ',
    'C/ LA VEGA, 18','MIERES','33600',to_date('10/01/1996','dd/mm/yyyy'),
    5500,null,'S',30700018,1111,10);

Insert into usuarios (num_socio, dni, nombre, apellidos, domicilio,
    localidad, cp, fecha_alta, cuota_socio, cuota_familiar, paga_banco, codigo_banco,
    cuenta, digito_control)
    values ( 'A2222','02222222','INES','PEREZ DIAZ',
    'C/DR. FLEMING, 14','MIERES','33600',to_date('10/01/1996','dd/mm/yyyy'),
    6800,2000,'S',20480070,2222,47);

Insert into usuarios (num_socio, dni, nombre, apellidos, domicilio,
    localidad, cp, fecha_alta, cuota_socio, cuota_familiar, paga_banco)
    values ( 'A3333','03333333','JOSE','RUIZ PEÑA',
    'C/ AVILES, 18','LUANCO', '33400',to_date('21/05/1996','dd/mm/yyyy'),
    6000,null,'N');

Insert into usuarios (num_socio, dni, nombre, apellidos, domicilio,
    localidad, cp, fecha_alta, cuota_socio, cuota_familiar, paga_banco, codigo_banco,
    cuenta, digito_control)
    values ( 'I1111','00111111','MARIA','EGIA SANTAMARINA',
    'C/GERNIKA, 3','MIERES','33600',
    to_date('10/11/1998','dd/mm/yyyy'),7000,null,'S',43000250,123456,21);

Insert into usuarios (num_socio, dni, nombre, apellidos, domicilio,
    localidad, cp, fecha_alta, cuota_socio, cuota_familiar, paga_banco, codigo_banco,
    cuenta, digito_control)
    values ( 'I2222','77896542','MARTA','ARIAS SANTOLAYA',
    'C/ ASTURIAS, 23 4º','MIERES','33600',to_date('12/12/1997','dd/mm/yyyy'),
    4500,2500,'S',20480070,2242,41);

Insert into usuarios (num_socio, dni, nombre, apellidos, domicilio,
    localidad, cp, fecha_alta, cuota_socio, cuota_familiar, paga_banco, codigo_banco,
    cuenta, digito_control)
    values ( 'J1111','11111111','ANA','GUTIERREZ ALONSO',
    'C/ ASTURIAS, 51 4º','LUANCO','33440',to_date('02/03/1998','dd/mm/yyyy'),
    5000,3500,'S',85002222,2342,61);

Insert into usuarios (num_socio, dni, nombre, apellidos, domicilio,
```

```
    localidad, cp, fecha_alta, cuota_socio, cuota_familiar, paga_banco, codigo_banco,
    cuenta, digito_control)
    values ( 'J2222','22222222','LAURA','FERNANDEZ ALONSO',
    'C/ TEVERGA, 17 2º','LUANCO','33440',to_date('11/12/1997','dd/mm/yyyy'),
    5000,null,'S',85002222,234255,34);

Insert into usuarios (num_socio, dni, nombre, apellidos, domicilio,
    localidad, cp, fecha_alta, cuota_socio, cuota_familiar, paga_banco, codigo_banco,
    cuenta, digito_control)
    values ( 'J3333','33333333','MARIA','ALONSO GUTIERREZ',
    'C/ OVIEDO, 8 1º','LUANCO','33440',to_date('10/11/1998','dd/mm/yyyy'),
    6000,2500,'S',22223333,425566,43);

Insert into usuarios (num_socio, dni, nombre, apellidos, domicilio,
    localidad, cp, fecha_alta, cuota_socio, cuota_familiar, paga_banco, codigo_banco,
    cuenta, digito_control)
    values ( 'J4444','44444444','MANUEL','ALONSO OVIES',
    'C/ OVIEDO, 18 4º','LUANCO','33440',to_date('05/06/1996','dd/mm/yyyy'),
    6000,null,'S',22223333,425566,43);

Insert into usuarios (num_socio, dni, nombre, apellidos, domicilio,
    localidad, cp, fecha_alta, cuota_socio, cuota_familiar, paga_banco, codigo_banco,
    cuenta, digito_control)
    values ( 'J5555','55555555','RAMON','ARBOLEYA GARCIA',
    'C/ LANGREO, 9 1º','AVILES','33400',to_date('03/04/1995','dd/mm/yyyy'),
    4000,2500,'S',85002222,4343,12);

Insert into usuarios (num_socio, dni, nombre, apellidos, domicilio,
    localidad, cp, fecha_alta, cuota_socio, cuota_familiar, paga_banco, codigo_banco,
    cuenta, digito_control)
    values ( 'J6666','66666666','DOLORES','MORENO RODRIGUEZ',
    'C/ OVIEDO, 23 6º','AVILES','33400',to_date('03/03/1998','dd/mm/yyyy'),null,null,'S',22223333,6675,21);

Insert into usuarios (num_socio, dni, nombre, apellidos, domicilio,
    localidad, cp, fecha_alta, cuota_socio, cuota_familiar, paga_banco, codigo_banco,
    cuenta, digito_control)
    values ( 'J7777','77777777','PABLO','RODRIGUEZ ARIAS',
    'C/ LA FLORIDA, 3 6º','AVILES','33400',to_date('06/09/1997','dd/mm/yyyy'),
    4000,2000,'S',33334444,6775,12);

Insert into usuarios (num_socio, dni, nombre, apellidos, domicilio,
    localidad, cp, fecha_alta, cuota_socio, cuota_familiar, paga_banco, codigo_banco,
    cuenta, digito_control)
    values ( 'J8888','88888888','MARTA','ARRIEN GONZALEZ',
    'PLAZA SAN JUAN, 9','MIERES','33600',to_date('06/09/1997','dd/mm/yyyy'),
    7000,null,'S',33334444,9975,11);

Insert into usuarios (num_socio, dni, nombre, apellidos, domicilio,
    localidad, cp, fecha_alta, cuota_socio, cuota_familiar, paga_banco, codigo_banco,
    cuenta, digito_control)
    values ( 'J9999','99999999','LUIS','BULNES BALBIN',
    'C/ LA VEGA, 49 2º','MIERES','33600',to_date('14/11/1996','dd/mm/yyyy'),
    7000,2000,'S',33334444,39975,22);

Insert into usuarios (num_socio, dni, nombre, apellidos, domicilio,
    localidad, cp, fecha_alta, cuota_socio, cuota_familiar, paga_banco, codigo_banco,
    cuenta, digito_control)
    values ( 'J1010','10101010','JOSE','ALVAREZ CASTRO',
    'C/ OÑON, 23 2º','MIERES','33600',to_date('01/04/1995','dd/mm/yyyy'),
    6000,null,'S',33334444,93375,42);

Insert into usuarios (num_socio, dni, nombre, apellidos, domicilio,
    localidad, cp, fecha_alta, cuota_socio, cuota_familiar, paga_banco)
    values ( 'J0011','11011011','PELAYO','ESLA CASARIEGO',
    'C/ LA PISTA, 14 1º','MIERES','33600',to_date('01/05/1997','dd/mm/yyyy'),
    8000,null,'N');
```

```
Insert into usuarios (num_socio, dni, nombre, apellidos, domicilio,
    localidad, cp, fecha_alta, cuota_socio, paga_banco, codigo_banco,
    cuenta, digito_control)
    values ( 'J0012','12121212','VICTOR','ALBA PRIETO',
    'C/ LA LILA, 49 2º','AVILES','33400',to_date('01/05/1998','dd/mm/yyyy'),
    8000,1800,'S',85002222,54679,32);

Insert into usuarios (num_socio, dni, nombre, apellidos, domicilio,
    localidad, cp, fecha_alta, cuota_socio, paga_banco, codigo_banco,
    cuenta, digito_control)
    values ( 'J0013','13131313','LUZ','CUETO ARROYO',
    'C/ MAYOR, 91 5º','AVILES','33400',to_date('01/06/1995','dd/mm/yyyy'),
    7000,2700,'S',22223333,66785,32);

Insert into usuarios (num_socio, dni, nombre, apellidos, domicilio,
    localidad, cp, fecha_alta, cuota_socio, paga_banco, codigo_banco,
    cuenta, digito_control)
    values ( 'J0014','14141414','MARIO','FERNANDEZ VEGA',
    'C/ PEZ, 19 2º','AVILES','33400',to_date('01/04/1998','dd/mm/yyyy'),
    3000,null,'S',33334444,3354679,53);

Insert into actividades values ('G000001', 'GIMNASIA DE MANTENIMIENTO', 1000);

Insert into actividades values ('G000002','GIMNASIA RITMICA',800);

Insert into actividades values ('AM00001','JUDO',1200);

Insert into actividades values ('AM00002','KARATE',1100);

Insert into actividades values ('N000001','NATACION 1',900);

Insert into actividades values ('N000002','NATACION 2',1000);

Insert into actividades values ('G000003','MUSCULACION',1300);

Insert into actividades_usuarios (codigo_actividad, codigo_usuario, fecha_alta) values (
'G000001','A1111',to_date('30/03/1999','dd/mm/yyyy'));

Insert into actividades_usuarios (codigo_actividad, codigo_usuario, fecha_alta) values (
'AM00002','A1111',to_date('30/03/1999','dd/mm/yyyy'));

Insert into actividades_usuarios (codigo_actividad, codigo_usuario, fecha_alta) values (
'AM00001','A2222',to_date('30/03/1999','dd/mm/yyyy'));

Insert into actividades_usuarios (codigo_actividad, codigo_usuario, fecha_alta) values (
'N000001','A2222',to_date('30/03/1999','dd/mm/yyyy'));

Insert into actividades_usuarios (codigo_actividad, codigo_usuario, fecha_alta) values (
'N000002','I2222',to_date('01/03/1999','dd/mm/yyyy'));

Insert into actividades_usuarios (codigo_actividad, codigo_usuario, fecha_alta) values (
'AM00002','I2222',to_date('11/02/1999','dd/mm/yyyy'));

Insert into pagos (codigo_usuario, numero_mes,cuota) values ('A1111',1,5000);

Insert into pagos (codigo_usuario, numero_mes,cuota) values ('A1111',2,5000);

Insert into pagos (codigo_usuario, numero_mes,cuota) values ('A1111',3,5000);

Insert into pagos (codigo_usuario, numero_mes,cuota) values ('A1111',4,5500);

Insert into pagos (codigo_usuario, numero_mes,cuota) values ('A1111',5,5500);

Insert into pagos (codigo_usuario, numero_mes,cuota) values ('A1111',6,5000);
```

```
Insert into pagos (codigo_usuario, numero_mes,cuota) values ('A2222',1,7800);
Insert into pagos (codigo_usuario, numero_mes,cuota) values ('A2222',2,7800);
Insert into pagos (codigo_usuario, numero_mes,cuota) values ('A2222',5,7800);
Insert into pagos (codigo_usuario, numero_mes,cuota) values ('A2222',6,8000);
Insert into pagos (codigo_usuario, numero_mes,cuota) values ('A2222',3,7800);
Insert into pagos (codigo_usuario, numero_mes,cuota) values ('A2222',4,9000);
Insert into pagos (codigo_usuario, numero_mes,cuota) values ('I1111',5,4500);
Insert into pagos (codigo_usuario, numero_mes,cuota) values ('I1111',7,5000);
Insert into pagos (codigo_usuario, numero_mes,cuota) values ('I1111',1,3456);
Insert into pagos (codigo_usuario, numero_mes,cuota) values ('A2222',7,8000);
Insert into pagos (codigo_usuario, numero_mes,cuota) values ('A3333',2,4500);
Insert into pagos (codigo_usuario, numero_mes,cuota) values ('I2222',1,5000);
Insert into pagos (codigo_usuario, numero_mes,cuota) values ('I2222',2,5000);
Insert into pagos (codigo_usuario, numero_mes,cuota) values ('I2222',3,5500);
Insert into pagos (codigo_usuario, numero_mes,cuota) values ('I2222',4,5500);
Insert into pagos (codigo_usuario, numero_mes,cuota) values ('I2222',6,6000);
Insert into pagos (codigo_usuario, numero_mes,cuota) values ('J1111',1,1000);
Insert into pagos (codigo_usuario, numero_mes,cuota) values ('J1111',2,10000);
Insert into pagos (codigo_usuario, numero_mes,cuota) values ('J1111',4,10000);
Insert into pagos (codigo_usuario, numero_mes,cuota) values ('J1111',5,10000);
Insert into pagos (codigo_usuario, numero_mes,cuota) values ('J1111',12,10000);
Insert into pagos (codigo_usuario, numero_mes,cuota) values ('J2222',12,8000);
Insert into pagos (codigo_usuario, numero_mes,cuota) values ('J2222',1,1000);
Insert into pagos (codigo_usuario, numero_mes,cuota) values ('J2222',3,10000);
Insert into pagos (codigo_usuario, numero_mes,cuota) values ('J2222',11,10000);
Insert into pagos (codigo_usuario, numero_mes,cuota) values ('J3333',1,1000);
Insert into pagos (codigo_usuario, numero_mes,cuota) values ('J3333',2,2000);
Insert into pagos (codigo_usuario, numero_mes,cuota) values ('J3333',3,3000);
Insert into pagos (codigo_usuario, numero_mes,cuota) values ('J3333',4,4000);
Insert into pagos (codigo_usuario, numero_mes,cuota) values ('J3333',5,500);
Insert into pagos (codigo_usuario, numero_mes,cuota) values ('J3333',11,800);
Insert into pagos (codigo_usuario, numero_mes,cuota) values ('J3333',12,1500);
```

Administración de sistemas gestores de Bases de datos Ejercicios

Actividades complementarias (Capitulo 8)

1. Construye un bloque PL/SQL que escriba el texto 'Hola'.

```
BEGIN
DBMS_OUTPUT.PUT_LINE ('Hola mundo');
END;
```

2. ¿Que hace el siguiente bloque PL/SQL?

```
DECLARE
        v_num NUMBER;
BEGIN
        SELECT count(*) INTO v_num
        FROM productos;
DBMS_OUTPUT.PUT_LINE (v_num);
END;
/
```

Primero declara la variable v_num
Segundo ejecuta la instrucción SELECT ,cuenta las filas de la tabla y mete el resultado en v_num
Tercero muestra en pantalla el contenido de la variable v_num

3. Introduce el bloque anterior desde SQL*Plus y guárdalo en un fichero.

```
DECLARE
        v_num NUMBER;
BEGIN
        SELECT count(*) INTO v_num
        FROM productos;
DBMS_OUTPUT.PUT_LINE (v_num);
END;
.
```

4. Ejecuta la orden SELECT especificada en el bloque anterior desde SQL*PLUS sin la clausula INTO.

```
SELECT count(*) v_num
        FROM productos;
```

V_NUM
8

5. Carga y ejecuta el bloque de nuevo, y comprueba que el resultado aparece en pantalla.

```
8

Statement processed.
```

5.

6. Escribe desde SQL*Plus el ejemplo numero 1 del epígrafe "Uso de subprogramas almacenados" y prueba a ejecutarlo con distintos valores. (ejemplo pag 227)
dado un numero de producto que nos de el precio.

```
CREATE OR REPLACE
PROCEDURE preciocodigo (numproducto NUMBER)
AS
        v_precio NUMBER;
BEGIN
        SELECT precio_actual INTO v_precio
        FROM productos
        WHERE producto_no = numproducto;
        DBMS_OUTPUT.PUT_LINE (v_precio);
EXCEPTION
        WHEN NO_DATA_FOUND THEN
                DBMS_OUTPUT.PUT_LINE ('No encontrado precio');

END preciocodigo;
/
```

consulta:

```
BEGIN
preciocodigo(10);
END;
```

resultado.

```
1050
Statement processed.
```

7. Identifica en el ejemplo numero 2 del epígrafe "Uso de subprogramas almacenados" los siguientes elementos: (ejercicio 7.2 de la pagina 231)

- La cabecera del procedimiento.

```
CREATE OR REPLACE
        PROCEDURE modificar_precio_producto (numproducto NUMBER, nuevoprecio NUMBER)
```

- El nombre del procedimiento.
modificar_precio_producto

- Los parámetros del procedimiento.
numproducto **NUMBER**, nuevoprecio **NUMBER**

- Las variables locales.
v_precioant **NUMBER**(5)

- El comienzo y el final del bloque PL/SQL.

```
AS
…...
END;
```

- El comienzo y el final de las secciones declarativa, ejecutable y de gestión de excepciones.
Sección declarativa:

```
v_precioant NUMBER(5);
```

Sección ejecutable

```
SELECT precio_actual INTO v_precioant
        FROM productos
        WHERE producto_no = numproducto;
    IF (v_precioant * 0.20) > (nuevoprecio – v_precioant) THEN
        UPDATE productos SET precio_actual = nuevoprecio
        WHERE producto_no = numproducto;
    ELSE
        DBMS_OUTPUT.PUT_LINE ('Error, modificacion supera 20%');
    END IF;
```

Sección excepciones

```
WHEN NO_DATA_FOUND THEN
    DBMS_OUTPUT.PUT_LINE ('No encontrado producto ' || numproducto);
```

- ¿Que hace la clausula INTO?
Mete el contenido de **precio_actual** dentro de la variable **v_precioant**

- ¿Que hace WHEN NO_DATA_FOUND?
Cuando no se han encontrado datos en la consulta realiza una acción (normalmente presentar un mensaje de aviso que se ha predefinido)

- ¿Por que no tiene la clausula DECLARE? ¿Que tiene en su lugar?
Porque es un procedimiento no un bloque anónimo, en su lugar tiene AS donde se declaran las variables.

Ejercicio 5 Tema 8 Pag.228 (Bajo Terminal):
El programa solicita la introducción de un numero de cliente y visualiza el nombre del cliente con el numero introducido.

```
DECLARE
    v_nom CLIENTES.NOMBRE%TYPE; --(Ejemplo de uso de %TYPE)
BEGIN
    SELECT nombre INTO v_nom
    FROM clientes
    WHERE CLIENTE_NO=&vn_cli;
DBMS_OUTPUT.PUT_LINE (v_nom);
END;
.
```

```
SQL> RUN
  1  DECLARE
  2  v_nom CLIENTES.NOMBRE%TYPE;
  3  BEGIN
  4  SELECT nombre INTO v_nom
  5  FROM clientes
  6  WHERE CLIENTE_NO=&vn_cli;
  7  DBMS_OUTPUT.PUT_LINE(v_nom);
  8* END;
Introduzca un valor para vn_cli: 102
antiguo   6: WHERE CLIENTE_NO=&vn_cli;
nuevo   6: WHERE CLIENTE_NO=102;

Procedimiento PL/SQL terminado correctamente.
```

Ejercicio 6 Tema 8 Pag.229:
Procedimiento sencillos para consultas de datos de un cliente.
Creación procedimiento:

```
CREATE OR REPLACE
PROCEDURE ver_depart (numdepart NUMBER)
AS
        v_dnombre VARCHAR2(14);
        v_localidad VARCHAR2(14);
BEGIN
        SELECT dnombre, Localidad INTO v_dnombre, v_localidad
        FROM DEPARTAMENTOS
        WHERE dep_no = numdepart;
        DBMS_OUTPUT.PUT_LINE ('Num depart: ' || numdepart || ' * Nombre dep: ' ||
        v_dnombre || ' * Localidad: ' || v_localidad);
EXCEPTION
        WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE ('No encontrado departamento ');
END ver_depart;
/
```

Llamada al procedimiento:

```
BEGIN
ver_depart(30);
END;
```

Resultado procedimiento:

```
Num depart: 30 * Nombre dep: VENTAS * Localidad: MADRID

Statement processed.
```

Vía terminal:

```
SQL> set serveroutput on
SQL> execute ver_depart (20)
Num depart: 20 * Nombre dep: INVESTIGACION * Localidad: VALENCIA

Procedimiento PL/SQL terminado correctamente.
```

Ejercicio 7.1 Tema 8 Pag.231:
Visualización del precio de un producto cuyo numero de producto se pasa como parámetro.

```
CREATE OR REPLACE
        PROCEDURE ver_precio (v_num_producto NUMBER)
AS
        v_precio NUMBER;
BEGIN
        SELECT precio_actual INTO v_precio
        FROM productos
        WHERE producto_no = v_num_producto;
        DBMS_OUTPUT.PUT_LINE ('Precio = ' || v_precio);
END ;
/
```

Llamada procedimiento,Vía terminal:

```
SQL> SET SERVEROUTPUT ON
SQL> EXECUTE VER_PRECIO(50);
Precio = 1050

Procedimiento PL/SQL terminado correctamente.
```

Llamada procedimiento, entorno gráfico:

```
BEGIN
ver_precio(50);
END
;
```

Resultado procedimiento:

```
Precio = 1050
Statement processed.
```

Ejercicio 7.2 Tema 8 Pag.231:
Procedimiento que modifica el precio de un producto, pasandole el numero de producto y el nuevo precio, el procedimiento comprobara que la variación de precio no supera el 20%.

```
CREATE OR REPLACE
        PROCEDURE modificar_precio_producto (numproducto NUMBER, nuevoprecio NUMBER)
AS
        v_precioant NUMBER(5);
BEGIN
        SELECT precio_actual INTO  v_precioant
                FROM productos
                WHERE producto_no = numproducto;
        IF ( v_precioant * 0.20) > ( nuevoprecio – v_precioant) THEN
                UPDATE productos SET precio_actual =  nuevoprecio
                WHERE producto_no = numproducto;
        ELSE
                DBMS_OUTPUT.PUT_LINE ('Error, modificacion supera 20%');
        END IF;
EXCEPTION
        WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE ('No encontrado producto ' || numproducto);
END  modificar_precio_producto;
/
```

Llamada procedimiento,Vía terminal:

```
SET SERVEROUTPUT ON
EXECUTE modificar_precio_producto(60,300);
```

Nota: Este procedimiento vía entorno gráfico da error hay que hacerlo todo vía terminal.

Ejercicio 7.3 Tema 8 Pag.232:
Función que devuelve el valor con IVA de una cantidad que se pasara como primer parámetro, la función también podrá recoger un segundo parámetro opcional, que sera el tipo de IVA, siendo el valor por defecto 16.

```
CREATE OR REPLACE
        FUNCTION con_iva (cantidad NUMBER, tipo NUMBER DEFAULT 16)
        RETURN NUMBER
AS
        v_resultado NUMBER (10,2) DEFAULT 0;
BEGIN
        v_resultado := cantidad * (1 + (tipo/100));
        RETURN (v_resultado);
END  con_iva;
/
```

Llamada al procedimiento entorno gráfico:

```
BEGIN
DBMS_OUTPUT.PUT_LINE(con_iva(200));
END
;
```

resultado:

```
232
```

Tablas utilizadas en los ejercicios

DEPARTAMENTOS

DEP_NO Number(2)	**DNOMBRE** Varchar2(14)	**LOCALIDAD** Varchar2(10)
10	CONTABILIDAD	BARCELONA
20	INVESTIGACION	VALENCIA
30	VENTAS	MADRID
40	PRODUCCION	SEVILLA

EMPLEADOS

EMP_NO Number(4)	**APELLIDO** Varchar2(8)	**OFICIO** Varchar2(10)	**DIRECTOR** Number(4)	**FECHA_ALTA** Date	**SALARIO** Number (6)	**COMISION** Number(6)	**DEP_NO** Number(2)
7499	ALONSO	VENDEDOR	7698	20-FEB-1990	1500	390	30
7521	LOPEZ	EMPLEADO	7782	22-FEB-1991	1625		10
7654	MARTIN	VENDEDOR	7698	28-SEP-1991	1600	1020	30
7698	GARRIDO	DIRECTOR	7839	01-MAY-1991	3005		30
7782	MARTINEZ	DIRECTOR	7839	09-JUN-1991	2885		10
7876	GIL	ANALISTA	7782	06-JUN-1993	1430		20
7839	REY	PRESIDENTE		17-NOV-1991	4100		10
7844	CALVO	VENDEDOR	7698	08-SEP-1991	1350	0	30
7900	JIMENEZ	EMPLEADO	7788	24-MAR-1993	3000		20

CLIENTES

CLIENTE_NO Number(4)	**NOMBRE** Varchar2(25)	**LOCALIDAD** Varchar2(14)	**VENDEDOR_NO** Varchar2(14)	**DEBE** Number(9)	**HABER** Number(9)	**LIMITE_CREDITO** Number(9)
101	DISTRIBUCIONES GOMEZ	MADRID	7499	0	0	5000
102	LOGITRONICA S.L.	BARCELONA	7654	0	0	5000
103	INDUSTRIAS LACTEAS S.A.	LAS ROZAS	7844	0	0	10000
104	TALLERES ESTESO S.A.	SEVILLA	7654	0	0	5000
105	EDICIONES SANZ	BARCELONA	7499	0	0	5000
106	SIGNOLOGIC S.A.	MADRID	7654	0	0	5000
107	MARTIN Y ASOCIADOS S.L.	ARAVACA	7844	0	0	10000
108	MANUFACTURAS ALI S.A.	SEVILLA	7654	0	0	5000

PRODUCTOS

PRODUCTO_NO Number(4)	**DESCRIPCION** Varchar2(30)	**PRECIO_ACTUAL** Number(8)	**STOCK_DISPONIBLE** Number(9)
10	MESA DESPACHO MOD. GAVIOTA	550	50
20	SILLA DIRECTOR MOD. BUFALO	670	25
30	ARMARIO NOGAL DOS PUERTAS	460	20
40	MESA MODELO UNION	340	15
50	ARCHIVADOR CEREZO	1050	20
60	CAJA SEGURIDAD MOD B222	280	15
70	DESTRUCTORA DE PAPEL A3	450	25
80	MODULO ORDENADOR MOD. ERGOS	550	25

PEDIDOS

PEDIDO_NO Number(4)	**PRODUCTO_NO** Number(4)	**CLIENTE_NO** Number(4)	**UNIDADES** Number(4)	**FECHA_PEDIDO** Date
1000	20	103	3	22-SEP-2010
1001	50	106	2	22-SEP-2010
1002	10	101	4	23-SEP-2010
1003	20	105	4	26-SEP-2010
1004	40	106	8	28-SEP-2010
1005	30	105	2	28-SEP-2010
1006	70	103	3	02-OCT-2010
1007	50	101	2	02-OCT-2010
1008	10	106	6	04-OCT-2010
1009	20	105	2	04-OCT-2010
1010	40	102	3	05-OCT-2010
1011	30	106	2	07-OCT-2010
1012	10	105	3	10-OCT-2010
1013	30	106	2	16-OCT-2010
1014	20	101	4	18-OCT-2010
1015	70	105	4	18-OCT-2010
1016	30	106	7	22-OCT-2010
1017	20	105	6	02-NOV-2010

Actividades complementarias (Capitulo 9 pag.271)

1. Escribe un procedimiento que reciba dos numero y visualice una suma.

```
CREATE OR REPLACE
PROCEDURE suma (num1 NUMBER, num2 NUMBER)
AS
        suma NUMBER(2);
BEGIN
        suma:=num1+num2;
        DBMS_OUTPUT.PUT_LINE ('La suma de '||num1||' con '||num2||' es: '||suma);
EXCEPTION
        WHEN TOO_MANY_ROWS THEN
        RAISE_APPLICATION_ERROR (-20000, 'ERROR demasiados datos');
END;
/
```

Ejecución por entorno gráfico

```
BEGIN
suma(10,5);
END;
```

El resultado:
La suma de 10 con 5 es: 15

Ejecución por consola:

```
SET SERVEROUTPUT ON
EXECUTE suma(10,5);
```

2. Codifica un procedimiento que reciba una cadena y la visualice al revés.

```
CREATE OR REPLACE
PROCEDURE alreves (cadena VARCHAR2)
AS
        r_cadena VARCHAR2(25);
BEGIN
        FOR i IN REVERSE 1..LENGTH (cadena)
        LOOP
        r_cadena := r_cadena || SUBSTR (cadena,i,1);
        END LOOP;
        DBMS_OUTPUT.PUT_LINE (r_cadena);
EXCEPTION
        WHEN TOO_MANY_ROWS THEN
        RAISE_APPLICATION_ERROR (-20000, 'ERROR demasiados datos');
END;
/
```

Ejecución por terminal:

```
execute alreves ('esto es una historia')
```

Resultado:
airotsih anu se otse

Ejecución por entorno gráfico:

```
BEGIN alreves ('esto es una historia');
END;
```

Procedimiento PL/SQL terminado correctamente.

3. Reescribe el código de los ejercicios anteriores para convertirlos en funciones que retornen los valores que mostraban los procedimientos.

#Ejercicio 1 en función

```
CREATE OR REPLACE
FUNCTION suma (num1 NUMBER, num2 NUMBER)
RETURN REAL
AS
        sumar NUMBER(2);
BEGIN
        sumar:=num1+num2;
        RETURN sumar;
EXCEPTION
        WHEN TOO_MANY_ROWS THEN
        RAISE_APPLICATION_ERROR (-20000, 'ERROR demasiados datos');
END;/
```

Nota: antes de ejecutar nada por terminal hay que poner: SET SERVEROUTPUT ON

Llamada a la función con otro programa (por terminal)

```
EXECUTE DBMS_OUTPUT.PUT_LINE (suma (10,20));
```

Llamada a la función con otro programa (entorno gráfico)

```
BEGIN
DBMS_OUTPUT.PUT_LINE (suma (10,20));
END;/
```

#Ejercicio 2 en función

```
CREATE OR REPLACE
FUNCTION alrevess (cadena VARCHAR2)
RETURN VARCHAR2
AS
        r_cadena VARCHAR2(25);
BEGIN
        FOR i IN REVERSE 1..LENGTH (cadena)
        LOOP
        r_cadena := r_cadena || SUBSTR (cadena,i,1);
        END LOOP;
        RETURN r_cadena;
EXCEPTION
        WHEN TOO_MANY_ROWS THEN
        RAISE_APPLICATION_ERROR (-20000, 'ERROR demasiados datos');
END;/
```

Llamada a la función con otro programa (por terminal)

```
EXECUTE DBMS_OUTPUT.PUT_LINE (alrevess ('programa'));
```

Llamada a la función con otro programa (entorno gráfico)

```
BEGIN
DBMS_OUTPUT.PUT_LINE (alrevess ('programa'));
END;/
```

4. Escribe una función que reciba una fecha y devuelva el año, en numero, correspondiente a esa fecha.

```
CREATE OR REPLACE
FUNCTION fecnum (fecha DATE)
RETURN NUMBER
AS
        num NUMBER;
BEGIN
        num:=TO_NUMBER(TO_CHAR (fecha, 'yyyy'));
        RETURN num;
EXCEPTION
        WHEN TOO_MANY_ROWS THEN
        RAISE_APPLICATION_ERROR (-20000, 'ERROR demasiados datos');
END;/
```

Para probar en entorno gráfico:

```
BEGIN
DBMS_OUTPUT.PUT_LINE (fecnum ('25/07/2012'));
END;/
```

5. Escribe un bloque PL/SQL que haga uso de la función anterior.

```
CREATE OR REPLACE
PROCEDURE fecha (fecha DATE)
AS
        num NUMBER;
BEGIN
        num:=fecnum (fecha);
        DBMS_OUTPUT.PUT_LINE ('La fecha: ' || fecha || ' pertenece al año: ' || num);
EXCEPTION
        WHEN TOO_MANY_ROWS THEN
        RAISE_APPLICATION_ERROR (-20000, 'ERROR demasiados datos');
END;
/
```

Para probar en entorno gráfico:

```
BEGIN
fecha ('25/07/2012');
END;/
```

6. Desarrolla una función que devuelva el numero de años completos que hay entre dos fechas que se pasan como parámetros.

```
CREATE OR REPLACE
FUNCTION numanio (fecha1 DATE, fecha2 DATE)
RETURN NUMBER
AS
        num NUMBER;
BEGIN
        num:=ABS(ROUND(MONTHS_BETWEEN (fecha1, fecha2)/12));
        RETURN num;
EXCEPTION
        WHEN TOO_MANY_ROWS THEN
        RAISE_APPLICATION_ERROR (-20000, 'ERROR demasiados datos');
END;/
```

Para probar en entorno gráfico:

```
BEGIN
DBMS_OUTPUT.PUT_LINE (numanio ('25/07/2012', '12/03/2010'));
END;/
```

7. Escribe una función que, haciendo uso de la función anterior, devuelva los trienios que hay entre dos fechas (un trienio son tres años).

```
CREATE OR REPLACE
PROCEDURE trienios (fecha1 DATE, fecha2 DATE)
AS
        num NUMBER;
BEGIN
        num:=FLOOR(numanio (fecha1,fecha2)/3);
        DBMS_OUTPUT.PUT_LINE (num);
EXCEPTION
        WHEN TOO_MANY_ROWS THEN
        RAISE_APPLICATION_ERROR (-20000, 'ERROR demasiados datos');
END;
/
```

Para probar en entorno gráfico:

```
BEGIN
trienios ('25/07/2012', '12/03/2010');
END;/
```

8. codifica un procedimiento que reciba una lista de hasta cinco números y visualice su suma.

```
CREATE OR REPLACE
PROCEDURE suma5 (num1 NUMBER, num2 NUMBER, num3 NUMBER, num4 NUMBER, num5
NUMBER )
AS
        suma NUMBER(4);
BEGIN
        suma:=num1+num2+num3+num4+num5;
        DBMS_OUTPUT.PUT_LINE ('La suma de ' ||num1|| ' + ' ||num2|| ' + ' ||num3|| ' + ' ||num4|| ' + ' ||
num5 ||' es: ' || suma);
EXCEPTION
        WHEN TOO_MANY_ROWS THEN
        RAISE_APPLICATION_ERROR (-20000, 'ERROR demasiados datos');
END;
/

BEGIN
suma5(10,5, 10, 5,10);
END;
```

10. Codifica un procedimiento que permita borrar un empleado cuyo numero se pasara en la llamada.

```
CREATE OR REPLACE
PROCEDURE borraemp (num NUMBER)
AS
        nombre EMPLEADOS.APELLIDO%TYPE;
BEGIN
        SELECT APELLIDO INTO nombre FROM EMPLEADOS
                WHERE  EMP_NO=num;

        DELETE FROM EMPLEADOS
                WHERE EMP_NO=num;

        DBMS_OUTPUT.PUT_LINE ('Se ha borrado el empleado ' || nombre || ' con nº empleado: ' || num);
EXCEPTION
        WHEN TOO_MANY_ROWS THEN
        RAISE_APPLICATION_ERROR (-20000, 'ERROR demasiados datos');
        WHEN NO_DATA_FOUND THEN
        RAISE_APPLICATION_ERROR (-20000, 'ERROR no hay datos');
END;
/

BEGIN
borraemp(1056);
END;
```

11. Escribe un procedimiento que modifique la localidad de un departamento. El procedimiento recibirá como parámetro el numero del departamento y la nueva localidad.

```
CREATE OR REPLACE
PROCEDURE cambiar_loc (num_depart NUMBER, nueva_loc VARCHAR2)
AS
        v_anterior_loc departamentos.localidad%TYPE;

BEGIN
        SELECT localidad INTO v_anterior_loc
        FROM departamentos
        WHERE dep_no = num_depart;

        UPDATE departamentos SET localidad = nueva_loc
        WHERE dep_no = num_depart;
        DBMS_OUTPUT.PUT_LINE ('Nº Departamento: ' || num_depart || ', localidad Anterior: ' ||
v_anterior_loc || ', localidad nueva : ' || nueva_loc);

EXCEPTION
        WHEN TOO_MANY_ROWS THEN
        RAISE_APPLICATION_ERROR (-20000, 'ERROR demasiados datos');
        WHEN NO_DATA_FOUND THEN
        RAISE_APPLICATION_ERROR (-20000, 'ERROR no hay datos');
        WHEN OTHERS THEN
        RAISE_APPLICATION_ERROR (-20000, 'ERROR en la aplicacion');

END cambiar_loc;
/
```

Ejecución por terminal:

```
set serveroutput on
execute cambiar_loc (10, 'SEVILLA');
```

Resultado:

```
Nº Departamento: 10, localidad Anterior: BARCELONA, localidad nueva : SEVILLA

Procedimiento PL/SQL terminado correctamente.
```

12. Visualiza todos los procedimientos y funciones del usuario almacenados en la base de datos y su situación (valid o invalid).

Ejercicios ejemplo del capitulo:

1. Supongamos que pretendemos modificar el salario de un empleado especificado en función del numero de empleados que tiene a su cargo: (ejercicio 1 de la pagina 250)

- Si no tiene ningún empleado a su cargo l SUBIDA SERA DE 50€.
- Si tiene 1 empleado la subida sera de 80€.
- Si tiene 2 empleados la subida sera de 100€.
- Si tiene mas de 3 empleados la subida sera de 110€.
- Ademas si el empleado es PRESIDENTE se incrementara el salario en 30€.

```
DECLARE
        v_empleado_no NUMBER (4,0);             -- Empleado al que subir salario
        v_c_empleados NUMBER (2);               -- Cantidad empleados dependen de el
        v_aumento NUMBER (7) DEFAULT 0;         -- Importe que vamos a aumentar
        v_oficio VARCHAR2 (10);

BEGIN
        v_empleado_no := &vt_empno;             -- variable de sustitucion, lee nº empleado

        SELECT oficio INTO v_oficio FROM empleados
        WHERE emp_no = v_empleado_no;

        IF v_oficio = 'PRESIDENTE' THEN         -- Alternativa simple
        v_aumento := 30;
        END IF;

        SELECT COUNT(*) INTO v_c_empleados FROM empleados
        WHERE director = v_empleado_no;

        IF v_c_empleados = 0 THEN               -- Alternativa multiple
        v_aumento := v_aumento + 50;
        ELSIF  v_c_empleados = 1 THEN
        v_aumento := v_aumento + 80;
        ELSIF  v_c_empleados = 2 THEN
        v_aumento := v_aumento + 100;
        ELSE
        v_aumento := v_aumento + 110;
        END IF;

        UPDATE empleados SET salario = salario + v_aumento
        WHERE emp_no = v_empleado_no;
        DBMS_OUTPUT.PUT_LINE(v_aumento);

END ;
/
```

Con CASE de búsqueda:

```
CASE
WHEN v_c_empleados = 0 THEN
        v_aumento := v_aumento + 50;
WHEN v_c_empleados = 1 THEN
        v_aumento := v_aumento + 80;
WHEN v_c_empleados = 2 THEN
        v_aumento := v_aumento + 100;
ELSE
        v_aumento := v_aumento + 110;
END CASE;
```

Con CASE de comprobación:

```
CASE v_c_empleados
WHEN 0 THEN
        v_aumento := v_aumento + 50;
WHEN 1 THEN
        v_aumento := v_aumento + 80;
WHEN 2 THEN
        v_aumento := v_aumento + 100;
ELSE
        v_aumento := v_aumento + 110;
END CASE;
```

Este ejercicio hay que hacerlo por consola y activar primero que imprima las variables con:
set serveroutput on

Una vez introducido todas las lineas al dar enter pide un numero de usuario y da la solución.

```
Introduzca un valor para vt_empno: 7499
antiguo   7:    v_empleado_no := &vt_empno;        -- variable de sustitucion, lee nº empleado
nuevo   7:      v_empleado_no := 7499;             -- variable de sustitucion, leenº empleado
50

Procedimiento PL/SQL terminado correctamente.
```

2. Supongamos que deseamos analizar una cadena que contiene los dos apellidos para guardar el primer apellido en una variable a la que llamaremos v_1apel . Entendemos que el primer apellido termina cuando encontremos cualquier carácter distinto de los alfabéticos (en mayúsculas): (ejercicio 2 de la pagina 252)

```
DECLARE
        v_apellidos VARCHAR2(25);
        v_1apel VARCHAR2(25);
        v_caracter CHAR;
        v_posicion INTEGER := 1;
BEGIN
        v_apellidos := '&vs_apellidos';

        v_caracter := SUBSTR(v_apellidos, v_posicion,1);
        WHILE v_caracter BETWEEN 'A' AND 'Z' LOOP
                v_1apel := v_1apel || v_caracter;
                v_posicion := v_posicion + 1;
                v_caracter := SUBSTR (v_apellidos,v_posicion,1);
        END LOOP;
        DBMS_OUTPUT.PUT_LINE('1er Apellido: '||v_1apel||'*');
END;
/
```

Este ejercicio hay que hacerlo por consola y activar primero que imprima las variables con:
set serveroutput on

En la ejecución pide el nombre y da el siguiente resultado

```
Introduzca un valor para vs_apellidos: GIL MORENO
antiguo   7:    v_apellidos := '&vs_apellidos';
nuevo   7:      v_apellidos := 'GIL MORENO';
1er Apellido: GIL*

Procedimiento PL/SQL terminado correctamente.
```

3. Vamos a construir de dos maneras un bloque PL/SQL que escriba la cadena "A donde vas" al revés mediante un for: (ejercicio 3 de la pagina 256).

```
DECLARE
        r_cadena VARCHAR2(15);
BEGIN
        FOR i IN REVERSE 1..LENGTH ('A DONDE VAS')
        LOOP
        r_cadena := r_cadena || SUBSTR ('A DONDE VAS',i,1);
        END LOOP;
        DBMS_OUTPUT.PUT_LINE (r_cadena);
END;
/
```

Funciona por consola y en entorno gráfico.

```
SAV EDNOD A
Procedimiento PL/SQL terminado correctamente.
```

3. Vamos a construir de dos maneras un bloque PL/SQL que escriba la cadena "A donde vas" al revés mediante un WHILE: (ejercicio 3 de la pagina 256).

```
DECLARE
        r_cadena VARCHAR2(15);
        i BINARY_INTEGER;
BEGIN
        i := LENGTH ('A DONDE VAS') ;
        WHILE i >= 1 LOOP
        r_cadena := r_cadena || SUBSTR ('A DONDE VAS',i,1);
        i := i - 1;
        END LOOP;
        DBMS_OUTPUT.PUT_LINE (r_cadena);
END;
/
```

Funciona por consola y en entorno gráfico.

```
SAV EDNOD A

Procedimiento PL/SQL terminado correctamente.
```

En vez de meter el texto que lo pida (solo por consola):

```
DECLARE
        r_cadena VARCHAR2(25);
        texto VARCHAR2(25);
BEGIN
        texto:='&cadena';
        FOR i IN REVERSE 1..LENGTH (texto)
        LOOP
        r_cadena := r_cadena || SUBSTR (texto,i,1);
        END LOOP;
        DBMS_OUTPUT.PUT_LINE (r_cadena);
END;
/
```

el resultado es:

```
Introduzca un valor para cadena: esto es una prueba
antiguo   5: texto:='&cadena';
nuevo   5: texto:='esto es una prueba';
abeurp anu se otse

Procedimiento PL/SQL terminado correctamente.
```

4. Crearemos un procedimiento que reciba un numero de empleado y una cadena correspondiente a su nuevo oficio. El procedimiento deberá localizar el empleado, modificar su oficio y visualizar los cambios realizados: (ejercicio 4 de la pagina 259).

```
CREATE OR REPLACE
PROCEDURE cambiar_oficio (num_empleado NUMBER, nuevo_oficio VARCHAR2)
AS
        v_anterior_oficio empleados.oficio%TYPE;
BEGIN
        SELECT oficio INTO v_anterior_oficio
        FROM empleados
        WHERE emp_no = num_empleado;

        UPDATE empleados SET oficio = nuevo_oficio
        WHERE emp_no = num_empleado;
        DBMS_OUTPUT.PUT_LINE (num_empleado || ', Oficio Anterior: ' || v_anterior_oficio || ', Oficio
Nuevo : ' || nuevo_oficio);
END cambiar_oficio;
/
```

Ejecución por terminal:

```
set serveroutput on
execute cambiar_oficio (7900, 'DIRECTOR');
```

Resultado:

```
7900Oficio Anterior:EMPLEADOOficio Nuevo :DIRECTOR
Procedimiento PL/SQL terminado correctamente.
```

Escribe un procedimiento con funcionalidad similar al ejemplo anterior, que recibirá un numero de empleado y numero de departamento y asignara el empleado al departamento indicado en el segundo parámetro (este llama al procedimiento anterior).

```
CREATE OR REPLACE
PROCEDURE cam_ofi (v_apellido VARCHAR, nue_oficio VARCHAR2)
AS
        v_n_empleado empleados.emp_no%TYPE;
BEGIN
        SELECT emp_no INTO v_n_empleado
        FROM empleados
        WHERE apellido = v_apellido;
        cambiar_oficio (v_n_empleado, nue_oficio);

END cam_ofi;
/
```

Ejecución por terminal:

```
execute cam_ofi ('JIMENEZ','ANALISTA');
```

Ejecución por entorno gráfico:

```
BEGIN cam_ofi_ ('JIMENEZ','ANALISTA');
END;
```

resultado:

```
7900Oficio Anterior:EMPLEADOOficio Nuevo :ANALISTA
Procedimiento PL/SQL terminado correctamente.
```

Actividades propuestas:

4. Dado el siguiente procedimiento (Actividad 4 pagina 263)**:**

```
CREATE OR REPLACE
PROCEDURE crear_depart (v_num_dept departamentos.dep_no%TYPE,
        v_dnombre departamentos.dnombre%TYPE DEFAULT 'PROVISINAL',
        v_loc departamentos.localidad%TYPE DEFAULT 'PROVISINAL')
AS
BEGIN
        INSERT INTO departamentos
        VALUES (v_num_dept, v_dnombre, v_loc);
END crear_depart;
/
```

Indica cuales de las siguientes llamadas son correctas y cuales son incorrectas. En el caso de que sean incorrectas, escribe la llamada correcta usando usando la notación posicional, siempre que sea posible:

- execute crear_depart;
  - ✔ incorrecto. execute crear_depart (50, 'COMPRAS', 'VALENCIA');
    execute crear_depart (50);
- execute crear_depart (50);
  - ✔ correcto. Las otras dos opciones se rellenan por defecto con 'PROVISINAL'
- execute crear_depart ('COMPRAS');
  - ✔ incorrecto. execute crear_depart (50, 'COMPRAS');
- execute crear_depart (50, 'COMPRAS');
  - ✔ correcto.
- execute crear_depart ('COMPRAS', 50);
  - ✔ incorrecto. execute crear_depart (50, 'COMPRAS');
- execute crear_depart ('COMPRAS', 'VALENCIA');
  - ✔ incorrecto. execute crear_depart (50, 'COMPRAS', 'VALENCIA');
- execute crear_depart (50, 'COMPRAS', 'VALENCIA');
  - ✔ correcto.
- execute crear_depart ('COMPRAS', 50, 'VALENCIA');
  - ✔ incorrecto. execute crear_depart (50, 'COMPRAS', 'VALENCIA');
- execute crear_depart ('VALENCIA', 'COMPRAS');
  - ✔ incorrecto. execute crear_depart (50, 'COMPRAS', 'VALENCIA');
- execute crear_depart ('VALENCIA', 50);
  - ✔ incorrecto. execute crear_depart (50, 'COMPRAS' , 'VALENCIA');

Nota: La columna localidad esta configurada en VARCHAR2(10) y PROVISIONAL tiene 11 por lo que se cambia por PROVISINAL para que cumpla las especificaciones de 10 caracteres.

Ejercicios adicionales:

Haz un programa que visualice los números pares entre el 2 al 50.

Bloque usando FOR (lista en ascendente):

```
DECLARE
        i NUMBER (2);
BEGIN
        FOR i IN 2..50 LOOP
        IF MOD (i,2)=0 THEN
        DBMS_OUTPUT.PUT_LINE (i);
        END IF;
        END LOOP;
END;
/
```

Bloque usando WHILE (lista en descendente):

```
DECLARE
        i NUMBER (2);
BEGIN
        i:= 50;
        WHILE i>1 LOOP
        IF MOD (i,2)=0 THEN
        DBMS_OUTPUT.PUT_LINE (i);
        END IF;
        i:= i - 1;
        END LOOP;
END;
/
```

Bloque usando LOOP (lista en ascendente):

```
DECLARE
i NUMBER(2):=1;
BEGIN
LOOP
IF MOD(i,2)=0 THEN
DBMS_OUTPUT.PUT_LINE ('Contador: '||i);
END IF;
i:=i + 1;
EXIT WHEN i>=50;
END LOOP;
END;
```

Programa usando FOR:

```
CREATE OR REPLACE
PROCEDURE numpar (num1 NUMBER, num2 NUMBER)
AS
        i NUMBER (2);
BEGIN
        FOR i IN num1..num2 LOOP
        IF MOD (i,2)=0 THEN
        DBMS_OUTPUT.PUT_LINE (i);
        END IF;
        END LOOP;
END;
/
```

Programa usando WHILE:

```
CREATE OR REPLACE
PROCEDURE numpar (num1 NUMBER, num2 NUMBER)
AS
        i NUMBER (2);
BEGIN
        i:= num2;
        WHILE i>num1 LOOP
        IF MOD (i,2)=0 THEN
        DBMS_OUTPUT.PUT_LINE (i);
        END IF;
        i:= i - 1;
        END LOOP;
END;
/
```

En entorno gráfico (que liste los números pares entre 2 y 50):

```
BEGIN        numpar (2,50);        END;
```

Por consola: `EXECUTE (2,50);`

EJERCICIOS DE REPASO

1. **Realizar un procedimiento que reciba 2 nº y los reste. Sea cual sea el orden de la resta el resultado sea positivo**

```
CREATE OR REPLACE
PROCEDURE restar(num1 NUMBER, num2 NUMBER)
AS
        resta NUMBER(5);
BEGIN
        resta:=ABS(num1-num2);
        DBMS_OUTPUT.PUT_LINE ('La resta de los numeros introducidos es: ' || resta);
END restar;
/
```

Por terminal:
```
EXECUTE restar(7,6);
```

Entorno gráfico:
```
BEGIN restar(7,6); END;
```

2. **Versión 1. Realizar un procedimiento que reciba una cadena (letras y cualquier otro carácter). La visualice al revés y en el momento que detecte un carácter que no sea una letra nos dé un mensaje de error**

```
CREATE OR REPLACE
PROCEDURE alreves (texto VARCHAR2)
AS
        cadena VARCHAR2(15);
        caracter VARCHAR2(1);
BEGIN
        FOR i IN REVERSE 1..LENGTH(texto)
        LOOP
        cadena:=cadena||SUBSTR(texto,i,1);
        caracter:=(SUBSTR(texto,i,1));
                IF LOWER(caracter) NOT BETWEEN 'a' AND 'z' THEN
                DBMS_OUTPUT.PUT_LINE ('Error el caracter: ' || caracter || ' No es una letra');
                END IF;
        END LOOP;
        DBMS_OUTPUT.PUT_LINE(texto || ' al reves es: ' || cadena);
END alreves;
/
```

Por terminal:
```
EXECUTE alreves ('pr&ueba');
```

Entorno gráfico:
```
BEGIN alreves('pr&ueba'); END;
```

Nota: Tal como esta el IF la ñ la toma como que no es una letra, esta contemplando un abecedario sin ñ, seguramente use el abecedario ingles.

Versión 2. Realizar un procedimiento que reciba una cadena (letras y cualquier otro carácter). La visualice al revés y en el momento que detecte un carácter que no sea una letra nos dé un mensaje de error.

El mismo pero que en vez de imprimir los errores, indique error y se pare la ejecución del programa:

```
CREATE OR REPLACE
PROCEDURE alreves (texto VARCHAR2)
AS
        cadena VARCHAR2(15);
        caracter CHAR;
        no_es_letra EXCEPTION;
BEGIN
        FOR i IN REVERSE 1..LENGTH(texto)
        LOOP
        cadena:=cadena||SUBSTR(texto,i,1);
        caracter:=(SUBSTR(texto,i,1));
                IF LOWER(caracter) NOT BETWEEN 'a' AND 'z' THEN
                        RAISE no_es_letra;
                END IF;
        END LOOP;
        DBMS_OUTPUT.PUT_LINE(texto || ' al reves es: ' || cadena);
        EXCEPTION
        WHEN no_es_letra THEN DBMS_OUTPUT.PUT_LINE ('Error el caracter: ' || caracter || '
        No es una letra, introduce texto formado por letras');
END alreves;
/
```

Nota: La ñ la considera que no es letra, seguramente porque tome como referencia el alfabeto ingles.

3. **Realizar los 2 ejercicios anteriores pero realizando funciones**

Ejercicio 1 en función:

```
CREATE OR REPLACE
FUNCTION restar2 (num1 NUMBER, num2 NUMBER)
RETURN NUMBER
AS
        resta NUMBER(5);
BEGIN
        resta:=ABS(num1-num2);
        RETURN resta:
END restar2;
/

BEGIN DBMS_OUTPUT.PUT_LINE ('La resta de los numeros introducidos es: ' || restar2(7,6));
END;
```

Ejercicio 2 en función:

```
CREATE OR REPLACE
FUNCTION alreves2 (texto VARCHAR2)
RETURN VARCHAR2
AS
cadena VARCHAR2(15);
caracter CHAR;
BEGIN
FOR i IN REVERSE 1..LENGTH(texto)
LOOP
cadena:=cadena||SUBSTR(texto,i,1);
caracter:=(SUBSTR(texto,i,1));
IF LOWER(caracter) NOT BETWEEN 'a' AND 'z' THEN
DBMS_OUTPUT.PUT_LINE ('Error el caracter: ' || caracter || ' No es una letra');
END IF;
END LOOP;
RETURN cadena;
END alreves2;
/
```

Por terminal:
```
EXECUTE alreves2('prueba')
```

Entorno gráfico
```
BEGIN DBMS_OUTPUT.PUT_LINE(alreves2('prueba')); END;
```

4. **Realizar un procedimiento que de la tabla EMPLEADOS dado un n_emple me devuelva el apellido y su salario. De no encontrar el emple me dé un mensaje de error de 'Empleado no encontrado'**

```
CREATE OR REPLACE
PROCEDURE apesal (num NUMBER)
AS
        v_apellido EMPLEADOS.APELLIDO%TYPE;
        v_salario EMPLEADOS.SALARIO%TYPE;
BEGIN
        SELECT APELLIDO INTO v_apellido FROM EMPLEADOS WHERE EMP_NO=num;
        SELECT SALARIO INTO v_salario FROM EMPLEADOS WHERE EMP_NO=num;
        DBMS_OUTPUT.PUT_LINE ('El nº empleado: ' || num || ' corresponde a: ' || v_apellido || ' y
        su salario es: ' || v_salario) ;
EXCEPTION
        WHEN NO_DATA_FOUND THEN RAISE_APPLICATION_ERROR (-20000,
        'Empleado no encontrado');
END apesal;
/
```

Por terminal:
```
EXECUTE apesal(7499);
```

Entorno gráfico:
```
BEGIN apesal (7499); END;
```

5. **Queremos modificar el salario de un empleado en función del dpto. en que trabaje, de tal forma que a los:**
   a. **Empleados del dpto. 10 se les va a incrementar el salario en 50€**
   b. **Empleados del dpto. 20 se les va a incrementar el salario en 100€**
   c. **Si tabajan en cualquier otro dpto. se les incrementa en 150€**

El nº del emple lo introduciremos por teclado

```
DECLARE
        n_emple NUMBER(4);                      -- empleado al que subimos el salario
        aumento NUMBER (7) DEFAULT 0;           -- cantidad que le aumentamos, por defecto 0
        depart NUMBER(2);                       -- departamento al que pertenece
        sueldo NUMBER(6);                       -- salario actual del empleado
        n_sueldo NUMBER(6);                     -- nuevo salario del empleado

BEGIN
        n_emple:=&num_emp;                      -- pide por teclado el numero de empleado

        SELECT DEP_NO INTO depart FROM EMPLEADOS WHERE EMP_NO=n_emple;
                                                -- busco departamento asignado al empleado

        SELECT SALARIO INTO sueldo FROM EMPLEADOS WHERE EMP_NO=n_emple;
                                                -- busco salario asignado al empleado

        IF depart=10 THEN                       -- determino el aumento segun departamento
        aumento:=aumento+50;                    -- asigno a aumento la cantidad establecida
        ELSIF depart=20 THEN
        aumento:=aumento+100;
        ELSE
        aumento:=aumento+150;
        END IF;

        n_sueldo:=sueldo+aumento;               -- asigno el nuevo salario a la variable
        UPDATE empleados SET SALARIO = n_sueldo
        WHERE n_emple=EMP_NO;                   -- actualizo la tabla con el nuevo salario

        DBMS_OUTPUT.PUT_LINE ('El empleado: ' || n_emple || ' Pertenece al dpt: ' || depart || ' y
        tendra un incremento de: ' || aumento || ' euros' || ' su salario era de : ' || sueldo || ' y con el
        aumento es de: ' || n_sueldo);

EXCEPTION
WHEN NO_DATA_FOUND THEN RAISE_APPLICATION_ERROR (-20000, 'Empleado no
encontrado');

END;
/
```

Una vez introducido por terminal, pide nº empleado y ejecuta las instrucciones:

```
Introduzca un valor para num_emp: 7698
antiguo   9:    n_emple:=&num_emp;                -- pide por teclado el numero de empleado
nuevo   9:      n_emple:=7698;                -- pide por teclado el numero de empleado
El empleado: 7698 Pertenece al dpt: 30 y        tendra un incremento de: 150 euros su
salario era de : 3005 y con el  aumento es de: 3155
Procedimiento PL/SQL terminado correctamente.
```

Compruebo la excepción metiendo un empleado que no existe:

```
Introduzca un valor para num_emp: 3567
antiguo   7:    n_emple:=&num_emp;
nuevo   7:      n_emple:=3567;
DECLARE
*
ERROR en linea 1:
ORA-20000: Empleado no encontrado
ORA-06512: en linea 24
```

6. **Crear una función que aplique el IVA a unos importes con las siguientes características. A la función se le pasará una cantidad y un tipo de IVA**
   a. **Tipo de IVA 1 21%**
   b. **Tipo de IVA 2 8%**
   c. **Tipo de IVA 3 4%**

   Nos devolverá Importe: Cantidad+Cantidad por IVA
   Por defecto el tipo de IVA será del 21%

```
CREATE OR REPLACE
FUNCTION calciva (cantidad NUMBER, tipoiva NUMBER DEFAULT 1)
RETURN NUMBER
AS
        importe NUMBER(4);
BEGIN
        IF tipoiva=1 THEN
        importe:=cantidad+(cantidad*21/100);
                ELSIF  tipoiva=2 THEN
                importe:=cantidad+(cantidad*8/100);
                ELSIF  tipoiva=3 THEN
                importe:=cantidad+(cantidad*4/100);
                ELSE
                DBMS_OUTPUT.PUT_LINE ('Error en tipo de IVA, introduce 1, 2 o 3');
        END IF;
        RETURN importe;
END calciva;
/

BEGIN DBMS_OUTPUT.PUT_LINE (calciva(100,1)); END;
Resultado:
121
Statement processed.

BEGIN DBMS_OUTPUT.PUT_LINE (calciva(100,4)); END;
Resultado:
Error en tipo de IVA, introduce 1, 2 o 3
Statement processed.
```

7. **Crear una función para el calculo de la potencia fiscal de un vehículo. La formula para el calculo es:** $P_f = TN(C/N)^{0.6}$

donde:
P_f = Potencia fiscal
C = Cilindrada
N = Numero de cilindros
T = 0.008 para motores de 4 tiempos y 0.11 para motores de 2 tiempos

Formula expresada en PL/SQL podría quedar P_f = T*N*POWER((C/N), 0.6);

A la función se le pasaran la cilindrada en cm^3, opcionalmente el nº de cilindros (por defecto se asumen 4) y el tipo de motor (por defecto se asumen 4 tiempos, en caso contrario se pondrá 2). La función debe devolver la potencia fiscal.

Solución propuesta profesor:

```
CREATE OR REPLACE
FUNCTION pot_fiscal(cilindrada NUMBER, n_cilindros NUMBER DEFAULT 4, tipo_motor
NUMBER DEFAULT 4)
        RETURN REAL
IS
        pot_fiscal REAL;
        T NUMBER (3,2);
BEGIN
        IF tipo_motor <> 4 THEN
                tipo_motor:=2;
        END IF;
        CASE tipo_motor
                WHEN 4 THEN T:=0.08;
                WHEN 2 THEN T:=0.11;
        END CASE;
        pot_fiscal:=T*n_cilindros*POWER((cilindrada/n_cilindros), 0.6);
        RETURN pot_fiscal;
END;
```

Mi solución:

```
CREATE OR REPLACE
FUNCTION potencia(cilindrada NUMBER, cilindros NUMBER DEFAULT 4, tipo NUMBER
DEFAULT 4)
        RETURN NUMBER
AS
        P NUMBER;
        T NUMBER (3,2);
BEGIN
        IF tipo = 4 THEN
                T:=0.08;
        ELSIF tipo = 2 THEN
                T:=0.11;
        ELSE
        T:=0.11;
        END IF;
        P:=T*cilindros*POWER((cilindrada/cilindros), 0.6);
        RETURN P;
END ;
```

Ejercicios ejemplo del capitulo (Capitulo 12):

4. Crea un usuario con nombre USUARIO1, la clave la misma, El tablespace por omisión es USERS (ya que no se indica) al que se le asigna 500Kilobytes. El tablespace para trabajos temporales es TEMP (ya que tampoco de indica): (ejercicio 4 de la pagina 387)

SQL> **CONNECT** system/manager
Conectado.

SQL> **CREATE USER** usuario1 **IDENTIFIED BY** usuario1 **QUOTA** 500K **ON USERS**;

Usuario creado.

5. El usuario MILAGROS tiene una tabla de nombre TABLA1 que contiene la temperatura de un serie de ciudades. Concede a FRANCISCO los privilegios SELECT e INSERT en TABLA1: (ejercicio 5 de la pagina 390)

- Creo los usuarios con SYSTEM (CONNECT SYSTEM/'contraseña')
  - ✔ **CONNECT** system/manager
  - ✔ **CREATE USER** milagros **IDENTIFIED BY** milagros **QUOTA** 500K **ON USERS**;
  - ✔ **CREATE USER** francisco **IDENTIFIED BY** francisco **QUOTA** 500K **ON USERS**;

- Se da permiso a los usuarios de conexión
  - ✔ **GRANT CREATE SESSION TO** milagros;
  - ✔ **GRANT RESOURCE TO** milagros;
  - ✔ **GRANT CREATE SESSION TO** francisco;

- Se entra con el usuario milagros y se crea la tabla TABLA1
  - ✔ **CONNECT** milagros/milagros
  - ✔ **CREATE TABLE** TABLA1
    (
    NOM_CIUDAD **VARCHAR2**(10) ,
    TEMPERATURA **NUMBER**(2),
    CONSTRAINT NOM_CIUDAD_PK **PRIMARY KEY** (NOM_CIUDAD)
    );

- Se introducen valores a la tabla
  - ✔ **INSERT INTO** TABLA1 (NOM_CIUDAD,TEMPERATURA) **VALUES** ('BILBAO',20);
  - ✔ **INSERT INTO** TABLA1 (NOM_CIUDAD,TEMPERATURA) **VALUES** ('MADRID',15);
  - ✔ **INSERT INTO** TABLA1 (NOM_CIUDAD,TEMPERATURA) **VALUES** ('SEVILLA',35);
  - ✔ **INSERT INTO** TABLA1 (NOM_CIUDAD,TEMPERATURA) **VALUES** ('HUELVA',27);

- Se da permiso a francisco de SELECT e INSERT
  - ✔ **GRANT SELECT**, **INSERT ON** TABLA1 **TO** francisco;

- Se comprueba que francisco puede hacer SELECT y INSERT en la tabla de milagros
  - ✔ **CONNECT** francisco/francisco
  - ✔ **SELECT * FROM** milagros.TABLA1;
  - ✔ **INSERT INTO** milagros.TABLA1 (NOM_CIUDAD,TEMPERATURA) **VALUES** ('TERUEL',20);

- milagros concede a francisco todos los privilegios sobre TABLA1:
  - ✔ **CONNECT** milagros/milagros
  - ✔ **GRANT ALL ON** TABLA1 **TO** francisco;

- milagros concede todos los privilegios sobre TABLA1 a todos los usuarios, incluidos a los que se creen después de ejecutar esta orden.
  - ✔ **GRANT ALL ON** TABLA1 **TO PUBLIC**;

- milagros concede a juan sobre TABLA1 para que pueda modificar solo la columna temperatura, si intenta modificar las dos columnas no podrá, pero si puede modificar la de temperatura.
  - ✔ **CONNECT** system/manager
  - ✔ **CREATE USER** juan **IDENTIFIED BY** juan **QUOTA** 500K **ON USERS**;
  - ✔ **GRANT CREATE SESSION TO** juan ;
  - ✔ **CONNECT** milagros/milagros
  - ✔ **GRANT UPDATE** (TEMPERATURA) **ON** TABLA1 **TO** juan ;
  - ✔ **CONNECT** juan/juan
  - ✔ **UPDATE** milagros.TABLA1 **SET** TEMPERATURA=19 **WHERE** NOM_CIUDAD='TERUEL' ;

- Con la opción WITH GRANT OPTION milagros da permiso a francisco para que por ejemplo pueda hacer inserciones en la tabla y ademas pueda pasar este privilegio a otros usuarios.
  - ✔ **CONNECT** milagros/milagros
  - ✔ **GRANT INSERT ON** TABLA1 **TO** francisco **WITH GRANT OPTION** ;
  - ✔ **CONNECT** francisco/francisco
  - ✔ **INSERT INTO** milagros.TABLA1 (NOM_CIUDAD,TEMPERATURA) **VALUES** ('SORIA',18) ;
  - ✔ **GRANT INSERT ON** milagros.TABLA1 **TO** juan ;
  - ✔ **CONNECT** juan/juan
  - ✔ **INSERT INTO** milagros.TABLA1 (NOM_CIUDAD,TEMPERATURA) **VALUES** ('VIGO',17) ;
  - ✔ **CONNECT** milagros/milagros
  - ✔ **SELECT * FROM** TABLA1 ;

```
NOM_CIUDAD TEMPERATURA
---------- -----------
BILBAO              20
MADRID              15
SEVILLA             35
HUELVA              27
TERUEL              19
SORIA               18
VIGO                17

7 filas seleccionadas.
```

- Ver con que usuario se esta logeado :
  - ✔ **SHOW** USER

6. Cuando creamos un usuario tenemos que darle privilegios para que, como mínimo, pueda iniciar sesión en la base de datos. Creamos al usuario pedro y le damos el privilegio de crear sesión (CREATE SESSION)(ejercicio 6 de la pagina 393)

- Creo el usuario con SYSTEM y le doy el privilegio de CREATE SESSION
  - ✔ **CONNECT** system/manager
  - ✔ **CREATE USER** pedro **IDENTIFIED BY** pedro **QUOTA** 500K **ON USERS** ;
  - ✔ **GRANT CREATE SESSION TO** pedro ;
- Se concede a pedro el rol CONNECT, lo que le permitira tener todos los privilegios descritos para este rol (ALTER SESSION, CREATE CLUSTER, CREATE DATABASE LINK, CREATE SEQUENCE, CREATE SESSION, CREATE SYNONYM, CREATE TABLE, y CREATE VIEW):
  - ✔ **GRANT CONNECT TO** pedro ;
- Ahora se concede a pedro y a usuario1 el privilegio de administrador del sistema (DBA):
  - ✔ **GRANT DBA TO** pedro, usuario1 ;
- Para hacer que milagros pueda borrar usuarios, y ademas pueda conceder este privilegio a otros usuarios, se utiliza la opcion WITH ADMIN OPTION:
  - ✔ **GRANT DROP USER TO** milagros **WITH ADMIN OPTION** ;
- Para hacer que todos los usuarios puedan hacer SELECT en cualquier tabla de cualquier usuario, escribimos:
  - ✔ **GRANT SELECT ANY TABLE TO** PUBLIC ;

7. milagros retira los privilegios SELECT/UPDATE en TABLA1 a francisco (ejer.7 pag. 393):

- Conecto el usuario y ejecuto las ordenes
  - ✔ **CONNECT** milagros/milagros
  - ✔ **REVOKE** SELECT, UPDATE **ON** TABLA1 **FROM** francisco ;
- REVOKE ALL elimina todos los privilegios concedidos anteriormente sobre algún objeto. La opción WITH GRANT OPTION desaparece con el privilegio con el cual fue asignada. En este ejemplo, milagros retira todos los privilegios concedidos a francisco y juan sobre TABLA1:
  - ✔ **REVOKE** ALL **ON** TABLA1 **FROM** francisco, juan ;
- Retiramos el privilegio de borrar usuarios a milagros:
  - ✔ **REVOKE** DROP USER **FROM** milagros ;
- Retiramos el privilegio de consultar cualquier tabla a todos los usuarios:
  - ✔ **REVOKE** SELECT ANY TABLE **FROM** PUBLIC ;
- Retiramos el privilegio de administrador (DBA) a los usuarios juan y pedro:
  - ✔ **REVOKE** DBA **FROM** juan, pedro ;

8. Por razones de seguridad, creamos el perfil PERFIL1, en el que limitamos a uno el numero máximo de sesiones concurrentes por usuario y a dos minutos el tiempo de conexión permitido por sesión (ejercicio 8 de la pagina 398):

- Creo el perfil
  - ✔ **CONNECT** system/manager
  - ✔ **CREATE PROFILE** PERFIL1 **LIMIT SESSIONS_PER_USER** 1 **CONNECT_TIME** 2 **;**
    Como los demás limites del recurso no se mencionan en la instrucción CREATE PROFILE, se utilizaran los valores por defecto del sistema.
- Creo un usuario de prueba
  - ✔ **CREATE USER** prueba1 **IDENTIFIED BY** prueba1 **QUOTA** 100K **ON USERS PROFILE** PERFIL1 ;

- Le asigno el perfil
  - **GRANT** CONNECT **TO** prueba1 ;

- Pruebo que si abro una sesión de oracle con el usuario funciona, pero si abro otra diferente da error.
  - **Nota: para que funcionen estas limitaciónes hay que activar el uso de perfiles en el sistema, esto se hace con la siguiente orden por un administrador:**
    - **ALTER SYSTEM** SET RESOURCE_LIMIT=TRUE ;
    - si no aplica esta orden no tienen efecto las limitaciones de los perfiles.

- Se crea un perfil que permite 3 intentos de acceso fallidos para la cuenta, el cuarto producirá el bloqueo de la cuenta:
  - **CONNECT** system/manager
  - **CREATE PROFILE** PERFIL2 **LIMIT FAILED_LOGIN_ATTEMPTS** 3 **;**

- Se asigna el perfil a usuario creado.
  - **ALTER USER** prueba1 **PROFILE** PERFIL2 ;

- Se prueba el perfil
  - **CONNECT** prueba1/prueba ; tantas veces hasta que se bloquee la cuenta.

- Para desbloquear una cuenta el administrador de la BD tiene que ejecutar:
  - **ALTER USER** prueba1 **ACCOUNT UNLOCK** ;

- Se prueba el funcionamiento del perfil creado y como se bloquea la cuenta tras tres intentos.

```
SQL*Plus: Release 10.2.0.1.0 - Production on Jue Ene 10 18:45:41 2013

Copyright (c) 1982, 2005, Oracle.  All rights reserved.

SQL> CONNECT prueba1/prueba ;
ERROR:
ORA-01017: invalid username/password; logon denied

SQL> CONNECT prueba1/prueba ;
ERROR:
ORA-01017: invalid username/password; logon denied

SQL> CONNECT prueba1/prueba ;
ERROR:
ORA-01017: invalid username/password; logon denied

SQL> CONNECT prueba1/prueba ;
ERROR:
ORA-28000: the account is locked

SQL> CONNECT system/manager
Conectado.
SQL> ALTER USER prueba1 ACCOUNT UNLOCK ;

Usuario modificado.

SQL> CONNECT prueba1/prueba1 ;
Conectado.
SQL>
```

- A continuación se modifica el perfil anterior para que obligue a los usuarios que lo tengan asignado a cambiar su contraseña cada 2 días.
  - **CONNECT** system/manager
  - **ALTER PROFILE** PERFIL2 **LIMIT PASSWORD_LIFE_TIME** 2 **;**

Actividades propuestas (Capitulo 12)

1. Concede el privilegio SELECT e INSERT sobre la tabla DEPARTAMENTOS a uno de tus compañeros de clase con la opción de que se lo pueda conceder a otros (pag.390).

- Creo el usuario javi con SYSTEM y con el usuario HR que es quien tiene la tabla le concedo permisos a javi
  - ✔ **CONNECT** system/manager
  - ✔ **CREATE USER** javi **IDENTIFIED BY** javi **QUOTA** 500K **ON USERS**;
  - ✔ **GRANT CREATE SESSION TO** javi;
  - ✔ **CONNECT** HR/HR
  - ✔ **GRANT SELECT, INSERT ON** DEPARTAMENTOS **TO** javi **WITH GRANT OPTION**;

Concede el privilegio UPDATE sobre la columna APELLIDO de la tabla EMPLEADOS a un compañero de clase.

- Con el usuario HR que es quien tiene la tabla le concedo el nuevo permiso a javi
  - ✔ **CONNECT** HR/HR
  - ✔ **GRANT UPDATE** (APELLIDO) **ON** EMPLEADOS **TO** javi;

Prueba los privilegios recibidos sobre las tablas.

- Pruebo los privilegios concedidos.
  - ✔ **CONNECT** javi/javi
  - ✔ **INSERT INTO** HR.DEPARTAMENTOS (DEP_NO,DNOMBRE,LOCALIDAD) **VALUES** (50,'PERSONAL','BILBAO');
  - ✔ **SELECT * FROM** HR.DEPARTAMENTOS;

DEP_NO	DNOMBRE	LOCALIDAD
10	CONTABILIDAD	BARCELONA
20	INVESTIGACION	VALENCIA
30	VENTAS	MADRID
40	PRODUCCION	SEVILLA
50	PERSONAL	BILBAO

  - ✔ **CONNECT** HR/HR
  - ✔ **INSERT INTO** EMPLEADOS (EMP_NO,APELLIDO,OFICIO,DIRECTOR,FECHA_ALTA,SALARIO,COMISION,DEP_NO) **VALUES** (8282,'NOSE','VENDEDOR',7698,SYSDATE,1350,0,30);

7839	REY	PRESIDENTE		17/11/91	4100		10
7698	GARRIDO	DIRECTOR	7839	01/05/91	3005		30
7782	MARTINEZ	DIRECTOR	7839	09/06/91	2885		10
7499	ALONSO	VENDEDOR	7698	20/02/90	1500	390	30
7521	LOPEZ	EMPLEADO	7782	22/02/91	1625		10
7654	MARTIN	VENDEDOR	7698	28/09/91	1600	1020	30
7844	CALVO	VENDEDOR	7698	08/09/91	1350	0	30
7876	GIL	ANALISTA	7782	06/06/93	1430		20
7900	JIMENEZ	ANALISTA	7782	24/03/82	3000		20
8282	NOSE	VENDEDOR	7698	10/12/12	1350	0	30

- ✔ **CONNECT** javi/javi
- ✔ **UPDATE** HR.EMPLEADOS **SET** APELLIDO='VALENCI' **WHERE** EMP_NO=8282;
- ✔ **CONNECT** HR/HR
- ✔ **SELECT * FROM** EMPLEADOS **WHERE** EMP_NO=8282;

8282	VALENCI	VENDEDOR	7698	10/12/12	1350	0	30

2. Escribe una secuencia de ordenes en la que se crea un usuario, se le asigna el privilegio de iniciar sesión en Oracle y de crear una tabla. Después conectate con ese usuario y consulta los privilegios del sistema que tiene. Concede permiso SELECT sobre la tabla creada a otro usuario y consulta los privilegios sobre objetos concedidos y recibidos por el.(pag.394).

- Creo los usuarios con SYSTEM
  - ✔ **CONNECT** system/manager
  - ✔ **CREATE USER** pepe **IDENTIFIED BY** pepe **QUOTA** 500K **ON USERS** ;

- Se da permiso al usuarios de conexión y creación de tablas
  - ✔ **GRANT CREATE SESSION TO** pepe ;
  - ✔ **GRANT RESOURCE TO** pepe ;

- Se entra con el usuario, se crea la tabla TABLA2 y se meten valores
  - ✔ **CONNECT** pepe/pepe
  - ✔ **CREATE TABLE** TABLA2
    (
    CIUDAD **VARCHAR2**(15) ,
    TEMP **NUMBER**(2),
    CONSTRAINT CIUDAD_PK **PRIMARY KEY** (CIUDAD)
    );
  - ✔ **INSERT INTO** TABLA2 (CIUDAD,TEMP) **VALUES** ('SANTANDER',20) ;
  - ✔ **INSERT INTO** TABLA2 (CIUDAD,TEMP) **VALUES** ('ALMERIA',15) ;
  - ✔ **INSERT INTO** TABLA2 (CIUDAD,TEMP) **VALUES** ('BARCELONA',35) ;
  - ✔ **INSERT INTO** TABLA2 (CIUDAD,TEMP) **VALUES** ('BADAJOZ',27) ;

- Se consultan privilegios del sistema que tiene
  - ✔ **SELECT * FROM** SESSION_PRIVS

```
SQL> SELECT * FROM SESSION_PRIVS ;

PRIVILEGE
----------------------------------------
CREATE SESSION
UNLIMITED TABLESPACE
CREATE TABLE
CREATE ANY TABLE
CREATE CLUSTER
CREATE SEQUENCE
CREATE PROCEDURE
CREATE TRIGGER
CREATE TYPE
CREATE OPERATOR
CREATE INDEXTYPE

10 filas seleccionadas.
```

- ✔ **SELECT * FROM** USER_SYS_PRIVS ;

```
SQL> SELECT * FROM USER_SYS_PRIVS ;

USERNAME   PRIVILEGE                                 ADM
---------- ---------------------------------------- -------
PEPE       UNLIMITED TABLESPACE                     NO
PUBLIC     SELECT ANY TABLE                         NO
PEPE       CREATE SESSION                           NO
```

- ➢ Se concede permiso de SELECT sobre la tabla a milagros
  - ✔ **GRANT SELECT ON** TABLA2 **TO** milagros ;

- ➢ Se hace una consulta de concesiones sobre objetos propiedad de pepe concedidos y recibidos
  - ✔ **SELECT * FROM** USER_TAB_PRIVS ;

```
SQL> SELECT * FROM USER_TAB_PRIVS ;

GRANTEE    OWNER      TABLE_NAME     GRANTOR    PRIVILEGE  GRA HIE
MILAGROS   PEPE       TABLA2         PEPE       SELECT     NO  NO
```

- ➢ Se hace una consulta de concesiones sobre objetos propiedad de pepe (asignadas)
  - ✔ **SELECT * FROM** USER_TAB_PRIVS_MADE ;

```
SQL> SELECT * FROM USER_TAB_PRIVS ;

GRANTEE    TABLE_NAME     GRANTOR    PRIVILEGE  GRA HIE
MILAGROS   TABLA2         PEPE       SELECT     NO NO
```

3. Crea un usuario y concedele el rol creado (ACCESO). Añade el privilegio CREATE TABLE al rol. Consulta los privilegios de sistema que tiene asignados el usuario creado.(pag.395).

- ➢ Me conecto como system y creo el rol ACCESO
  - ✔ **CONNECT** system/manager
  - ✔ **CREATE** ROLE ACCESO ;

- ➢ Creo una una tabla para las pruebas (sobre esta tabla se concederán los privilegios al rol)
  - ✔ **CREATE TABLE** TABLA3
    (
    CIUDAD **VARCHAR2**(15) ,
    TEMP **NUMBER**(2),
    CONSTRAINT CIUDAD_PK **PRIMARY KEY** (CIUDAD)
    );
  - ✔ **INSERT INTO** TABLA3 (CIUDAD,TEMP) **VALUES** ('SANTANDER',20) ;
  - ✔ **INSERT INTO** TABLA3 (CIUDAD,TEMP) **VALUES** ('ALMERIA',15) ;
  - ✔ **INSERT INTO** TABLA3 (CIUDAD,TEMP) **VALUES** ('BARCELONA',35) ;
  - ✔ **INSERT INTO** TABLA3 (CIUDAD,TEMP) **VALUES** ('BADAJOZ',27) ;

- ➢ Concedo privilegios al rol creado
  - ✔ **GRANT** SELECT, INSERT **ON** TABLA3 **TO** ACCESO ;
  - ✔ **GRANT** CREATE SESSION **TO** ACCESO ;

- ➢ Creo el usuario lurdes y le concedo el rol creado
  - ✔ **CREATE USER** lurdes **IDENTIFIED BY** lurdes **QUOTA** 500K **ON USERS** ;
  - ✔ **GRANT** ACCESO **TO** lurdes ;

- Añado un privilegio al rol creado
  - ✔ **GRANT** CREATE TABLE **TO** ACCESO ;

- Me conecto como el usuario lurdes y consulto sus privilegios de sistema
  - ✔ **CONNECT** lurdes/lurdes
  - ✔ **SELECT * FROM** SESSION_PRIVS ;

```
SQL> SELECT * FROM SESSION_PRIVS ;

PRIVILEGE
----------------------------------------
CREATE SESSION
CREATE TABLE
```

  - ✔ **SELECT * FROM** USER_SYS_PRIVS ;

```
SQL> SELECT * FROM USER_SYS_PRIVS ;

ninguna fila seleccionada
```

Actividades complementarias (Capitulo 12 pag.401)

1. Crea una base de datos de forma manual llamada ORACLE10. Sigue el procedimiento del caso practico 2.
No hay que hacerlo.

2. Una vez creada la base de datos, crea un usuario y dale permiso para conectarse y crear objetos. Crea una tabla con el formato que desees. Resuelve las situaciones que se presenten.
No hay que hacerlo

3. Crea un rol que tenga los siguientes privilegios:
INSERT y SELECT en DEPAR y EMPLE, CREATE SESSION, CREATE TYPE, CREATE TABLE y CREATE VIEW.

- Me conecto como system y creo el rol ROL1
  - ✔ **CONNECT** system/manager
  - ✔ **CREATE** ROLE ROL1 ;

- Creo las tablas
  - ✔ **CREATE** TABLE DEPARTAMENTOS
    (DEP_NO **NUMBER**(2),
    DNOMBRE **VARCHAR2**(14),
    LOCALIDAD **VARCHAR2**(10),
    CONSTRAINT PK_DEPARTAMENTOS_DEP_NO PRIMARY KEY (DEP_NO));

  - ✔ **CREATE** TABLE EMPLEADOS
    (EMP_NO **NUMBER**(4),
    APELLIDO **VARCHAR2**(8),
    OFICIO **VARCHAR2**(10),
    DIRECTOR **NUMBER**(4),
    FECHA_ALTA **DATE**,
    SALARIO **NUMBER**(6),
    COMISION **NUMBER**(6),
    DEP_NO **NUMBER** (2),

CONSTRAINT PK_EMPLEADOS_EMP_NO PRIMARY KEY (EMP_NO),
CONSTRAINT FK_EMP_DIRECTOR FOREIGN KEY (DIRECTOR)
REFERENCES EMPLEADOS(EMP_NO),
CONSTRAINT FK_EMP_DEP_NO FOREIGN KEY (DEP_NO)
REFERENCES DEPARTAMENTOS(DEP_NO));

- ✔ **INSERT INTO** DEPARTAMENTOS **VALUES**(10, 'CONTABILIDAD', 'BARCELONA') ;
  INSERT INTO DEPARTAMENTOS **VALUES**(20, 'INVESTIGACION', 'VALENCIA') ;
  INSERT INTO DEPARTAMENTOS **VALUES**(30, 'VENTAS', 'MADRID') ;
  INSERT INTO DEPARTAMENTOS **VALUES**(40, 'PRODUCCION', 'SEVILLA') ;

- ✔ **INSERT INTO** EMPLEADOS **VALUES** (7839,'REY', 'PRESIDENTE',NULL,'17-NOV-1991',4100, NULL, 10) ;
  INSERT INTO EMPLEADOS **VALUES** (7698,'GARRIDO', 'DIRECTOR', 7839,'01-MAY-1991',3005, NULL, 30) ;
  INSERT INTO EMPLEADOS **VALUES** (7782,'MARTINEZ','DIRECTOR', 7839,'09-JUN-1991',2885, NULL, 10) ;
  INSERT INTO EMPLEADOS **VALUES** (7499,'ALONSO','VENDEDOR', 7698,'20-FEB-1990',1500,390,30) ;
  INSERT INTO EMPLEADOS **VALUES** (7521,'LOPEZ', 'EMPLEADO', 7782,'22-FEB-1991',1625, NULL,10) ;
  INSERT INTO EMPLEADOS **VALUES** (7654,'MARTIN', 'VENDEDOR', 7698,'28-SEP-1991',1600, 1020, 30) ;
  INSERT INTO EMPLEADOS **VALUES** (7844,'CALVO', 'VENDEDOR', 7698,'08-SEP-1991',1350, 0, 30) ;
  INSERT INTO EMPLEADOS **VALUES** (7876,'GIL', 'ANALISTA', 7782,'06-JUN-1993',1430, NULL, 20) ;
  INSERT INTO EMPLEADOS **VALUES** (7900,'JIMENEZ', 'EMPLEADO', 7782,'24-MAR-1982',3000, NULL, 20) ;

- ➢ Concedo privilegios al rol creado
  - ✔ **GRANT** INSERT, SELECT **ON** DEPARTAMENTOS **TO** ROL1 ;
  - ✔ **GRANT** INSERT, SELECT **ON** EMPLEADOS **TO** ROL1 ;
  - ✔ **GRANT** CREATE SESSION, CREATE TYPE, CREATE TABLE, CREATE VIEW **TO** ROL1 ;

4. CREA un usuario llamado comprador. El tablespace por defecto, es USERS. Se le asigna 1 MB en el tablespace USERS. El tablespace temporal sera TEMP. Se le asigna el rol anterior.

- ➢ Me conecto como system y creo el usuario
  - ✔ **CONNECT** system/manager
  - ✔ **CREATE USER** comprador **IDENTIFIED BY** comprador **DEFAULT** TABLESPACE USERS **TEMPORARY** TABLESPACE TEMP **QUOTA** 1M **ON USERS** ;
  - ✔ **GRANT** ROL1 **TO** comprador ;

5. Realiza la siguiente secuencia de instrucciones en el orden indicado:

1. **Crea un usuario de base de datos que tenga funciones de administrador.**
   - ✔ **CONNECT** system/manager
   - ✔ **CREATE USER** admin **IDENTIFIED BY** admin **QUOTA** 1M **ON USERS** ;
   - ✔ **GRANT** DBA **TO** admin ;

2. **Conectate con el nombre del usuario creado.**
   - ✔ **CONNECT** admin/admin

3. **Crea varias tablas en el propio esquema.**
   - ✔ **CREATE** TABLE DEPART
     (DEP_NO **NUMBER**(2),
     DNOMBRE **VARCHAR2**(14),
     LOCALIDAD **VARCHAR2**(10),
     CONSTRAINT PK_DEPART_DEP_NO PRIMARY KEY (DEP_NO));

   - ✔ **CREATE** TABLE EMPLE
     (EMP_NO **NUMBER**(4),
     APELLIDO **VARCHAR2**(8),
     OFICIO **VARCHAR2**(10),
     DIRECTOR **NUMBER**(4),
     FECHA_ALTA **DATE**,
     SALARIO **NUMBER**(6),
     COMISION **NUMBER**(6),
     DEP_NO **NUMBER** (2),
     CONSTRAINT PK_EMPLE_EMP_NO PRIMARY KEY (EMP_NO),
     CONSTRAINT FK_EMP_DIRECTOR FOREIGN KEY (DIRECTOR)
     REFERENCES EMPLE(EMP_NO),
     CONSTRAINT FK_EMP_DEP_NO FOREIGN KEY (DEP_NO)
     REFERENCES DEPART(DEP_NO));

   - ✔ **INSERT INTO** DEPART **VALUES**(10, 'CONTABILIDAD', 'BARCELONA') ;
     INSERT INTO DEPART **VALUES**(20, 'INVESTIGACION', 'VALENCIA') ;
     INSERT INTO DEPART **VALUES**(30, 'VENTAS', 'MADRID') ;
     INSERT INTO DEPART **VALUES**(40, 'PRODUCCION', 'SEVILLA') ;

   - ✔ **INSERT INTO** EMPLE **VALUES** (7839,'REY', 'PRESIDENTE',NULL,'17-NOV-1991',4100, NULL, 10) ;
     INSERT INTO EMPLE **VALUES** (7698,'GARRIDO', 'DIRECTOR', 7839,'01-MAY-1991',3005, NULL, 30) ;
     INSERT INTO EMPLE **VALUES** (7782,'MARTINEZ','DIRECTOR', 7839,'09-JUN-1991',2885, NULL, 10) ;
     INSERT INTO EMPLE **VALUES** (7499,'ALONSO','VENDEDOR', 7698,'20-FEB-1990',1500,390,30) ;
     INSERT INTO EMPLE **VALUES** (7521,'LOPEZ', 'EMPLEADO', 7782,'22-FEB-1991',1625, NULL,10) ;
     INSERT INTO EMPLE **VALUES** (7654,'MARTIN', 'VENDEDOR', 7698,'28-SEP-1991',1600, 1020, 30) ;
     INSERT INTO EMPLE **VALUES** (7844,'CALVO', 'VENDEDOR', 7698,'08-SEP-1991',1350, 0, 30) ;
     INSERT INTO EMPLE **VALUES** (7876,'GIL', 'ANALISTA', 7782,'06-JUN-1993',1430, NULL, 20) ;
     INSERT INTO EMPLE **VALUES** (7900,'JIMENEZ', 'EMPLEADO', 7782,'24-MAR-1982',3000, NULL, 20) ;

4. **Crea cinco usuarios nuevos asignándoles un tablespace por defecto y cuota (USU1, USU2, USU3, USU5 y USU5).**
   - ✔ **CREATE USER** USU1 **IDENTIFIED BY** USU1 **QUOTA** 500K **ON USERS ;**
   - ✔ **CREATE USER** USU2 **IDENTIFIED BY** USU2 **QUOTA** 500K **ON USERS ;**
   - ✔ **CREATE USER** USU3 **IDENTIFIED BY** USU3 **QUOTA** 500K **ON USERS ;**
   - ✔ **CREATE USER** USU4 **IDENTIFIED BY** USU4 **QUOTA** 500K **ON USERS ;**
   - ✔ **CREATE USER** USU5 **IDENTIFIED BY** USU5 **QUOTA** 500K **ON USERS ;**

5. **Da permiso a uno de los usuarios (USU1) solo para que pueda conectarse a la base de datos.**
   - ✔ **GRANT** CREATE SESSION **TO** USU1**;**
   - ✔ Se prueba el permiso otorgado.

```
SQL> CONNECT USU1/USU1 ;

Conectado.
```

6. **Crea un rol que permita conectarse a la base de datos y hacer SELECT sobre algunas tablas.**
   - ✔ **CREATE ROLE** ROL2 **;**
   - ✔ **GRANT** CREATE SESSION **TO** ROL2 **;**
   - ✔ **GRANT** SELECT **ON** DEPART **TO** ROL2 ;

7. **Concede el rol creado a dos de los usuarios creados anteriormente (USU2 y USU3).**
   - ✔ **GRANT** ROL2 **TO** USU2, USU3 **;**
   - ✔ Se prueba el ROL otorgado.

```
SQL> CONNECT USU2/USU2
Conectado.

SQL> SELECT * FROM admin.DEPART ;

DEP_NO     DNOMBRE          LOCALIDAD
----------  --------------  ----------
10         CONTABILIDAD     BARCELONA
20         INVESTIGACION    VALENCIA
30         VENTAS           MADRID
40        PRODUCCION        SEVILLA
```

8. **Concede al usuario USU4 privilegios sobre algunas tablas con la opción de poder concedérselos a otros usuarios.**
   - ✔ **GRANT** CREATE SESSION **TO** USU4 **;**
   - ✔ **GRANT SELECT, INSERT ON** EMPLE **TO** USU4 **WITH GRANT OPTION ;**
   - ✔ Se prueba el privilegio otorgado.

```
SQL> CONNECT USU4/USU4 ;

Conectado.

SQL> GRANT SELECT ON admin.EMPLE TO USU5 ;

Concesion terminada correctamente.
```

9. **Concede al usuario USU5 cuatro privilegios de sistema, dos de ellos, con la opción de poder concedérselos a otros usuarios.**
   - ✔ **GRANT** CREATE ROLE **TO** USU5 **;**
   - ✔ **GRANT** CREATE USER **TO** USU5 **;**
   - ✔ **GRANT** CREATE SESSION **TO** USU5 **WITH ADMIN OPTION ;**
   - ✔ **GRANT** CREATE TABLE **TO** USU5 **WITH ADMIN OPTION ;**

   - ✔ Se prueban algunos de los privilegios otorgados.

```
SQL> CONNECT USU5/USU5
Conectado.

SQL> CREATE ROLE ROL3 ;

Rol creado.

SQL> CREATE USER USU6 IDENTIFIED BY USU6 QUOTA 500K  ON USERS ;

Usuario creado.
```

10. **Concede a todos los usuarios de la base de datos privilegios para que puedan modificar ciertas columnas de algunas tablas.**
    - ✔ **GRANT UPDATE** (APELLIDO) **ON** EMPLE **TO** PUBLIC **;**
    - ✔ Se prueba el privilegio otorgado.

```
SQL> CONNECT USU1/USU1
Conectado.

SQL> UPDATE admin.EMPLE SET APELLIDO='VALEN' WHERE EMP_NO=7900 ;

1 fila actualizada.
```

11. **Quita a los usuarios USU3 y USU4 todos los privilegios que tenían asignados.**
    - ✔ **CONNECT** admin/admin
    - ✔ **REVOKE** ROL2 **FROM** USU3 **;**
    - ✔ **REVOKE** SELECT, INSERT **ON** EMPLE **FROM** USU4 **;**

12. **Haz que USU5 solo pueda conectarse en dos sesiones concurrentes a la vez.**
    - ✔ **CONNECT** admin/admin
    - ✔ **CREATE PROFILE** PERFIL3 **LIMIT SESSIONS_PER_USER** 2 **;**
    - ✔ **ALTER USER** USU5 **PROFILE** PERFIL3 ;
    - ✔ **Nota: para que funcionen esta limitación hay que activar el uso de perfiles en el sistema, esto se hace con la siguiente orden por un administrador:**
      1. **ALTER SYSTEM** SET RESOURCE_LIMIT=TRUE ;

13. **Limita el tiempo de conexión a la base de datos a cinco minutos a los usuarios USU2 y USU3.**
    - ✔ **CONNECT** admin/admin
    - ✔ **CREATE PROFILE** PERFIL4 **LIMIT CONNECT_TIME** 2 **;**
    - ✔ **ALTER USER** USU2 **PROFILE** PERFIL4 **;**
    - ✔ **ALTER USER** USU3 **PROFILE** PERFIL4 **;**
    - ✔ **Nota: para que funcionen esta limitación hay que activar el uso de perfiles en el sistema, esto se hace con la siguiente orden por un administrador:**
      1. **ALTER SYSTEM** SET RESOURCE_LIMIT=TRUE ;

Ejercicios adicionales Cap.12

1. **Crear un usuario de nombre milagritos y asignarle una cuota en el tablespace USER de 100k.**
   - ✔ **CONNECT** system/manager
   - ✔ **CREATE USER** milagritos **IDENTIFIED BY** milagritos **DEFAULT TABLESPACE** USERS **QUOTA** 500K **ON USERS** ;

2. **Crear un rol ROL1 y concederle los permisos de SELECT y DELETE en las tablas que se quieran (emple y depart).**
   - ✔ **CREATE ROLE** ROL1 **;**
   - ✔ **GRANT** SELECT, DELETE **ON** DEPARTAMENTOS **TO** ROL1 ;
   - ✔ **GRANT** SELECT, DELETE **ON** EMPLEADOS **TO** ROL1 ;

3. **Añadir INSERT y UPDATE al ROL1 en otros dos tablas.**
   - ✔ **GRANT** INSERT, UPDATE **ON** TABLA3 **TO** ROL1 ;
   - ✔ **GRANT** INSERT, UPDATE **ON** BANCOS **TO** ROL1 ;

4. **Añade al rol creado 2 privilegios del sistema.**
   - ✔ **GRANT** CREATE SESSION, CREATE TABLE **TO** ROL1 **;**

5. **Concede el rol (ROL1) creado al usuario creado (milagritos) con la opción de que pueda concedérselo a otros usuarios.**
   - ✔ **GRANT** ROL1 **TO** milagritos **WITH ADMIN OPTION ;**

6. **¿Como podemos ver los privilegios del sistema del usuario activo?**
   - ✔ **CONNECT** milagritos/milagritos
   - ✔ **SELECT * FROM** USER_SYS_PRIVS ;
     Este comando da como resultado:
     - ninguna fila seleccionada

   - ✔ **SELECT * FROM** SESSION_PRIVS ;
     Sin embargo este si da resultados:
     ```
     PRIVILEGE
     --------------------------
     CREATE SESSION
     CREATE TABLE
     ```

   - ✔ Para comprobar los privilegios se hacen consultas, borrados, actualizaciones e inserciones:
     - **SELECT * FROM** system.DEPARTAMENTOS ;
     - **SELECT * FROM** system.EMPLEADOS ;
     - **DELETE FROM** system.DEPARTAMENTOS **WHERE** DEP_NO=40 ;
     - **DELETE FROM** system.EMPLEADOS **WHERE** EMP_NO=7900 ;
     - **INSERT INTO** system.TABLA3 **VALUES** ('SOMERA',20) ;
     - **UPDATE** system.TABLA3 SET TEMP=25 **WHERE** CIUDAD='SOMERA' ;
     - **INSERT INTO** system.BANCOS **VALUES** (30700019,'BANK','MOLLERA, 30','MINAS',0) **;**
     - **UPDATE** system.BANCOS **SET** LOCALIDAD='MOSER' **WHERE** ENT_SUC= 30700019 **;**

7. **Quita el permiso de INSERT en la tabla xx**
   - ✔ **CONNECT** system/manager
   - ✔ **REVOKE** INSERT **ON** TABLA3 **FROM** ROL1 ;

- ✔ Para comprobar que no tiene el privilegio se intenta hacer una inserción:
  - ✔ **CONNECT** milagritos/milagritos
  - ✔ **INSERT INTO** system.TABLA3 **VALUES** ('SALAS',28) ;
    Da como resultado:
    ERROR en Lynea 1:
    ORA-01031: privilegios insuficientes

8. ¿Como podemos ver solo los privilegios que hemos concedido sobre nuestros objetos a otros usuarios?.

- ✔ **SELECT * FROM** USER_TAB_PRIVS_MADE ;
- ✔ Este comando no da resultados si antes no concedo permisos a otro usuario sobre objetos propios.

- ✔ Para probar este comando creo una tabla con el usuario milagritos:
- ✔ **CONNECT** milagritos/milagritos
- ✔ **CREATE TABLE** TABLA4
  (CIUDAD VARCHAR2(15) ,
  TEMP NUMBER(2),
  CONSTRAINT CIUDAD_PK **PRIMARY KEY** (CIUDAD)) ;

  INSERT INTO TABLA4 (CIUDAD,TEMP) **VALUES** ('SANTANDER',20) ;
  INSERT INTO TABLA4 (CIUDAD,TEMP) **VALUES** ('ALMERIA',15) ;
  INSERT INTO TABLA4 (CIUDAD,TEMP) **VALUES** ('BARCELONA',35) ;

- ✔ Concedo permiso de **SELECT** sobre la tabla al usuario francisco :
- ✔ **GRANT SELECT ON** TABLA4 **TO** francisco;

- ✔ Consulto los permisos concedidos:
- ✔ **SELECT * FROM** USER_TAB_PRIVS_MADE ;

GRANTEE	TABLE_NAME	GRANTOR	PRIVILEGE	GRA	HIE
FRANCISCO	TABLA4	MILAGRITOS	SELECT	NO	NO

9. ¿Como podemos ver solo los privilegios que hemos recibido de otros usuarios?.

- ✔ **SELECT * FROM** USER_TAB_PRIVS_RECD ;
- ✔ Este comando no da resultados si ates no recibo permisos sobre objetos de otro usuario.

- ✔ Para probar este comando me conecto con otro usuario y doy permisos al usuario milagritos:
- ✔ **CONNECT** pepe/pepe
- ✔ **GRANT SELECT ON** TABLA2 **TO** milagritos;

- ✔ Consulto los permisos concedidos:
- ✔ **SELECT * FROM** USER_TAB_PRIVS_RECD ;

OWNER	TABLE_NAME	GRANTOR	PRIVILEGE	GRA	HIE
PEPE	TABLA2	PEPE	SELECT	NO	NO

Tabla DEPARTAMENTOS

```
CREATE TABLE DEPARTAMENTOS
( DEP_NO                NUMBER(2),
DNOMBRE                 VARCHAR2(14),
LOCALIDAD    VARCHAR2(10),
CONSTRAINT PK_DEPARTAMENTOS_DEP_NO PRIMARY KEY (DEP_NO) );

INSERT INTO DEPARTAMENTOS VALUES(10, 'CONTABILIDAD', 'BARCELONA') ;
INSERT INTO DEPARTAMENTOS VALUES(20, 'INVESTIGACION', 'VALENCIA') ;
INSERT INTO DEPARTAMENTOS VALUES(30, 'VENTAS', 'MADRID') ;
INSERT INTO DEPARTAMENTOS VALUES(40, 'PRODUCCION', 'SEVILLA') ;
```

Tabla EMPLEADOS

```
CREATE TABLE EMPLEADOS
( EMP_NO                NUMBER(4),
APELLIDO                VARCHAR2(8),
OFICIO                  VARCHAR2(10),
DIRECTOR                NUMBER(4),
FECHA_ALTA   DATE,
SALARIO                 NUMBER(6),
COMISION                NUMBER(6),
DEP_NO                  NUMBER (2),
CONSTRAINT PK_EMPLEADOS_EMP_NO PRIMARY KEY (EMP_NO),
CONSTRAINT FK_EMP_DIRECTOR FOREIGN KEY (DIRECTOR)
REFERENCES EMPLEADOS(EMP_NO),
CONSTRAINT FK_EMP_DEP_NO FOREIGN KEY (DEP_NO)
REFERENCES DEPARTAMENTOS(DEP_NO) );

INSERT INTO EMPLEADOS VALUES (7839,'REY', 'PRESIDENTE',NULL,'17-NOV-1991',4100, NULL, 10) ;
INSERT INTO EMPLEADOS VALUES (7698,'GARRIDO', 'DIRECTOR', 7839,'01-MAY-1991',3005, NULL, 30) ;
INSERT INTO EMPLEADOS VALUES (7782,'MARTINEZ','DIRECTOR', 7839,'09-JUN-1991',2885, NULL, 10) ;
INSERT INTO EMPLEADOS VALUES (7499,'ALONSO','VENDEDOR', 7698,'20-FEB-1990',1500,390,30) ;
INSERT INTO EMPLEADOS VALUES (7521,'LOPEZ', 'EMPLEADO', 7782,'22-FEB-1991',1625, NULL,10) ;
INSERT INTO EMPLEADOS VALUES (7654,'MARTIN', 'VENDEDOR', 7698,'28-SEP-1991',1600, 1020, 30) ;
INSERT INTO EMPLEADOS VALUES (7844,'CALVO', 'VENDEDOR', 7698,'08-SEP-1991',1350, 0, 30) ;
INSERT INTO EMPLEADOS VALUES (7876,'GIL', 'ANALISTA', 7782,'06-JUN-1993',1430, NULL, 20) ;
INSERT INTO EMPLEADOS VALUES (7900,'JIMENEZ', 'EMPLEADO',  7782,'24-MAR-1982',3000, NULL, 20) ;
```

Tabla TABLA3

```
CREATE TABLE TABLA3
(
CIUDAD VARCHAR2(15) ,
TEMP NUMBER(2),
CONSTRAINT CIUDAD_PK PRIMARY KEY (CIUDAD));

INSERT INTO TABLA3 (CIUDAD,TEMP) VALUES ('SANTANDER',20) ;
INSERT INTO TABLA3 (CIUDAD,TEMP) VALUES ('ALMERIA',15) ;
INSERT INTO TABLA3 (CIUDAD,TEMP) VALUES ('BARCELONA',35) ;
INSERT INTO TABLA3 (CIUDAD,TEMP) VALUES ('BADAJOZ',27) ;
```

Tabla BANCOS:

```
CREATE TABLE BANCOS
(ENT_SUC          NUMBER (8) NOT NULL,
NOMBRE            VARCHAR2 (50),
DIRECCION          VARCHAR2 (50),
LOCALIDAD          VARCHAR2 (30),
TELEFONOS          VARCHAR2 (30),
CONSTRAINT BANCOS_PK PRIMARY KEY(ENT_SUC)) ;

Insert into bancos (ent_suc, nombre, direccion, localidad) values (30700018,'BANESTO','MANUEL LLANEZA, 33','MIERES');
Insert into bancos (ent_suc, nombre, direccion, localidad) values (20480070,'CAJASUR','MANUEL LLANEZA, 17','MIERES');
Insert into bancos (ent_suc, nombre, direccion, localidad) values (43000250,'HERRERO','MANUEL LLANEZA, 22','MIERES');
Insert into bancos (ent_suc, nombre, direccion, localidad) values (85002222,'SANTANDER','LA CAMARA, 13','AVILES');
Insert into bancos (ent_suc, nombre, direccion, localidad) values (22223333,'BBV','LA RIBERA, 17','LUANCO');
Insert into bancos (ent_suc, nombre, direccion, localidad) values (33334444,'ATLANTICO','GIJON, 56','LUANCO');
```

Ejercicios ejemplo del capitulo (Capitulo 13):

1. A continuación se crea un tablespace de 15 Megabytes llamado TRABAJO. El tamaño inicial para el objeto que se cree en el tablespace (por ejemplo, un tabla) es de 10K. El tamaño de la siguiente extensión del objeto también es 10K; cada extensión subsiguiente sera un 25 por 100 mas grande que la anterior. Asignamos dos archivos a este tablespace 'TRABAJO1.ORA', de 10M y 'TRABAJO2.ORA', de 5M (si no ponemos la ubicación, el tablespace lo creara en C:\WINNT\SYSTEM32\): (ejercicio 1 de la pagina 405)

- **CONNECT** system/manager
- **CREATE TABLESPACE** TRABAJO DATAFILE 'TRABAJO1.ORA' SIZE 10M, 'TRABAJO2.ORA' SIZE 5M DEFAULT STORAGE (INITIAL 10K NEXT 10K PCTINCREASE 25) ;

Se crea un tablespace de 100k llamado PEQUE. Asignamos el archivo 'PEQUE.ORA', habilitando el crecimiento automático de 120k para la extensión siguiente dentro de un espacio máximo de 1M:

- **CREATE TABLESPACE** PEQUE DATAFILE 'PEQUE.ORA' SIZE 100K AUTOEXTEND ON NEXT 120K MAXSIZE 1M ;

Se crea un tablespace de deshacer de 10M llamado DESHACER. Asignamos el archivo 'DESHACER.ORA', habilitando el crecimiento automático de 512k para la extensión siguiente dentro de un espacio máximo ilimitado.

- **CREATE TABLESPACE** DESHACER DATAFILE 'DESHACER.ORA' SIZE 10M REUSE AUTOEXTEND ON NEXT 512K MAXSIZE UNLIMITED ;

***Consulta que muestra la descripción de todos los tablespaces del usuario:**

- **CONNECT** system/manager
- **SELECT** TABLESPACE_NAME, CONTENTS **FROM** DBA_TABLESPACES ;

***Consulta que muestra informacion sobre los archivos utilizados por los tablespaces (de todos):**

- **CONNECT** system/manager
- **SELECT * FROM** DBA_DATA_FILES ;

***Consulta que muestra información sobre los bytes utilizados por cada usuario en cada tablespaces:**

- **CONNECT** system/manager
- **SELECT * FROM** DBA_TS_QUOTAS ;

2. Se agrega un archivo al tablespace TRABAJO de 6 Megabytes llamado 'TRABAJO3.ORA': (ejercicio 2 de la pagina 408)

- **CONNECT** system/manager
- **ALTER TABLESPACE** TRABAJO ADD DATAFILE 'TRABAJO3.ORA' SIZE 6M ;

Renombramos los archivos TRABAJO1.ORA y TRABAJO2.ORA del tablespace TRABAJO, se llamaran TRABA1.ORA y TRABA2.ORA respectivamente. los pasos para renombrar los tablespaces son:

- Se desactiva el tablespace TRABAJO.
- **CONNECT system/manager**
- **ALTER TABLESPACE** TRABAJO OFFLINE ;

- ✔ Se cambian de nombre los archivos con el nombre especificado (con una terminal de dos).
  - E:\Documents and Settings\juan>cd \oraclexe\app\oracle\product\10.2.0\server\database\
  - E:\oraclexe\app\oracle\product\10.2.0\server\database>copy TRABAJO1.ORA TRABA1.ORA
  - E:\oraclexe\app\oracle\product\10.2.0\server\database>copy TRABAJO2.ORA TRABA2.ORA

- ✔ Con la orden ALTER TABLESPACE y la orden RENAME DATAFILE se renombran
  - **ALTER TABLESPACE** TRABAJO **RENAME** DATAFILE 'TRABAJO1.ORA', 'TRABAJO2.ORA' **TO** 'TRABA1.ORA', 'TRABA2.ORA' ;

- ✔ Se activa el tablespace
  - **ALTER TABLESPACE** TRABAJO ONLINE ;

Ejercicios de repaso

Conectate con los privilegios de SELECT y UPDATE sobre la tabla EMPLE y concedele a tu compañero la opción de concederle los privilegios que tienes a los demás.

- **Creo las tablas que voy a usar en este ejercicio:**
  - ✔ **CONNECT** system/manager
  - ✔ **CREATE** TABLE DEPART
    (DEP_NO **NUMBER**(2),
    DNOMBRE **VARCHAR2**(14),
    LOCALIDAD **VARCHAR2**(10),
    CONSTRAINT PK_DEPART_DEP_NO PRIMARY KEY (DEP_NO));
  - ✔ **CREATE** TABLE EMPLE
    (EMP_NO **NUMBER**(4),
    APELLIDO **VARCHAR2**(8),
    OFICIO **VARCHAR2**(10),
    DIRECTOR **NUMBER**(4),
    FECHA_ALTA **DATE**,
    SALARIO **NUMBER**(6),
    COMISION **NUMBER**(6),
    DEP_NO **NUMBER** (2),
    CONSTRAINT PK_EMPLE_EMP_NO PRIMARY KEY (EMP_NO),
    CONSTRAINT FK_EMPLE_DIRECTOR FOREIGN KEY (DIRECTOR)
    REFERENCES EMPLE(EMP_NO),
    CONSTRAINT FK_EMPLE_DEP_NO FOREIGN KEY (DEP_NO)
    REFERENCES DEPART(DEP_NO));
  - ✔ **INSERT INTO** DEPART **VALUES**(10, 'CONTABILIDAD', 'BARCELONA') ;
    INSERT INTO DEPART **VALUES**(20, 'INVESTIGACION', 'VALENCIA') ;
    INSERT INTO DEPART **VALUES**(30, 'VENTAS', 'MADRID') ;
    INSERT INTO DEPART **VALUES**(40, 'PRODUCCION', 'SEVILLA') ;
  - ✔ **INSERT INTO** EMPLE **VALUES** (7839,'REY', 'PRESIDENTE',NULL,'17-NOV-1991',4100, NULL, 10) ;
    INSERT INTO EMPLE **VALUES** (7698,'GARRIDO', 'DIRECTOR', 7839,'01-MAY-1991',3005, NULL, 30) ;
    INSERT INTO EMPLE **VALUES** (7782,'MARTINEZ','DIRECTOR', 7839,'09-JUN-1991',2885, NULL, 10) ;
    INSERT INTO EMPLE **VALUES** (7499,'ALONSO','VENDEDOR', 7698,'20-FEB-1990',1500,390,30) ;
    INSERT INTO EMPLE **VALUES** (7521,'LOPEZ', 'EMPLEADO', 7782,'22-FEB-1991',1625, NULL,10) ;
    INSERT INTO EMPLE **VALUES** (7654,'MARTIN', 'VENDEDOR', 7698,'28-SEP-1991',1600, 1020, 30) ;
    INSERT INTO EMPLE **VALUES** (7844,'CALVO', 'VENDEDOR', 7698,'08-SEP-1991',1350, 0, 30) ;
    INSERT INTO EMPLE **VALUES** (7876,'GIL', 'ANALISTA', 7782,'06-JUN-1993',1430, NULL, 20) ;
    INSERT INTO EMPLE **VALUES** (7900,'JIMENEZ', 'EMPLEADO', 7782,'24-MAR-1982',3000, NULL, 20) **;**

- **Creo los usuario y le concedo privilegios de SELECT Y UPDATE sobre la tabla EMPLE a juan y que ademas se los pueda conceder a otro**
  - ✔ **CONNECT** system/manager
  - ✔ **CREATE USER** juan1 **IDENTIFIED BY** juan1 **QUOTA** 500K **ON USERS** ;
  - ✔ **CREATE USER** javi1 **IDENTIFIED BY** javi1 **QUOTA** 500K **ON USERS** ;
  - ✔ **GRANT CREATE SESSION TO** juan1 ;
  - ✔ **GRANT CREATE SESSION TO** javi1 ;
  - ✔ **GRANT SELECT**, **UPDATE ON** EMPLE **TO** juan1 **WITH GRANT OPTION** ;

- **Concedo los privilegios a otro usuario y que ademas se los pueda conceder a los demás.**
  - ✔ **CONNECT** juan1/juan1
  - ✔ **GRANT SELECT, UPDATE ON** system.EMPLE **TO** javi1 **WITH GRANT OPTION ;**

Concede el privilegio UPDATE al atributo sueldo de tabla EMPLE a un compañero tuyo.

- ✔ **CONNECT** system/manager
- ✔ **CREATE USER** jose1 **IDENTIFIED BY** jose1 **QUOTA** 500K **ON USERS** ;
- ✔ **GRANT CREATE SESSION TO** jose1 ;
- ✔ **GRANT UPDATE** (SALARIO) **ON** EMPLE **TO** jose1 **;**

Prueba los privilegios anteriores

- ✔ **CONNECT** juan1/juan1
- ✔ **SELECT** * F**ROM** system.EMPLE ;
- ✔ **UPDATE** system.EMPLE **SET** SALARIO=1300 **WHERE** EMP_NO=7876 ;
- ✔ **CONNECT** javi1/javi1
- ✔ **SELECT * FROM** system.EMPLE ;
- ✔ **UPDATE** system.EMPLE **SET** SALARIO=1400 **WHERE** EMP_NO=7876 ;
- ✔ **GRANT SELECT, UPDATE ON** system.EMPLE **TO** jose1 **;**
- ✔ **CONNECT** jose1/jose1
- ✔ **UPDATE** system.EMPLE **SET** SALARIO=1500 **WHERE** EMP_NO=7876 ;

Crea un usuario llamado bego y crea el rol 'ACCESO'.

- ✔ **CONNECT** system/manager
- ✔ **CREATE USER** bego1 **IDENTIFIED BY** bego1 **QUOTA** 500K **ON USERS** ;
- ✔ **CREATE ROLE** ACCESO ;

Añádele 2 privilegios de sistema a ese rol y 2 privilegios de usuario a ese rol y concedeselos al usuario bego.

- ✔ **CONNECT** system/manager
- ✔ **GRANT CREATE SESSION, CREATE TABLE TO** ACCESO **;**
- ✔ **GRANT SELECT, DELETE ON** DEPART **TO** ACCESO **;**
- ✔ **GRANT** ACCESO **TO** bego1 **;**

Consulta los privilegios

- ✔ **CONNECT** bego1/bego1
- ✔ Privilegios del sistema que tiene:
- ✔ **SELECT * FROM** SESSION_PRIVS **;** **SELECT * FROM** USER_SYS_PRIVS **;**
- ✔ Privilegios sobre tablas que tiene (no da resultados porque solo tiene concedido un rol).
- ✔ **SELECT * FROM** USER_TAB_PRIVS **;**
- ✔ Roles que tiene asignados (aquí el resultado es ACCESO que es el único rol que tiene)
- ✔ **SELECT * FROM** USER_ROLE_PRIVS ;
- ✔ Privilegios del sistema que tiene el rol o roles asignados
- ✔ **SELECT * FROM** ROLE_SYS_PRIVS ;
- ✔ Privilegios sobre tablas que tiene el rol o roles asignados
- ✔ **SELECT * FROM** ROLE_TAB_PRIVS ;
- ✔ También se puede consultar privilegios sobre tablas si los tiene desde el usuario system
- ✔ **CONNECT** system/manager
- ✔ **SELECT * FROM** USER_TAB_PRIVS **WHERE UPPER(**GRANTOR)='BEGO1' **;**

Crea un rol con los siguientes privilegios:

- **SELECT y UPDATE en las tablas DEPART y EMPLE**
- **Que tenga el privilegio de CREATE SESSION, CREATE TABLE y CREATE VIEW .**
  - **CONNECT** system/manager
  - **CREATE ROLE** ROL1 ;
  - **GRANT** SELECT, UPDATE **ON** DEPART **TO** ROL1 ;
  - **GRANT** SELECT, UPDATE **ON** EMPLE **TO** ROL1 ;
  - **GRANT** CREATE SESSION, CREATE TABLE, CREATE VIEW **TO** ROL1 ;

Crea un tablespace llamado ventas, asignale un archivo llamado ventas.ora de 5MB

- **CREATE TABLESPACE** VENTAS DATAFILE 'VENTAS.ORA' SIZE 5M ;

Modifica el anterior tablespace para que pueda auto-extenderse sin limite de espacio.

- **ALTER TABLESPACE** VENTAS AUTOEXTEND ON NEXT 512K MAXSIZE UNLIMITED ;
- me da el siguiente error al ejecutar la instrucción:
- ORA-32773: operación no soportada para tablespace de archivo pequeño VENTAS

Crea un usuario con funciones de administrador, conectate, introduce varias tablas (TABLA1, TABLA2 y TABLA3)

- **CONNECT** system/manager
- **CREATE USER** admin1 **IDENTIFIED BY** admin1 **QUOTA** 500K **ON** USERS ;
- **GRANT** DBA **TO** admin1 ;
- **CONNECT** admin1/admin1 ;
- **CREATE TABLE** TABLA1
  (CIUDAD **VARCHAR2**(15) ,
  TEMP **NUMBER**(2),
  CONSTRAINT CIUDAD_PK **PRIMARY KEY** (CIUDAD)) ;
- **CREATE TABLE** TABLA2
  (PROVINCIA **VARCHAR2**(15) ,
  TEMP **NUMBER**(2),
  CONSTRAINT PROVINCIA_PK **PRIMARY KEY** (PROVINCIA)) ;
- **CREATE TABLE** TABLA3
  (PAIS **VARCHAR2**(15) ,
  TEMP **NUMBER**(2),
  CONSTRAINT PAIS_PK **PRIMARY KEY** (PAIS)) ;

Crea tres usuarios en el tablespace por defecto de 1MB

- **CREATE USER** usuario1 **IDENTIFIED BY** usuario1 **QUOTA** 1M **ON** USERS ;
- **CREATE USER** usuario2 **IDENTIFIED BY** usuario2 **QUOTA** 1M **ON** USERS ;
- **CREATE USER** usuario3 **IDENTIFIED BY** usuario3 **QUOTA** 1M **ON** USERS ;

Da permiso al primer usuario para conectarse y hacer SELECT

- **GRANT CREATE SESSION TO** usuario1 ;
- **GRANT SELECT ON** TABLA1 **TO** usuario1 **;**

Crea un rol para conectarse a la base de datos que realice SELECT a la tabla 1, DELETE a la tabla 2 y UPDATE a la tabla 3.

- ✔ **CREATE ROLE** ROL2 ;
- ✔ **GRANT** SELECT **ON** TABLA1 **TO** ROL2 ;
- ✔ **GRANT** DELETE **ON** TABLA2 **TO** ROL2 ;
- ✔ **GRANT** UPDATE **ON** TABLA3 **TO** ROL2 ;
- ✔ **GRANT** CREATE SESSION **TO** ROL2 ;

Concede ese rol al usuario 2 y usuario3

- ✔ **GRANT** ROL2 **TO** usuario2 ;
- ✔ **GRANT** ROL2 **TO** usuario3 ;

concede al usuario 1 privilegios sobre la tabla 1 y tabla 2 y que luego se los pasen a usuario 2 y usuario3.

- ✔ **GRANT DELETE, UPDATE ON** TABLA1 **TO** usuario1 **WITH GRANT OPTION** ;
- ✔ **GRANT SELECT, UPDATE ON** TABLA2 **TO** usuario1 **WITH GRANT OPTION** ;

Quitale a usuario 3 todos los privilegios.

- ✔ **REVOKE** ROL2 **FROM** usuario3 ;

Haz que usuario 2 pueda conectarse en tres sesiones concurrentemente.

- ✔ **CREATE PROFILE** PERF1 **LIMIT SESSIONS_PER_USER** 3 ;
- ✔ **ALTER USER** usuario2 **PROFILE** PERF1 ;

```
SQL> connect usuario2/usuario2
ERROR:
ORA-02391: exceeded simultaneous SESSIONS_PER_USER limit
```

- ✔ **Nota: para que funcionen esta limitación hay que activar el uso de perfiles en el sistema, esto se hace con la siguiente orden por un administrador:**
  - ✗ **ALTER SYSTEM** SET RESOURCE_LIMIT=TRUE ;

Limita el tiempo de conexión a 3 minutos a usuario 1 y usuario 3

- ✔ **CREATE PROFILE** PERF2 **LIMIT CONNECT_TIME** 3 ;
- ✔ **ALTER USER** usuario1 **PROFILE** PERF2 ;
- ✔ **ALTER USER** usuario3 **PROFILE** PERF2 ;

```
SQL> connect usuario1/usuario1
Conectado.
SQL> select * from tabla1 ;
select * from tabla1
*
ERROR en lýnea 1:
ORA-02399: ha excedido el tiempo mßximo de conexi¾n, desconectando
```

- ✔ **Nota: para que funcionen esta limitación hay que activar el uso de perfiles en el sistema, esto se hace con la siguiente orden por un administrador:**
  - ✗ **ALTER SYSTEM** SET RESOURCE_LIMIT=TRUE ;

INSTRUCCIONES

- Los valores nulos en las expresiones siempre dan como resultado un valor nulo

Instrucción	Definición	Ejemplos
. **ED**	Invoca al Editor del Sistema.	
. **CLEAR SCR**	Borra la pantalla	
. **START** fichero	Ejecuta el contenido almacenado en ***fichero.sql***. Hay que especificar la ruta completa	
. **RUN**	Repite la ejecución de la última sentencia o de lo que hay en el buffer	. RUN . R
. **LIST**	Visualiza el contenido del buffer ó el contenido de la linea n	. LIST . LIST n . L n
. **SAVE** fichero	Almacena el contenido del buffer en *fichero.sql*	
. **GET** fichero	Recupera al buffer el contenido del fichero *fichero.sql*	
. **INPUT**	Añade una línea a continuación de la actual activa	
. **DEL**	Elimina la línea actual	
. **SPOOL** fichero	Todas las salidas por pantalla se almacenan en *fichero.lst*	
. **SPOOL OFF**	Libera el almacenamiento de salidas por pantalla	

Sentencia de creación de tablas y vistas		
. **CREATE TABLE** tabla1 (col1 tipo_dato [NOT NULL], col2 tipo_dato [NOT NULL], ...) ;	Crea la tabla1 con los campos especificados como col1, col2..., siendo cada campo del tipo y extensión definidos con tipo_dato.	
. **DESC** tabla1	Muestra la descripción de la tabla1, listando los nombres de los campos, tipo y extensión.	
. **CREATE TABLE** tabla1 col1 tipo_dato CONSTRAINT nombre_constraint restricciones, col2 tipo_dato [NOT NULL] CONSTRAINT nombre_constraint restricciones, ... ; . **CREATE TABLE** tabla1 (col1 tipo_dato, col2 tipo_dato, ..., CONSTRAINT nombre_constraint PRIMARY KEY (col1, col2...), CONSTRAINT nombre_constraint FOREIGN KEY (col1,col2...) REFERENCES tabla2 (col1, col2...) [ON DELETE CASCADE], CONSTRAINT nombre_constraint CHECK cond ...) ;	. Creación de una tabla con restricciones de columna. . Restricciones: . NOT NULL . UNIQUE . PRIMARY KEY . DEFAULT VALUE . REFERENCES tabla2 (col1, ...) [ON DELETE CASCADE] . CHECK cond En este otro caso las restricciones se ponen al final.	

. **CREATE TABLE** tabla1 (col1, col2,...) **AS** consulta ;	Creación de una tabla a partir de los datos recogidos mediante una consulta en otra tabla.	
. **DROP TABLE** [usuario.]tabla1 [CASCADE CONSTRAINTS] ;	Se borra la tabla1 del usuario especificado [con restricciones incluidas]	
. **ALTER TABLE** tabla1 [**ADD** (col1 tipo_dato, col2 tipo_dato)] [**MODIFY** (col1 tipo_dato, col2 tipo_dato)] [**ADD CONSTRAINT** restricción] [**DROP CONSTRAINT** restricción] ;	Modifica la tabla, añadiendo nuevos campos, cambiando características de los campos, añadiendo restricciones, suprimiendo restricciones	
. **CREATE VIEW** vista1 [(col1, col2,...] **AS** consulta ;	Se crea una vista con unos campos que se llamarán col1, col2... que se rellenan con los datos provenientes de una consulta	
. **DROP VIEW** vista1 ;	Se elimina la vista1	
. **CREATE** [PUBLIC] **SYNONIM** sinonimo1 **FOR** [usuario.]tabla1 ;	Crea un sinónimo para una tabla	
. **DROP SYNONIM** usuario.sinonimo1 ;	Se borra el sinonimo1	
. **RENAME** nombreanterior **TO** nombrenuevo ;	Se renombra una tabla con un nombre nuevo	

Sentencia de consulta de datos		
. **SELECT** [ALL|DISTINCT] [campo1, campo2,... | *] **FROM** [tabla1 alias, tabla2 alias, ...] **WHERE** cond **GROUP BY** exp, exp **HAVING** cond **ORDER** BY campo1 [DESC | ASC], campo2 [DESC | ASC], ...	selecciona campos; de la(s) tabla(s); selecciona filas; agrupa las filas (los campos deben estar en la SELECT); filtra los grupos seleccionando y eliminando; clasifica la salida. Ordena los grupos.	. Se usan comillas dobles para poner cabecera a cada columna en el resultado: . **SELECT** campo1 **"cabecera1"** FROM tabla1
	. **condiciones**: where NOTA = 5 where EDAD < 26 ; EDAD > 26 where PRECIO >= 50 ; PRECIO <= 50 where DESC <> 15 ; DESC != 15 where a=8 **AND** b=3 where a=8 **OR** b=3 where **NOT** a=8 where APE **LIKE** 'A%' where GRUPO **LIKE** 'A_' where GRUPO **IS NULL** where GRUPO **IS NOT NULL** where DEPT **IN** (20,30,40) where DEPT **BETWEEN** 10 **AND** 50	.**NOTAS**: . Los literales van entre comillas simples . Si se usan comodines en una comparación (%, _) hay que utilizar **LIKE** y no = . Si el campo a comparar es de tipo CHAR, se puede utilizar: APE ='expr' ó APE LIKE 'expr ' Se rellena a espacios toda la longitud de la variable CHAR

Funciones de selección de datos		
OUTER JOIN (+)	Selecciona algunas filas de una tabla aunque no tengan correspondencia con las filas de la otra tabla. Se pone + a la tabla donde están las filas que no tendrán correspondencia en la otra	. **SELECT** A.DATO1, B.DATO2 FROM A B **WHERE** A.CLAVE = B.CLAVE**(+)** **(en este caso habrá filas de la tabla A que no tengan correspondencia en la tabla B, por lo que los campos de la tabla B se rellenarán como NULL)**
. **UNION** (ALL)	Une los resultados de dos consultas. Las filas duplicadas se reducen a una excepto si se indica ALL	. **SELECT** campo1, campo2 **FROM** tabla1 **UNION** **SELECT** campo1, campo2 **FROM** tabla2
. **INTERSECT**	Devuelve las filas comunes de dos consultas	. **SELECT** campo1, campo2 **FROM** tabla1 **INTERSECT** **SELECT** campo1, campo2 **FROM** tabla2
. **MINUS**	Devuelve las filas de tabla1 que no están en tabla2	. **SELECT** campo1, campo2 **FROM** tabla1 **MINUS** **SELECT** campo1, campo2 **FROM** tabla2

Funciones de manipulación de datos		
. **INSERT INTO** tabla (col1, col2,…) **VALUES** (val1, val2,…)	Inserta en las columnas los valores especificados. Si no se indican las col se insertarán valores en todos los campos	. INSERT INTO EMPLE NOM, TEL VALUES 'Jorge', '945566778' ;
. **INSERT INTO** tabla1 (col1, col2,…) **SELECT** (kol1, kol2,…) **FROM** tabla2 (**WHERE** cond) ;	Inserta en las col de la tabla1 los valores de las kol de la tabla2 que cumplan la condición	. INSERT INTO EMPLE (NOM, TEL) SELECT (NOM1, TEL1) FROM EMPLE2 WHERE POB='Bilbao';
. **UPDATE** tabla **SET** col1=val1, col2=val2 **WHERE** cond ;	Modifica las columnas de la tabla con los valores indicados en las filas que cumplan la condición	. UPDATE EMPLE SET PROV='Bizkaia' WHERE PROV='Vizcaya' ;

. **UPDATE** tabla **SET** col1=val1, col2=val2 **WHERE** col3 = (SELECT…) ;	Modifica las columnas de la tabla con los valores indicados en las filas que cumplan la condición fruto de la SELECT	. UPDATE EMPLE SET PROV='Bizkaia' WHERE PROV=(SELECT PROV FROM PROVINCIAS WHERE CP=48) ;
. **UPDATE** tabla1 **SET** (col, col2,…) = (select kol1,kol2… FROM tabla2) **WHERE** cond ;	Modifica las columnas de la tabla1 con los valores seleccionados de la tabla2, en las filas que cumplan la condición	. UPDATE EMPLE SET (APENOM) = (SELECT UPPER(APENOM) FROM EMPLE WHERE DNI=30456546) WHERE PROV='Bizkaia' ;
. **DELETE (FROM)** tabla **WHERE** cond ;	Borra de la tabla las filas que cumplan la condición	. DELETE FROM EMPLE WHERE PROV IN (SELECT PROV FROM DEPART) ;
. **COMMIT** ;	Valida todos los cambios hechas en la BD desde que abrimos la última sesión o desde el último COMMIT	
. **ROLLBACK** ;	Deshace los cambios en la BD desde el último COMMIT	
. **AUTOCOMMIT**	Es un COMMIT automático que se ejecuta tras cada sentencia	Para activar AUTOCOMMIT: (SET AUTOCOMMIT ON)
Otras funciones		
. **DECODE** (var,val1,cod1, val2, cod2..., valdefecto)	Si 'var' es igual a algún 'val', lo sustituye por su 'cod', y si no coincide, por el valor por defecto	. DECODE (SEXO,'M', 'MUJER','H', 'HOMBRE','SINDEFINIR')
. **VSIZE** (expresión)	Devuelve el número de bytes que ocupa exp	. SELECT VSIZE ('AMOREBIETA') FROM DUAL 10
. **USER**	Muestra el usuario que está conectado	. SELECT USER FROM DUAL SCOTT
. **SHOW USER**	Muestra qué usuario somos	. SHOW USER el usuario es 'SCOTT'
. **UID**	Devuelve el identificador del usuari actual	
. **CONNECT** usuario/password	Conecta el usuario con su password a la BD	. CONNECT SCOTT/TIGER CONECTADO

Funciones aritméticas			
. **ABS** (n)	Devuelve el valor absoluto de "n"	. SELECT ABS(2) FROM DUAL . SELECT ABS(-2) FROM DUAL	2 2
. **CEIL** (n)	Obtiene el valor entero inmdiatamente superior o igual a "n"	. SELECT CEIL(13.2) FROM DUAL . SELECT CEIL(13) FROM DUAL . SELECT CEIL(-13.2) FROM DUAL	14 13 -13
. **FLOOR** (n)	Obtiene el valor entero inmdiatamente inferior o igual a "n"	. SELECT FLOOR(13.2) FROM DUAL . SELECT FLOOR(13) FROM DUAL . SELECT FLOOR(-13.2) FROM DUAL	13 13 -14
. **MOD** (m, n)	Devuelve el resto de dividir m/n	. SELECT MOD(11,4) FROM DUAL . SELECT MOD(11,0) FROM DUAL	3 11
. **NVL** (valor, expresión)	Si "valor" es NULL, lo sustituye por "expresión"; si no, devuelve "valor"	. SELECT SALARIO, COMISION, SALARIO + NVL(COMISION, 0) FROM EMPLE;	
. **POWER** (m, n)	Devuelve m^n	. SELECT POWER(2,3) FROM DUAL . SELECT POWER(3,2) FROM DUAL	8 9
. **ROUND** (numero [,n])	Redondea número con "n" decimales	. SELECT ROUND(1.56, 1) FROM DUAL . SELECT ROUND(1.56) FROM DUAL . SELECT ROUND(127.56, -1) FROM DUAL	1.6 2 130
. **SIGN** (valor)	Indica el signo de "valor"	. SELECT SIGN(8.2) FROM DUAL . SELECT SIGN(-8.2) FROM DUAL	1 -1
. **SQRT** (n)	Devuelve la raiz cuadrada de n	. SELECT SQRT(9) FROM DUAL	3
. **TRUNC** (numero [,m])	Trunca número con "n" decimales	. SELECT TRUNC(1.56, 1) FROM DUAL . SELECT TRUNC(1.56) FROM DUAL . SELECT TRUNC(127.56, -1) FROM DUAL	1.5 1 120
. **VARIANCE** ([DISTINCT] valor)	Devuelve la varianza de los valores	. SELECT VARIANCE(SALARIO) FROM EMPLE	
. **AVG** (n)	Calcula el valor medio de n ignorando los valores nulos	. SELECT AVG(EDAD) FROM EMPLE	

. **COUNT** (* | expresión)	Cuenta todas las filas (*) o las que no tienen valor nulo	. SELECT COUNT (*) FROM EMPLE 24 . SELECT COUNT (COMISION) FROM EMPLE 10 . SELECT COUNT (DISTINCT TEL) FROM EMPLE 16
. **MAX** (expresión)	Calcula el máximo valor de la expresión	. SELECT MAX (SUELDO) FROM EMPLE 2500
. **MIN** (expresión)	Calcula el mínimo valor de la expresión	. SELECT MIN (SUELDO) FROM EMPLE 600
. **SUM** (expresión)	Obtiene la suma de los valores de la expresión	. SELECT SUM (SUELDO) FROM EMPLE 130000
Funciones de listas		
. **GREATEST** (valor1, valor2...)	Obtiene el mayor valor de la lista	. SELECT GREATEST (NOTA1, NOTA2, NOTA3) FROM NOTAS
. **LEAST** (valor1, valor2...)	Obtiene el menor valor de la lista	. SELECT LEAST (NOTA1, NOTA2, NOTA3) FROM NOTAS
Funciones de cadenas de caracteres		
. **CHR** (n)	Devuelve el carácter de código ASCII n	. SELECT CHR (65) FROM DUAL A
. **ASCII** (cad)	Devuelve el código ASCII de la primera letra de 'cad'	. SELECT SCII ('Andoni') FROM DUAL 65
. **CONCAT**(cad1, cad2) ó cad' || cad'	Concatena 'cad1' con 'cad2'	. SELECT CONCAT ('Soy ',NOM) FROM EMPLE . SELECT 'Soy ' || NOM FROM EMPLE Soy LANDER
. **LOWER** (cad)	Devuelve 'cad' todo en minúsculas	. SELECT LOWER (NOM) FROM EMPLE lander
. **UPPER** (cad)	Devuelve 'cad' todo en mayúsculas	. SELECT UPPER (NOM) FROM EMPLE LANDER
. **INITCAP** (cad)	Devuelve el 1º carácter de cad en mayúsculas y el resto en minúsculas	. SELECT INITCAP (NOM) FROM EMPLE Lander
. **LPAD** (cad1, n [, cad2])	Añade a la izquierda de 'cad1' los caracteres de 'cad2' hasta tener una cadena de longitud n	. SELECT LPAD ('Lander', 8, '-') FROM DUAL --Lander . SELECT LPAD ('Lander', 8) FROM DUAL ' Lander'
. **RPAD** (cad1, n [, cad2])	Añade a la derecha de 'cad1' los caracteres de 'cad2' hasta tener una cadena de longitud n	. SELECT RPAD ('Lander', 8, '-') FROM DUAL Lander-- . SELECT RPAD ('Lander', 8) FROM DUAL 'Lander '
. **LTRIM** (cad [, set])	Devuelve 'cad' con el grupo de caracteres de 'set' omitidos por la izquierda	. SELECT LTRIM ('abaLLL', 'a') FROM DUAL bLLL . SELECT LTRIM (' abaLLL') FROM DUAL 'abaLLL'
. **RTRIM** (cad [, set])	Devuelve 'cad' con el grupo de caracteres	. SELECT RTRIM ('LLL;;;;', ';') FROM DUAL LLL

	de 'set' omitidos por la derecha	. SELECT RTRIM (' abaLLL ') FROM DUAL	' abaLLL'
. **REPLACE** (cad, cad1 [, cad2])	Devuelve 'cad' con cada ocurrencia 'cad1' sustituida por 'cad2'	. REPLACE ('abcdabe','ab','xx') FROM DUAL . REPLACE ('abcdabe','b') FROM DUAL	xxcdxxe 'a cda e'
. **SUBSTR** (cad, inicio [,n])	Devuelve el trozo de 'cad' que empieza en la posición *inicio* y tiene de longitud n	. SUBSTR ('ABCDEFG', 3, 2) FROM DUAL . SUBSTR ('ABCDEFG', -3, 2) FROM DUAL . SUBSTR ('ABCDEFG', 3) FROM DUAL	CD EF CDEFG
. **TRANSLATE** (cad1, cad2, cad3)	Devuelve 'cad1' con los caracteres encontrados en 'cad2' sustituidos por los caracteres de 'cad3'	. TRANSLATE ('abcbaf', 'ab', '12') . TRANSLATE ('abcbaf', 'ab')	12c21f 'cf'
. **INSTR** (cad1, cad2 [,comienzo [,m]])	Devuelve la posición de la m-ésima ocurrencia de 'cad2' en 'cad1' empezando la búsqueda en la posición *comienzo*	. INSTR ('Guadalupe', 'a', 2, 2) . INSTR ('Guadalupe', 'a', -5, 2) . INSTR ('Guadalupe', 'u')	5 3 2
. **LENGTH** (cad)	Devuelve el número de caracteres de 'cad'	. LENGTH ('Urritxe')	7
Funciones para manejo de fechas			
. **SYSDATE**	Devuelve la fecha del sistema	. SELECT SYSDATE FROM DUAL	02/10/02
. **ADD_MONTHS** (fecha, n)	Añade n meses a la fecha	. SELECT ADD_MONTHS ('17/01/02', 2) FROM DUAL	17/03/02
. **LAST_DAY** (fecha)	Devuelve la fecha del último día de ese mes	. SELECT LAST_DAY ('01/03/02') FROM DUAL	31/03/02
. **MONTHS_BETWEEN** (fecha1, fecha2)	Devuelve a diferencia en meses entre ambas fechas	. SELECT MONTHS_BETWEEN (SYSDATE, fechaNac)/12	Calcularía la edad
. **NEXT_DAY** (fecha, cad)	Devuelve la fecha del primer día de la semana indicado por 'cad' después de la fecha indicada por *fecha*	. SELECT NEXT_DAY (SYSDATE, ´JUEVES')	03/10/02

Funciones de conversión		
. **TO_CHAR** (FECHA, 'formato')	Convierte la fecha de tipo DATE a tipo VARCHAR2 con el formato especificado	Si SYSDATE=17/10/02, haciendo SELECT TO_CHAR... ... (SYSDATE, 'yyyy') FROM DUAL 2002 ... (SYSDATE, 'yy') FROM DUAL 02 ... (SYSDATE, 'q') FROM DUAL 3 ... (SYSDATE, 'mm') FROM DUAL 10 ... (SYSDATE, 'month') FROM DUAL OCTUBRE ... (SYSDATE, 'q') FROM DUAL 3 ... (SYSDATE, 'dd') FROM DUAL 17 ... (SYSDATE, 'mon') FROM DUAL OCT
. **TO_CHAR** (número, 'formato')	Convierte un número de tipo NUMBER a tipo VARCHAR2 con el formato especificado Máscaras: 9 → Devuelve el valor, sin ceros a la izquierda 0 → Muestra 0 si es 0, con ceros al principio $ → Muestra $ a la izquierda del valor B → Muestra espaciosi es 0 MI → Si es un valor negativo, - sigue al número S → + si es positivo, - si es negativo PR → Si es negativo se muestra entre < > D → Carácter decimal en la posición de D	Más máscaras: G → Carácter de miles en la posición de G C → Devuelve el símbolo ISO del territorio L → Devuelve el símbolo de la moneda , → Devuelve la coma en esa posición . → Devuelve el punto en esa posición V → Devuelve el valor multiplicado por 10^n, donde n es el número de nueves después de la 'V' EEEE → Devuelve el valor usando notación científica RN → Devuelve el valor en números romanos FM → Devuelve el valor alineado a la izquierda
. **TO_NUMBER** (cad [, 'formato'])	Convierte 'cad' a tipo NUMBER con el formato especificado. No puede haber espacios entre números ni otros caracteres, excepto el carácter decimal o el signo menos a la izquierda	. SELECT TO_NUMBER ('123,99', '999d99') FROM DUAL123,99
. **TO_DATE** (cad, 'formato')	Convierte 'cad' de tipo VARCHAR2 o CHAR, a un valor de tipo DATE con el formato especificado	. SELECT TO_DATE ('01012001', 'DD/MM/YYYY') 01/01/2001

Funciones e instrucciones de administración de Oracle		
. **CREATE USER** usu1 IDENTIFIED BY password **[DEFAULT TABLESPACE tablespace]** **[TEMPORARY TABLESPACE tablespace]** **[QUOTA {entero {K|M} | UNLIMITED}** **ON tablespace]** **[PROFILE perfil]**;	Creación del usuario USU1 con contraseña de acceso PASSWORD. TABLESPACEs por defecto y temporal. Tamaño de cada tablespace en Kbytes o Mbytes. Puede ser ilimitado. Asignación de un perfil para USU1.	. CREATE USER josu IDENTIFIED BY Josu DEFAULT TABLESPACE trabajo QUOTA 550K ON trabajo TEMPORARY TABLESPACE trabajo ; .CREATE USER jose IDIDENTIFIED BY Jose QUOTA 1M ON USERS ;
. **ALTER USER** usu1 IDENTIFIED BY password **[DEFAULT TABLESPACE tablespace]** **[TEMPORARY TABLESPACE tablespace]** **[QUOTA {entero {K|M} | UNLIMITED}** **ON tablespace]** **[PROFILE perfil]**;	Modificación de la definición del usuario USU1.	.ALTER USER usuario1 IDENTIFIED BY nuevaclave ; .ALTER USER usuario1 PROFILE nuevoperfil ;
. **DROP USER** usu1 [CASCADE]	Eliminación del usuario USU1. La opción CASCADE suprime todos los objetos del usuario antes de borrarlo	.DROP USER nombreusuario
. **GRANT {priv_obj1 [,priv_obj2]... | ALL}** **[(col1 [, col2]...)]** **ON [**usuario.]objeto TO {usuario1 | rol1 | PUBLIC} [,{usuario2 | rol2 | PUBLIC}...] **[WITH GRANT OPTION]** ;	Adjudica privilegios sobre objetos o columnas de objetos a un usuario o rol. PUBLIC adjudica los privilegios a todos los usuarios actuales o futuros. WITH GRANT OPTION permite que el receptor del privilegio o rol se lo asigne a otros usuarios o roles.	.GRANT SELECT ON tabla TO usuario1 ; .GRANT SELECT ON tabla TO nombre_ROL ; .GRANT SELECT ANY TABLE TO PUBLIC ; .GRANT SELECT ON tabla TO usuario1 WITH GRANT OPTION ; .GRANT UPDATE (columna) ON tabla TO usuario1 ;

. **GRANT {priv1 | rol1} [,{priv2 | rol2},...]** TO {usuario1 | rol1 | PUBLIC} [,{usuario2 | rol2 | PUBLIC}]... **[WITH ADMIN OPTION]** ;	Adjudica privilegios de sistema a un usuario o rol. PUBLIC adjudica los privilegios a todos los usuarios actuales o futuros. WITH ADMIN OPTION permite que el receptor del privilegio o rol pueda asignar esos mismos privilegios de administrador a otros usuarios o roles.	.GRANT CREATE SESSION TO usuario1 ; .GRANT CREATE SESSION TO usuario1 WITH ADMIN OPTION ; .GRANT CREATE SESSION TO nombre_ROL ; .GRANT nombre_ROL TO usuario1 ;
. **REVOKE {priv_obj1 [,priv_obj2]...| ALL]}** **ON [**usuario.]objeto FROM {usuario1 | rol1 | PUBLIC} [,{usuario2 | rol2 | PUBLIC}]...;	Retira privilegios sobre objetos o columnas de objetos a un usuario o rol. PUBLIC retira los privilegios a todos los usuarios actuales o futuros.	. REVOKE INSERT ON tabla FROM nombre_ROL/usuario :
. **REVOKE {priv1 | rol1} [,{priv2 | rol2}]...** FROM {usuario1 | rol1 | PUBLIC} [,{usuario2 | rol2 | PUBLIC}]...;	Retira privilegios de sistema a un usuario o rol. PUBLIC retira los privilegios a todos los usuarios actuales o futuros.	.REVOKE CREATE SESSION FROM nombre_ROL ; .REVOKE nombre_ROL FROM usuario ;
. **CREATE ROLE** rol [IDENTIFIED BY password];	Crea el rol ROL con, opcionalmente, contraseña PASSWORD. A este rol habrá que adjudicarle privilegios con la instrucción GRANT	
. **DROP ROLE** rol;	Elimina el rol ROL.	
. **ALTER USER** usu1 **DEFAULT** ROLE rol;	Adjudicación de un rol por defecto al usuario USU1	

. **CREATE PROFILE** perfil1 **LIMIT** {SESSIONS_PER USER | CPU_PER_SESSION | CPU_PER_CALL | CONNECT_TIME | IDLE_TIME | LOGICAL_READS_PER_SESSION | LOGICAL_READS_PER_CALL | PRIVATE_SGA | COMPOSITE_LIMIT } {**Entero** {K | M} | UNLIMITED | DEFAULT} [{SESSIONS_PER USER | CPU_PER_SESSION | CPU_PER_CALL | CONNECT_TIME | IDLE_TIME | LOGICAL_READS_PER_SESSION | LOGICAL_READS_PER_CALL | PRIVATE_SGA | COMPOSITE_LIMIT } {**Entero** {K | M} | UNLIMITED | DEFAULT}]… ;	Creación del perfil PERFIL1 con las limitaciones indicadas.	.CREATE PROFILE nombre_perfil LIMIT SESSIONS_PER_USER 1 ; .CREATE PROFILE nombre_perfil LIMIT CONNECT_TIME 5 ; .CREATE PROFILE nombre_perfil LIMIT FAILED_LOGIN_ATTEMPTS 3 ; .ALTER USER usuario ACCOUNT UNLOCK ;
. **DROP PROFILE** perfil1 [CASCADE];	Borra el perfil PERFIL1. CASCADE borrará los usuarios con ese perfil	

. **CREATE TABLESPACE** tablespace1 **DATAFILE** 'arch1' [SIZE entero [K|M] [REUSE] [,'arch2' [SIZE entero [K|M] [REUSE]… [DEFAULT STORAGE (INITIAL tamaño NEXT tamaño MINEXTENTS tamaño MEXEXTENTS tamaño PCTINCREASE valor)] [ONLINE | OFLINE] ;	Creación del tablespace TABLESPACE1	.CREATE TABLESPACE nombre DATAFILE 'nombre.ora' SIZE 5M ; .CREATE TABLESPACE nombre DATAFILE 'nombre.ora' SIZE 1M AUTOEXTEND ON NEXT 512K MAXSIZE UNLIMITED ;
. **ALTER TABLESPACE** tablespace1 { [**ADD DATAFILE** 'arch1' [SIZE entero [K|M] [REUSE] [AUTOEXTEND ON | OFF] [,'arch2' [SIZE entero [K|M] [REUSE] [AUTOEXTEND ON | OFF]…] [RENAME DATAFILE 'arch1' [,'arch2']… TO 'archi1' ['archi2']…] [DEFAULT STORAGE clausAlmacenamto] [ONLINE | OFFLINE] } ;	Modificación de la tablespace.	ALTER TABLESPACE nombre ADD DATAFILE 'nombre' SIZE 5M ; ALTER TABLESPACE nombre AUTOEXTEND ON NEXT 1M MAXSIZE UNLIMITED ; ALTER TABLESPACE nombre OFFLINE ; ALTER TABLESPACE nombre ONLINE ;
. **DROP TABLESPACE** tbalespace1 [INCLUDING CONTENTS] ;	Elimina el tablespace. INCLUDING CONTENTS permite borrar un tablespace que tenga datos.	

CREATE OR REPLACE
PROCEDURE 'nombre' (variable NUMBER)
AS
var1 VARCHAR2(14); var2 VARCHAR2(14);
BEGIN
SELECT col.1 col2. INTO var1, var2
FROM tabla WHERE col = num
DBMS_OUTPUT.PUT_LINE ('text1' ||
variable || 'text2' || var1 || 'text3' || var2);
EXCEPTION
Aqui van las excepciones
END 'nombre' ; /
	Creación de un procedimiento	**EXCEPTION**
		WHEN TOO_MANY_ROWS THEN RAISE_APLICATION_ERROR ('texto1')
		WHEN NO_DATA_FOUND THEN RAISE_APLICATION_ERROR ('texto2')
		WHEN OTHERSTHEN RAISE_APLICATION_ERROR ('texto3')
		
		v_oficio empleados.oficio**%TYPE**;
		v_apellidos := '&vs_apellidos';
		v_dato:=&dato

CREATE OR REPLACE
FUNCTION 'nombre' (dato1 NUMBER, dato2 NUMBER)
RETURN REAL
AS
var NUMBER(2);
BEGIN
var := dato1 + dato2; **RETURN** var;
EXCEPTION
Aqui van las excepciones;
END 'nombre' ; /
	Creación de una función	A esta función se llamaría con un procedimiento que llama a la función, ejemplo:
		EXECUTE DBMS_OUTPUT.PUT_LINE (suma (10,20));
		
		Llamada a la función con otro programa (entorno gráfico)
		BEGIN DBMS_OUTPUT.PUT_LINE (suma (10,20)); **END;/**

DECLARE aquí declaro las variables, constantes etc. (opcional) **BEGIN** conjunto de instrucciones a ejecutar **EXCEPTION** Aquí van las excepciones (opcional) **END** ;	Estructura bloque PL/SQL	

IF <condición> **THEN**
instrucciones;
ELSIF <condicion2> **THEN**
instrucciones;
ELSIF <condicion3> **THEN**
instrucciones;
…..
ELSE
instrucciones;
END IF;
	Alternativa múltiple IF ELSIF	**IF** <condición> **THEN**
		instrucciones;
		END IF;
		
		IF <condición> **THEN**
		instrucciones;
		ELSE
		END IF;
			Alternativa simple IF
			
			Alternativa doble IF ELSE

WHILE <condición> **LOOP** instrucciones; **END LOOP;**	Estructura repetitiva WHILE	**CASE [<expresión>]** **WHEN** <test1> **THEN** <instrucciones1>; **WHEN** <test2> **THEN** <instrucciones2>; **WHEN** <test3> **THEN** <instrucciones3>; [**ELSE** <otras-instrucciones>;] **END CASE;**	Alternativa múltiple con CASE (de comprobación)
FOR <variablecontrol> **IN** <inicio>...<final> **LOOP** instrucciones; **END LOOP**; **FOR** <variablecontrol> **IN REVERSE** <inicio>...<final> **LOOP** instrucciones; **END LOOP**;	Estructura FOR Estructura FOR en incrementos negativos	**CASE** **WHEN** <condicion1> **THEN** <instrucciones1>; **WHEN** <condicion2> **THEN** <instrucciones2>; **WHEN** <condicion3> **THEN** <instrucciones3>; [**ELSE** <otras-instrucciones>;] **END CASE;**	Alternativa múltiple con CASE (de busqueda)

Funciones Aritméticas		
Valores simples:num, vble, col		
ABS(n)	- Valor absoluto de n	
CEIL(n)	-Sgte valor entero >= a n	
FLOOR(n)	-Sgte valor entero <= a n	
MOD(m,n)	-Resto de m entre n	
NVL(valor, expr)	-Susti valor nulo por otro	
POWER(m, exp)	-Potencia de un num	
ROUND(num, [,m]	-Redondea num con m decimales. Si es -, redondea parte entera	ROUND (1.564,1)=1,6; ROUND (1.564,1)=1,6; ROUND (145.5,-1)=150; ROUND (141,-2)=100;
SIGN(valor)	-Devuelv el signo de valor	
SQRT(n)	-Raiz de n	
TRUNC(num, [,m]	-Trunca num y lo deja con m decimales. Si m – trunca por izq de decimal	TRUNC (1.563,2)=1.56 TRUNC (178.5,-2)=100
Grupos de valores:Actuan sobre un grupo de filas para obtener el valor. Ignoran NULLS		
AVG (n)	-Media de "n".	
COUNT (* | expr)	-Cuenta nº veces que hay NO NULLS, ed, hay datos	-COUNT (*) → Cuenta nº de filas - COUNT (comision)→ nº de comisiones no nulas
MAX (expr)	-Máximo valor de la expr	
MIN (expr)	-Minimo valor de la expr	
SUM (expr)	-Suma valores de 1 col	
DISTINCT		
VARIANCE(Valor)	-Varianza de cjto de valores	
De listas: Trabajan sobre un grupo de columnas dentro de la misma fila		
GREATEST(v1,v2..)	- > valor de la lista	- GREATEST (nota1, nota2, nota3)
LEAST(v1,v2, v3...)	- < valor de la lista	– LEAST ('Benito', 'Julio', 'Andres') = 'Andres'

Funciones de cadenas de caracteres		
Funciones que devuelven **valores carácter**		
CHR (n)	-Da carácter de un ASCII	- CHR(65) = 'A'
CONCAT(cad1,cad2)	-Une 2 cadenas de caracteres. Si quiero unir más cadenas, anido concats	- CONCAT ('El apellido es ', APELLIDO). OPER || - CONCAT(CONCAT(apellido, 'es'), oficio) select apellido || 'es' || oficio from emple
UPPER (cad)	- Convierte cad a mayúsc	
LOWER (cad)	Convierte cad a minúscul	
INITCAP(cad)	-1ªletra de cada palabra mayúsculas. Resto a minus	
LPAD(cd1,n,[,cad2]	- Rellena la cadena cd1 con cad2 a la izq o dcha dejando en total n caract	LPAD ('hola',10,'.-') = .-.-.-hola. Si se suprime cad2
RPAD(cd1,n,[,cad2]		RPAD ('apellido,7,'*') = Diz***** se rellena a blancos
LTRIM(cad [,set)	-Omite la cadena set desde la izq o dcha.Si omite set y hay blancos, se omiten. No palabras.	LTRIM (RTRIM(titulo, '.'"), ' " ') from mistextos Quita de la decha el punto y comilla y de la izq, la comilla
RTRIM(cad [,set)		
REPLACE(cad,cbus, [cadsust])	-Sustituye en cad, cbus por cadbus. Si se omie cadbus, lo sustituye por nada Busca cadenas enteras y reemplaza por cadsust. Translate busca caracteres por posición	- REPLACE ('OGRO', 'O', 'AS') → ASGRAS - TRANSLATE ('OGRO', 'O', 'AS') → AGRA - REPLACE ('OGRON', 'ON', 'AS') → OGRAS -TRANSLATE ('OGRON', 'ON', 'AS') → AGRAS

SUBSTR(Cad,m[,n]	-A partir de m, da n caracteres de cad. n>1 y m + ó -	
TRANSLATE(cad1;cad2; cad3)	- Sustituye en cad1, los caract de cad2 por los de cad3 según su posición	TRANSLATE ('SQLPLUS', 'SQL', '123') = 123P3U1
SOUNDEX(cad)		
Funciones que devuelven **valores numéricos**		
ASCII (cad)	-Da ASCII del 1er carácter de cad	- ASCII('A') = 65
INSTR(cad1,cad2 [,com [, m]])	-Da la posición de la m-sima ocurrenc de cad2 en cad1 empezando por com	- INSTR ('Abracadabra ', 'bra', 2, 2) = 9 - INSTR ('Abracadabra ', 'BRA, 2, 2) = 0 - INSTR ('Abracadabra ', 'bra') = 2
LENGTH (cad)	- Nº caracteres de cad	- Si cad tipo char, la long es la fijad en el diseño
Funciones para **manejo de fechas.** *las fechas entre comillas*		
SYSDATE	- Da fecha actual (hoy)	- SELECT SYSDATE FROM DUAL;
ADD_MONTHS (fecha, n)	-Da la fecha incrementada en n meses	- ADD_MONTHS (FECHA_ALT, 2). Si n es negativo, resta meses a la fecha
LAST-DAY (fecha)	-Da último dia del mes q se indica en fecha	-LAST_DAY('5/2/01)=28/2/01;LAST_DAY (FECH_ALT) - LAST_DAY ('5/2/00) = 29/2/00
MONTHS_BETWEEN (fech1,fech2)	- Da diferencia de meses entre dos fechas	- MONTHS_BETWEEN('5/5/2000', '1/1/2000')=4,1 - MONTHS_BETWEEN(SYSDATE,'21/12/70') / 12
NEXT_DAY (fecha, cad)	- Da la sgte fecha que toque el dia indicada en cad	- NEXT_DAY (SYSDATE, 'domingo') Dara la fecha del sgte domingo
Funciones de **conversión**		
TO_CHAR (fecha, formato)	-Convierte un date a varchar2 en el formato especificado	- TO_CHAR ('12/enero/04', 'month DD, YYYY') = enero 12, 2004 - TO_CHAR (SYSDATE, ' "Hoy es" dd " de " month " de " yyyy ') = Hoy es 3 de octubre de 2004
TO_CHAR (numero, formato)	-Convierte un number a varchar2 en el formato especificado	- TO_CHAR ('12/enero/04', 'month DD, YYYY') = enero 12, 2004 - TO_CHAR (SYSDATE, ' "Hoy es" dd " de " month " de " yyyy ') = Hoy es 3 de octubre de 2004
TO_NUMBER (cadena [,formato])		
TO_DATE (cadena, 'formato')	- Convierte cad de tipo char o varchar2 a date	
Otras **funciones**		
DECODE (variable, valo1, codigo1, valo2, codigo2, ..., valor por defecto)	- Sustituye un valor por otro. Es como una case o como un IF	- SELECT APELLIDO, DECODE(UPPER(OFICIO, 'PRESIDENTE', 1, 'EMPLEADO', 2, 5). Saca el apellido y si es presi un 1, si es emple un 2 y si no un 5. En vez de 5, si ponemos oficio, nos saldrá el oficio que tenía
TO_NUMBER	- Convierte un char o date a number	

MÁSCARAS DE FORMATO NUMERICAS	
yyyy	Año sin signo
yyy	Ultimos 3 digitos del año
yy	Ultimos 2 digitos del año
y	Ultimos digito del año
q	Numero de trimestre
ww	Número de semana del año
w	Numero de semana del mes
mm	Número de mes
ddd	Número del dia del año
dd	Número de dia del mes
d	Número de dia de la semana

MASCARAS DE FORMATO DE CARACTERES	
Year	Año
Month	Nombre del mes (Enero)
Mon	Abreviatura de tres letras del nombre del mes
Day	Nombre del dia de la semana (Lunes)
Dy	Abreviatura de tres letras del nombre del dia (Lun)