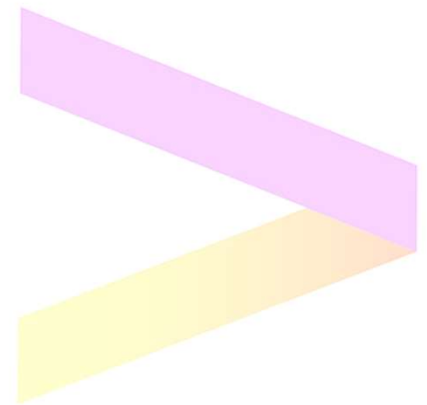




# Clase 7

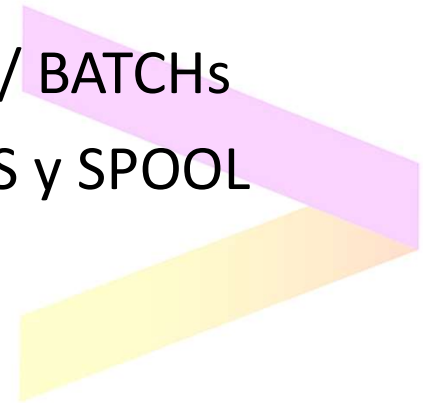
## Batch





# Temario

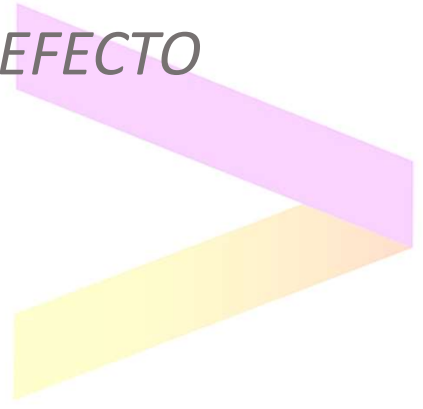
- 01-Usuarios
- 02-Permisos
- 03-Crear Base de Datos Oracle
- 04-Secuencias
- 05-Indices
- 06-Procedure
- 07-Funciones
- 08-If
- 09-Case
- 10-Loop
- 11-For
- 12-While
- 13-Trigger
- 14-Cursores
- 15 – SHELLs / BATCHs
- 16 - SQL PLUS y SPOOL



## 01 - USUARIOS

- Puede haber varios usuarios diferentes de la base de datos. Cada uno es propietario de sus objetos.
- Para crear un usuario debemos conectarnos a la base de datos como administradores (por ejemplo "system").
- Sintaxis básica para crear un usuario:

```
create user NOMBREUSUARIO identified by CONTRASEÑA  
default tablespace NOMBRETABLESPACEPORDEFECTO  
quota CANTIDAD on TABLESPACE;  
** [default role ROLE, ALL];
```

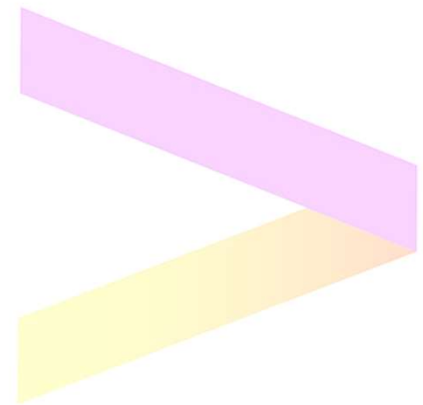




- Ejemplos:

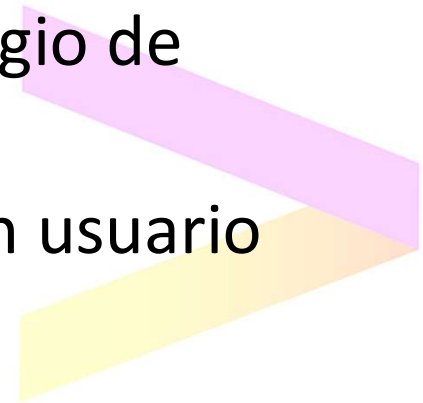
*create user ana identified by anita;*

*create user juan identified by juancito  
default tablespace system  
quota 100M on system;*



## 02 - PERMISOS

- Los usuarios necesitan permisos para poder acceder a la base de datos y a los objetos de la misma.
- Los privilegios pueden ser de dos tipos: del sistema y sobre objetos.
- Como mínimo, un usuario debe tener permiso para conectarse.
- El permiso "create session" es un privilegio de sistema.
- Para conceder permiso de conexión a un usuario empleamos la instrucción "grant".



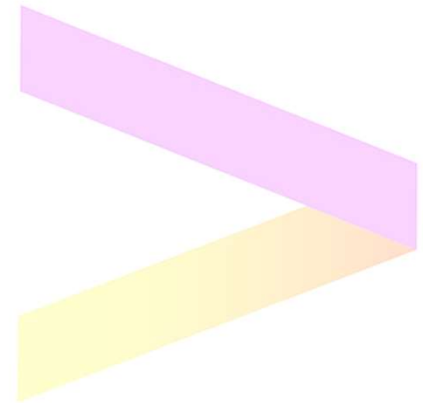


- Sintaxis básica:

*grant create session  
to USUARIO;*

- En el siguiente ejemplo concedemos al usuario "juan" permiso para conectarse:

- *grant create session to juan;*



## 03 – CREACION DE BD

- **Ejemplo:**

```
CREATE DATABASE sample
CONTROLFILE REUSE
LOGFILE
  GROUP 1 ('diskx:log1.log',
'disky:log1.log') SIZE 50K,
  GROUP 2 ('diskx:log2.log',
'disky:log2.log') SIZE 50K
MAXLOGFILES 5
MAXLOGHISTORY 100
MAXDATAFILES 10
MAXINSTANCES 2
ARCHIVELOG
```

```
CHARACTER SET AL32UTF8
NATIONAL CHARACTER SET
AL16UTF16
DATAFILE
  'disk1:df1.dbf' AUTOEXTEND
ON,
  'disk2:df2.dbf' AUTOEXTEND ON
NEXT 10M MAXSIZE UNLIMITED
DEFAULT TEMPORARY
TABLESPACE temp_ts
UNDO TABLESPACE undo_ts
SET TIME_ZONE = '+02:00';
```



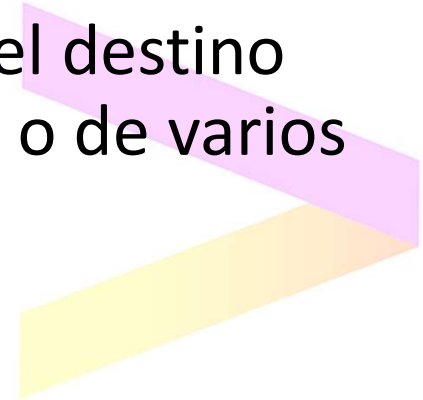
- **CLAUSULAS**

- Especifique el nombre de la base de datos que se creará. El nombre debe coincidir con el valor del parámetro de inicialización DB\_NAME . El nombre puede tener hasta 8 bytes de longitud y solo puede contener caracteres ASCII. Oracle Database escribe este nombre en el archivo de control.
- **Clausula USER SYS ..., USER SYSTEM ...**
- Use estas cláusulas para establecer contraseñas para los usuarios SYS y SYSTEM . Estas cláusulas no son obligatorias en esta versión. Sin embargo, si especifica una cláusula, debe especificar ambas cláusulas.





- **Clausula CONTROLFILE REUSE**
- Especifique el CONTROLFILE para reutilizar los archivos de control existentes identificados por el parámetro de inicialización CONTROL\_FILES , sobrescribiendo la información que contienen actualmente. Normalmente, utiliza esta cláusula solo cuando está recreando una base de datos, en lugar de crear una por primera vez. Cuando crea una base de datos por primera vez, Oracle Database crea un archivo de control en el destino predeterminado, que depende del valor o de varios parámetros de inicialización.



- **Cláusula MAXDATAFILES**
- Especifica el tamaño inicial de la sección de archivos de datos del archivo de control en CREATE DATABASE o CREATE CONTROLFILE . Al agregar un archivo cuyo número es mayor que MAXDATAFILES pero inferior o igual a DB\_FILES hace que el archivo de control de Oracle Database se expanda automáticamente para que la sección de archivos de datos pueda acomodar más archivos.



# accenture

- **Cláusula MAXINSTANCES**
- Especifique la cantidad máxima de instancias que pueden tener simultáneamente esta base de datos montada y abierta. Este valor tiene prioridad sobre el valor del parámetro de inicialización INSTANCES . El valor mínimo es 1. Los valores máximo y predeterminado dependen de su sistema operativo.
- 
- **Clausula CHARACTER SET**
- Especifique los CHARACTER SET que la base de datos usa para almacenar datos. Los CHARACTER SET compatibles y el valor predeterminado de este parámetro dependen de su sistema operativo.



- **NATIONAL CHARACTER SET**
- Especifique el NATIONAL CHARACTER SET utilizado para almacenar datos en columnas específicamente definidas como NCHAR , NCLOB o NVARCHAR2 . Los valores válidos son AL16UTF16 y UTF8 . El valor predeterminado es AL16UTF16 .
- 
- **Clausula Database logging**
- Usa database\_logging\_clauses para determinar cómo la base de datos Oracle manejará los archivos de registro (Logs).



# accenture

- **Cláusula LOGFILE**

- Especifique uno o más archivos para usar como logs. Usa el formulario *redo\_log\_file\_spec* de *file\_specification* para crear archivos logs en un sistema de archivos del sistema operativo.

- 

- **Cláusula MAXLOGFILES**

- Especifique la cantidad máxima de grupos de archivos logs que pueden crearse para la base de datos. Los valores predeterminados, mínimos y máximos dependen de su sistema operativo.



- **ARCHIVELOG**

- Especifica ARCHIVELOG si desea que los contenidos de un grupo de logs se archiven antes de que el grupo pueda reutilizarse. Esta cláusula se prepara para la posibilidad de recuperación de medios.

- **Cláusula MAXLOGMEMBERS**

- Especifique la cantidad máxima de miembros o copias para un grupo de archivos log. El valor mínimo es 1. Los valores máximo y predeterminado dependen de su sistema operativo.





- **Cláusula de MAXLOGHISTORY**
- Este parámetro es útil solo si está utilizando Oracle Database en el modo de registro de archivo con Real Application Clusters. Especifique la cantidad máxima de archivos de registro redo archivados para la recuperación automática de medios de Real Application Clusters. La base de datos usa este valor para determinar cuánto espacio asignar en el archivo de control para los nombres de los logs archivados. El valor mínimo es 0. El valor predeterminado es un múltiplo del valor MAXINSTANCES y depende de su sistema operativo. El valor máximo está limitado solo por el tamaño máximo del archivo de control.



- **NOARCHIVELOG**
- Especifique NOARCHIVELOG si el contenido de un grupo de logs no necesita ser archivado antes de que el grupo pueda ser reutilizado. Esta cláusula no permite la posibilidad de recuperación de medios.
- **FORCE LOGGING**
- Use esta cláusula para poner la base de datos en el modo FORCE LOGGING . Oracle Database registrará todos los cambios en la base de datos, excepto los cambios en espacios de tablas temporales y segmentos temporales.





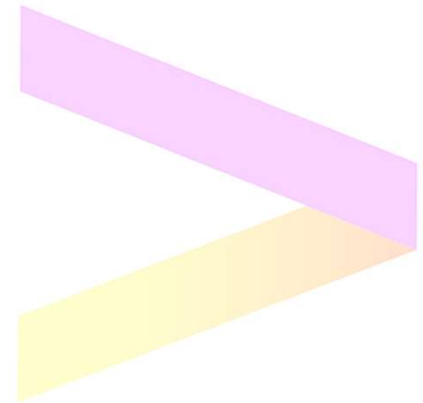
# accenture

- **tablespace\_clauses**
- Utilice las cláusulas de espacio de tabla para configurar los SYSAUX tabla SYSTEM y SYSAUX y para especificar un espacio de tabla temporal predeterminado y un tablespace de deshacer.
- **extent\_management\_clause**
- Use esta cláusula para crear un espacio de tabla del SYSTEM administrado localmente. Si omite esta cláusula, el espacio de tabla SYSTEM será administrado por diccionario.



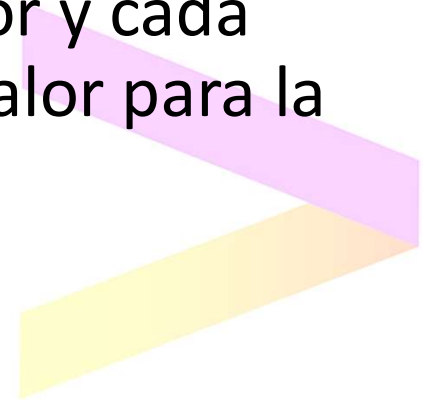


- **Cláusula SYSAUX**
- Oracle Database crea los SYSAUX tabla SYSTEM y SYSAUX como parte de cada base de datos. Utilice esta cláusula si no está utilizando archivos administrados por Oracle y desea especificar uno o más archivos de datos para el espacio de SYSAUX .



## 04 - SECUENCIAS

- Una secuencia (sequence) se emplea para generar valores enteros secuenciales únicos y asignárselos a campos numéricos; se utilizan generalmente para las claves primarias de las tablas garantizando que sus valores no se repitan.
- Una secuencia es una tabla con un campo numérico en el cual se almacena un valor y cada vez que se consulta, se incrementa tal valor para la próxima consulta.





- Sintaxis general:

*create sequence NOMBRESECUENCIA*

*start with VALORENTERO*

*increment by VALORENTERO*

*maxvalue VALORENTERO*

*minvalue VALORENTERO*

*cycle / nocycle;*

- - La cláusula "start with" indica el valor desde el cual comenzará la generación de números secuenciales. Si no se especifica, se inicia con el valor que indique "minvalue".
- - La cláusula "increment by" especifica el incremento, es decir, la diferencia entre los números de la secuencia; debe ser un valor numérico entero positivo o negativo diferente de 0. Si no se indica, por defecto es 1.



- le la



- será 1, el  
999999 y

## > accenture

- En el siguiente ejemplo creamos una secuencia llamada "sec\_codigolibros", estableciendo que comience en 1, sus valores estén entre 1 y 99999 y se incrementen en 1, por defecto, será "nocycle":

*create sequence sec\_codigolibros*

*start with 1*

*increment by 1*

*maxvalue 99999*

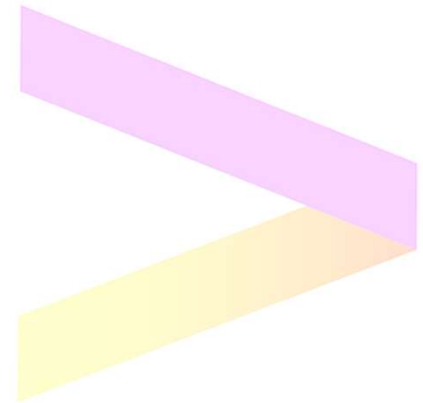
*minvalue 1;*

- Si bien, las secuencias son independientes de las tablas, se utilizarán generalmente para una tabla específica, por lo tanto, es conveniente darle un nombre que referencie a la misma.



- Para recuperar los valores de una secuencia empleamos las pseudocolumnas "currval" y "nextval".
- Primero debe inicializarse la secuencia con "nextval". La primera vez que se referencia "nextval" retorna el valor de inicio de la secuencia; las siguientes veces, incrementa la secuencia y nos retorna el nuevo valor:

```
NOMBRESECUENCIA.NEXTVAL;
```







- Para recuperar el valor actual de una secuencia usamos:

`NOMBRESECUENCIA.CURRVAL;`

- Los valores retornados por "currval" y "nextval" pueden usarse en sentencias "insert" y "update".





- Veamos un ejemplo completo:
- Creamos una secuencia para el código de la tabla "libros", especificando el valor máximo, el incremento y que no sea circular:

```
create sequence sec_codigolibros  
maxvalue 999999  
increment by 1  
nocycle;
```

- Luego inicializamos la secuencia

```
select sec_codigolibros.nextval from dual;
```

- Recuerde que la primera vez que se reference la secuencia debe emplearse "nextval" para inicializarla.



# accenture

- Ingresamos un registro en "libros", almacenando en el campo "codigo" el valor actual de la secuencia:

```
insert into libros values
```

```
(sec_codigolibros.currval, 'El aleph', 'Borges', 'Emece');
```

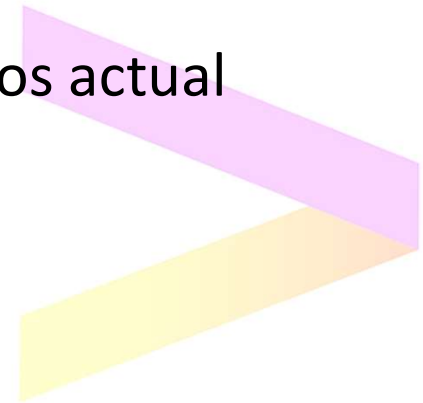
- Ingresamos otro registro en "libros", almacenando en el campo "codigo" el valor siguiente de la secuencia:

```
insert into libros values
```

```
(sec_codigolibros.nextval, 'Matematica estas ahi',  
'Paenza', 'Nuevo siglo');
```

- Para ver todas las secuencias de la base de datos actual realizamos la siguiente consulta:

```
select *from all_sequences;
```



## 05 - INDICES

- Otros objetos de base de datos son los índices.
- Los índices sirven para acceder a los registros de una tabla rápidamente, acelerando la localización de la información.
- Los índices se emplean para facilitar la obtención de información de una tabla. El índice de una tabla desempeña la misma función que el índice de un libro: permite encontrar datos rápidamente; en el caso de las tablas, localiza registros.

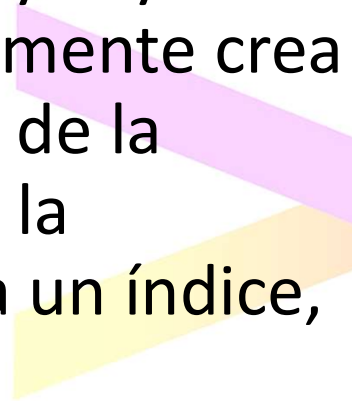


- Oracle accede a los datos de dos maneras:
  - 1) recorriendo las tablas; comenzando el principio y extrayendo los registros que cumplen las condiciones de la consulta; lo cual implica posicionar las cabezas lectoras, leer el dato, controlar si coincide con lo que se busca (como si pasáramos una a una las páginas de un libro buscando un tema específico).
  - 2) empleando índices; recorriendo la estructura de árbol del índice para localizar los registros y extrayendo los que cumplen las condiciones de la consulta (comparando con un libro, diremos que es como leer el índice y luego de encontrar el tema buscado, ir directamente a la página indicada).



- Un índice posibilita el acceso directo y rápido haciendo más eficiente las búsquedas. Sin índice, Oracle debe recorrer secuencialmente toda la tabla para encontrar un registro.
- Los índices son estructuras asociadas a tablas, una tabla que almacena los campos indexados y se crean para acelerar las consultas.
- Entonces, el objetivo de un índice es acelerar la recuperación de información. La indexación es una técnica que optimiza el acceso a los datos, mejora el rendimiento acelerando las consultas y otras operaciones. Es útil cuando la tabla contiene miles de registros, cuando se realizan operaciones de ordenamiento y agrupamiento y cuando se combinan varias tablas (tema que veremos más adelante).
- La desventaja es que consume espacio en el disco y genera costo de mantenimiento (tiempo y recursos).

# **accenture**

- Es importante identificar el o los campos por los que sería útil crear un índice, aquellos campos por los cuales se realizan búsquedas con frecuencia: claves primarias, claves externas o campos que combinan tablas.
  - No se recomienda crear índices sobre campos que no se usan con frecuencia en consultas o en tablas muy pequeñas.
  - Cuando creamos una restricción "primary key" o "unique" a una tabla, Oracle automáticamente crea un índice sobre el campo (o los campos) de la restricción y le da el mismo nombre que la restricción. En caso que la tabla ya tenga un índice, Oracle lo usa, no crea otro.
- 

# accenture

- Para crear índices empleamos la instrucción "create index".
- La sintaxis básica es la siguiente:

```
create TIPOdeINDICE index NOMBREINDICE  
on NOMBRETABLA(CAMPOS);
```

- En el siguiente ejemplo creamos un índice único sobre el campo "documento" de la tabla "empleados"

```
create unique index I_empleados_documento  
on empleados(documento);
```

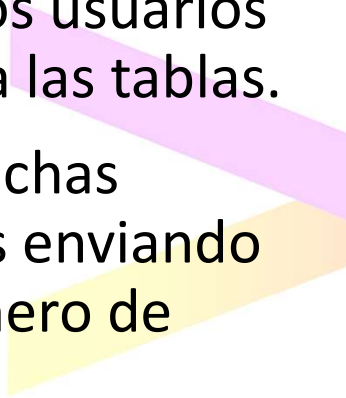
- Los índices se eliminan con "drop index"; la siguiente es la sintaxis básica:

```
drop index NOMBREINDICE;
```





## 06 - PROCEDURES

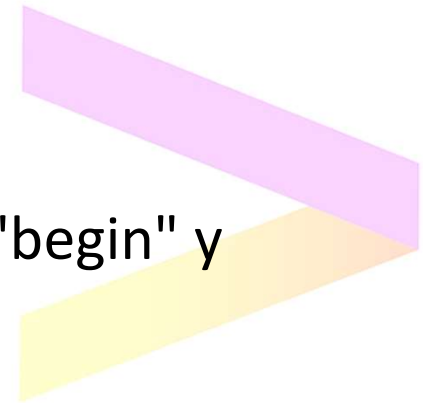
- Un procedimiento almacenado es un conjunto de instrucciones a las que se les da un nombre, se almacena en la base de datos activa. Permiten agrupar y organizar tareas repetitivas.
  - Ventajas:
    - - comparten la lógica de la aplicación con las otras aplicaciones, con lo cual el acceso y las modificaciones de los datos se hacen en un solo sitio.
    - - permiten realizar todas las operaciones que los usuarios necesitan evitando que tengan acceso directo a las tablas.
    - - reducen el tráfico de red; en vez de enviar muchas instrucciones, los usuarios realizan operaciones enviando una única instrucción, lo cual disminuye el número de solicitudes entre el cliente y el servidor.
- 
- A decorative graphic element consisting of two overlapping triangles pointing to the right. The top triangle is light purple and the bottom triangle is light yellow.

# accenture

- Un procedimiento almacenado pueden incluir cualquier cantidad y tipo de instrucciones DML (de manipulación de datos, como insert, update, delete), no instrucciones DDL (de definición de datos, como create..., drop... alter...).
- Para crear un procedimiento almacenado empleamos la instrucción "create procedure". La sintaxis básica parcial es:

```
create or replace procedure NOMBREPROCEDIMIENTO  
as  
begin  
INSTRUCCIONES  
end;
```

- El bloque de instrucciones comienza luego de "begin" y acaba con "end".





- Con las siguientes instrucciones creamos un procedimiento almacenado llamado "pa\_libros\_aumentar10" que incrementa en un 10% el precio de todos los libros:

```
create procedure pa_libros_aumentar10  
as
```

```
update libros set precio=precio+precio*0.1;
```

- Para ejecutar el procedimiento almacenado creado anteriormente tipeamos:

```
execute pa_libros_aumentar10;
```



# accenture

- Los procedimientos almacenados se eliminan con "drop procedure". Sintaxis:

*drop procedure NOMBREPROCEDIMIENTO;*

- Eliminamos el procedimiento almacenado llamado "pa\_libros\_aumentar10":

*drop procedure pa\_libros\_aumentar10;*

- Los procedimientos almacenados pueden recibir y devolver información; para ello se emplean parámetros.





- Veamos los primeros. Los parámetros de entrada posibilitan pasar información a un procedimiento. Para que un procedimiento almacenado admita parámetros de entrada se deben declarar al crearlo. La sintaxis es:

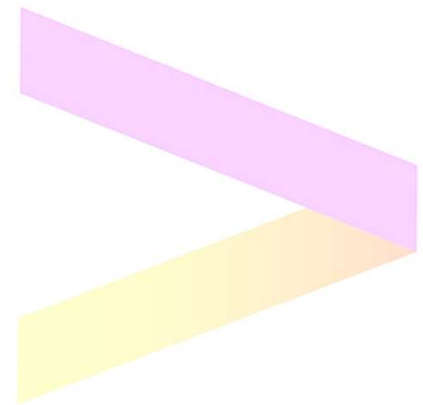
```
create or replace procedure  
NOMBREPROCEDIMIENTO (PARAMETRO in  
TIPODEDATO)
```

```
as
```

```
begin
```

```
INSTRUCCIONES;
```

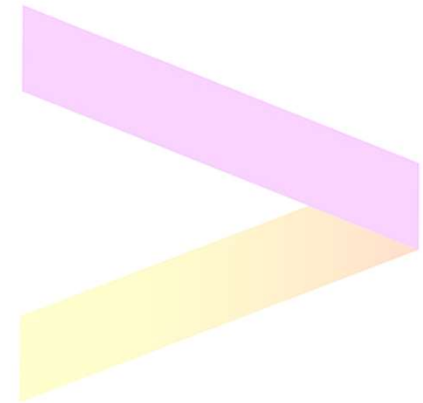
```
end;
```



# accenture

- Creamos otro procedimiento que recibe 2 parámetros, el nombre de una editorial y el valor de incremento (que tiene por defecto el valor 10):

```
create or replace procedure  
pa_libros_aumentar(aeditorial in  
varchar2,aporcentaje in number default 10)  
as  
begin  
    update libros set  
    precio=precio+(precio*aporcentaje/100)  
    where editorial=aeditorial;  
end;
```



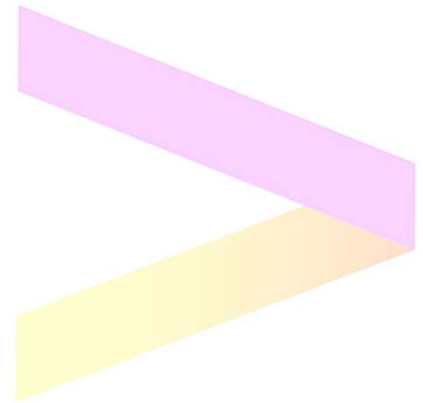
## **accenture**

- El procedimiento se ejecuta colocando "execute" (o "exec") seguido del nombre del procedimiento y los valores para los parámetros separados por comas:

*execute pa\_libros\_aumentar('Planeta',30);*

- Podemos omitir el segundo parámetro al invocar el procedimiento porque tiene establecido un valor por defecto:

*execute pa\_libros\_aumentar('Planeta');*



# **accenture**

- Los procedimientos almacenados pueden contener en su definición, variables locales, que existen durante el procedimiento.
- La sintaxis para declarar variables dentro de un procedimiento almacenado es la siguiente:

```
create or replace procedure NOMBREPROCEDIMIENTO  
(PARAMETRO in TIPODEDATO)
```

```
as
```

```
NOMBREVARIABLE TIPO;
```

```
begin
```

```
INSTRUCCIONES;
```

```
end;
```

- Las variables se definen antes del bloque de sentencias; pueden declararse varias.






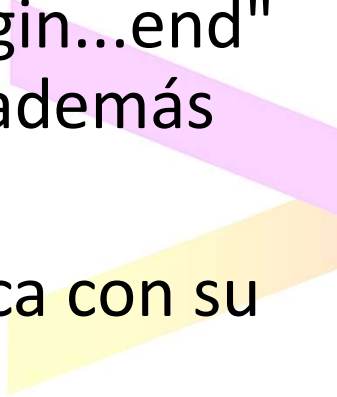
# > accenture

- Creamos un procedimiento que recibe el nombre de un libro, en una variable guardamos el nombre del autor de tal libro, luego buscamos todos los libros de ese autor y los almacenamos en una tabla:

```
create or replace procedure
pa_autorlibro(atitulo in varchar2)
as
  vautor varchar2;
begin
  vautor= select autor from libros where
titulo=atitulo;
drop table tabla1;
create table tabla1(
  titulo varchar2(40),
  precio number(6,2));
insert into tabla1
  select titulo,precio
  from libros
  where autor=vaautor;
end;
```



Ejecutamos el procedimiento: execute  
pa\_autorlibro('Ilusiones');

- **FUNCIONES:**
  - Las funciones, como los procedimientos almacenados son bloques de código que permiten agrupar y organizar sentencias SQL que se ejecutan al invocar la función.
  - Las funciones tienen una estructura similar a la de los procedimientos. Como los procedimientos, las funciones tienen una cabecera, una sección de declaración de variables y el bloque "begin...end" que encierra las acciones. Una función, además contiene la cláusula "return".
  - Una función acepta parámetros, se invoca con su nombre y retorna un valor.
- 
- A large, stylized arrow pointing to the right, composed of two overlapping triangles. The top triangle is light purple and the bottom triangle is light yellow.

# accenture

- Para crear una función empleamos la instrucción "create function" o "create or replace function". Si empleamos "or replace", se sobrescribe (se reemplaza) una función existente; si se omite y existe una función con el nombre que le asignamos, Oracle mostrará un mensaje de error indicando tal situación.
- La sintaxis básica parcial es:

```
create o replace function NOMBREFUNCION(PARAMETRO1 TIPODATO,  
PARAMETRON TIPODATO)
```

```
return TIPODEDATO is
```

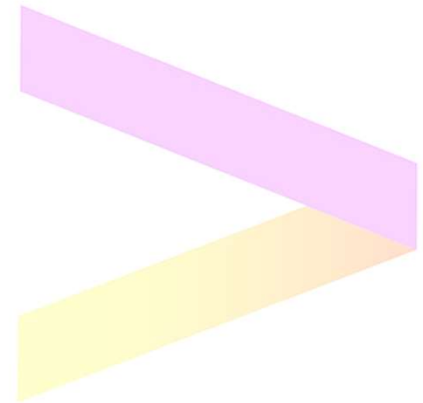
```
DECLARACION DE VARIABLES
```

```
begin
```

```
ACCIONES;
```

```
return VALOR;
```

```
end;
```



# accenture

- La siguiente funcion recibe 1 parámetro, un valor a incrementar y retorna el valor ingresado como argumento con el incremento del 10%:

*create or replace function f\_incremento10 (avalor  
number)*

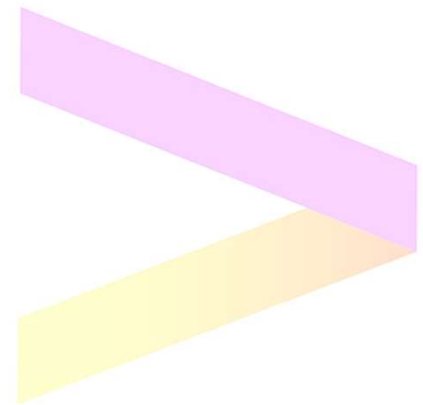
*return number*

*is*

*begin*

*return avalor+(avalor\*0.1);*

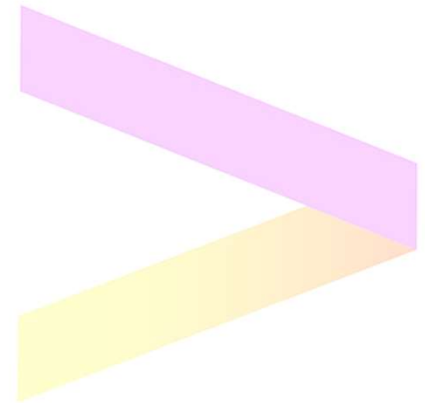
*end;*





- Podemos emplear las funciones en cualquier lugar en el que se permita una expresión en una sentencia "select", por ejemplo:

```
select titulo,precio,f_incremento10(precio) from  
libros;
```



- "if... else" testea una condición; se emplea cuando un bloque de sentencias debe ser ejecutado si una condición se cumple y si no se cumple, se debe ejecutar otro bloque de sentencias diferente.
- Sintaxis:

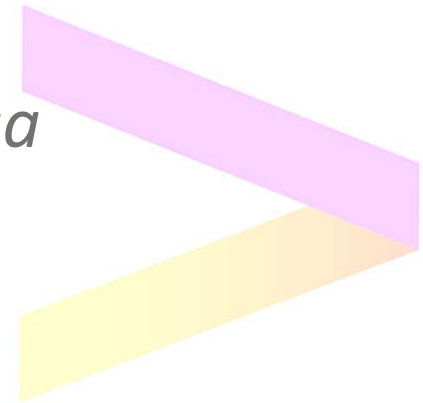
*if (CONDICION) then*

*SENTENCIAS-- si la condición se cumple*

*else*

*SENTENCIAS-- si la condición resulta falsa*

*end if;*



# accenture

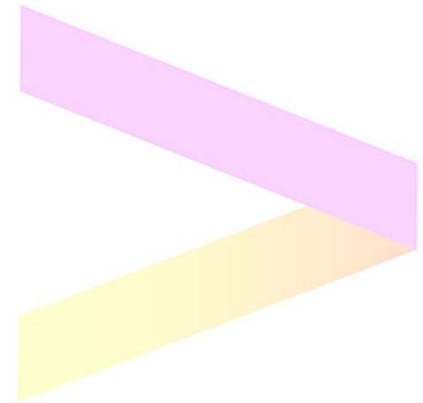
- Ejemplo:
- Un profesor almacena las notas de sus alumnos en una tabla denominada "notas".
- Creamos o reemplazamos la función "f\_condicion" que recibe una nota y retorna una cadena de caracteres indicando si aprueba o no:

```
create or replace function  
f_condicion (anota number)  
return varchar2  
is  
condicion varchar2(20);  
begin  
condicion:="";  
if anota<4 then
```

```
condicion:='desaprobado';  
elsif anota<8 then  
condicion:='regular';  
else  
condicion:='promocionado';  
end if;  
return condicion;  
end;
```

- La estructura "case" es similar a "if", sólo que se pueden establecer varias condiciones a cumplir. Con el "if" solamente podemos obtener dos salidas, cuando la condición resulta verdadera y cuando es falsa, si queremos más opciones podemos usar "case".
- Sintaxis:

```
case VALORACOMPARAR  
  when VALOR1 then SENTENCIAS;  
  when VALOR2 then SENTENCIAS;  
  when VALOR3 then SENTENCIAS;  
  else SENTENCIAS;  
end case;
```





# accenture

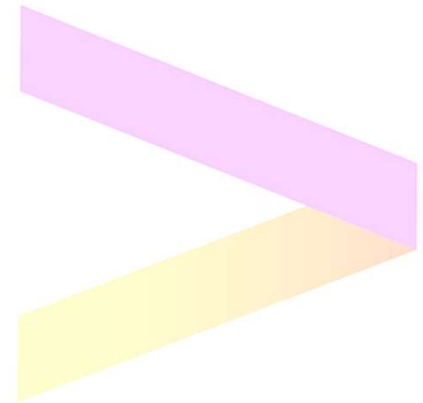
- Necesitamos, dada una fecha, obtener el nombre del mes en español. Podemos utilizar la estructura condicional "case". Para ello crearemos una función que reciba una fecha y retorne una cadena de caracteres indicando el nombre del mes de la fecha enviada como argumento:

```
create or replace function  
f_mes(afecha date)  
return varchar2  
is  
mes varchar2(20);  
begin  
mes:='enero';  
case extract(month from afecha)  
when 1 then mes:='enero';  
when 2 then mes:='febrero';  
when 3 then mes:='marzo';
```

```
when 4 then mes:='abril';  
when 5 then mes:='mayo';  
when 6 then mes:='junio';  
when 7 then mes:='julio';  
when 8 then mes:='agosto';  
when 9 then mes:='setiembre';  
when 10 then mes:='octubre';  
when 11 then mes:='noviembre';  
else mes:='diciembre';  
end case;  
return mes;  
end;
```

- Es una estructuras repetitivas que permiten ejecutar una secuencia de sentencias varias veces.
- Sintaxis:

```
loop  
SENTENCIAS;  
exit when CONDICION;  
SENTENCIAS;  
end loop;
```





- Cuando se llega a la línea de código en la que se encuentra la condición "exit when", se evalúa dicha condición, si resulta cierta, se salta a la línea donde se encuentra "end loop", saliendo del bucle, omitiendo las sentencias existentes antes del "end loop"; en caso contrario, si la condición resulta falsa, se continúa con las siguientes sentencias y al llegar a "end loop" se repite el bucle.

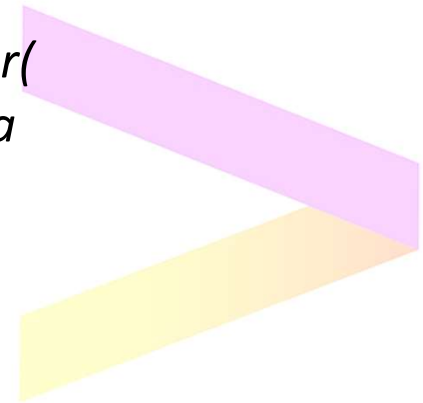


# accenture

- En este ejemplo se muestra la tabla del 3. Se va incrementando la variable "multiplicador" y se almacena en una variable "resultado"; el ciclo se repite hasta que el multiplicador llega a 5, es decir, 6 veces.

```
set serveroutput on;
declare
    resultado number;
    multiplicador number:=0;
begin
    loop
        resultado:=3*multiplicador;

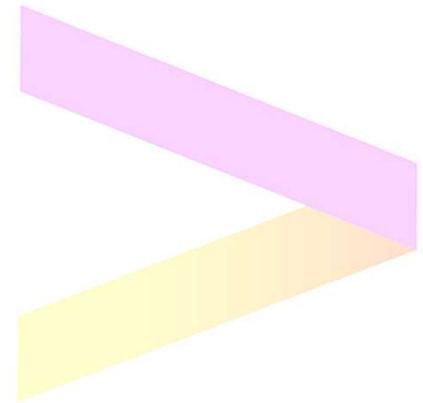
        dbms_output.put_line('3x' || to_char(
multiplicador) || '=' || to_char(resultado));
        multiplicador:=multiplicador+1;
        exit when multiplicador>5;
    end loop;
end;
```



## 11 - FOR

- En la sentencia "for... loop" se especifican dos enteros, un límite inferior y un límite superior, es decir, un rango de enteros, las sentencias se ejecutan una vez por cada entero; en cada repetición del bucle, la variable contador del "for" se incrementa en uno.
- Sintaxis:

```
for VARIABLECONTADOR in  
LIMITEINFERIOR..LIMITESUPERIOR loop  
    SENTENCIAS;  
end loop;
```





- En el siguiente ejemplo se muestra la tabla del 3. La variable "f" comienza en cero (límite inferior del for) y se va incrementando de a uno; el ciclo se repite hasta que "f" llega a 5 (límite superior del for), cuando llega a 6, el bucle finaliza.

```
for f in 0..5 loop  
dbms_output.put_line('3x' || to_char(f) || '=' || to_char  
(f*3));  
end loop;
```



## 12 - WHILE

- "while...loop" (mientras) ejecuta repetidamente una instrucción (o bloque de instrucciones) siempre que la condición sea verdadera.
- Sintaxis básica:

```
while CONDICION loop
```

```
SENTENCIAS
```

```
end loop;
```

- La diferencia entre "while...loop" y "for...loop" es que en la segunda se puede establecer la cantidad de repeticiones del bucle con el valor inicial y final. Además, el segundo siempre se ejecuta, al menos una vez, en cambio el primero puede no ejecutarse nunca, caso en el cual al evaluar la condición por primera vez resulte falsa.

# accenture

- En el siguiente ejemplo se muestra la tabla del 3 hasta el 5:

```
numero number:=0;
```

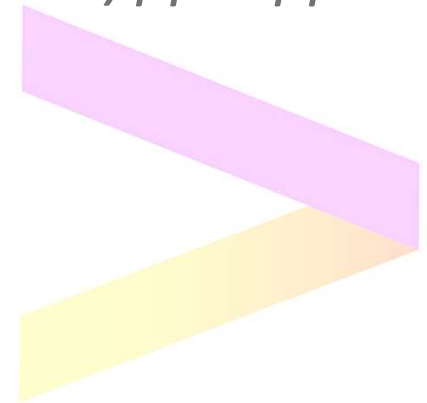
```
while numero<=5 loop
```

```
resultado:=3*numero;
```

```
dbms_output.put_line('3*' || to_char(numero) || '=' || t  
o_char(resultado));
```

```
numero:=numero+1;
```

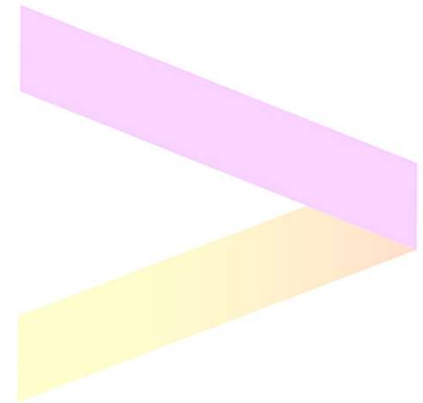
```
end loop;
```





## 13 - TRIGGER

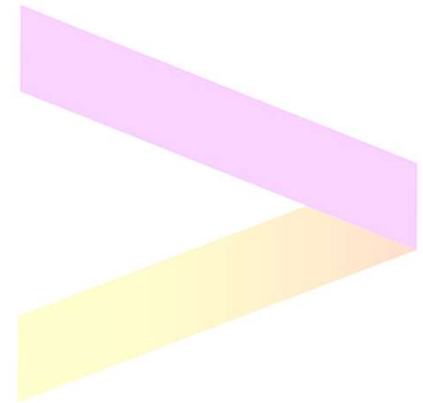
- Un "trigger" (disparador o desencadenador) es un bloque de código que se ejecuta automáticamente cuando ocurre algún evento (como inserción, actualización o borrado) sobre una determinada tabla (o vista); es decir, cuando se intenta modificar los datos de una tabla (o vista) asociada al disparador.





Sintaxis general para crear un disparador:

*create or replace trigger NOMBREDISPARADOR  
MOMENTO-- before, after o instead of  
EVENTO-- insert, update o delete  
of CAMPOS-- solo para update  
on NOMBRETABLA  
NIVEL--puede ser a nivel de sentencia (statement)  
o de fila (for each row)  
when CONDICION--opcional  
begin  
CUERPO DEL DISPARADOR--sentencias  
end NOMBREDISPARADOR;*



# **accenture**

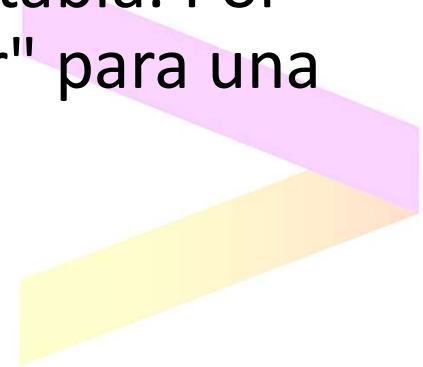
- "MOMENTO" indica cuando se disparará el trigger en relación al evento, puede ser BEFORE (antes), AFTER (después) o INSTEAD OF (en lugar de). "before" significa que el disparador se activará antes que se ejecute la operación (insert, update o delete) sobre la tabla, que causó el disparo del mismo. "after" significa que el trigger se activará después que se ejecute la operación que causó el disparo. "instead of" sólo puede definirse sobre vistas, anula la sentencia disparadora, se ejecuta en lugar de tal sentencia (ni antes ni después).
- "EVENTO" especifica la operación (acción, tipo de modificación) que causa que el trigger se dispare (se active), puede ser "insert", "update" o "delete"; DEBE colocarse al menos una acción, puede ser más de una, en tal caso se separan con "or". Si "update" lleva una lista de atributos, el trigger sólo se ejecuta si se actualiza algún atributo de la lista.



- "on NOMBRETABLA" indica la tabla (o vista) asociada al disparador;
- "NIVEL" puede ser a nivel de sentencia o de fila. "for each row" indica que el trigger es a nivel de fila, es decir, se activa una vez por cada registro afectado por la operación sobre la tabla, cuando una sola operación afecta a varios registros. Los triggers a nivel de sentencia, se activan una sola vez (antes o después de ejecutar la operación sobre la tabla). Si no se especifica, o se especifica "statement", es a nivel de sentencia.
- "CUERPO DEL DISPARADOR" son las acciones que se ejecutan al dispararse el trigger, las condiciones que determinan cuando un intento de inserción, actualización o borrado provoca las acciones que el trigger realizará. El bloque se delimita con "begin... end".



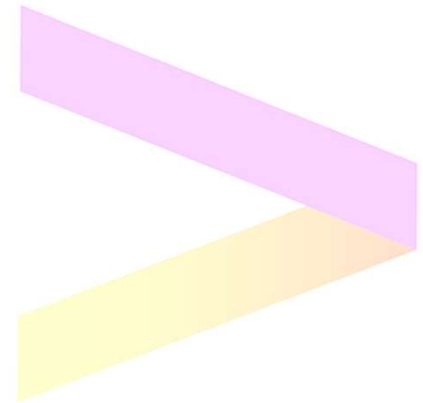
- Consideraciones generales:
  - Las siguientes instrucciones no están permitidas en un desencadenador: create database, alter database, drop database, load database, restore database, load log, reconfigure, restore log, disk init, disk resize.
  - Se pueden crear varios triggers para cada evento, es decir, para cada tipo de modificación (inserción, actualización o borrado) para una misma tabla. Por ejemplo, se puede crear un "insert trigger" para una tabla que ya tiene otro "insert trigger".





- Ejemplo: Creamos un desencadenador que se dispara cada vez que se ejecuta un "insert" sobre la tabla "libros":

```
create or replace trigger tr_ingresar_libros  
before insert  
on libros  
begin  
insert into Control values(user,sysdate);  
end tr_ingresar_libros;
```



# > accenture

- Creamos un desencadenador a nivel de fila, que se dispara una vez por cada fila afectada por un "update" sobre la tabla "libros". Se ingresa en la tabla "control" el nombre del usuario que altera la tabla "libros" (obtenida mediante la función "user") y la fecha en que lo hizo (mediante la función "sysdate"):

```
create or replace trigger tr_actualizar_libros
```

```
before update
```

```
on libros
```

```
for each row
```

```
begin
```

```
insert into control values(user,sysdate);
```

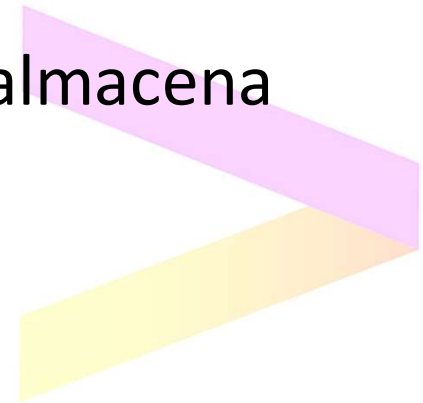
```
end tr_actualizar_libros;
```

- Un trigger puede definirse sobre más de un evento; en tal caso se separan con "or".





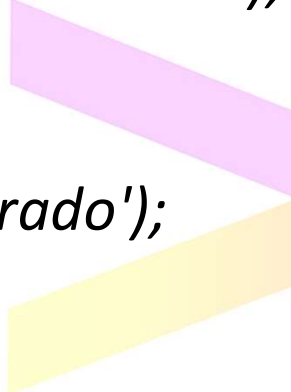
- "update" y "delete"; cuando se modifican los datos de "libros", se almacena en la tabla "control" el nombre del usuario, la fecha y el tipo de modificación que alteró la tabla:
  - si se realizó una inserción, se almacena "inserción";
  - si se realizó una actualización (update), se almacena "actualización" y
  - si se realizó una eliminación (delete) se almacena "borrado".





# accenture

```
create or replace trigger tr_cambios_libros
before insert or update or delete
on libros
for each row
begin
  if inserting then
    insert into control values (user, sysdate, 'inserción');
  end if;
  if updating then
    insert into control values (user, sysdate, 'actualización');
  end if;
  if deleting then
    insert into control values (user, sysdate, 'borrado');
  end if;
end tr_cambios_libros;
```

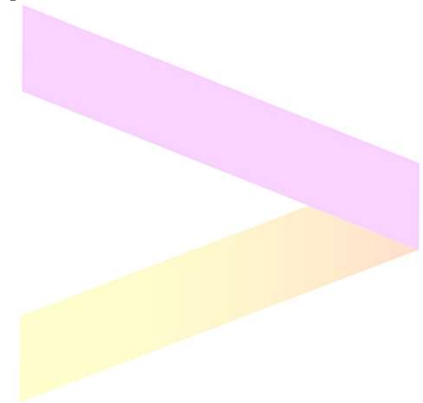


# accenture

- Cuando trabajamos con trigger a nivel de fila, Oracle provee de dos tablas temporales a las cuales se puede acceder, que contienen los antiguos y nuevos valores de los campos del registro afectado por la sentencia que disparó el trigger. El nuevo valor es ":new" y el viejo valor es ":old". Para referirnos a ellos debemos especificar su campo separado por un punto ":new.CAMPO" y ":old.CAMPO".
- Creamos un trigger a nivel de fila que se dispara "antes" que se ejecute un "update" sobre el campo "precio" de la tabla "libros". En el cuerpo del disparador se debe ingresar en la tabla "control", el nombre del usuario que realizó la actualización, la fecha, el código del libro que ha sido modificado, el precio anterior y el nuevo:



```
create or replace trigger  
tr_actualizar_precio_libros  
before update of precio  
on libros  
for each row  
begin  
insert into control  
values(user,sysdate,:new.codigo,:old.precio,:new.precio);  
end tr_actualizar_precio_libros;
```



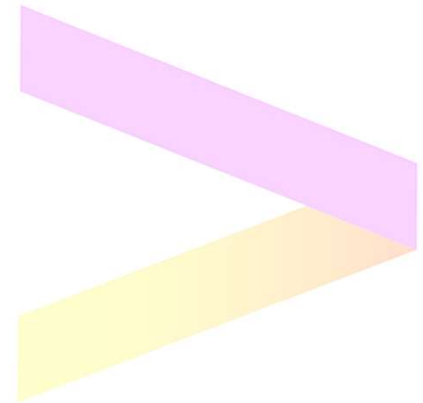
# **accenture**

- En los triggers a nivel de fila, se puede incluir una restricción adicional, agregando la cláusula "when" con una condición que se evalúa para cada fila que afecte el disparador; si resulta cierta, se ejecutan las sentencias del trigger para ese registro; si resulta falsa, el trigger no se dispara para ese registro.
- Limitaciones de "when":
  - no puede contener subconsultas, funciones agregadas ni funciones definidas por el usuario;
  - sólo se puede hacer referencia a los parámetros del evento;
  - no se puede especificar en los triggers "instead of" ni en trigger a nivel de sentencia.

# accenture

- Creamos el siguiente disparador:

```
create or replace trigger tr_precio_libros  
before insert or update of precio  
on libros  
for each row when(new.precio>50)  
begin  
:new.precio := round(:new.precio);  
end tr_precio_libros;
```





- Se puede deshabilitar un trigger para que no se ejecute. Un trigger deshabilitado sigue existiendo, pero al ejecutar una instrucción que lo dispara, no se activa.

- Sintaxis para deshabilitar un trigger:

```
alter trigger NOMBREDISPARADOR disable;
```

- Ejemplo: Deshabilitamos el trigger "tr\_ingresar\_empleados":

```
alter trigger tr_ingresar_empleados disable;
```

- Sintaxis para habilitar un trigger que está deshabilitado:

```
alter trigger NOMBREDISPARADOR enable;
```

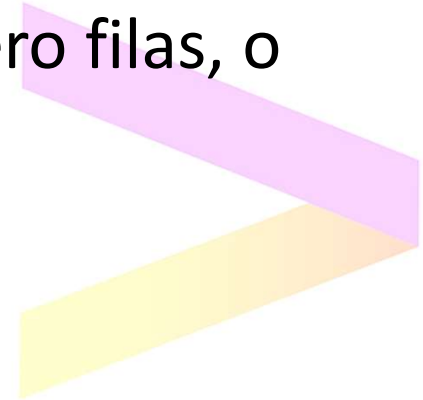
- Ejemplo: Habilitamos el trigger "tr\_actualizar\_empleados":

```
alter trigger tr_actualizar_empleados enable;
```

A decorative graphic consisting of two overlapping triangles pointing to the right. The top triangle is light purple and the bottom triangle is light yellow.

## 14 - CURSORES

- PL/SQL utiliza cursores para gestionar las instrucciones SELECT. Un cursor es un conjunto de registros devuelto por una instrucción SQL. Técnicamente los cursores son fragmentos de memoria que reservados para procesar los resultados de una consulta SELECT.
- Los cursores explicitos se emplean para realizar consultas **SELECT** que pueden devolver cero filas, o más de una fila.





- Para declarar un cursor debemos emplear la siguiente sintaxis:

```
CURSOR nombre_cursor IS  
instrucción_SELECT
```

- También debemos declarar los posibles parametros que requiera el cursor:

```
CURSOR nombre_cursor(param1 tipo1, ..., paramN  
tipoN) IS instrucción_SELECT
```





# accenture

- Para abrir el cursor

```
OPEN nombre_cursor;
```

- o bien (en el caso de un cursor con parámetros)

```
OPEN nombre_cursor(valor1, valor2, ..., valorN);
```

- Para recuperar los datos en variables PL/SQL.

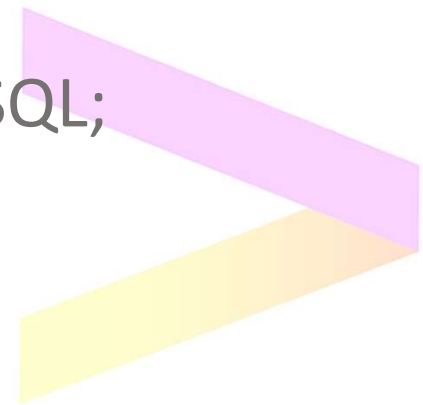
```
FETCH nombre_cursor INTO lista_variables;
```

-- o bien ...

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

- Para cerrar el cursor:

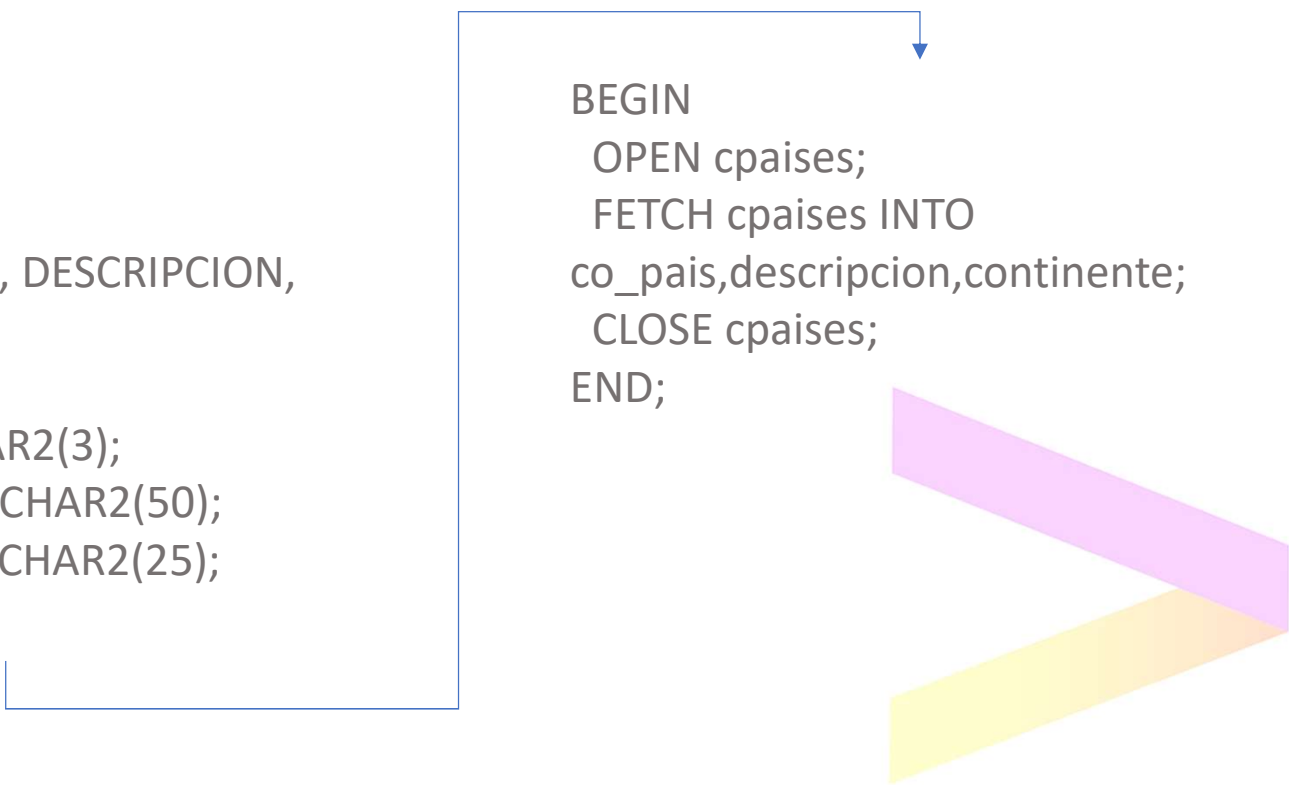
```
CLOSE nombre_cursor;
```



# accenture

- El siguiente ejemplo ilustra el trabajo con un cursor explícito. Hay que tener en cuenta que al leer los datos del cursor debemos hacerlo sobre variables del mismo tipo de datos de la tabla (o tablas) que trata el cursor.

```
DECLARE
  CURSOR cpaíses
  IS
    SELECT CO_PAIS, DESCRIPCION,
    CONTINENTE
    FROM PAISES;
  co_pais VARCHAR2(3);
  descripcion VARCHAR2(50);
  continente VARCHAR2(25);
```



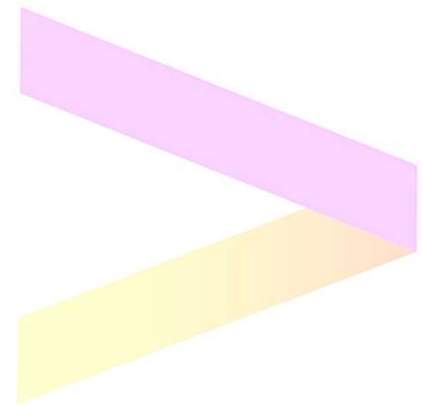
```
BEGIN
  OPEN cpaíses;
  FETCH cpaíses INTO
  co_pais,descripcion,continente;
  CLOSE cpaíses;
END;
```



- Podemos simplificar el ejemplo utilizando el atributo de tipo %ROWTYPE sobre el cursor.

```
DECLARE  
  CURSOR cpaises  
  IS  
    SELECT CO_PAIS, DESCRIPCION,  
           CONTINENTE  
    FROM PAISES;
```

```
registro cpaises%ROWTYPE;  
BEGIN  
  OPEN cpaises;  
  FETCH cpaises INTO registro;  
  CLOSE cpaises;  
END;
```

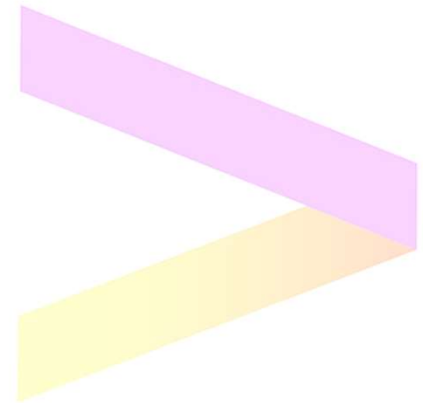


# accenture

- El mismo ejemplo, pero utilizando parámetros:

```
DECLARE
  CURSOR cpaises (p_continente
VARCHAR2)
  IS
  SELECT CO_PAIS, DESCRIPCION,
CONTINENTE
  FROM PAISES
  WHERE CONTINENTE = p_continente;

registro cpaises%ROWTYPE;
BEGIN
  OPEN cpaises('EUROPA');
  FETCH cpaises INTO registro;
  CLOSE cpaises;
END;
```





- Cuando trabajamos con cursores debemos considerar:
  - Cuando un cursor está cerrado, no se puede leer.
  - Cuando leemos un cursor debemos comprobar el resultado de la lectura utilizando los atributos de los cursores.
  - Cuando se cierra el cursor, es ilegal tratar de usarlo.
  - Es ilegal tratar de cerrar un cursor que ya está cerrado o no ha sido abierto



## 15 – SHELLs / BATCHs

- Una Shell de Unix o también shell, es el término usado en informática para referirse a un intérprete de comandos, el cual consiste en la interfaz de usuario tradicional de los sistemas operativos basados en Unix y similares como GNU/Linux.

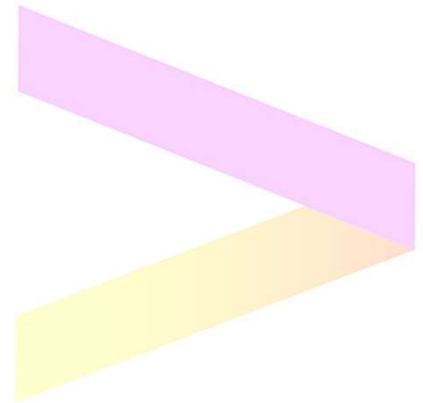
```
chealer@vinci:/usr/share/doc/bash$ export LC_ALL=C
chealer@vinci:/usr/share/doc/bash$ cd ~chealer/
chealer@vinci:~$ ls
Cloutier Ido      Musique logs      skolo sources
Desktop  Mes images boston ncix.png smb4k vieux
chealer@vinci:~$ #Why is there color when calling ls without arguments?
chealer@vinci:~$ which ls
/bin/ls
chealer@vinci:~$ $(!)
$(which ls)
Cloutier Ido      Musique logs      skolo sources
Desktop  Mes images boston ncix.png smb4k vieux
chealer@vinci:~$ type ls # "ls" doesn't just run /bin/ls
ls is aliased to `ls --color=auto'
chealer@vinci:~$ echo $PS1
${debian_chroot:+($debian_chroot)}\u@h:\w\$
chealer@vinci:~$ sh
sh-3.1$ echo $PS1
\s-\v\$
sh-3.1$ echo $BASH_VERSION
3.1.17(1)-release
sh-3.1$ ls
Cloutier Ido      Musique logs      skolo sources
Desktop  Mes images boston ncix.png smb4k vieux
sh-3.1$ echo $SHELLOPTS # ls isn't an alias in POSIX mode
braceexpand:emacs:hashall:histexpand:history:interactive-comments:monitor:posix
sh-3.1$ kill
kill: usage: kill [-s sigspec | -n signum | -sigspec] pid | jobspec ... or kill
-l [sigspec]
sh-3.1$ /bin/kill &> killerror # collect stdout and stderr of $ /bin/kill; in ki
llerror
sh-3.1$ wc -l !$
wc -l killerror
7 killerror
sh-3.1$ type kill # kill doesn't just run /bin/kill, even in POSIX mode.
kill is a shell builtin
sh-3.1$ !$_ -n 9 $$ # OK, kill self
kill -n 9 $$ # OK, kill self
Killed
chealer@vinci:~$
```



- Mediante las instrucciones que aporta el intérprete, el usuario puede comunicarse con el núcleo y por extensión, ejecutar dichas órdenes, así como herramientas que le permiten controlar el funcionamiento de la computadora.

The UNIX logo, consisting of the word "UNIX" in a bold, black, sans-serif font, followed by a registered trademark symbol (®). The logo is centered within a light green rectangular background.

**UNIX®**





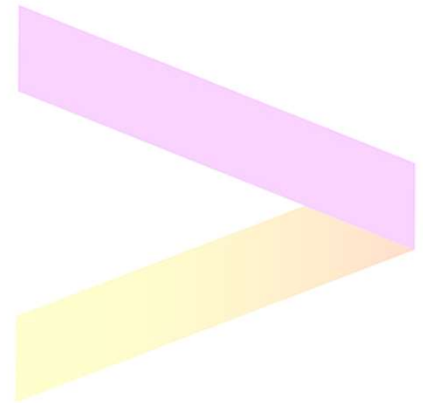
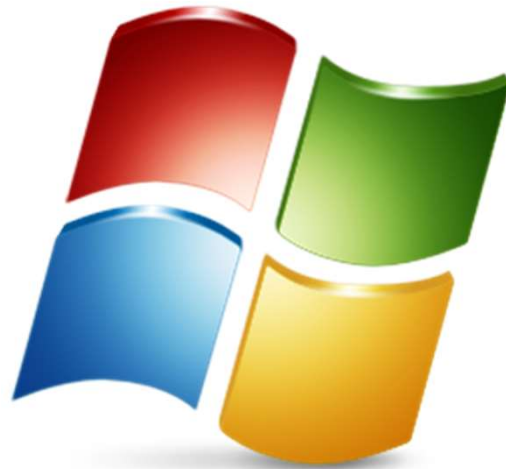
- En DOS, OS/2 y Microsoft Windows, un archivo batch es un archivo de procesamiento por lotes. Se trata de archivos de texto sin formato, guardados con la extensión .BAT que contienen un conjunto de instrucciones MS-DOS. Cuando se ejecuta este archivo, las órdenes contenidas son ejecutadas en grupo, de forma secuencial, permitiendo automatizar diversas tareas. Cualquier orden reconocible por MS-DOS puede ser utilizado en un archivo batch.







- Esta es la forma de automatizar procesos (copiar, pegar, renombrar y enviar datos) en MS-DOS. De este modo, evitamos procesos rutinarios y monótonos, acelerando los mismos. Tiene la funcionalidad de conectarse con otras interfaces por línea de comandos.



# accenture 16 - SQL PLUS y SPOOL

- SQL\*PLUS es una herramienta de Oracle que reconoce y envía sentencias SQL al servidor Oracle para su ejecución.
- Contiene su propio lenguaje de comandos.
- Permite abreviatura de palabras claves de SQL\*PLUS.
- Permite guardar y recuperar sentencias SQL en archivos.
- Desde la línea de comandos: sqlplus [username[/password[@database]]]

```
sqlplus [username[/password[@database]]]
```



# accenture

- Para guardar en un fichero la salida de uno o varios comandos SQL en una base de datos Oracle con sqlplus podemos usar el comando spool.
- Para empezar a almacenar los datos en un fichero le deberemos indicar al comando spool con que nombre lo queremos guardar, por ejemplo “spool ejemplo”:

```
SQL> spool ejemplo  
SQL> select sysdate from dual;
```

```
SYSDATE
```

```
-----
```

```
10/03/10
```

```
SQL> spool off
```

- Mediante spool off dejamos de guardar los comandos SQL en el fichero.