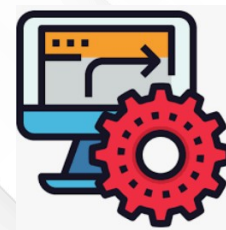
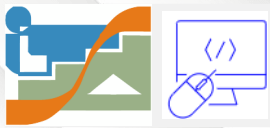




# Automatització de tasques





## Objectius de la unitat

- Reconéixer la importància d'automatitzar tasques administratives.
- Descriure els diferents mètodes d'execució de guions.
- Identificar les eines disponibles per redactar guions.
- Definir i utilitzar guions per automatitzar tasques.
- Identificar els esdeveniments susceptibles d'activar disparadors.
- Definir disparadors.
- Fer servir estructures de control de flux.
- Adoptar mesures per mantenir la integritat i la consistència de la informació.

# Automatització de tasques

---



Quan veges este símbol, hi ha un enllaç, pots punxar per ampliar informació



Quan veges este símbol, hi ha prop un nom de vista de Diccionari de Dades

Prova-ho

Quan veges este símbol, es recomana provar i practicar el codi que hi ha prop



# Rutines de BD

Blocs anònims

Estructures de programació

Procediments

Funcions

Disparadors (triggers)

Seqüències

Tasques automatitzables

Automatització



L'automatització consisteix a fer tasques de manera sistemàtica i repetitiva sense que estiga involucrat un usuari en la seua execució

## Avantatges Automatització

- Estalvi de temps
- Reducció costos administració
- Reducció errors ( humana)

## Tipus

- Programa extern (al SGBD)
- Programa intern : rutina de bbdd + job

Programador de tasques  
en Windows, o  
cron/crontab en Linux

Procediments  
Funcions  
Disparadors



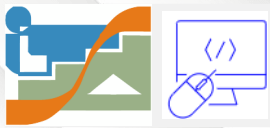


## Rutina

- Script, guió, programa o seqüència de comandos que permeten dur a terme el processament d'unes certes accions.
- Quan és creada rep un **nom** que permet que siga invocada tantes vegades com siga necessari
- Van ser introduïdes en la versió SQL3, o SQL:1999

## El SGBD deu proporcionar:

- Eines necessàries per a crear rutines.
- Eines per a executar les rutines automàticament.



## Avantatges rutina interna

- Rendiment
- Reutilització (codi)
- Encapsula regles de negoci
- Major seguretat

## Les rutines: **S'han de Documentar**

Descripció de la tasca, descripció de paràmetres d'entrada i eixida, Autor, Versió, Data d'última modificació



COM?? Fent servir comentaris  
en el codi de les rutines.  
amb /\* \*/ --



## Bloc anònim

Permet encadenar una o més sentències SQL que s'executen en seqüència. També permeten l'opció de la gestió d'excepcions

. punt

🔊 En **SQL\*Plus**, tot bloc ha d'acabar en **.** perquè siga emmagatzemat en el buffer SQL. Una vegada guardat el podem executar amb l'ordre **run** (en **SQL\*Plus**)

Practica amb l'exemple següent:

```
set serveroutput on
DECLARE
  Vnom VARCHAR2(15) := '&nom';
BEGIN
  DBMS_OUTPUT.PUT_LINE ('Hola ' || Vnom);
  DBMS_OUTPUT.PUT_LINE ('Benvingut a la programació en PL/SQL');
END;
```

Variable de substitució

En **SQL Developer** no es posa el .

Prova-ho en sql-developer  
Observa com funciona

set serveroutput on +  
paquet dbms\_output  
procediment put\_line





## Comandos bàsics en SQL\*Plus

```
SQL> edit
SQL> list
SQL> run (r o /)
SQL> save fitxer.ext [replace]
SQL> get fitxer.ext
SQL> run
SQL> start fitxer.ext
SQL>
SQL> @fitxer.ext (@ equival a start)
```

SQL\*Plus sols guarda la última ordre, que pot tindre diverses línies..  
Esta es pot editar, llistar, executar, etc...

tutorial SQL\*Plus



Practica: amb diversos CREATE TABLE de 2 o 3 línies !! en sql\*plus !!

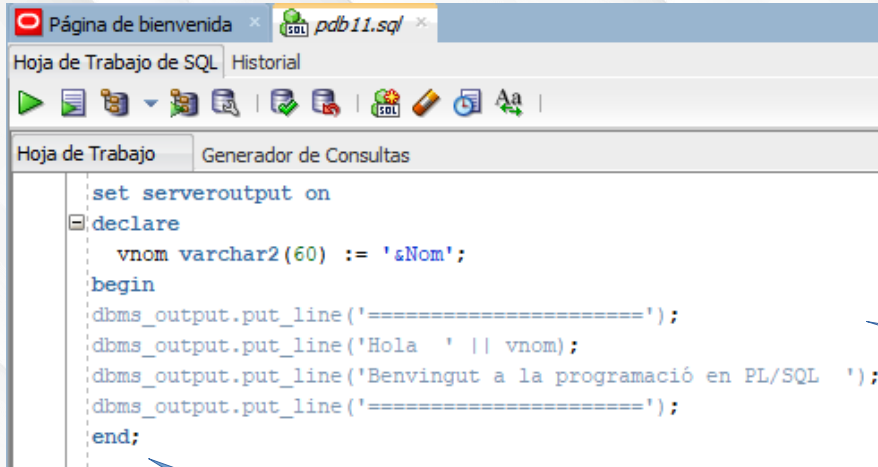
El comando start carrega i executa un script d'un fitxer



## Bloc anònim

🔊 En **SQL Developer** s'executa amb F5

F5 Executa script  
F9 Executa sentència



```
set serveroutput on
declare
  vnom varchar2(60) := '&Nom';
begin
  dbms_output.put_line('=====');
  dbms_output.put_line('Hola ' || vnom);
  dbms_output.put_line('Benvingut a la programació en PL/SQL ');
  dbms_output.put_line('=====');
end;
```

En SQL Developer  
no es posa el .

En este script hi ha dos  
sentències !!  
Un set i un bloc anònim



<codi>



El codi dels procediments, funcions i disparadors pot ser des d'una línia o sentència, fins a un programa complex amb estructures de repetició, selecció i seqüenciació, seguint les regles de PL/SQL

PL/SQL (Procedural Language/Structured Query Language) és un llenguatge de programació incrustat en Oracle (va ser el primer SGBD en incloure un llenguatge dins)

PL/SQL suportarà totes les consultes, ja que la manipulació de dades que s'usa és la mateixa que en SQL

PostgreSQL dona suport a una variant, el PL/pgSQL  
SQL Server utilitza una altra variant, el TSQL

Son tots molt pareguts



# Estructures de programació en PL/SQL

- Comentaris
- Variables
- Execució condicional
  - Bucles
  - Blocs
  - Cursors
- Transaccions
- Excepcions





## Estructures de programació en PL/SQL

- Comentaris
- Variables
- Execució condicional
  - Bucles
  - Blocs
  - **Cursors**
- **Transaccions**
- **Excepcions**



Repàs de  
primer curs

Altres  
estructures  
en PL/SQL





## - Comentaris

Una línia --

Més d'una línia /\* .... \*/

Molt important, per documentar el codi



## - Variables

`v_nom := 'Francisco';`      `v_empno := 10;`

tutorial tipos de dades

Assignació de valors

**Tipus :** CHAR() varchar2() number() boolean DATE tutorial dates

`anynou DATE:='01/ene/2024';`      `B1 boolean := true;`  
`dataactual DATE:=SYSDATE;`      `B2 boolean := false;`

Declaració  
( i assignació )

## - Operacions

`+` `-` `*` `/` `**` `||` `:=` tutorial operadors



Un bloc condicional sols s'executa si es compleix la condició

## - Execució condicional (simple)

**IF** condicion1 **THEN** instrucció/es; **END IF**;

Una condició sempre s'avalua a **Vertader** o **Fals**. És una pregunta amb resposta V o F

## - Condicions

### Exemples

**var1 > 1000**      **1000 > var1**      **vnum3<> vnum5 + 100**

**var2 >= num and var3=var4 or not( var7 > 55)**

**vdata1 > sysdate**      **vdata2 = '01/01/2024'**      **to\_char(vdata2,'dd/mm/yyyy') = '01/01/2024'**

**vnum5 BETWEEN 1 and 10**

**vchar LIKE expr**      **vprovin LIKE 'VAL%'**

**var5 IN (6 ,7 ,8 )**      **var7 NOT IN (1 ,6 ,9 )**

**Compte amb el valor NULL**      **var9 IS NOT NULL**

SYSDATE és la data/hora actual

Més correcte !!l

El valor NULL no es pot comparar.  
Sols es pot comprovar.



## - Execució condicional (estesa)

```
IF condicion1 THEN instrucció/es;  
[ELSIF condicion2 THEN instrucció/es; ]  
[ELSE instrucció/es; ]  
END IF;
```

+Info

tutorial condicionals



```
CASE expr  
WHEN valor THEN  
    instrucciones1  
[WHEN valor THEN  
    instrucciones2]  
[ ELSE  
    instrucciones3]  
END CASE;
```

+Info

```
CASE             
WHEN condicion1 THEN  
    instrucciones1  
[WHEN condicion2 THEN  
    instrucciones2]  
[ ELSE  
    instrucciones3]  
END CASE;
```



## - Bucles o Estructures de repetició

### LOOP

sentències

EXIT (dins d'un if )

END LOOP;

+Info

WHILE condició LOOP

sentències

END LOOP;

+Info

FOR v\_comptador IN [REVERSE] liminf..limsup LOOP  
sentències  
END LOOP;

+Info

CONTINUE  
EXIT

Un bucle, en programació, és una seqüència d'instruccions de codi que s'executa repetides vegades, fins que la condició assignada a aquest bucle deixa de complir-se

Tutorial bucles





## Precedència d'operadors

$2+3*6-5$  **no és**  $(2+3)*(6-5)$   
 $2**3+2+1$  **no és**  $2**(3+2)+1$

Operator	Operation
()	parenthesis inside out
**	exponentiation
+, -	identity, negation ( <b>unario</b> )
*, /	multiplication, division
+, -,	addition, subtraction, concatenation
=, <, >, <=, >=, <>, !=, ~=, ^=, IS NULL, LIKE, BETWEEN, IN	comparison
NOT	logical negation
AND	conjunction (logical)
OR	inclusion (logical)

tutorial precedència







## Precedència d'operadors

$2+3*6-5=15$  no és  $(2+3)*(6-5)=5$   
 $2**3+2+1=11$  no és  $2**(3+2)+1=33$

Operator	Operation
()	parenthesis inside out
**	exponentiation
+, -	identity, negation (unario)
*, /	multiplication, division
+, -,	addition, subtraction, concatenation
=, <, >, <=, >=, <>, !=, ~=, ^=, IS NULL, LIKE, BETWEEN, IN	comparison
NOT	logical negation
AND	conjunction (logical)
OR	inclusion (logical)

$-3**2=-9$  no es  $(-3)**2=9$

Prova-ho en sql-developer  
Observa com funciona



## Procediments - Funcions - Triggers

Un procediment emmagatzemat o STORED PROCEDURE és un codi SQL preparat que es pot guardar, per la qual cosa el codi pot reutilitzar-se una vegada i una altra. (tindrà un nom)

Així que, si es té una consulta SQL que s'ha d'escriure una vegada i una altra, es podrà guardar com un procediment emmagatzemat i després cridar-la per a executar-la.

També pot passar paràmetres a un procediment emmagatzemat, de manera que el procediment emmagatzemat pugui actuar en funció dels valors de paràmetre que es passen.



## Procediments Funcions Triggers

### Permisos

- Creació i execució
- Sols execució
- Cap permís

🔊 Un usuari necessita tindre permís per executar rutines, o per crear-les i executar-es !!

El permís val per procediments  
I també per funcions

procediments  
funcions  
triggers

`GRANT CREATE PROCEDURE, CREATE TRIGGER TO <usu>;`  
(si te permís de crear, també pot executar)

`GRANT EXECUTE ON <esquema>.<procedim> TO <usu> [WITH GRANT OPTION];`



## Procediments [emmagatzemats]

- No tornen cap informació (valor)

- Com crear-los -

```
CREATE [OR REPLACE] PROCEDURE <nom_proc> [(  
    <param1> [ IN | OUT | IN OUT ] <tipus>,...)]  
[AUTHID current_user | definer]  
AS  
    [<declaració variables>]  
BEGIN  
    <codi pl/sql>  
[EXCEPTION]  
    [<codi excepció>]  
END;  
/
```

Per defecte, definer  
(creador).  
Authid determina amb  
quins permisos  
s'executarà l'script

Un procediment [emmagatzemat] és un subprograma que executa una acció específica i que no retorna cap valor per si mateix, com succeeix amb les funcions. Un procediment té un nom, un conjunt de paràmetres (opcional) i un bloc de codi.



## Procediments - exemple

```
CREATE OR REPLACE  
PROCEDURE Actualiza_Saldo(cuenta NUMBER, new_saldo NUMBER)  
IS  
    -- Lloc per a Declaració de variables locals  
BEGIN  
    UPDATE SALDOS_CUENTAS  
        SET SALDO = new_saldo, DATA_ACTUALITZACIO = SYSDATE  
        WHERE CO_CUENTA = cuenta;  
END Actualiza_Saldo;  
/
```

En SQL\*Plus, finalitza la definició del procediment (també es pot posar un punt . )





## Procediments

### - Com executar-los

```
execute <nom_proc> ( <param1>, <param2> ...); // oracle També amb exec  
execute <nom_proc> ;           execute <nom_esquema>.<nom_proc> ;
```

### - Com esborrar-los

```
DROP PROCEDURE nom_proc;
```

### - Com explorar-los

En DD user\_procedures

```
select object_name, object_type from user_procedures;  
select object_name, object_type, status from user_objects;
```

user\_procedures  
user\_source  
user\_objects





## Procediments – exemple senzill

```
CREATE OR REPLACE PROCEDURE dimehora  
IS BEGIN  
    DBMS_OUTPUT.PUT_LINE ('L''hora en este moment es ' || to_char(sysdate,'hh:mi:ss'));  
    DBMS_OUTPUT.PUT_LINE ('Sols hora ' || to_char(sysdate,'hh'));  
    DBMS_OUTPUT.PUT_LINE ('El diames ' || to_char(sysdate,'dd'));  
    DBMS_OUTPUT.PUT_LINE ('El diasem ' || to_char(sysdate,'d'));  
END ;
```

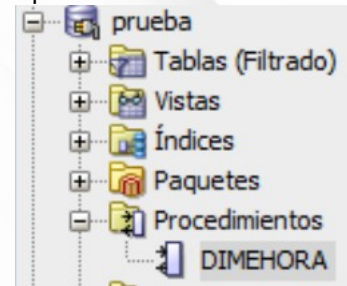
execute dimehora;

execute dimehora;

execute dimehora;

Prova-ho en SQL Developer  
Crea el procediment  
Executa'l

Editar procediment



Compilar procediment





## Procediments

### - Com executar-los

#### Notació posicional

Es passen els valors dels paràmetres en el mateix ordre en que el procedure els defineix.

```
BEGIN  
  actualiza_Saldo(200501,2500);  
  COMMIT;  
  DBMS_OUTPUT.put ('Saldo act');  
END;
```

En sql\*plus es deu activar prèviament amb  
SET serveroutput ON

Des de qualsevol rutina interna d'Oracle ( inclús des d'un bloc d'instruccions) es pot cridar a un procediment o funció invocant-la directament com una instrucció, sense la necessitat d'utilitzar **execute** o **exec**. També es poden utilitzar funcions i procediments de paquets, posant el nom del paquet, punt, el nom de la funció o procediment.



## Procediments

### - Com executar-los

#### Notació nominal

Es passen els valors en qualsevol orde, nominant explícitament el paràmetre i el seu valor separats pel símbol =>.

```
BEGIN  
    actualiza_Saldo(cuenta => 200501,new_saldo => 2500);  
    COMMIT;  
END;
```

Prova-ho en SQL Developer



## Funcions - Tornen informació (un únic valor, d'un tipus)

Existeixen funcions predefinides que podem utilitzar en oracle

Com se  
solen usar  
les funcions

```
select sysdate from dual;  
select sqrt(5) from dual;  
update taula set camp=lower('DadesDe1CaMp') ; <-- sense WHERE ,tota la taula!
```

Funcions numèriques: round,trunc,mod,sqrt,power,sign,abs,...

de cadenes (strings): lower,upper,trim,substr,length,replace,instr,translate,chr,ascii...

de treball en NULLs: nvl,nvl2, nullif, coalesce

de dates: sysdate,last\_day, extract, add\_months, ....

de conversió: to\_number, to\_date, to\_char,...

i altres avançades...

A més a més, **Oracle permet definir noves funcions** que podem utilitzar.







## Funcions - Tornen informació (un únic valor, d'un tipus)

Exemples d'ús de funcions predefinides

```
declare vnum number := &dame_un_numero;  vcad := '&dame_un_string';
begin
dbms_output.put_line('Su cuadrado es ' || power(vnum,2) );
dbms_output.put_line('El resto de dividir por 3 es ' || mod(vnum,3 );
dbms_output.put_line('La raiz cuadrada es ' || sqrt(vnum) );
dbms_output.put_line('La longitud de la cadena es ' || length(vcad) );
dbms_output.put_line('La ultima letra es' || substr(vcad ,length(vcad),1
));
dbms_output.put_line('El ultimo dia del mes es ' || last_day(sysdate));
end;
```



A més a més, **Oracle permet definir noves funcions** que podrem utilitzar.

## Funcions - Tornen informació (un únic valor, d'un tipus)

### - Com definir noves funcions / crear-les

```
CREATE [OR REPLACE] FUNCTION <nom_func> [(  
    <param> [ IN | OUT | IN OUT ] <tipus>,.. )]  
RETURN <tipus>  
[AUTHID current_user | definer]  
IS  
    [ <variables> ]  
BEGIN  
    <codi>  
    RETURN <valor>  
END;  
/
```

Sempre ha  
de tornar un  
valor



## Funcions

En Oracle existeix una taula de sistema anomenada **dual** que permet executar consultes que no accedeixen a cap taula de la bbdd, la qual cosa és molt útil per a comprovar el resultat d'invocar una funció.

```
select sysdate from dual;  
select funcio_de_usuari(valor_parametre) from dual;  
select funcio_de_usuari(nom_par => valor) from dual;  
-> notació posicional o nominal <-
```



- Com explorar-les en el DD  
**vista** `user_procedures`

```
select object_name, object_type from user_procedures;
```

Funcions i procediments estan junts en el DD, el camp `object_type` els diferencia

`user_procedures`  
`user_source`  
`user_objects`



## Funcions

### -exemple-

```
create or replace function f_incremento (avalor number,  
aincremento number)  
return number  
is  
begin  
return avalor + (avalor*aincremento/100);  
end;
```

Prova-ho en SQL Developer  
Crea la funció  
Executa-la

-----  
**Utilitzar una funció.**

```
select titulo,precio,f_incremento(precio,20) from libros;
```

-----

**Esborrar una funció.**

```
DROP FUNCTION f_incremento;
```



## Funcions - Exemple

```
create or replace function notaCHAR (avalor number)
return varchar2
is
    valorretornado varchar2(20);
begin
    valorretornado:='';
    if avalor>=5 then
        valorretornado:='APROBADO';
    else valorretornado:='NO APROBADO';
    end if;
    return valorretornado;
end;
/
```

Podriem utilitzar la funció, per exemple, dins d'un select..

```
select count(*),notaCHAR(nota) from taula_notes group by notaCHAR(nota);
```

```
if avalor>=5 then
    return 'APROBADO';
else
    return 'NO APROBADO';
end if;
```

Prova-ho en SQL Developer  
Crea la taula \_notes i ompli-la  
Crea la funció  
Executa-la



## Funcions - Resum

### Tipos de bloque PL/SQL

#### Anónimo

```
[DECLARE]  
  
variables  
  
BEGIN  
    --statements  
  
[EXCEPTION]  
  
END;
```

#### Función

```
FUNCTION name  
RETURN datatype  
IS  
BEGIN  
    --statements  
    RETURN value;  
[EXCEPTION]  
  
END;
```

#### Procedimiento

```
PROCEDURE name  
IS  
BEGIN  
    --statements  
  
[EXCEPTION]  
  
END;
```



## Funcions - Resum

- **DECLARE (optional)**
  - Variables, cursors, user-defined exceptions
- **BEGIN (Obligatorio)**
  - Instrucciones SQL
  - Instrucciones PL/SQL
- **EXCEPTION (optional)**
  - Actions a ejecutar cuando ocurran errores
- **END; (obligatorio)**





## Disparadors (trigger)

Un trigger o disparador en una Base de Dades , és un bloc de codi que s'executa (automàticament) quan es compleix una condició establida, com per exemple, realitzar una **operació** (INSERT, UPDATE, DELETE) sobre una taula (o sobre un camp d'una taula)

Els triggers poden ser d'inserció (INSERT), actualització (UPDATE) o esborrat (DELETE).

El procediment s'executarà abans (BEFORE), després (AFTER) de que es realitzi l'**operació**, o (INSTEAD OF) en compte de l'**operació**.

Segons el cas, es poden utilitzar valors d'una fila abans de la operació o després de l'**operació**: :new i :old



## Disparadors (trigger)

```
CREATE [OR REPLACE] TRIGGER <nom_disp>
BEFORE | AFTER | INSTEAD OF
INSERT | DELETE | UPDATE | UPDATE OF <column1> [, column2,...]
ON <nom_taula>
[REFERENCING OLD <nomold> NEW <nomnew> ]
[FOR EACH ROW ]      <- si se omite actua como STATEMENT
[WHEN condició]
[DECLARE]
BEGIN
    <codi>
END <nom_disp>;
/
```

Per defecte

## Disparadors (trigger)

**FOR EACH ROW**

El codi s'executa tantes vegades com files afectades per la sentència que ha disparat el trigger (abans o després de cada fila)

~~FOR EACH STATEMENT~~ (per defecte) s'indica no posant res

El codi s'executa **una vegada**, abans o després de la sentència que ha disparat el trigger

**WHEN condició**

El codi s'executa si es compleix la condició, en el moment que li haguera tocat executar-se.





BeforeStatement

Suposem un UPDATE que afecta a 3 files d'una taula. Observem quan s'executaria el trigger depenent del tipus.

BeforeRow

AfterRow

BeforeRow

AfterRow

BeforeRow

AfterRow

7839	KING	1200	30
7698	BLAKE	2100	30
7788	SMITH	2300	30

AfterStatement



## Disparadors (trigger) exemple

```
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;
/
```

Prova-ho en SQL Developer  
Crea taula **libros** i ompli  
Crea taula **control**  
Crea trigger i compila  
Actualitza preus  
Oberva com s'ompli la  
taula **control**

El trigger s'activarà quan es llance una sentència que vaja a actualitzar el valor del camp «precio» de la taula «libros» (update of precio on libros)  
El codi s'executara abans (before) d'actualitzar el valor de «precio» per cada fila que es vaja a actualitzar (for each row)

Exemple: `UPDATE libros SET precio=precio*0.95 WHERE editorial='McGraw Hill';`



## Disparadors (trigger) raise\_application\_error

El procediment "raise\_application\_error" permet emetre un missatge d'error. El NUMERO de missatge ha de ser un número negatiu entre -20000 i -20999 i el missatge de TEXT una cadena de caràcters de fins a 2048 bytes.

Si durant l'execució d'un trigger es produeix un error definit per l'usuari, **s'anul·len totes les actualitzacions** realitzades per l'acció del trigger així com l'esdeveniment que la va activar, és a dir, es reprén qualsevol efecte retornant un missatge i es desfà l'ordre executada.

```
create or replace trigger tr_actualitzar_preu_neg
before update of precio on libros
for each row
begin
    if :new.preu<0 then
        raise_application_error(-20020,'No es permet preu negatiu');
    end if;
end tr_actualitzar_preu_neg;
/
```

Prova-ho en SQL Developer  
Crea trigger i compila  
Actualitza preus en negatiu  
Oberva si actualitza tots,  
alguns o cap

**Prova: UPDATE libros SET precio= -50 ;**

## Disparadors (trigger)

Per a un activador INSERT, :OLD no conté valors, i :NEW conté els valors nous.

Per a un activador UPDATE, :OLD conté els valors antics, i :NEW conté els valors nous.

Per a un activador DELETE, :OLD conté els valors antics, i :NEW no conté valors.



**- Com explorar-los**

**En DD user\_triggers**

```
select * from user_triggers where trigger_name='TR_MITRIGGER';
```

**dba\_triggers**  
**dba\_source**

## Disparadors múltiples (trigger) exemple

```
create or replace trigger tr_actualizar_precio_libros
before insert or delete on llibres
for each row
Begin
    If inserting then
        insert into control values(user,sysdate,:new.codigo,'INSERTAR');
    Else
        insert into control values(user,sysdate,:old.codigo,'ESBORRAR');
    End if;
end tr_actualizar_precio_libros;
```

INSERTING  
UPDATING  
DELETING

El trigger s'activarà quan es llance una sentència insert o una sentència delete sobre la taula llibres

El codi s'executara abans (before) de cada fila afectada.





## Disparadors (trigger)

En oracle existeixen altres tipus de triggers, segons l'esdeveniment que el dispara:

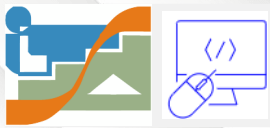
- Un INSERT, UPDATE o DELETE en una taula específica ← **Hem vist**
- Un CREATE, ALTER o DROP en qualsevol objecte d'esquema
- Una arrancada(startup) de Base de Dades o un tancament (shutdown) d'instància
- Un missatge d'error
- Un inici o final de sessió d'usuari de Base de Dades

Els triggers es poden deshabilitar temporalment:

```
alter trigger tr_trig1 disable;  
alter trigger tr_trig1 enable;
```

Es poden habilitar/deshabilitar tots els triggers d'una taula:

```
alter table nom_taula disable all triggers;  
alter table nom_taula enable all triggers;
```



## Disparadors (trigger)

### Restriccions en l'ús de disparadors

- No poden executar-se instruccions DDL
- No poden executar-se instruccions de TCL
- Per sentència, no te sentit l'ús de :old i :new
- Per fila. No es poden consultar les dades de la taula que ha disparat el trigger, es a dir, no es pot fer un SELECT
- **Oracle no deixa crear triggers en objectes de l'esquema de SYS !!**



Arregla  
problema de  
concurrència

## Seqüències

Una **seqüència (sequence)** s'empra per a generar valors sencers seqüencials **únics** i assignar-li'ls a camps numèrics; s'utilitzen generalment per a les claus primàries de les taules garantint que els seus valors no es repetisquen.

Una seqüència és una taula amb un camp numèric en el qual s'emmagatzema un valor i cada vegada que es consulta, s'incrementa tal valor per a la pròxima consulta.

Permís:

```
GRANT CREATE SEQUENCE TO <usu>;
```

Crear:

```
CREATE SEQUENCE <NOM_SEQ>;
```

Esborrar:

```
DROP SEQUENCE <NOM_SEQ>;
```

Utilitzar seqüència

```
INSERT INTO taula VALUES  
(nom_seq.nextval, 'valor1',  
'valor2', ...);
```

Prova-ho en SQL Developer  
Crea la seqüència  
Prova-la



Vista del DD: **dba\_sequences**



Arregla  
problema de  
concurrència

## Identity Column

Una **identity column** és un altre mecanisme d'oracle per generar valors sencers seqüencials **únics** i assignar-li'ls a camps numèrics; s'utilitzen generalment per a les claus primàries de les taules garantint que els seus valors no es repetisquen.

```
CREATE TABLE empleados (  
  id NUMBER  
  GENERATED BY DEFAULT AS IDENTITY  
  PRIMARY KEY,  
  .....);
```

Prova-ho en SQL Developer  
Crea la seqüència  
Prova-la

## Diferències entre Identity i Sequence

Característica	IDENTITY	SEQUENCE
Creació	Es defineix dins de la taula en crear una columna amb <code>GENERATED AS IDENTITY</code>	Es crea per separat amb <code>CREATE SEQUENCE</code> i s'utilitza manualment en les insercions
Associació amb taula	Lligada directament a una columna específica	Independent, es pot utilitzar en múltiples taules o columnes
Inserció de valors	Oracle assigna automàticament els valors	S'ha de cridar manualment amb <code>NEXTVAL</code>
Control de valors	Opcions limitades (es pot personalitzar <code>START WITH</code> , <code>INCREMENT BY</code> , etc.)	Més flexible, permet modificar valors en qualsevol moment
Reutilització en altres taules	No es pot compartir entre múltiples taules	Es pot utilitzar en diverses taules per a mantindre numeracions consistents
Eliminació	Si s'elimina la taula, la identitat desapareix amb ella	La seqüència persisteix encara que la taula siga eliminada



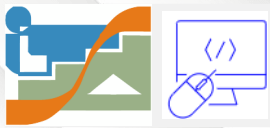
## Estructures de programació en PL/SQL

- Comentaris
- Variables
- Execució condicional
  - Bucles
  - Blocs
  - **Cursors**
- **Transaccions**
- **Excepcions**



Repàs de  
primer curs

Altres  
estructures  
en PL/SQL



## Cursors

En **PL/SQL** no es poden utilitzar sentències SELECT de sintaxi bàsica ( SELECT <lista> FROM <tabla> ).

PL/SQL utilitza cursors per a gestionar les instruccions SELECT.

Un cursor és un conjunt de registres retornat per una instrucció SQL.

Dos tipus

Els cursors implícits només poden retornar una única fila. En cas que es retorne més d'una fila (o cap fila) es produirà una excepció: **NO\_DATA\_FOUND** o **TOO\_MANY\_ROWS**

- Implícits => select into ( no es declaren, sols tornen un resultat o fila)
- Explícits => es declaren i controlen pel programador. Poden tornar més d'un resultat



## Cursors

Per a treballar amb un cursor ( **implícit** ) cal realitzar els següents passos:

```
SET SERVEROUTPUT ON;
declare
  vdescripcion VARCHAR2(50);
begin
  SELECT DESCRIPCION INTO vdescripcion from PAISES WHERE CO_PAIS = 'ESP';
  dbms_output.put_line('La lectura del cursor es: ' || vdescripcion);
End;
```

TIP: Per evitar errors, podem utilitzar funcions d'agregació quan siga possible.

```
*SELECT max(cant) INTO vcant from factura WHERE idcli = 'V00923';
```

TIP: Utilitzar EXCEPTION

Es pot produir un error si no torna cap resultat o si torna més d'un resultat

Si es produeix un error, podem tractar-lo amb una excepció: ( **h**o **v**orem més endavant)  
En este cas podem utilitzar  
SQL%FOUND - SQL%NOTFOUND  
- SQL%ROWCOUNT

## Cursors

Per a treballar amb un cursor ( **explícit** ) cal realitzar els següents passos:

- Declarar el cursor

```
CURSOR .nom. IS .....
```

- Obrir el cursor en el servidor

```
OPEN .nom.
```

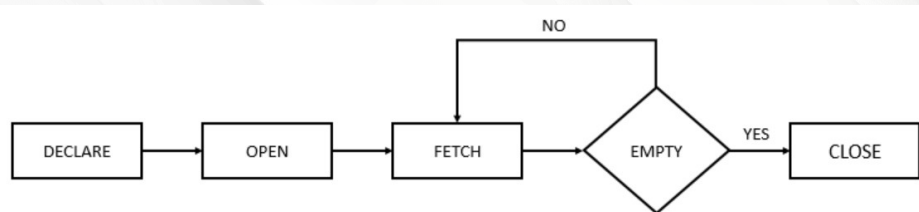
- Recuperar cadascuna de les seues files ( **bucle** )

```
FETCH .nom. INTO ..variable/s.
```

- Tancar el cursor

```
CLOSE .nom.
```

nom\_cursor%NOTFOUND  
nom\_cursor%FOUND



En cursors implícits podem utilitzar  
SQL%FOUND  
SQL%NOTFOUND  
SQL%ROWCOUNT  
Després d'executar l'ordre



## Cursors

### Definir (en el DECLARE)

```
CURSOR nom_cursor (par1 tipus1, par2 tipus2, ..) IS SELECT ...;
```

### Utilitzar (en el BEGIN)

```
OPEN nom_cursor ( par1, par2) ;  
FETCH nom_cursor INTO v_aux;  
CLOSE nom_cursor;
```

Carrega la primera fila en v\_aux

Important !! tancar el cursor

Vista  
dinàmica

v\$open\_cursor







## Cursors

### Utilitzar cursor amb sentència FOR

```
cursor c_articulos is select idarticulo from ....
```

```
BEGIN
```

```
FOR datos IN c_articulos LOOP
```

```
    sentències
```

```
END LOOP;
```

```
FOR datos IN nom_cursor(par1,par2) LOOP
```

```
    update personal set nom=datos.nom, coddep=datos.coddep
```

```
    where codemp=datos.codemp;
```

```
END LOOP;
```

Quan un cursor torna moltes files, es pot processar amb una bucle FOR

També es pot processar amb un bucle LOOP o un WHILE

El bucle FOR tanca el cursor automàticament en acabar

El bucle FOR fa el OPEN, FETCH, CLOSE cursor internament



## Cursors - exemple

### Utilitzar cursor amb sentència FOR

```
create or replace procedure pr_guardar_preu as
  cursor cli is select num1,preu from taula1 where estat='actiu';
begin
  for x in cli loop
    insert into taula2 values ( x.num1, sysdate, x.preu, 'estat');
  end loop;
end;
/
```

x és una variable  
de tipus 'registre'

Registre

	num1	preu
X	254	10,50

C  
u  
r  
s  
o  
r

cli	
num1	preu
254	10,50
344	12,00
23	23,00
556	7,00
87	21,40
321	60,00



## Cursors - exemple

### Dades estructurades vs Dades escalars

```
a number(8,2);  
b varchar2(40);  
c date;
```

a	56,80
b	'hola mon'
c	30/12/24

Tipus escalars

```
type llibre is record (  
titol varchar2(50),  
autor varchar2(50),  
preu number );
```

```
LL1 llibre;
```

LL1 és una variable Registre

Accedim amb LL1.titol , LL1.autor , LL1.preu

LL1	titol	Oracle 19c Multitenant Architecture for Beginners
	autor	Suchit Kumar Pati
	preu	10,5

Les dades estructurades en PL/SQL son registres.  
**Un registre és com una fila d'una taula !**



- **Transaccions** -  
tablespace de UNDO 'limitat'  
commit en procediments

Si s'inicia una transacció i no es fa commit, UNDO creix i s'ompli

!

- **Excepcions** -

**BEGIN**

sentències

**EXCEPTION WHEN error THEN**

sentències quan error

**[RAISE\_APPLICATION\_ERROR(SQLCODE, SQLERRM)]**

**END;**

Els procediments no inclouen per defecte un "commit". És important recordar-se d'executar un commit després de l'execució d'un procediment perquè els canvis siguin persistents en la base de dades.

\*Para la execució de la rutina



## - Excepcions -

**EXCEPTION WHEN DIVISION BY ZERO THEN ...**

**EXCEPTION WHEN OTHERS THEN ...**

sentències quan error

**DBMS\_OUTPUT.PUT\_LINE('Se ha producido el error ' || SQLERRM);**  
**[RAISE\_APPLICATION\_ERROR(SQLCODE, SQLERRM)]** \*Para la execució de la rutina

Des de qualsevol rutina interna de Oracle es pot cridar a qualsevol funció o procediment invocant-ho directament sense necessitat d'incloure "execute"  
Es poden utilitzar les funcions o procediments inclosos en paquets del DD, precedint el nom de procediment o funció del nom del paquet al qual pertany

SQLCODE i SQLERRM  
existeixen dins dels  
blocs de captura  
d'excepcions



Oracle deshabilita per defecte l'eixida per pantalla. Per a veure els missatges emesos per "dbms\_output.put\_line" s'haurà d'habilitar prèviament l'eixida per pantalla amb la sentència "SET SERVEROUTPUT ON"





## - Excepcions - exemple

```
create or replace procedure preullibre (vcod number)
as
    vpreu number;
begin
    select preu into vpreu from llibres where codllibre=vcod;
    dbms_output.put_line(vpreu);
exception when others then
    dbms_output.put_line('error capturado. ' || sqlerrm);
end;
=====
set serveroutput on
exec preullibre(1);
exec preullibre(3);
```

SQLCODE i SQLERRM  
existeixen dins dels  
blocs de captura  
d'excepcions

Prova-ho en SQL Developer  
Crea el procediment  
Executa'l amb un codi que  
no estiga en la taula

CODLLIBRE	TITOL	PREU
1	java	24
2	pl/sql	35



## Tasques automatitzables

- Tasques de BD
- Tasques d'administració

- Còpies de Seguretat
- Particionament
- Execució d'estadístiques – en horari de càrrega mínima
  - Desfragmentació - **alter table <nom> move;**
  - Reconstrucció d'índex - **alter index <nom> rebuild;**
  - purgat i passe a històric

Estes  
tasques  
es veuen  
més  
endavant  
... en la  
unitat 5



## Tasques d'administració

- S'utilitzarà el diccionari de dades (amb un cursor)
- Per a cada objecte de DD s'aplica la sentència corresponent

DDL: Data Definition Language

Dins d'un procediment no es poden executar sentències de DDL directament.  
S'utilitzarà la sentència «execute immediate»

```
execute immediate ('alter table' || par_taula || 'move');
```





## Tasques d'administració - exemple



```
create procedure defrag_taulas (vuser varchar2(20))
-- Este proc desfragmenta totes les taules d'un usuari/schema
as
  cursor c is select table_name,tablespace_name
    from dba_tables where user=vuser ;
begin
  for x in c loop
    execute immediate ('alter table ' || x.table_name || ' move');
    dbms_output.put_line('Taula desfragmentada:' || x.table_name);
  end loop;
end;
```

Prova-ho en SQL Developer  
Crea el procediment  
Executa'l



## Particionament

Permet dividir una taula gran en subtaules

Avantatges:

- Optimitza l'accés a la taula
- Optimitza les tasques d'administració
- Facilita el purgat de dades
- Facilita el manteniment
- Permet la realització de operacions d'optimització avançades

Formes de particionar:

- Particions fixes (es definix quan es crea la taula)
- Particions variables (es definix al llarg del temps)

El particionament es  
veu més endavant...  
en la unitat 5



# **Automatització de tasques en ORACLE**

---



## Eines del SO

```
sqlplus usuari/password@servei @script.sql > log.txt
```

I esta tasca la llancem amb el programador de tasques de SO

➡ problema de seguretat. El password està en clar

Solució: Utilitzar eines del SGBD : dbms\_scheduler



## DBMS Scheduler - Permisos

🔊 Un usuari necessita tindre permís per crear jobs (treballs) i executar-los !!

```
grant create job to <usuari>
```

```
grant execute on dbms_scheduler to <usuari>;
```



DD  
Vista del DD :  
**dba\_scheduler\_jobs**



## DBMS Scheduler

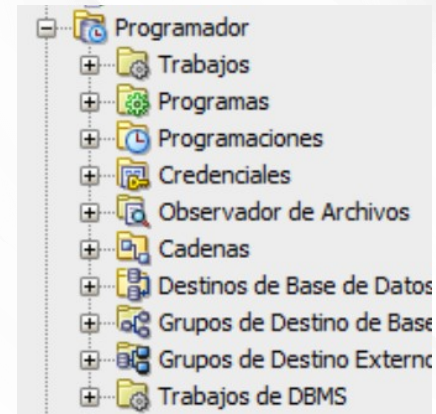
DBMS\_JOB en versions anteriors a 10g.  
Ara, **DBMS\_SCHEDULER**

Programa «jobs» i «chains»

```
DBMS_SCHEDULER.CREATE_JOB ( atribut=>valor, .....);  
DBMS_SCHEDULER.SET_JOB_ARGUMENT_VALUE('nom_job',1,'valor');  
DBMS_SCHEDULER.SET_JOB_ARGUMENT_VALUE('nom_job',2,'valor2');  
DBMS_SCHEDULER.ENABLE('nom_job');  
DBMS_SCHEDULER.DISABLE('nom_job');  
DBMS_SCHEDULER.DROP_JOB('nom_job');
```

```
dbms_scheduler.set_attribute_null( name=>'nom', attribute=>'a');  
dbms_scheduler.set_attribute(name=>'nom',attribute=>'a',value=>'v');
```

En SQL Developer  
-conexiones





## DBMS Scheduler - exemple

```
BEGIN
DBMS_SCHEDULER.CREATE_JOB (
  job_name      => 'nom_del_job',
  job_type      => 'STORED_PROCEDURE',
  job_action    => 'usuari1.actualitza_preus',
  start_date    => sysdate,
  repeat_interval => 'FREQ=MONTHLY;BYMONTHDAY=1',
  auto_drop     => FALSE,
  enabled       => TRUE,
  comments      => 'Aclariments del treball...' );
END;
/
```

Atributs

Valors

Nom del treball

esquema.procedure

MENSUAL, CADA DIA 1

-Prova-ho en SQL Developer  
-Crea el procediment que  
  executarà el job  
  -Crea el job  
-Comprova els resultats

YEARLY  
MONTHLY  
WEEKLY  
DAILY  
HOURLY  
MINUTELY  
SECONDLY

També es pot crear des de  
SQL Developer





## DBMS Scheduler - exemple

El paràmetre `repeat_interval` és molt versàtil.....

`FREQ=MONTHLY` → S'executa un cop al mes. (Defineix la repetició mensual)

`BYMONTHDAY=1` El dia 1 de cada mes

`BYDAY=TU` → Els dimarts.

`BYSETPOS=1` → Només el primer dimarts del mes.

`BYHOUR=1; BYMINUTE=5` → A la 01:05 AM.

`FREQ={SECONDLY | MINUTELY | HOURLY | DAILY | WEEKLY | MONTHLY | YEARLY};`

A partir d'aquí, pots afegir modificadors com `BYSECOND`, `BYMINUTE`, `BYHOUR`, `BYDAY`, `BYMONTH`, etc

- Prova-ho en SQL Developer
- Crea el procediment que executarà el job
- Crea el job
- Comprova els resultats



## DBMS Scheduler - exemple

### repeat\_interval **Exemples d'ús**

Cada dia a les 8:30 AM

REPEAT\_INTERVAL => 'FREQ=DAILY; BYHOUR=8; BYMINUTE=30;'

Cada dilluns i dimecres a les 10:00 AM i 4:00 PM

REPEAT\_INTERVAL => 'FREQ=WEEKLY; BYDAY=MON,WED; BYHOUR=10,16; BYMINUTE=0;'

El primer dimarts de cada mes a les 09:00 AM

REPEAT\_INTERVAL => 'FREQ=MONTHLY; BYDAY=TU; BYSETPOS=1; BYHOUR=9; BYMINUTE=0;'

L'últim divendres de cada mes a les 18:00

REPEAT\_INTERVAL => 'FREQ=MONTHLY; BYDAY=FR; BYSETPOS=-1; BYHOUR=18; BYMINUTE=0;'

Cada 6 hores

REPEAT\_INTERVAL => 'FREQ=HOURLY; INTERVAL=6;'

-Prova-ho en SQL Developer  
-Crea el procediment que  
  executarà el job  
  -Crea el job  
  -Comprova els resultats