

I. VISTAS

1. Concepto

Una vista es una representación lógica de una o más tablas. Una vista no contiene datos. Todos los datos son derivados de la(s) tabla(s) subyacentes.

2. VENTAJAS DE LAS VISTAS

- Seguridad.
- Conveniencia.
- Perspectiva.

3. SINTAXIS CREAR VISTA

```
CREATE OR REPLACE VIEW nombre_VISTA  
AS sentencia_select;
```

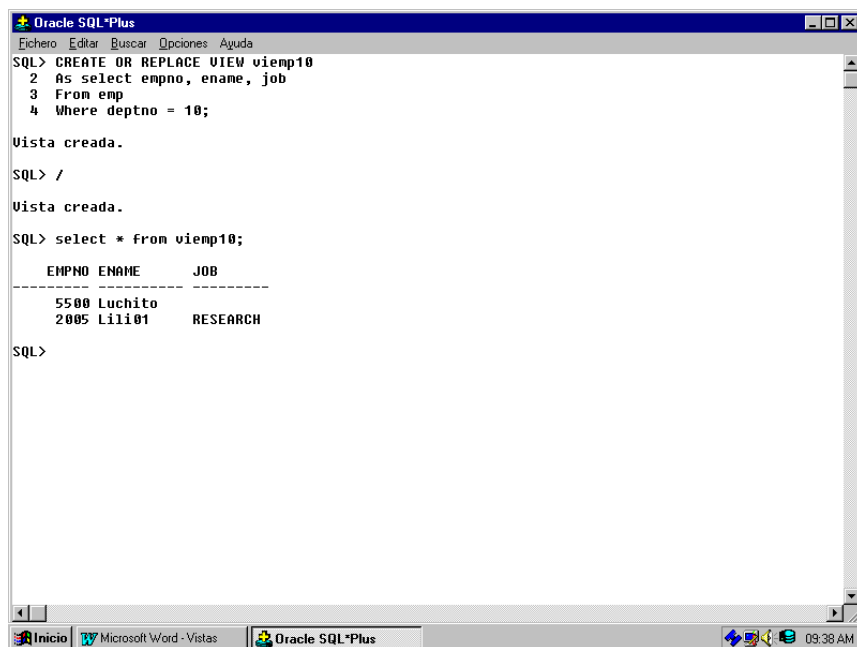
4. EJEMPLOS

- ♦ Cree una vista que contenga el nro, nombre y puesto para todos los empleados en el departamento 10.

```
CREATE OR REPLACE VIEW viemp10  
As select empno, ename, job  
From emp  
Where deptno = 10;
```

/

SALIDA



```
Oracle SQL*Plus
Eichero  Editar  Buscar  Opciones  Ayuda
SQL> CREATE OR REPLACE VIEW viemp10
2  As select empno, ename, job
3  From emp
4  Where deptno = 10;

Vista creada.

SQL> /

Vista creada.

SQL> select * from viemp10;

  EMPNO ENAME      JOB
-----
  5500 Luchito
  2005 Lili01      RESEARCH

SQL>
```

5. PARA BORRAR VISTA

```
Drop view nombre_vista;
```

EJEMPLO

```
Drop view viemp10;
```

6. VISTA CON ALIAS PARA COLUMNAS

```
Create view viemp10 (id, empleado, titulo)
As select empno, ename, job
From emp
Where deptno = 10;
```

7. MODIFICANDO DATOS A PARTIR DE LA VISTA

```
Create view viemp20
As Select *
From emp;

Update viemp20
Set deptno = 30
Where empno = 2010;
```

8. USO DE LA OPCION WITH CHECK OPTION

La opción with check option asegura que los inserts y updates ejecutados a través de las vistas no afecten a los datos a los que la vista no tiene permitido seleccionar.

```
Create view viemp30
As Select *
From emp
Where deptno = 30
With check option;

Update viemp30
Set deptno = 20
Where empno = 2010
```

II. SECUENCIA

Se crean secuencias para generar números enteros únicos para múltiples usuarios. Los números de secuencia pueden ser usados para generar PK automáticamente.

1. SINTAXIS

```
Create sequence nombre_secuencia  
Increment by 1/n  
Start with n  
Maxvalue n  
Minvalue n;
```

2. EJEMPLO

```
Create sequence sq_emp  
Increment by 2  
Start with 0  
Maxvalue 10  
Minvalue 0;
```

3. MOSTRANDO DATOS

```
Select sq_emp.nextval from dual;  
Select sq_emp.currval from dual;
```

4. ELIMINANDO UNA SECUENCIA

```
Drop sequence sq_emp;
```

III. SINÓNIMOS

El Oracle Database de datos permite realizar operaciones en sinónimos. Un sinónimo es un alias o nombre descriptivo para los objetos de base de datos (por ejemplo, tablas, vistas, procedimientos almacenados, funciones y paquetes). Para obtener más información sobre los sinónimos en Oracle.

1. Ventajas de usar sinónimos

Los sinónimos son útiles en los escenarios siguientes:

- **Trabajar con esquemas** diferentes: si está trabajando con esquemas diferentes y necesita tener acceso a los objetos entre esquemas, debe usar diferentes instrucciones SQL para acceder a esos objetos. Puede crear un sinónimo para un objeto en un esquema y usar el sinónimo en la instrucción SQL para acceder al objeto. Si necesita tener acceso al objeto subyacente en un esquema diferente, modifique la definición del sinónimo para que apunte al objeto en un esquema diferente. Por lo tanto, las aplicaciones basadas en el sinónimo siguen funcionando sin modificaciones en la SQL aplicación.

Por ejemplo, suponga que tiene dos esquemas idénticos para los entornos de prueba y producción: "**Test**" y "**Prod**". Para acceder a una tabla denominada "Employee" en el esquema "Test", `Test.EmployeeEmployee` debe usar o (si "Test" es el esquema predeterminado) en la instrucción SQL configuración. Si desea usar la tabla "Employee" en el esquema de producción, `Prod.EmployeeEmployee` ahora debe usar o (cambiar el esquema predeterminado a "Prod") en la instrucción SQL producción. Para evitar este problema, puede crear un sinónimo para la tabla "Test.Employee" (por ejemplo, "EMP") y, a continuación, usarlo en las SQL instrucciones. Siempre que necesite realizar la operación en la tabla "Prod.Employee", modifique la definición del sinónimo "EMP" para que apunte a la tabla "Prod.Employee". Esto garantiza que no tiene que modificar las instrucciones SQL para realizar operaciones en el objeto en esquemas diferentes.

- **Cambios en los objetos subyacentes**: los sinónimos le aíslan de los cambios en el nombre o la ubicación de los objetos subyacentes en los que se realiza una operación. Puede modificar la definición de sinónimos para dar cabida a los cambios en el nombre o la ubicación de los objetos subyacentes.

Por ejemplo, suponga que usa una tabla en uno de los procedimientos almacenados. Ahora, si el nombre de la tabla cambia o la tabla se mueve a otra ubicación, el procedimiento almacenado dejará de funcionar. Para evitarlo, puede usar un sinónimo para la tabla en el procedimiento almacenado y actualizar la definición de sinónimo si hay un cambio en el nombre o la ubicación de la tabla.

- **Acceso simplificado y seguro:** en un entorno distribuido, debe usar el nombre de esquema junto con los nombres de objeto para asegurarse de que está accediendo al objeto correcto. Además, también debe asegurarse de que el usuario tiene privilegios necesarios en el objeto de destino. Para simplificar esto, puede asignar un nombre simple para un objeto mediante la creación de un sinónimo que tenga la ruta de acceso completa al objeto y, a continuación, conceder los privilegios adecuados en el sinónimo.

2. Trabajar con sinónimos en el adaptador

El Oracle Database expone los sinónimos en Oracle para:

- Tablas
- Vistas
- Procedimientos almacenados
- Funciones
- Paquetes

Los sinónimos de cada uno de estos artefactos se exponen junto con el artefacto subyacente correspondiente en el complemento Consume Adapter Service Add-in, Add Adapter Metadata Wizard y Add Adapter Service Reference Plug-in. Por ejemplo, el nodo Tabla de un esquema mostrará todos los sinónimos de las tablas junto con las tablas de base de datos de un esquema, el nodo Ver bajo un esquema mostrará todos los sinónimos de las vistas junto con las vistas de base de datos de un esquema, y así sucesivamente.

- En el caso de los sinónimos creados en tablas y vistas, se exponen las mismas operaciones que para las tablas y vistas subyacentes, respectivamente. Por ejemplo, si las tablas y vistas subyacentes contienen columnas LOB, los sinónimos de esas tablas y vistas también expondrán las operaciones ReadLOB y UpdateLOB.
- En el caso de los sinónimos creados en procedimientos almacenados, funciones y paquetes, los sinónimos se exponen como operaciones junto con los respectivos procedimientos almacenados, funciones y paquetes subyacentes en un esquema.

3. SINTAXIS

Crear sinónimos en Oracle es muy sencillo, basta con utilizar la siguiente sentencia:

```
CREATE SYNONYM <NOMBRE_SINONIMO> FOR  
<ESQUEMA_PROPIETARIO>.<NOMBRE_OBJETO>;
```

Con esta sentencia podemos crear un sinónimo para cualquier objeto de la base de datos. Podemos hacerlo desde el mismo esquema o usuario que va a ser propietario del sinónimo o desde el usuario *system* de la base de datos (si este fuera el caso tendríamos que poner el nombre del esquema que va a ser propietario del sinónimo justo antes del nombre del sinónimo).

Tened en cuenta que **el nombre del sinónimo puede ser el mismo que el del objeto al que estamos apuntando** (siempre y cuando no exista ya en el nuevo esquema).

También podemos utilizar la coetilla pública, para crear un sinónimo que pueda ser visto por todos los usuarios de Oracle:

```
CREATE PUBLIC SYNONYM <NOMBRE_SINONIMO> FOR  
<ESQUEMA_PROPIETARIO>.<NOMBRE_OBJETO>;
```

4. Crear o reemplazar sinónimos

En ocasiones es probable que queramos actualizar sinónimos ya creados, en cuyo caso si intentamos lanzar la sentencia de creación de sinónimo nos devolverá un bonito error ORA-00955 advirtiéndonos que *este nombre ya lo está utilizando otro objeto existente*. El motivo, por supuesto, es que ya existe un sinónimo con ese nombre. Para evitar esta problemática yo siempre recomiendo lanzar la sentencia de la siguiente forma:

```
CREATE OR REPLACE SYNONYM <NOMBRE_SINONIMO> FOR  
<ESQUEMA_PROPIETARIO>.<NOMBRE_OBJETO>;
```

Añadiendo el *or replace* le decimos al motor de base de datos que cree o reemplace al sinónimo, de esta forma, si éste ya existiera previamente lo actualizaría, sin devolvernos ningún error.

5. Dar permisos (grants) para ese sinónimo

Con la creación del sinónimo no termina todo, ya que después tendremos que asignar los permisos que queremos brindar al usuario. Lo haremos de la siguiente forma:

```
GRANT SELECT ON <ESQUEMA_PROPIETARIO>.<NOMBRE_OBJETO> TO  
<ESQUEMA_PROPIETARIO_DEL_SINONIMO>;
```

De esta forma le diremos que el esquema o usuario propietario del sinónimo puede seleccionar (leer) la información que contiene el objeto del esquema propietario, ya que ahí es a donde apunta nuestro sinónimo. Podríamos darle más privilegios si quisiéramos como, por ejemplo:

```
GRANT INSERT, ALTER, DELETE, SELECT, UPDATE ON  
<ESQUEMA_PROPIETARIO>.<NOMBRE_OBJETO> TO  
<ESQUEMA_PROPIETARIO_DEL_SINONIMO>;
```

Así ya no solo podría leer la información sino también insertar datos, eliminarlos o actualizarlos.

6. Ejemplo

Voy a poner un ejemplo muy sencillo. Imaginaos que en una base de datos tenemos dos esquemas y uno de ellos contiene una tabla llamada *EMPLEADO*.

- ESQUEMA1
 - EMPLEADO
- ESQUEMA2
 - (no contiene tablas)

Imaginemos que queremos que el *ESQUEMA2* puede leer la información de la tabla *EMPLEADO* cuyo propietario es el *ESQUEMA1*. Nos conectaríamos a la base de datos como *system* y lanzaríamos lo siguiente:

```
CREATE SYNONYM ESQUEMA2.EMPLEADO FOR ESQUEMA1.NINJA;
```

```
GRANT SELECT ESQUEMA1.EMPLEADO TO ESQUEMA2;
```

Con esto creamos un sinónimo para la tabla *EMPLEADO* del *ESQUEMA1* en el *ESQUEMA2*. Este sinónimo también se llama *EMPLEADO*, como la tabla original de *ESQUEMA1*.

7. Eliminar

DROP SYNONYM

IV. CURSORES

1. CURSOR

- Las ordenes PL/SQL, se amplían con el uso de CURSORES, que permiten a un programa tomar el control explícitamente de la ejecución SQL. Oracle asigna un área de memoria para ejecutar las consultas. Un cursor es un puntero al área de memoria, que permite controlar lo que en ella sucede a medida que se procesa la orden.

2. TIPOS DE CURSORES

- Explicitos, Se utiliza cuando el cursor devuelve mas de una fila.
- Implícitos, Se utiliza cuando el cursor devuelve una fila.

3. SINTAXIS DEFINIR UN CURSOR - EXPLICITO

```
DECLARE
CURSOR NOMBRE_CURSOR is orden_select;
```

4. EJEMPLOS

- ◆ Cursor sin parámetros

```
declare
Cursor c_emp is
Select empno, ename, job from emp;
```

```
Begin
    Null;
End;
/
```

- ◆ Cursor con parámetros

```
declare
Cursor c_emp(pdeptno emp.deptno%type) is
Select empno, ename, job from emp where deptno = pdeptno;
```

```
Begin
    Null;
End;
/
```

5. ATRIBUTOS DE LOS CURSORES EXPLICITOS

Son cuatro los atributos en PL/SQL que pueden aplicarse a los cursores y son:

%FOUND es un atributo booleano, devuelve TRUE si la última orden FETCH devolvió una fila y false en caso contrario.

%NOTFOUND opuesto a %FOUND.

%ISOPEN Determina si el cursor se encuentra abierto y devuelve el valor TRUE.
%ROWCOUNT Este atributo numérico devuelve el número de filas extraídas por el cursor hasta el momento.

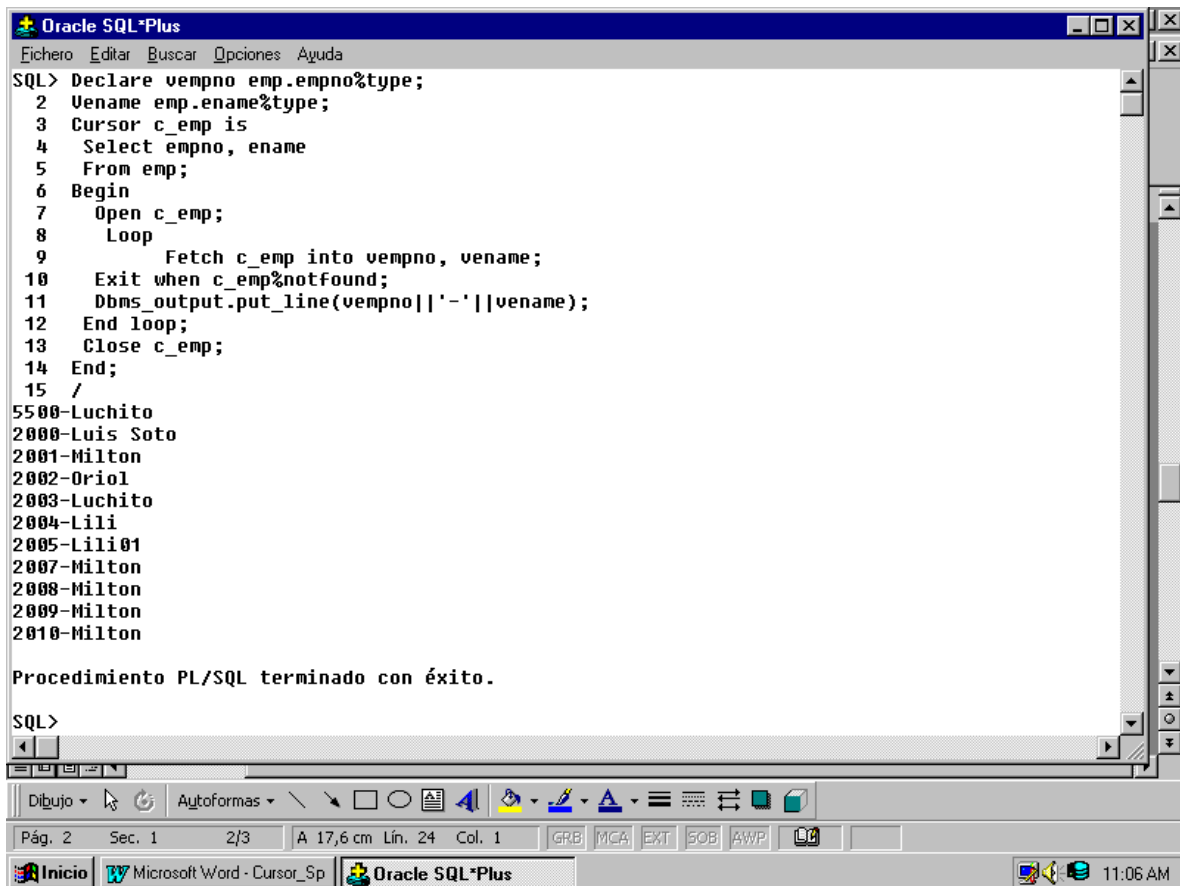
6. APERTURA DEL CURSOR Y FETCH

La apertura del cursor es con OPEN nombre del cursor.

Fetch permite extraer datos de un CURSOR; FETCH NOMBRE DEL CURSOR INTO lista de variables

7. EJEMPLO

```
Declare vempno emp.empno%type;
Vename emp.ename%type;
Cursor c_emp is
  Select empno, ename
  From emp;
Begin
  Open c_emp;
  Loop
    Fetch c_emp into vempno, vename;
    Exit when c_emp%notfound;
    Dbms_output.put_line(vempno||'-'||vename);
  End loop;
  Close c_emp;
End;
```



```
Oracle SQL*Plus
Fichero  Editar  Buscar  Opciones  Ayuda

SQL> Declare vempno emp.empno%type;
2  Vename emp.ename%type;
3  Cursor c_emp is
4  Select empno, ename
5  From emp;
6  Begin
7  Open c_emp;
8  Loop
9      Fetch c_emp into vempno, vename;
10 Exit when c_emp%notfound;
11 Dbms_output.put_line(vempno||'-'||vename);
12 End loop;
13 Close c_emp;
14 End;
15 /
5500-Luchito
2000-Luis Soto
2001-Milton
2002-Oriol
2003-Luchito
2004-Lili
2005-Lili01
2007-Milton
2008-Milton
2009-Milton
2010-Milton

Procedimiento PL/SQL terminado con éxito.

SQL>
```

8. FOR

Declare

Cursor c_emp is

Select empno, ename

From emp;

Begin

for r_emp in c_emp loop

Dbms_output.put_line(r_emp.empno||'-'||r_emp.ename);

end loop;

End;

/

Otra variante:

```
Declare
Cursor c_emp(PEMPNO EMP.EMPNO%TYPE) is
  Select empno, ename
  From emp WHERE EMPNO = PEMPNO;
Begin
  for r_emp in c_emp('2009') loop
    Dbms_output.put_line(r_emp.empno||'-'||r_emp.ename);
  end loop;
End;
/
```

9. CURSORES IMPLICITOS

El cursor implicito sirve para procesar las ordenes INSERT, UPDATE, DELETE Y SELECT .. INTO de una sola fila. Puesto que el motor PL/SQL quien abre y cierra el cursor SQL. Las ordenes open, fetch y close no son relevantes para este tipo de cursor. Pero si se pueden aplicar al cursor SQL los atributos arriba descritos.

Ejemplo

```
Declare
  PEMPNO EMP.ename%TYPE;
Begin
  Select ename
  into pempno
  From emp WHERE EMPNO = '2009';
  if sql%found then
    dbms_output.put_line(pempno);
  else
    dbms_output.put_line('No hay registro');
  end if;
End;
/
```

Miscelanea :

```
declare
vdeptno dept.deptno%type;
vdname dept.dname%type;
vempno emp.empno%type;
vename emp.ename%type;
cursor c_dept is
    select deptno, dname from dept;
CURSOR C_EMP(pdeptno dept.deptno%type) IS
SELECT EMPNO, ENAME
FROM EMP
where deptno = pdeptno;
begin
    open c_dept;
    loop
        fetch c_dept into vdeptno, vdname;
        Exit when c_dept%notfound;
        Dbms_output.put_line('Registro Nro '||c_dept%rowcount||' Codigo '||vdeptno||' Desc '||vdname);
        Dbms_output.put_line('=====');
        open c_emp(vdeptno);
        loop
            Fetch c_emp into vempno, vename;
            if c_emp%found then
                --Exit when c_emp%notfound;
                Dbms_output.put_line('Registro '||c_emp%rowcount||' Codigo '||vempno||' Nombre '||vename);
            else
                if c_emp%rowcount = 0 then
                    Dbms_output.put_line('No hay personal');
                end if;
                Exit when c_emp%notfound;
            end if;
        end loop;
        close c_emp;
    end loop;
    close c_dept;
end;
```

**

Declare

PEMPNO EMP.ename%TYPE;

Begin

Select ename

into pempno

From emp WHERE EMPNO = '2009';

if sql%found then

 update emp

 set ename = 'Milton S. S.'

 WHERE EMPNO = '2009';

if sql%rowcount = 0 then

 dbms_output.put_line('Ningun registro ha sido actualizado');

else

 dbms_output.put_line('Registro actualizados: '||to_char(sql%rowcount));

end if;

else

 dbms_output.put_line('No hay Personal para actualizar');

end if;

End;

/

Otro ejemplo

declare

vempno emp.empno%type;

vename emp.ename%type;

vsal emp.sal%type;

cursor c_emp is

SELECT EMPNO, ENAME, sum(sal) salario FROM EMP group by EMPNO, ENAME;

begin

 open c_emp;

 loop

 Fetch c_emp into vempno, vename, vsal;

 if c_emp%found then

 --Exit when c_emp%notfound;

 Dbms_output.put_line('Registro '||c_emp%rowcount||'Codigo '||vempno||' Nombre '||vename||'
Salario '||to_char(vsal));

 else

 Exit when c_emp%notfound;

 end if;

 end loop;

 close c_emp;

end;

V. PROGRAMACIÓN CON PL/SQL

1. INTRODUCCION

- Es un lenguaje procedural como extensión del SQL ANSI. Con PL/SQL se puede lograr encapsulamiento, ocultamiento de información y administración de excepciones.

2. BLOQUE ANONIMO

- ♦ También conocido como PL.
- ♦ Un bloque anónimo es un grupo de declaraciones y sentencias lógicamente relacionadas.
- ♦ Las declaraciones en un bloque solo existen hasta que se concluya su ejecución.

3. PARTES DE UN BLOQUE PL/SQL

```
DECLARE
■ Declaración de excepciones.
BEGIN
■ Sentencias DE EJECUCIÓN
EXCEPCIONES
■ Excepciones
END
```

La unidad mínima del bloque PL/SQL es la sección de ejecución. Dentro de las secciones de ejecución y de excepciones se puede colocar otros bloques PL/SQL a manera de anidamiento. El ámbito de las variables definidas en la sección de declaración quedan determinadas por el nivel de anidamiento de los bloques PL/SQL.

4. DECLARACION DE VARIABLES

```
Create or replace procedure sp_bd01 is
begin
    DECLARE WC_C_EMPLEADO NUMBER;
    WC_T_EMPLEADO VARCHAR2(100);
    Begin
        Null;
    End;
End;
```

- ♦ Las variables se declaran en la sección de declaración.
- ♦ Los tipos de variables pueden ser de cualquier tipo SQL y/o PL/SQL.
- ♦ Los tipos de variables SQL son definidos por el SQL ANSI.

- ◆ Los tipos de variables PL/SQL son RECORD y TABLE.
- ◆ En un bloque PL/SQL se pueden utilizar el tipo BOOLEAN.

Ejemplo

```
DECLARE    WN_C_EMPLEADO NUMBER;
           WC_T_EMPLEADO VARCHAR2(100);
```

5. ATRIBUTOS

Para la definición de los tipos de datos de las variables se pueden referenciar a tipos de datos definidos en el diccionario de datos.

```
Wc_c_departamento dept.dname%type;
```

```
Wt_departamento dept%rowtype;
```

6. DECLARACION E INICIALIZACION DE VARIABLES

```
Wc_c_departamento dept.dname%type := 'Gerencia de Producción';
```

7. DECLARACION DE CONSTANTES

```
Wc_c_dpto CONSTANT number := 20;
```

Ejemplo:

```
DECLARE WC_C_EMPLEADO NUMBER;
```

```
WC_T_EMPLEADO VARCHAR2(100);
```

```
Wc_c_departamento dept.dname%type := 'hola';
```

```
Wc_c_dpto CONSTANT number := 20;
```

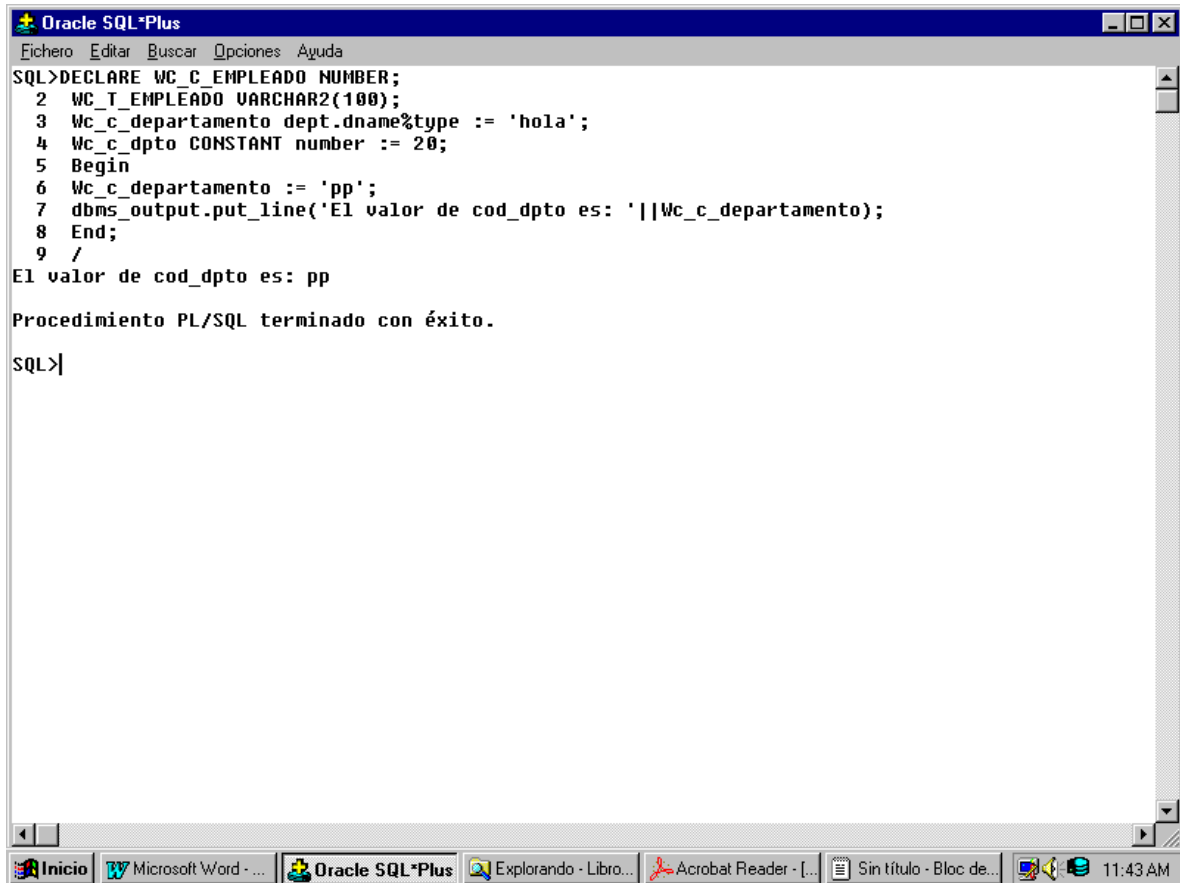
```
Begin
```

```
Wc_c_departamento := 'pp';
```

```
dbms_output.put_line('El valor de cod_dpto es: '||Wc_c_departamento);
```

```
End;
```


Salida:



The screenshot shows the Oracle SQL*Plus application window. The title bar reads "Oracle SQL*Plus". The menu bar includes "Archivo", "Editar", "Buscar", "Opciones", and "Ayuda". The main text area contains the following PL/SQL code and its output:

```
SQL>DECLARE WC_C_EMPLEADO NUMBER;  
2  WC_T_EMPLEADO VARCHAR2(100);  
3  Wc_c_departamento dept.dname%type := 'hola';  
4  Wc_c_dpto CONSTANT number := 20;  
5  Begin  
6  Wc_c_departamento := 'pp';  
7  dbms_output.put_line('El valor de cod_dpto es: '||Wc_c_departamento);  
8  End;  
9  /  
El valor de cod_dpto es: pp  
Procedimiento PL/SQL terminado con éxito.  
SQL>|
```

The Windows taskbar at the bottom shows the following open applications: "Inicio", "Microsoft Word - ...", "Oracle SQL*Plus", "Explorando - Libro...", "Acrobat Reader - [...]", and "Sin título - Bloc de...". The system clock in the bottom right corner displays "11:43 AM".

Ejemplo:

```
DECLARE WC_C_EMPLEADO NUMBER;
```

```
WC_T_EMPLEADO VARCHAR2(100);
```

```
Wc_c_departamento dept.dname%type := 'hola';
```

```
wn_registro number;
```

```
Wc_c_dpto CONSTANT number := 20;
```

```
Begin
```

```
Wc_c_departamento := 'pp';
```

```
dbms_output.put_line('El valor de p es: '||Wc_c_departamento);
```

```
select count(*)
```

```
into wn_registro
```

```
from dept;
```

```
dbms_output.put_line('Nro de registros: '||wn_registro);
```

```
End;
```

Salida:

The screenshot shows the Oracle SQL*Plus interface. The main window contains the following text:

```
SQL>DECLARE WC_C_EMPLEADO NUMBER;  
2 WC_T_EMPLEADO VARCHAR2(100);  
3 Wc_c_departamento dept.dname%type := 'hola';  
4 wn_registro number;  
5 Wc_c_dpto CONSTANT number := 20;  
6 Begin  
7 Wc_c_departamento := 'pp';  
8 dbms_output.put_line('El valor de p es: '||Wc_c_departamento);  
9 select count(*)  
10 into wn_registro  
11 from dept;  
12 dbms_output.put_line('Nro de registros: '||wn_registro);  
13 End;  
14 /  
El valor de p es: pp  
Nro de registros: 7  
  
Procedimiento PL/SQL terminado con éxito.  
  
SQL>select * from dept;
```

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON
50	sistemas	lima
51	SISTEMAS	LIMA
52	SISTEMAS	LIMA

7 filas seleccionadas.
SQL>|

The taskbar at the bottom shows the following open applications: Inicio, Oracle SQL*Plus, Explorando - Libros Oracle, Sin título - Bloc de notas, Microsoft Word - PL_Sql, and a system clock showing 12:00 PM.

8. ESTRUCTURAS DE CONTROL

♦ FOR..LOOP SINTAXIS

FOR I IN 1..20 LOOP

 NULL;

END LOOP

EJEMPLO

DECLARE I NUMBER;

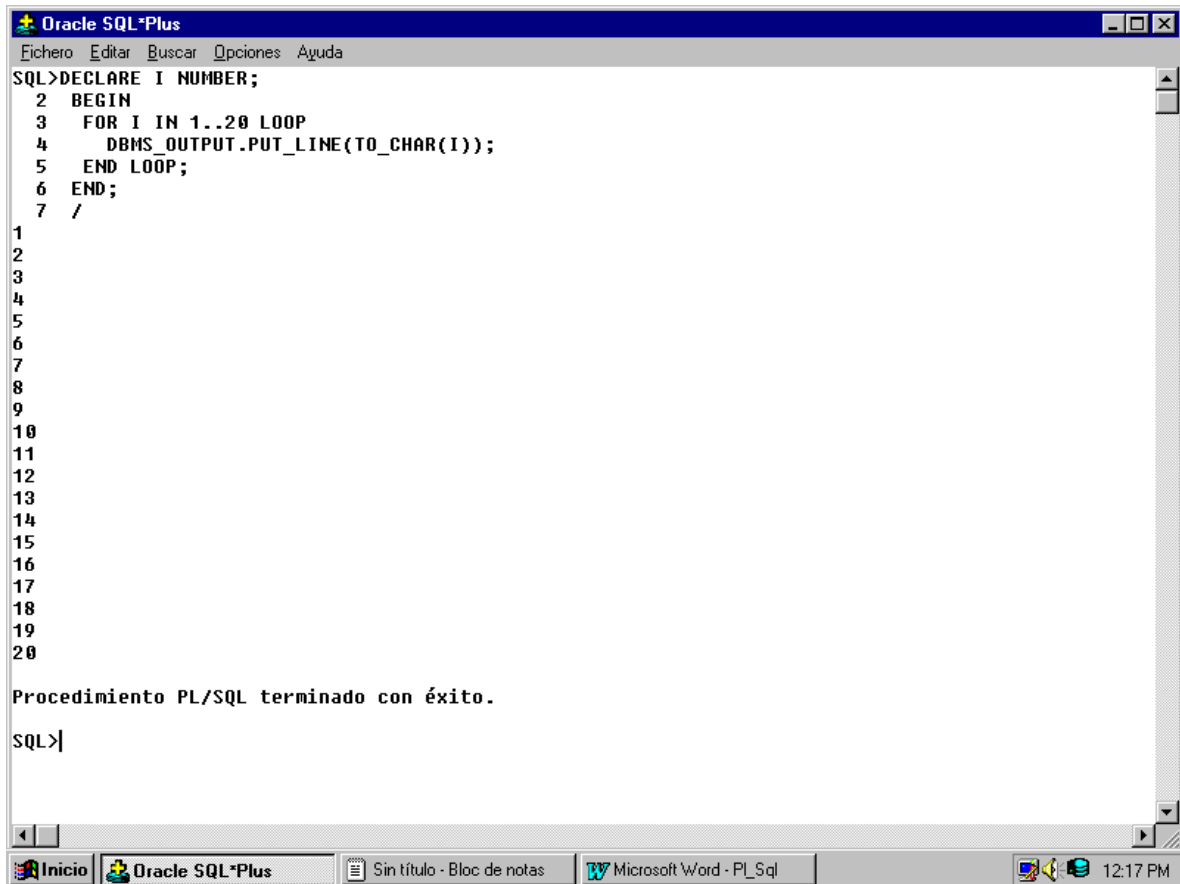
BEGIN

FOR I IN 1..20 LOOP

```
DBMS_OUTPUT.PUT_LINE(TO_CHAR(I));
```

```
END LOOP;
```

```
END;
```



The screenshot shows the Oracle SQL*Plus application window. The title bar reads "Oracle SQL*Plus". The menu bar includes "Fichero", "Editar", "Buscar", "Opciones", and "Ayuda". The main text area contains the following PL/SQL code:

```
SQL>DECLARE I NUMBER;  
2 BEGIN  
3   FOR I IN 1..20 LOOP  
4     DBMS_OUTPUT.PUT_LINE(TO_CHAR(I));  
5   END LOOP;  
6 END;  
7 /
```

To the left of the code, line numbers 1 through 20 are listed. Below the code, the message "Procedimiento PL/SQL terminado con éxito." is displayed, followed by the prompt "SQL>|". The status bar at the bottom shows the taskbar with icons for "Inicio", "Oracle SQL*Plus", "Sin título - Bloc de notas", and "Microsoft Word - PL_Sql". The system clock on the right indicates "12:17 PM".

Analizar

```
DECLARE II NUMBER;  
  
BEGIN  
  
FOR I IN REVERSE 1..20 LOOP  
  
    DBMS_OUTPUT.PUT_LINE(TO_CHAR(I));  
  
END LOOP;  
  
END;
```

♦ LOOP .. END LOOP EJEMPLO

```
DECLARE I NUMBER :=0;  
  
BEGIN  
  
LOOP  
  
    I := I + 1;  
  
    EXIT WHEN I = 6;  
  
    DBMS_OUTPUT.PUT_LINE(TO_CHAR(I));  
  
END LOOP;  
  
END;
```

WHILE END .. LOOP

EJEMPLO

```
DECLARE I NUMBER :=1;  
  
BEGIN  
  
WHILE I <= 6 LOOP
```

```
DBMS_OUTPUT.PUT_LINE(TO_CHAR(I));
```

```
I := I + 1;
```

```
END LOOP;
```

```
END;
```

9.

SENTENCIAS DE DECISION

IF .. THEN .. ELSE END

EJEMPLO

```
BEGIN

FOR I IN 1..4 LOOP

    IF I = 1 THEN

        DBMS_OUTPUT.PUT_LINE('UNO '||TO_CHAR(I));

    ELSIF I = 2 THEN

        DBMS_OUTPUT.PUT_LINE('DOS '||TO_CHAR(I));

    ELSIF I = 3 THEN

        DBMS_OUTPUT.PUT_LINE('TRES '||TO_CHAR(I));

    ELSE

        DBMS_OUTPUT.PUT_LINE('OTRO '||TO_CHAR(I));

    END IF;

END LOOP;

END;
```

10. CLASES DE PROGRAMAS PL/SQL

◆ BLOQUE PL/SQL ANONIMO

```
BEGIN

FOR I IN 1..4 LOOP

    IF I = 1 THEN

        DBMS_OUTPUT.PUT_LINE('UNO '||TO_CHAR(I));

    ELSIF I = 2 THEN
```

```
        DBMS_OUTPUT.PUT_LINE('DOS '||TO_CHAR(I));

ELSIF I = 3 THEN

        DBMS_OUTPUT.PUT_LINE('TRES '||TO_CHAR(I));

ELSE

        DBMS_OUTPUT.PUT_LINE('OTRO '||TO_CHAR(I));

END IF;

END LOOP;

END;
```

Ejemplo

DECLARE

J NUMBER :=3000;

ERROR_1 EXCEPTION;

BEGIN

IF J > 100 THEN

 RAISE ERROR_1;

ELSE

FOR I IN 1..4 LOOP

 IF I = 1 THEN

 DBMS_OUTPUT.PUT_LINE('UNO '||TO_CHAR(I));

 ELSIF I = 2 THEN

 DBMS_OUTPUT.PUT_LINE('DOS '||TO_CHAR(I));

 ELSIF I = 3 THEN

 DBMS_OUTPUT.PUT_LINE('TRES '||TO_CHAR(I));

 ELSE

 DBMS_OUTPUT.PUT_LINE('OTRO '||TO_CHAR(I));

 END IF;

END LOOP;

END IF;

EXCEPTION WHEN ERROR_1 THEN

 DBMS_OUTPUT.PUT_LINE('ERROR J FUERA DE RANGO');

END;

◆ **TRIGGERS**

Un trigger es una tipo especial de PL/SQL anónimo.

◆ **STORE PROCEDURE**

Un store procedure o function es una unidad de programa que tiene:

Nombre del procedimiento.

Opcionalmente parámetros y devolver resultados.

Se almacena en la base de datos.

Los Procedure pueden ser invocados por muchos usuarios.

```
CREATE OR REPLACE PROCEDURE NN_UNO(J NUMBER) AS
```

```
BEGIN
```

```
DECLARE
```

```
ERROR_1 EXCEPTION;
```

```
BEGIN
```

```
IF J > 100 THEN
```

```
    RAISE ERROR_1;
```

```
ELSE
```

```
FOR I IN 1..4 LOOP
```

```
    IF I = 1 THEN
```

```
        DBMS_OUTPUT.PUT_LINE('UNO '||TO_CHAR(I));
```

```
    ELSIF I = 2 THEN
```

```
        DBMS_OUTPUT.PUT_LINE('DOS '||TO_CHAR(I));
```

```
ELSIF I = 3 THEN

    DBMS_OUTPUT.PUT_LINE('TRES '||TO_CHAR(I));

ELSE

    DBMS_OUTPUT.PUT_LINE('OTRO '||TO_CHAR(I));

END IF;

END LOOP;

END IF;

EXCEPTION WHEN ERROR_1 THEN

    DBMS_OUTPUT.PUT_LINE('ERROR J FUERA DE RANGO');

END;

END;

EJECUTANDO

EXEC NN_UNO(3000);
```

ANALIZAR

CREATE OR REPLACE PROCEDURE NN_DOS(J IN NUMBER, RESUL OUT CHAR) AS

BEGIN

DECLARE

ERROR_1 EXCEPTION;

BEGIN

IF J > 100 THEN

 RESUL := '0';

 RAISE ERROR_1;

ELSE

 RESUL := '1';

FOR I IN 1..4 LOOP

 IF I = 1 THEN

 DBMS_OUTPUT.PUT_LINE('UNO '||TO_CHAR(I));

 ELSIF I = 2 THEN

 DBMS_OUTPUT.PUT_LINE('DOS '||TO_CHAR(I));

 ELSIF I = 3 THEN

 DBMS_OUTPUT.PUT_LINE('TRES '||TO_CHAR(I));

 ELSE

 DBMS_OUTPUT.PUT_LINE('OTRO '||TO_CHAR(I));

 END IF;

END LOOP;

END IF;

```
EXCEPTION WHEN ERROR_1 THEN
```

```
    DBMS_OUTPUT.PUT_LINE('ERROR J FUERA DE RANGO');
```

```
END;
```

```
END;
```

Invocando procedimiento desde otro procedimiento.

```
DECLARE HH CHAR;  
  
BEGIN  
  
    NN_dos(200, HH);  
  
    DBMS_OUTPUT.PUT_LINE('RESUL '||HH);  
  
END;  
  
/
```

CREANDO FUNCIONES EN EL SERVIDOR

create or replace function sf_uno(p in number) return char as

begin

declare resul char;

begin

if p = 1 then

resul := '0';

DBMS_OUTPUT.PUT_LINE('P es uno');

else

resul := '1';

DBMS_OUTPUT.PUT_LINE('P es diferente de uno');

end if;

return resul;

end;

end;

INVOCANDO FUNCIONES

```
declare rr char;
```

```
begin
```

```
    rr := sf_uno(1);
```

```
    DBMS_OUTPUT.PUT_LINE('El valor de rr es '||rr);
```

```
end;
```

VI. PACKAGE

Un Paquete es un objeto PL/Sql que agrupa lógicamente otros objetos PL/Sql relacionados entre sí, encapsulándolos y convirtiéndolos en una unidad dentro de la base de datos.

Los Paquetes están divididos en 2 partes: especificación (obligatoria) y cuerpo (no obligatoria). La especificación o encabezado es la interfaz entre el Paquete y las aplicaciones que lo utilizan y es allí donde se declaran los tipos, variables, constantes, excepciones, cursores, procedimientos y funciones que podrán ser invocados desde fuera del paquete.

SINTAXIS

Como hemos dicho anteriormente, la creación de un paquete pasa por dos fases:

- Crear la cabecera del paquete donde se definen que procedimientos, funciones, variables, cursores, etc. Disponibles para su uso posterior fuera del paquete. En esta parte solo se declaran los objetos, no se implementa el código.
- Crear el cuerpo del paquete, donde se definen los bloques de código de las funciones y procedimientos definidos en la cabecera del paquete.

Para crear la cabecera del paquete utilizaremos la siguiente instrucción:

```
CREATE {OR REPLACE} PACKAGE nombre_de_paquete IS
-- Declaraciones
END;
```

Para crear el cuerpo del paquete utilizaremos la siguiente instrucción:

```
CREATE {OR REPLACE} PACKAGE BODY nombre_paquete IS
--Bloques de código
END;
```

Cuando se quiera acceder a las funciones, procedimientos y variables de un paquete se debe anteponer el nombre de este:

```
Nombre_paquete.función(x);
Nombre_paquete.procedimiento(x);
Nombre_paquete.variable;
```

Ejemplo

```
CREATE OR REPLACE PACKAGE PK1 IS
  PROCEDURE xLis (xfamilia IN Articulos.cArtFml%TYPE);
END;

CREATE OR REPLACE PACKAGE BODY PK1 IS
  procedure xLis (xfamilia Articulos.cArtCdg%TYPE)
  IS
    xfam Articulos.cArtFml%type;
    xCod Articulos.cArtCdg%TYPE;
    xDes Articulos.cArtDsc%TYPE;

    CURSOR cArticulos IS SELECT cArtCdg,cArtDsc FROM Articulos
```

```
        WHERE cArtFml = xfam;  
BEGIN  
    xfam := xfamilia;  
    OPEN cArticulos;  
    LOOP  
        FETCH cArticulos INTO xCod,xDes;  
        EXIT WHEN cArticulos%NOTFOUND;  
        DBMS_OUTPUT.PUT_LINE (xDes);  
    END LOOP;  
    CLOSE cArticulos;  
END;  
END;
```

VII. EXCEPCIONES

1. EXCEPCIONES

- Las excepciones permiten tratar los errores de un programa PL/SQL, lo que permite construir PL/SQL robustos que permita tratar errores previstos e inesperados que se producen durante la ejecución.
- Cuando se produce un error, se genera una excepción. Cuando esto sucede, el control pasa al gestor de excepciones, que es una sección independiente del programa, que es una sección independiente del programa. Esto permite que sea más fácil de entender la lógica del programa y asegura que los errores sean interceptados.

2. DECLARACION DE EXCEPCIONES

- Las excepciones se declararán en la sección declarativa de un bloque PL/SQL.

3. TIPOS DE EXCEPCIONES

- **Definidas por el usuario**, Son excepciones que manejan errores generalmente asociados a los datos y en algunos casos asociados a SQL comunes.

Por ejemplo, declare error_nro_alumnos EXCEPTION;

- **Predefinidas**, Oracle ha predefinido diversas excepciones que se corresponden con los errores Oracle comunes. Al igual que los tipos predefinidos (Number, Varchar2, etc) los identificadores de estas excepciones se definen en el paquete SYS.STANDARD.

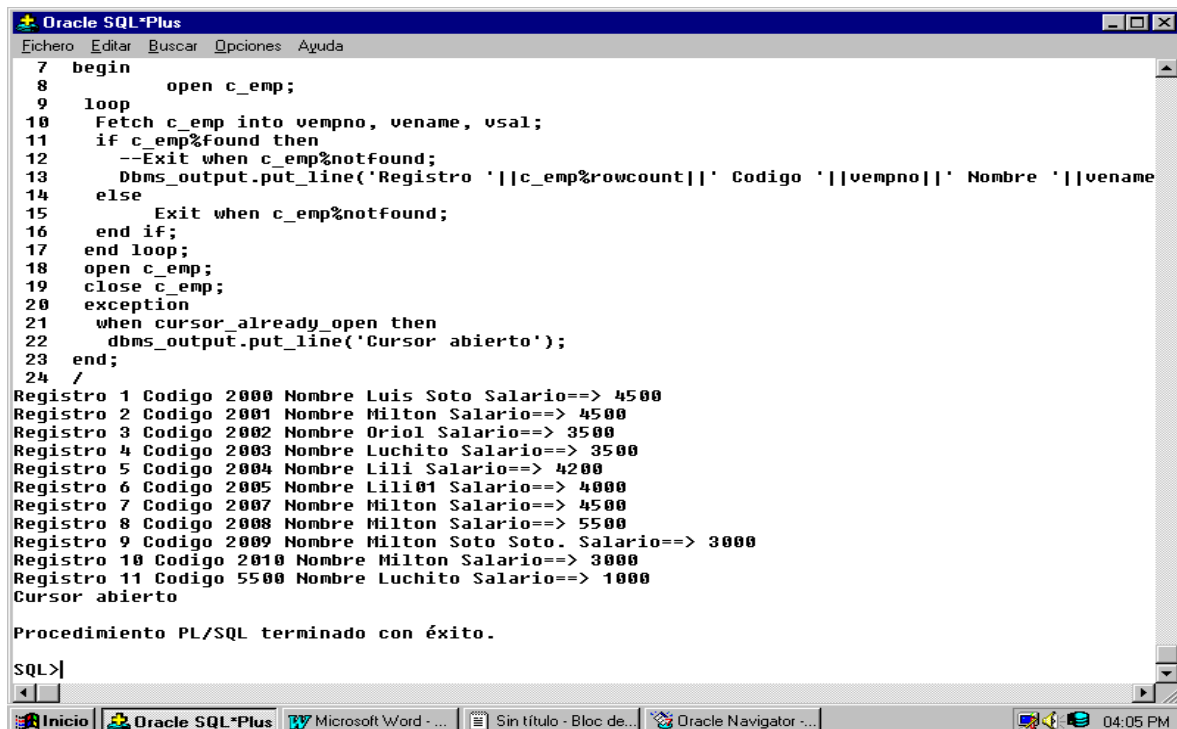
ERROR ORACLE	EXCEPCIÓN
ORA-0001	DUP_VAL_ON_INDEX
ORA-0051	TIMEOUT_ON_RESOURCE
ORA-0061	TRANSACTION_BACKED_OUT
ORA-1001	INVALID_CURSOR
ORA-1012	NOT_LOGGED_ON
ORA-1017	LOGIN_DENIED
ORA-1403	NO_DATA_FOUND
ORA-1422	TOO_MANY_ROWS
ORA-1476	ZERO_DIVIDE
ORA-1722	INVALID_NUMBER
ORA-6500	STORAGE_ERROR
ORA-6501	PROGRAM_ERROR
ORA-6502	VALUE_ERROR
ORA-6504	ROWTYPE_MISMATCH
ORA-6511	CURSOR_ALREADY_OPEN
ORA-6530	ACCESS_INTO_NULL
ORA-6531	COLLECTION_IS_NULL
ORA-6532	SUBSCRIPT_OUTSIDE_LIMIT
ORA-6533	SUBSCRIPT_BEYOND_COUNT

EXCEPCIONES PREDEFINIDAS POR ORACLE

4. EJEMPLO

```
declare
  vempno EMPLOYEES.employee_id%type;
  vename EMPLOYEES.last_name%type;
  vsal EMPLOYEES.salary%type;
cursor c_emp is
SELECT  a.employee_id,  a.last_name,  sum(salary)  salario  FROM
EMPLOYEES A group by a.employee_id, a.last_name;
begin
  open c_emp;
  loop
    Fetch c_emp into vempno, vename, vsal;
    if c_emp%found then
      --Exit when c_emp%notfound;
      Dbms_output.put_line('Registro '||c_emp%rowcount||' Codigo
'||vempno||' Nombre '||vename||' Salario==> '||to_char(vsal));
    else
      Exit when c_emp%notfound;
    end if;
  end loop;
  open c_emp;
  close c_emp;
exception
  when cursor_already_open then
    dbms_output.put_line('Cursor abierto');
end;/
```

Salida



```
7  begin
8      open c_emp;
9  loop
10     Fetch c_emp into vempno, vename, vsal;
11     if c_emp%found then
12         --Exit when c_emp%notfound;
13         Dbms_output.put_line('Registro '||c_emp%rowcount||' Codigo '||vempno||' Nombre '||vename
14     else
15         Exit when c_emp%notfound;
16     end if;
17 end loop;
18 open c_emp;
19 close c_emp;
20 exception
21     when cursor_already_open then
22         dbms_output.put_line('Cursor abierto');
23 end;
24 /
Registro 1 Codigo 2000 Nombre Luis Soto Salario==> 4500
Registro 2 Codigo 2001 Nombre Milton Salario==> 4500
Registro 3 Codigo 2002 Nombre Oriol Salario==> 3500
Registro 4 Codigo 2003 Nombre Luchito Salario==> 3500
Registro 5 Codigo 2004 Nombre Lili Salario==> 4200
Registro 6 Codigo 2005 Nombre Lili01 Salario==> 4000
Registro 7 Codigo 2007 Nombre Milton Salario==> 4500
Registro 8 Codigo 2008 Nombre Milton Salario==> 5500
Registro 9 Codigo 2009 Nombre Milton Soto Soto. Salario==> 3000
Registro 10 Codigo 2010 Nombre Milton Salario==> 3000
Registro 11 Codigo 5500 Nombre Luchito Salario==> 1000
Cursor abierto

Procedimiento PL/SQL terminado con éxito.

SQL>
```

Otro ejemplo:

```
begin
    insert into dept
    values (60, 'U de Lima',null);
    if sql%found then
        dbms_output.put_line('Registro insertado');
    end if;
    insert into dept
    values (60, 'U de Lima',null);
    if sql%found then
        dbms_output.put_line('Registro insertado');
    end if;
    exception
        when dup_val_on_index then
            dbms_output.put_line('Dato duplicado');
end;
```

5. EJEMPLO DE EXCEPCION DEFINIDA POR EL USUARIO

```
DECLARE
    SALARIO_ALTO EXCEPTION;
    SALARIO_BAJO EXCEPTION;
    -- Cursor
BEGIN
    FOR I IN ( SELECT LAST_NAME, NVL(SALARY,0) SAL FROM EMPLOYEES )
    LOOP
        BEGIN
            IF I.SAL > 3500 THEN
                RAISE SALARIO_ALTO;
            ELSE
                RAISE SALARIO_BAJO;
            END IF;
        EXCEPTION
            WHEN SALARIO_ALTO THEN
                dbms_output.put_line(' Apellido '||I.LAST_NAME||'    Salario Alto
                '||to_char(I.sal));
            WHEN SALARIO_BAJO THEN
                dbms_output.put_line(' Apellido '||I.LAST_NAME||'    Salario Bajo
                '||to_char(I.sal));
            END;
        END LOOP;
    END;
```

6.

PROPAGACIÓN DE ERRORES

```
begin
  declare wsalario number;
  begin
    --wsalario := 10/2;
    wsalario := 10/0;
    exception
    when zero_divide then
      dbms_output.put_line('Interno - Div entre cero');
      --raise;
    end;
    dbms_output.put_line('Interno - Bloque externo');
    exception
    when zero_divide then
      dbms_output.put_line('Externo - Div entre cero');
  end;
/
```

7. RAISE: PERMITE VOLVER ACTIVAR LA EXCEPCIÓN

```
begin
  declare wsalario number;
  begin
    --wsalario := 10/2;
    wsalario := 10/0;
    exception
    when zero_divide then
      dbms_output.put_line('Interno - Div entre cero');
      raise;
    end;
    dbms_output.put_line('Interno - Bloque externo');
    exception
    when zero_divide then
      dbms_output.put_line('Externo - Div entre cero');
  end;
/
```

8.

EL PRAGMA EXCEPTION_INIT

Es posible asociar una excepción nominada con un error ORACLE determinado, lo que nos permite interceptar de forma específica dicho error, en lugar del gestor de la excepción OTHERS. Por ejemplo:

```
begin
  declare wsalario number;
  begin
    wsalario := 10/0;
  end;
  dbms_output.put_line('Interno - Bloque externo');
  exception
  when others then
    dbms_output.put_line('Externo      -      Error' || to_Char(sqlcode) || '
' || substr(sqlerrm,1,100));
end;
/
Externo - Error-1476 ORA-01476: el divisor es igual a cero

Procedimiento PL/SQL terminado con éxito.
```

◆ EJEMPLO CON PRAGMA EXCEPTION_INIT

```
declare
  wsalario number;
  division_por_cero exception;
  pragma exception_init(division_por_cero,-01476);
begin
  wsalario := 10/0;
  exception
  when division_por_cero then
    dbms_output.put_line('Divisor es cero');
  when others then
    dbms_output.put_line('Externo      -      Error' || to_Char(sqlcode) || '
' || substr(sqlerrm,1,100));
end;
/
Divisor es cero

Procedimiento PL/SQL terminado con éxito.
```

9. UTILIZACION DE LA FUNCION RAISE_APPLICATION_ERROR

Se puede emplear esta función predefinida para personalizar los mensajes de error que las excepciones nominadas. Su sintaxis es:

```
RAISE_APPLICATION_ERROR(número_error, mensaje_error, {preservar_errores});
```

Donde: **número_error**, es un parámetro comprendido entre 20000 y 20999; **mensaje_error**, es el texto asociado con este error, cuyo texto debe ser máximo 512 caracteres; **preservar_errores**, es un valor booleano y es opcional, si toma el valor de TRUE, el nuevo error se añade a la lista de errores que ya han sido generados (si es que existe) y si toma el valor de false, como lo hace de forma predeterminada, el nuevo error reemplaza a la lista actual de errores.

Ejemplo:

```
declare
  wsalario number;
begin
  wsalario := 10/0;
exception
  when zero_divide then
    dbms_output.put_line('Dbms Mensaje --> Divisor es cero');
    raise_application_error(-20001, 'Raise --> Divisor es cero');
  when others then
    dbms_output.put_line('Externo      -      Error' || to_Char(sqlcode) || '
' || substr(sqlerrm,1,100));
end;
/
```

VIII. TRIGGERS

1. TRIGGERS

- Un TRIGGER es un subprograma que se ejecuta cuando se modifica los datos de una tabla.
- Los TRIGGERS se utilizan para implementar reglas de negocio e integridad de datos.

2. CARACTERISTICAS

- Están asociados a las tablas.
- Se ejecutan en forma automática cuando se modifican los datos de la tabla.
- No pueden recibir parámetros.
- Los TRIGGERS se pueden ejecutar en cascada.
- No se pueden crear TRIGGERS recursivos.
- Los TRIGGERS se compilan la primera vez que son ejecutados.
- Para no afectar el rendimiento del TRIGGER este debe tener máximo una página (60 líneas) como máximo, si existe la necesidad de tener TRIGGERS extensos se debe crear un STORE PROCEDURE.

3. TIPOS DE TRIGGERS

Existen dos tipos de TRIGGERS:

- BEFORE: Se ejecutan antes que se modifiquen los datos.
- AFTER: Se ejecutan después que se modifiquen los datos.

4. SINTAXIS

```
CREATE OR REPLACE TRIGGER nombre_TRIGGER
Before insert or update of col1, col2 on nombre_tabla
For each row
Begin
    :NEW.col1 := upper(:new.col1)
    :NEW.col2 := upper(:new.col2)
End;
```

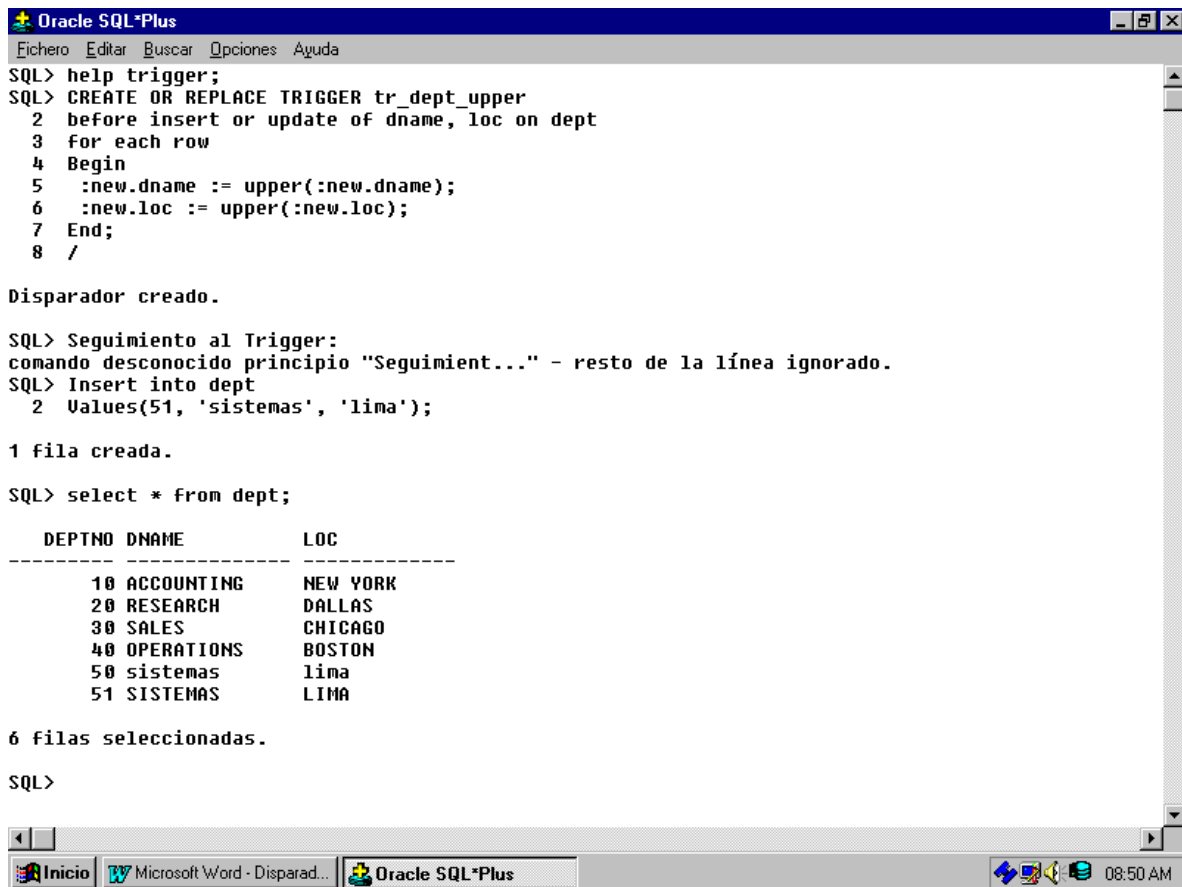
5. EJEMPLOS

◆ TRIGGER BEFORE

◆ INSERTAR EL DEPARTAMENTO FINANZAS EN LA TABLA DEPT.

```
CREATE OR REPLACE TRIGGER tr_dept_upper
before insert or update of dname, loc on dept
for each row
Begin
    :new.dname := upper(:new.dname);
    :new.loc := upper(:new.loc);
End;
/
```

Seguimiento al Trigger:
Insert into dept
Values(52, 'sistemas', 'lima');



The screenshot shows the Oracle SQL*Plus interface. The title bar reads 'Oracle SQL*Plus'. The menu bar includes 'Archivo', 'Editar', 'Buscar', 'Opciones', and 'Ayuda'. The command window contains the following text:

```
SQL> help trigger;
SQL> CREATE OR REPLACE TRIGGER tr_dept_upper
2 before insert or update of dname, loc on dept
3 for each row
4 Begin
5   :new.dname := upper(:new.dname);
6   :new.loc := upper(:new.loc);
7 End;
8 /
```

Disparador creado.

```
SQL> Seguimiento al Trigger:
comando desconocido principio "Seguimient..." - resto de la línea ignorado.
SQL> Insert into dept
2 Values(51, 'sistemas', 'lima');
```

1 fila creada.

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

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON
50	sistemas	lima
51	SISTEMAS	LIMA

6 filas seleccionadas.

SQL>

The taskbar at the bottom shows 'Inicio', 'Microsoft Word - Disparad...', 'Oracle SQL*Plus', and a system clock showing '08:50 AM'.

♦ Borrar TRIGGER

```
drop trigger tr_dept_upper;
```

REGLA DE NEGOCIO:

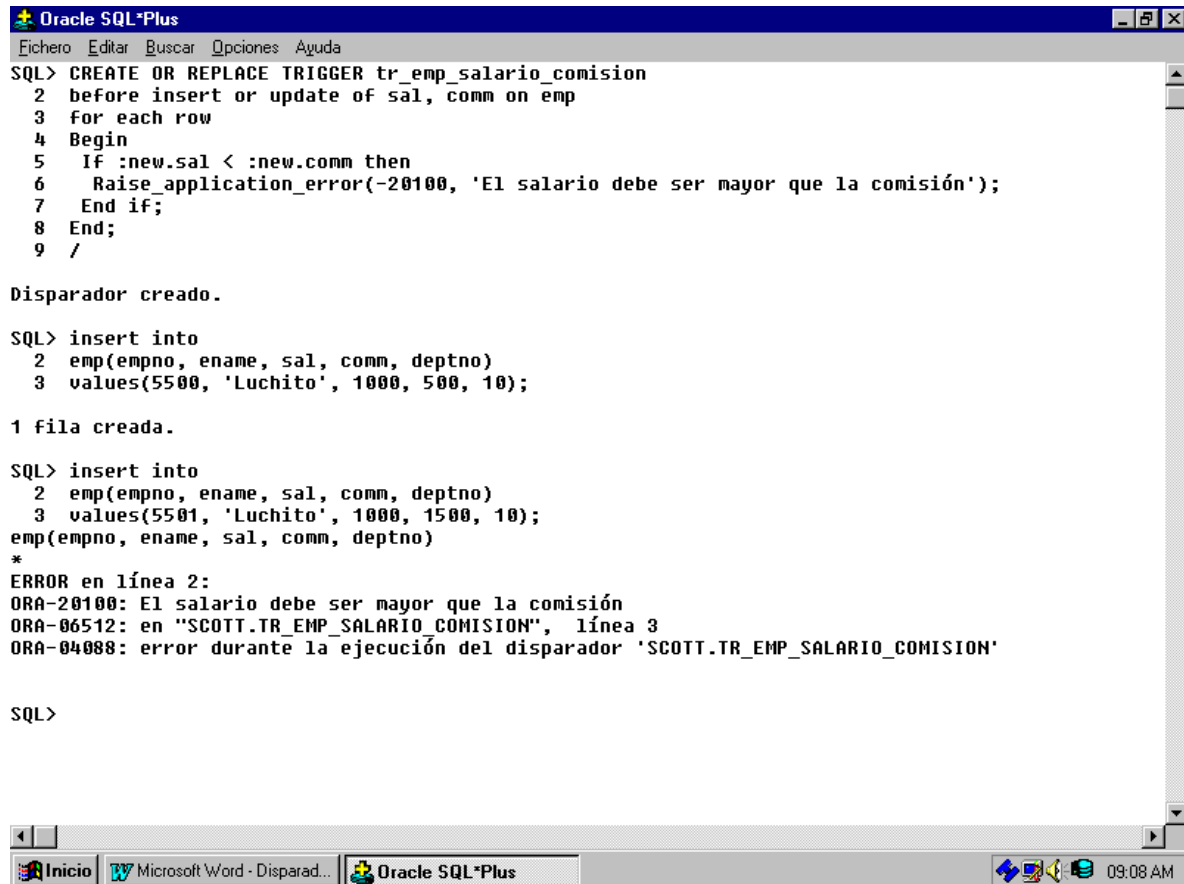
El salario siempre mayor que la comisión

```
CREATE OR REPLACE TRIGGER tr_emp_salario_comision
before insert or update of sal, comm on emp
for each row
Begin
  If :new.sal < :new.comm then
    Raise_application_error(-20100, 'El salario debe ser
mayor que la comisión');
  End if;
End;
/
```

insert into

```
emp(empno, ename, sal, comm, deptno)
values(5500, 'Luchito', 1000, 500, 10);
```

```
insert into
emp(empno, ename, sal, comm, deptno)
values(5501, 'Luchito', 1000, 1500, 10);
```



The screenshot shows the Oracle SQL*Plus interface. The title bar reads "Oracle SQL*Plus". The menu bar includes "Archivo", "Editar", "Buscar", "Opciones", and "Ayuda". The main text area contains the following SQL commands and their output:

```
SQL> CREATE OR REPLACE TRIGGER tr_emp_salario_comision
  2 before insert or update of sal, comm on emp
  3 for each row
  4 Begin
  5   If :new.sal < :new.comm then
  6     Raise_application_error(-20100, 'El salario debe ser mayor que la comisi3n');
  7   End if;
  8 End;
  9 /
```

Disparador creado.

```
SQL> insert into
  2 emp(empno, ename, sal, comm, deptno)
  3 values(5500, 'Luchito', 1000, 500, 10);
```

1 fila creada.

```
SQL> insert into
  2 emp(empno, ename, sal, comm, deptno)
  3 values(5501, 'Luchito', 1000, 1500, 10);
emp(empno, ename, sal, comm, deptno)
*
```

ERROR en lnea 2:
ORA-20100: El salario debe ser mayor que la comisi3n
ORA-06512: en "SCOTT.TR_EMP_SALARIO_COMISION", lnea 3
ORA-04088: error durante la ejecuci3n del disparador 'SCOTT.TR_EMP_SALARIO_COMISION'

SQL>

The taskbar at the bottom shows the "Inicio" button, a taskbar with "Microsoft Word - Disparad..." and "Oracle SQL*Plus", and a system tray with icons for network, volume, and power, along with the time "09:08 AM".

```
select * from emp order by 1;
```

```

Oracle SQL*Plus
Fichero  Editor  Buscar  Opciones  Ayuda
SQL> select * from emp order by 1;

  EMPNO ENAME      JOB      MGR HIREDATE          SAL        COMM     DEPTNO
-----
  2000 Luis Soto  RESEARCH  7698 15/07/01        4500         800         20
  2001 Milton    RESEARCH  7698 15/07/01        4500         800         20
  2002 Oriol     RESEARCH  7698 15/07/01        3500          00         20
  2003 Luchito   RESEARCH  7698 15/07/01        3500          00         30
  2004 Lili      RESEARCH  7698 15/07/01        4200          00         30
  2005 Lili01    RESEARCH  7698 15/07/01        4000          00         10
  2007 Milton    RESEARCH  7698 15/07/01        4500         800         20
  2008 Milton    RESEARCH  7698 15/07/01        5500         800         20
  2009 Milton    RESEARCH  7698 15/07/01        3000         800         20
  2010 Milton    RESEARCH  2001 15/07/01        3000         800         20
  5500 Luchito   RESEARCH  7698 15/07/01        1000         500         10

11 filas seleccionadas.

SQL> |

```

Observe que no se ha insertado el empleado de código 5001.

◆ TRIGGER AFTER.

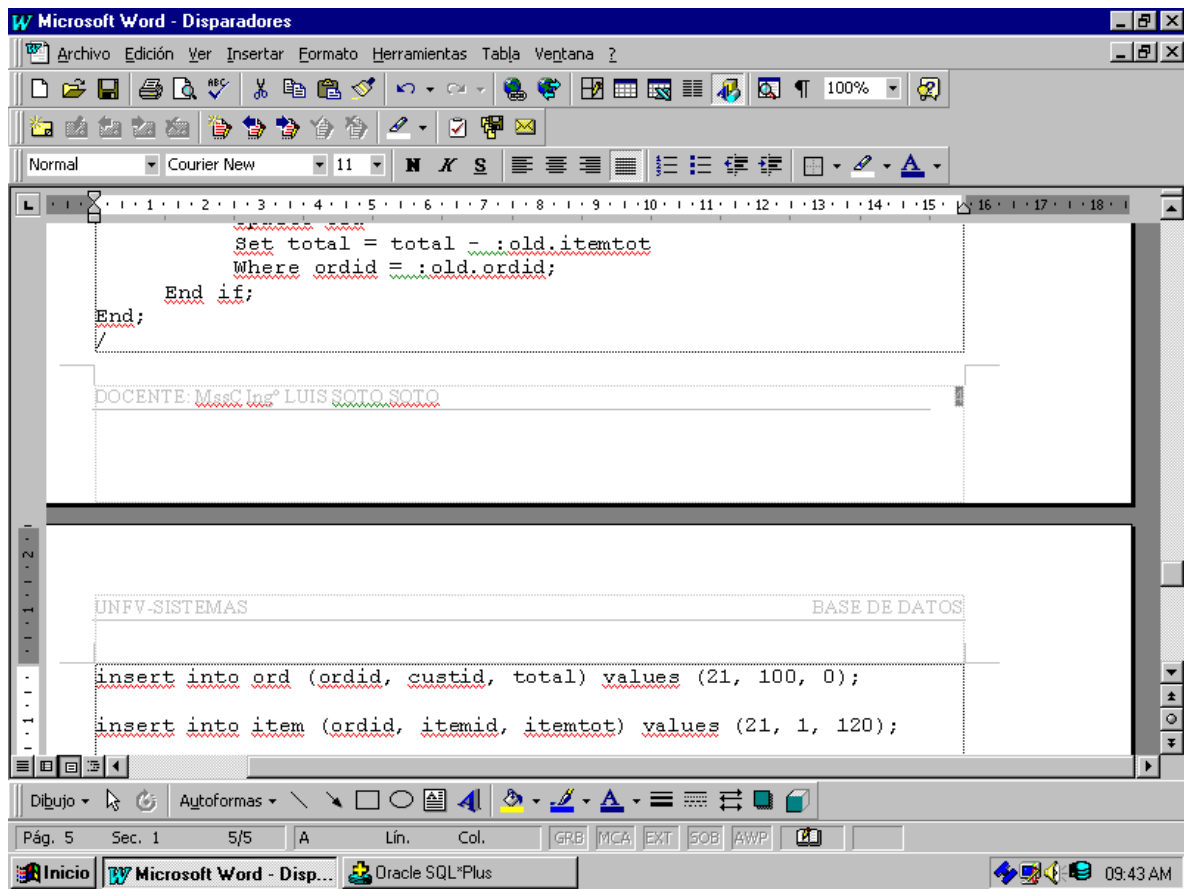
```

CREATE OR REPLACE TRIGGER TR_ORD_ACT_TOTAL
After insert or update or delete on item
For each row
Begin
    If inserting then
        Update ord
        Set total = total + :new.itemtot
        Where ordid = :new.ordid;
    Elsif updating then
        Update ord
        Set total = total + :new.itemtot - :old.itemtot
        Where ordid = :new.ordid;
    Elsif deleting then
        Update ord
        Set total = total - :old.itemtot
        Where ordid = :old.ordid;
    End if;
End;
/

```

```
insert into ord (ordid, custid, total) values (23, 100, 0);

insert into item (ordid, itemid, itemtot) values (23, 1, 120);
```



```
drop TRIGGER TR_ORD_ACT_TOTAL;
```

DESHABILITAR TRIGGER

```
alter trigger nombre disable;
alter trigger TR_ORD_ACT_TOTAL disable;
```

HABILITAR TRIGGER

```
alter trigger nombre enable;
alter trigger TR_ORD_ACT_TOTAL enable;
```