M2. BASES DE DADESlínea horizontal  
Jordi Quesada

jordi.quesada@iesjoandaustria.org

dam1m2

<https://meet.google.com/rmp-ngnh-cgk>

**Estructuració del curs**

| UF 1. Introducció a les bases de dades |
| --- |
| UF 2. Llenguatge SQL |
| **UF 3. Assegurament** |

1. Transaccions

Imagina que tenim una base de dades bancària amb, entre altres coses, la taula:

**CUENTA(codigo, saldo)**

* Un client disposa en total de 5000€ repartits (cuenta 1 y cuenta 2) • Vol realitzar una transacció de 1000€ desde la cuenta 1 a la cuenta 2
* Necesitaría realitzar 2 operaciones SQL del tipo:

| UPDATE cuenta SET saldo=saldo-1000 WHERE codigo=1; |
| --- |

| UPDATE cuenta SET saldo=saldo+1000 WHERE codigo=2; |
| --- |

Què passaria si després d’executar la primera instrucció INSERT hi ha alguna incidència?:

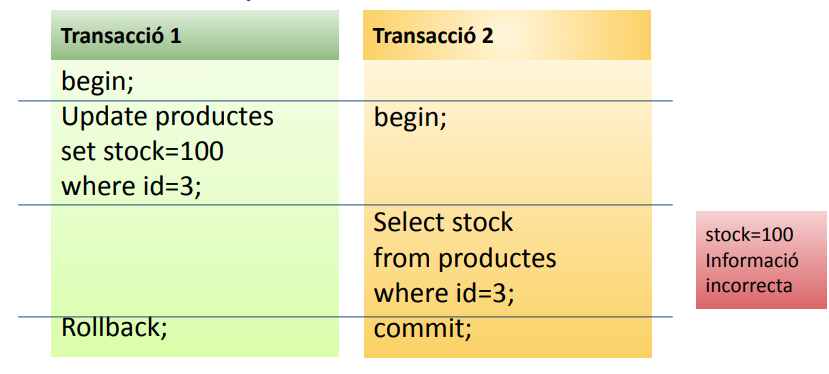
* Errors de hardware (tall de llum, disc dur, RAM…)
* Perdem la connexió
* Ens distraiem (trucades, mails…) i no executem la segona instrucció

O bé, executem la segona instrucció INSERT però:

* Es produeix un error perquè el compte 2 no existeix
* Algú borra el compte 2 just en aquest moment o tots els comptes
* No tenim permisos per modificar el compte 2
* – Etc

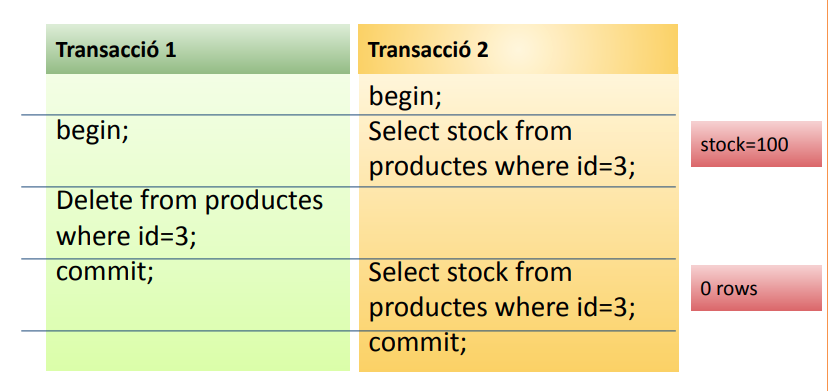
Casos d’error: accions llargues en el temps que es solapen.

1. **Dirty read**: Es produeix quan una transacció llegeix canvis fets per una transacció que no ha estat confirmada.



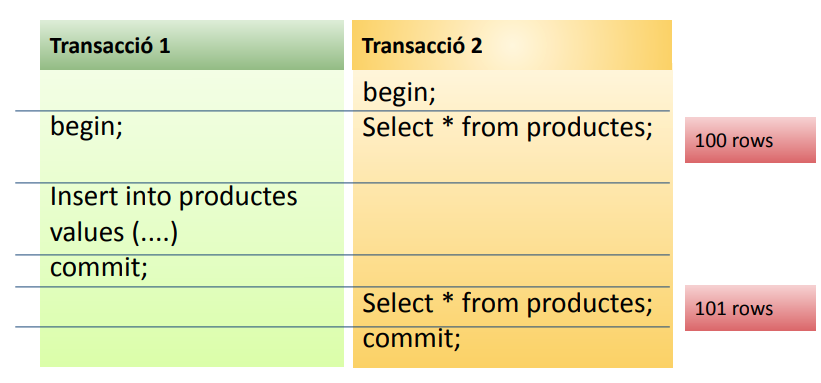
Perquè si es fa rollback, tornaria al seu estat inicial, no seria un stock=100, NO SERIA LA REALITAT.

2. **Non-repeatable read**: Es produeix quan una lectura de dades no pot ser repetida dins d’una mateixa transacció perquè una altra transacció els ha eliminat.



A la meitat, s’han borrat els productes.

3. **Phantom read:** Es produeix quan apareix una nova fila entre dos lectures de la mateixa taula dins de la mateixa transacció.



Del primer select al segon, apareix un valor ‘fantasma’. Abans no existia.

## 5. Solucions

* Per solucionar els problemes anteriors els SGBD ens ofereixen diferents nivells d'aïllament de les transaccions
* Aquests nivells s’anomenen **ISOLATION LEVEL** i es poden configurar a nivell global o a nivell de sessió
* Podem saber el nivell que tenim amb les comandes:

| SELECT @@GLOBAL.transaction\_isolation; |
| --- |

| SELECT @@SESSION.transaction\_isolation; |
| --- |

Ordenats de menys seguretat a més seguretat.

Els modes disponibles per a operar amb transaccions són:

* **READ UNCOMMITTED:** permet que la transacció vegi els canvis no confirmats realitzats per altres transaccions. Per tant, permet els tres problemes comentats anteriorment. Pot veure qualsevol transacció que s’estigui fent. (menos recursos).
* **READ COMMITTED**: només permet que la transacció vegi canvis confirmats. És a dir, no permet els 'dirty reads'. Un nivell més de seguretat. Cada transacció només pot veure els canvis que realment estiguin confirmats. (Si no s’ha fet un commit) MySQL es crea un “snapshop” - copia, i llegeix d’això.
* **REPEATABLE READ**: Assegura que la transacció sempre llegirà les mateixes dades independentment de si hi ha canvis o no durant el transcurs de la transacció. Per tant, no permet els 'dirty reads' i els 'non-repeatable read'. Aquest és el mode per defecte. Està com “aïllat”, es fa una còpia del moment, no tornarà a llegir si hi ha canvis nous.
* **SERIALIZABLE**: assegura que la transacció està completament aillada de qualsevol efecte produït per altres transaccions. Per tant, qualsevol dada consultada por una transacció serializable romandrà invariable durant tota la transacció.

Podem canviar el mode d’aillament de transaccions amb la comanda:

| SET [GLOBAL | SESSION] TRANSACTION ISOLATION LEVEL REPEATABLE READ | READ COMMITTED | READ UNCOMMITTED | SERIALIZABLE |
| --- |

UF3: PL/SQL

# 1.- Introducció

La majoria dels SGBD incorporen utilitats per ampliar /millorar la seva funcionalitat

Oracle -> PL/SQL = procedural Language / SQL

Informix → 4GL

SQL-Server → Transact-SQL

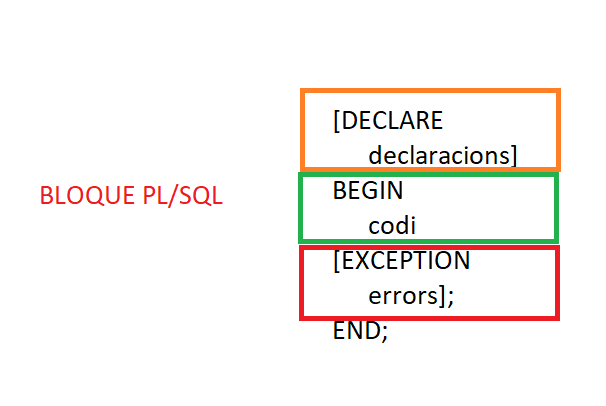
# 2.- Estructura de PL/SQL

La estructura mínima de treball de pl/SQL s’anomena **bloc**. Un bloc és un conjunt d’instruccions sense nom, s’executen i ja està.

També podrem crear blocs amb nom que seran procediments o funcions.

Tots els blocs tenen 3 parts:

* **Declaracions**: Declarem totes les variables, cursors i altres que necessitem
* **Codi**
* **Excepcions**: tractament d’errors



A més cal tenir present:

* El codi no distingeix majúscules/minúscules
* Totes les instruccions acaben en ;
* Una instrucció pot ocupar varies línies
* Comentaris:
  + -- comentari
  + /\* comentari \*/

## 2.1.- Declaració de variables

Per declarar variables:

| Nom [CONSTANT] tipus [:= valor\_inicial]; |
| --- |

Per exemple:

| numeros number(3):=5; strings varchar2(10):='pepito'; variable\_constante CONSTANT number(3,2):=3.14; |
| --- |

També podem declarar una variable amb el mateix tipus de dades que una columna d’una taula:

| Nom taula.variable % type |
| --- |

| x empleats.sou%type;  y empleats.nom\_Empl%type; |
| --- |

## 2.2.- Mostrar per pantalla (print)

Hem d’accedir a una llibreria:

| DBMS\_OUTPUT.PUT\_LINE |
| --- |

Programa holamón:

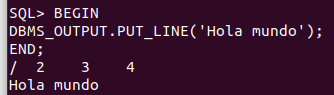
| BEGIN  DBMS\_OUTPUT.PUT\_LINE('Hola mundo'); END; / |
| --- |

Això així donaria error (no error, però no fa res). → La sortida per pantalla está desactivada.

Per activar la sortida per pantalla, a **cada sessió** cal fer:

| **SET** SERVEROUTPUT **ON**; |
| --- |

Ara sí que mostra per pantalla:



## 2.3.- Operadors

Ja els coneixem, +, -, \*, /

Apareixen:

* \*\* (potència) 2\*\*3
* || (concatenació)

| DECLARE  x integer; BEGIN  x:=3\*5;  DBMS\_OUTPUT.PUT\_LINE('x val: ' || x); END; / |
| --- |

| x val: 15 |
| --- |

## 2.4.- Funcions

| DECLARE x varchar2 (20); BEGIN  x:='Jordi';   x:=Upper(x);  DBMS\_OUTPUT.PUT\_LINE('x val: ' || x); END; / |
| --- |

| x val: JORDI |
| --- |

S’ha de tenir en compte la diferencia entre varchar i char:

| DECLARE x **char**(20); BEGIN  x:='Jordi';   x:=Upper(x);  DBMS\_OUTPUT.PUT\_LINE('Hola: ' || x || '!'); END; / |
| --- |

| Hola: JORDI ! |
| --- |

| DECLARE  x varchar2(20);  y number(6); BEGIN  x:='Jordi';  y:=length(x);  DBMS\_OUTPUT.PUT\_LINE('longitud de: '||x|| ' es ' || y); END; / |
| --- |

| longitud de: Jordi es 5 |
| --- |

## 2.5.- Estructures de control

a PL/SQL disposem de:

1. **IF**

| **IF** condició **THEN**  instruccions **END IF;** |
| --- |

| DECLARE  y number(6); BEGIN  y:=12;  if y > 10 THEN  DBMS\_OUTPUT.PUT\_LINE('És major que 10');  END IF; END; / |
| --- |

| És major que 10 |
| --- |

1. **IF .. ELSE**

| **IF** condició THEN  instruccions **ELSE**  instruccions **END** **IF**; |
| --- |

| DECLARE  y number(6); BEGIN  y:=12;  if y > 10 THEN  DBMS\_OUTPUT.PUT\_LINE('Es major que 10');  else  DBMS\_OUTPUT.PUT\_LINE('Es menor que 10');  END if; END; |
| --- |

1. **CASE**

Serveix per retornar un valor avaluant el que val una variable

| **CASE** expressió  **WHEN** valor1 **THEN** resultat1  **WHEN** valor2 **THEN** resultat2  ...  **ELSE** resultat **END**; |
| --- |

| BEGIN  x:=1;  y:=CASE x  WHEN 1 THEN 'Lunes'  WHEN 2 THEN 'Martes'  WHEN 3 THEN 'Miercoles'  WHEN 4,5 THEN 'Juernes'  ELSE ‘Finde’  end;  DBMS\_OUTPUT.PUT\_LINE(y); END; / |
| --- |

## 2.6.- Bucles

1. **While**

| **while** condició **LOOP**  instrucciones  **end** **LOOP**; |
| --- |

Si tenim un bucle infinit, hem d’apagar la base de datos. Però això no es solucio.

Hauriem d’anar al senyor de sistemes i que mati el procés.

| DECLARE  x number; BEGIN  while x<10 loop  DBMS\_OUTPUT.PUT\_LINE(x);  x:=x+1;  END loop; END;  / |
| --- |

Si no inicialitzem la variable, per **defecte es NULL.**

| DECLARE  x number:=0; BEGIN  while x<10 loop  DBMS\_OUTPUT.PUT\_LINE(x);  x:=x+1;  END loop; END;  / |
| --- |

1. **For**

| **for** contador **in [REVERSE]** min**..**max **loop** |
| --- |

0..9: Valor més petit al valor més gran, inclosos.

| DECLARE  x integer; BEGIN  for x in 0..9 loop  DBMS\_OUTPUT.PUT\_LINE('x vale: ' || x);  END loop; END; / |
| --- |

Al bucle for, **no** fa falta **inicialitzar**, perquè ja li diu que comenci amb 0. (Li assigna el valor més petit) I no fa falta el x+1: perquè ja ho fa.

Per decrementar: (Igualment segueix la regla de min-max)

| DECLARE  x integer; BEGIN  for x in **reverse** 0..9 loop  DBMS\_OUTPUT.PUT\_LINE('x vale: ' || x);  END loop; END; / |
| --- |

1. **Loop**

| **loop**  **exit** **WHEN** condició **END** **loop**; |
| --- |

Bucle infinit i acaba amb la condició (És al revés, quan no sabem quan acaba)

| DECLARE  x number:=0; BEGIN  loop  DBMS\_OUTPUT.PUT\_LINE('x vale: ' || x);  x:=x+1;  exit when x>=10;  END loop; END; / |
| --- |

**Exercici 3.**

| DECLARE  x varchar(20):='hola k ase'; BEGIN  for i in 1.. loop  letra=:substr(texto, i, 1);  DBMS\_OUTPUT.PUT\_LINE(letra); END; / |
| --- |

h

o

l

a

k

a

s

e

| DECLARE  texto varchar(20):='hola k ase';  letra char(1);  resultado varchar2(20):=''; BEGIN  for i in 1..20 loop  letra:=substr(texto, i, 1);  resultado:=resultado || letra;  end loop;  DBMS\_OUTPUT.PUT\_LINE(resultado); END; / |
| --- |

hola k ase

| BEGIN  for i in reverse 1..20 loop  letra:=substr(texto, i, 1);  resultado:=resultado || letra;  end loop;  DBMS\_OUTPUT.PUT\_LINE(resultado); END; / |
| --- |

esa k aloh

| BEGIN  for i in reverse 1..length(texto) loop  letra:=substr(texto, i, 1);  resultado:=resultado || letra;  end loop;  DBMS\_OUTPUT.PUT\_LINE(resultado); END; / |
| --- |

## 2.7.- Procediments

Un procediment és un bloc PL/SQL al que li podem posar un nom. La sintaxis és:

| **CREATE** [**OR** **REPLACE**] **PROCEDURE** nom **(**parámetres**)** **IS**  bloc\_plsql  **BEGIN**  **END;** |
| --- |

or replace → si existeix, el torna a crear, sinò no el modificaria.

Si no te paràmetres, no es posen els ().

| CREATE or REPLACE PROCEDURE prova IS  x number; BEGIN  x:=3;  DBMS\_OUTPUT.PUT\_LINE(x); END; / |
| --- |

**Per executar:**

| execute prova |
| --- |

| CREATE or replace PROCEDURE prova(y number) IS BEGIN  DBMS\_OUTPUT.PUT\_LINE(y); END; / |
| --- |

| execute prova(30); |
| --- |

**Problema 2:**

| CREATE PROCEDURE escriu(x varchar2) IS BEGIN  DBMS\_OUTPUT.PUT\_LINE(x); END; / |
| --- |

| execute escriu ('Hola mundo') |
| --- |

**Problema 3:**

| CREATE PROCEDURE problema3(texto varchar2) IS  letra char(1);  resultado varchar2(20):=''; BEGIN  for i in reverse 1..length(texto) loop  letra:=substr(texto, i, 1);  resultado:=resultado || letra;  end loop;  DBMS\_OUTPUT.PUT\_LINE(resultado); END; / |
| --- |

## 2.8.- Funcions

| **CREATE** [**OR** **REPLACE**] **FUNCTION** nom (parámetres) **RETURN** tipus **IS**  bloc\_plsql **BEGIN** **END**; |
| --- |

| CREATE OR REPLACE FUNCTION augment\_100(x number) RETURN number IS BEGIN  return x+100; END;  / |
| --- |

Les funcions no es poden cridar aixi:

| execute prueba2(200); |
| --- |

Perquè com hem retornen algo, necesito guardar això en algún lloc.

Dues opcions:

| DECLARE  x number:=20; BEGIN  x:=augment\_100(200);  escriu(x); END; / |
| --- |

| SELECT nom, sou, augment\_100(sou) from mitabla; |
| --- |

* **Normalment** les funcions no incoporen DBMS\_OUTPUT.PUT\_LINE.
* Les funciones fabriquen resultats pero no els mostren.

### Exercici 5.

Crea un procediment que rebi un número i mostra per pantalla la taula de multiplicar d’aquest número

| CREATE or REPLACE PROCEDURE tabla\_multiplicar(numero integer) IS BEGIN  for i in 0..10 loop  DBMS\_OUTPUT.PUT\_LINE(i ||' \* ' || numero || ' = ' || numero\*i);  end loop; END; / |
| --- |

## 2.9.- Paràmetres

Els paràmetres tant de procediments com de funcions tenen tres modes:

1. **Paràmetres d’entrada** [IN]: Són l’opció per defecte. Son paràmetres que dins del procediments o funció només podem consultar el valor, no podem modificar-los. Són paràmetres anomenats per **valor** a altres llenguatges de programació.
2. **Paràmetres de sortida** [OUT]: Els utilitzem quan volem tornar més d’un valor. No són paràmetres per **referència**.

Als paràmetres de sortida podem assignar valor , però no podem

(Escritura pero no llegir)

| CREATE OR REPLACE PROCEDURE prova3(x IN integer, y OUT integer) IS BEGIN  y:=x+5; END; / |
| --- |

| execute prova3 (5,6) |
| --- |

**Això donará error**

Li tindrem que donar un valor

| prova3(5,z); |
| --- |

| DECLARE z integer; BEGIN  prova3(5,**z**);  escriu(x); END; / |
| --- |

1. Paràmetres d’**entrada** i de **sortida [IN OUT]:** Són els paràmetres per referència d’altres llenguatges de programació. Són paràmetres que es poden llegir i també modificar dins del procediment o funció. Anomets paràmetres per **referència.**

## 2.10.-

Des dels procediments PLSQL podem executar comandes SQL directament.

La única instrucció que modificarem serà la de select.

## 2.11. Consultes simples

Des de plsql podem realitzar consultes simples a la base de dades.

Una consulta simple és la que retorna 1 única fila!

La sintaxis serà:

* SELECT ….
* **INTO ...**
* FROM
* WHERE
* GROUP BY
* HAVING
* ORDER BY

A la clàusula INTO el que farem serà indicar variables. Tantes com columnes tingui la consulta

CREATE OR REPLACE PROCEDURE cons\_empl(codi number) IS

v\_codi NUMBER(3);

v\_nom VARCHAR2(20);

v\_cognom VARCHAR2(50);

v\_sou NUMBER(8,2);

BEGIN

SELECT count(\*) into quants

from empleats where codi\_empl=codi;

IF quants=1 THEN

SELECT CODI\_EMPL, NOM\_EMPL, COGNOM\_EMPL, SOU

INTO v\_codi, v\_nom, v\_cognom, v\_sou

FROM empleats

WHERE codi\_empl=codi;

DBMS\_OUTPUT.PUT\_LINE('------------------------------------');

DBMS\_OUTPUT.PUT\_LINE(' DADES DE L''EMPLEAT ');

DBMS\_OUTPUT.PUT\_LINE('------------------------------------');

DBMS\_OUTPUT.PUT\_LINE('CODI: ' || v\_codi);

DBMS\_OUTPUT.PUT\_LINE('NOM: ' || v\_nom || ' ' || v\_cognom);

DBMS\_OUTPUT.PUT\_LINE('SOU: ' || v\_sou);

DBMS\_OUTPUT.PUT\_LINE('------------------------------------');

else

DBMS\_OUTPUT.PUT\_LINE('Error: No existeix cap empleat amb aquest codi');

END IF;

END;

/

### 

### 

### 

### 

### 

### 

### 

### 

### Exercici 13

### 

/\* PROBLEMA 13- Eliminar usuario \*/

CREATE OR REPLACE PROCEDURE eliminar\_user (codi number) IS

BEGIN

DELETE FROM empleats WHERE codi\_empl = codi;

END;

/

### Exercici 14

CREATE OR REPLACE PROCEDURE problema14 (codi number) IS

existe number;

BEGIN

existe:= existeix\_empl(codi);

IF existe=1 THEN

cons\_empl(codi);

eliminar\_user(codi);

ELSE

DBMS\_OUTPUT.PUT\_LINE('Error: No existeix un empleat amb aquest codi');

END IF;

END;

/

## 2.12.- Excepcions

Una excepció la definirem como un codi que provoca un error inesperat. Aquest error provoca que el programa acabi de forma inesperada.

En oracle existeixen 3 tipus d’excepcionss:

* Predefinides
* Sense definir
* D’ usuari

### Excepcions predefinides

Son excepcions que oracle ya te incorporades i que podemos utilizar

Exemple:

| ORA-01403: no data found |
| --- |

Entre les que anem a utilitzar están:

* CASE\_NOT\_FOUND: Es produeix quan no entrem en cap opció
* INVALID\_NUMBER: Falla la conversió a número
* NO\_DATA\_FOUND: Quan una consulta SELECT INTO no retorna files
* TOO\_MANY\_ROWS:
* ZERO\_DIVIDE: Divisió per 0

CREATE OR REPLACE PROCEDURE excepciones(x number, y number) IS

aux number;

BEGIN

aux:= x/y;

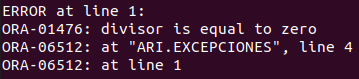
DBMS\_OUTPUT.PUT\_LINE(aux);

END;

/

Si fem:

| execute excepciones(7,0); |
| --- |



Dues maneres de solucionar això:

* **Programació defensiva** (posant condicions)
* **Programació amb exceptions**

| CREATE PROCEDURE procediment IS BEGIN **EXCEPTION** END; / |
| --- |

Les excepcions es declaren en la zona **exception** del codi i tenen el format:

| EXCEPTION  WHEN nom\_excepció THEN  accions |
| --- |

CREATE OR REPLACE PROCEDURE excepciones(x number, y number) IS

aux number;

BEGIN

aux:= x/y;

DBMS\_OUTPUT.PUT\_LINE(aux);

EXCEPTION

WHEN zero\_divide THEN

DBMS\_OUTPUT.PUT\_LINE('Error: Division por 0');

END;

/



Hi ha una excepció que podem utilitzar, pero no es recomanable (en la resta de casos)

| EXCEPTION  WHEN OTHERS THEN   DBMS\_OUTPUT.PUT\_LINE('Error: contacti amb l'administrador') |
| --- |

CREATE OR REPLACE PROCEDURE excepciones(x number, y number) IS

aux number;

BEGIN

aux:= x/y;

DBMS\_OUTPUT.PUT\_LINE(aux);

EXCEPTION

WHEN zero\_divide THEN

DBMS\_OUTPUT.PUT\_LINE('Error: Division por 0');

WHEN others THEN

DBMS\_OUTPUT.PUT\_LINE('Error: Contacti amb l''administrador');

END;

/

### Excepcions sense definir

Algunes excepcions com per exemple les de restriccció de integritat (foreign key), no tenen un nom assignat, però sí un número.

Per exemple, les d'integritat son el codi ORA-02292

Per tractar-les, el que farem será definir una excepció nostra i asociar-la:

En la zona de declaracions:

nom EXCEPTION;

pragma exception\_init (nom, num);

En la zona d’excepcions

when nom then

CREATE OR REPLACE PROCEDURE prob15 (codi number, nom varchar2, cognom varchar2, diners number, proj number) IS

meu\_error exception;

pragma exception\_init(meu\_error, -2291);

BEGIN

INSERT INTO Empleats (codi\_empl, nom\_empl, cognom\_empl, sou, codi\_proj)

VALUES (codi, nom, cognom, diners, proj);

EXCEPTION

WHEN meu\_error THEN

DBMS\_OUTPUT.PUT\_LINE('Error: el projecte ha d''existir');

-- una altre opció seria insertar el projecte

END;

/

### D’usuari

Podem treballar definint les nostres pròpies excepcions. Per fer-ho caldrà:

* Declarar una variable de tipus exception
* Quan vulguem la provocarem amb el raise
* La capturarem a la zona d’exception

CREATE OR REPLACE PROCEDURE excepcion IS

x exception;

BEGIN

DBMS\_OUTPUT.PUT\_LINE('entramos');

raise x; --provocar l'excepció

DBMS\_OUTPUT.PUT\_LINE('llego aquí?');

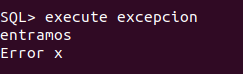
EXCEPTION

when x THEN

DBMS\_OUTPUT.PUT\_LINE('Error x');

END;

/



Exemple d’ús:

CREATE OR REPLACE PROCEDURE excepcion (nom varchar2) IS

x number;

y exception;

BEGIN

SELECT count(\*) INTO x

FROM jugador WHERE nombre\_equipo=nom;

if x>0 then -- si el equipo tiene jugadores

raise y; -- provoca excepció

end if;

delete from equipo where nombre\_equipo=nom;

EXCEPTION

when y then

DBMS\_OUTPUT.PUT\_LINE('Error: El equipo tiene jugadores');

END;

/

Els **delete no mostren error** encara que no trobem dades. (No te exceptions)

| 0 rows delete. |
| --- |

Els **update** no mostren error encara que no compleixi.

| 0 rows update. |
| --- |

CREATE OR REPLACE PROCEDURE excepcion (x number) IS

BEGIN

DELETE from mitable where id = x;

DBMS\_OUTPUT.PUT\_LINE('Borrat');

END;

/

Però encara que no borri, seguirá posant “borrat”.

CREATE OR REPLACE PROCEDURE excepcion (x number) IS

y exception;

BEGIN

DELETE from mitable where id = x;

if sql%notfound then

raise y;

end if;

DBMS\_OUTPUT.PUT\_LINE('Borrat');

EXCEPTION

DBMS\_OUTPUT.PUT\_LINE('No borrat');

END;

/

**SQL%NOTFOUND → quan el where no troba res.**

Dubte: si demanem un numero i ens fiquen un string, hi ha alguna excepcio per aixo?

## 2.13.- Cursors

Els cursos ens permeten fer consultes que retornin varies files. Per treballar amb cursors necessitarem seguir els següents pasos:

* Declarar el cursor
* Obrir-lo
* Tractar-lo
* Tancar-lo

### 

### a) Declaració de cursors

A la zona de DECLARE podem declarar cursors amb la sintaxis:

| CURSOR nom IS consulta; |
| --- |

CURSOR cempl IS SELECT \* FROM empleats;

### b) Obrir el cursor

Per obrir el cursor farem servir la comanda open i el nom del cursor

| OPEN nom\_cursor; |
| --- |

El open és realment quan s’executa el Select, si no el faig no tinc informació.

### 

### c) Declarar el cursor

Necessitarem un bucle ja que no sabem quants registres ens retorna la consulta.

Però amb el bucle necessitarem una condició de finalització.

Per gestionar el bucle, disposem d’una instrucció anomenada FETCH. La sintaxis és:

| FETCH nom\_cursor INTO llista\_variables; |
| --- |

Cada vegada agafa la informació d’una fila i les fica en les variables declarades.

### d) Tancar el cursor

El cursor es tanca amb close i el nom del cursor.

| CLOSE nom\_cursor; |
| --- |

S’allibera la memòria, “ja no ho necessito”

### Propietats dels cursors

Els cursors, a part de les dades, tenen una sèrie de propietats:

* nom%ISOPEN: Retorna cert si el cursor esta obert.
* nom%NOTFOUND: Retorna cert si la ultima instrucció FETCH no retorna informació.
* nom%NOTCOUNT: Retorna el número de files (número de FETCH) que s’han tractat. Inicialment val 0. Em retorna true quan el número de files hagi acabat.

CREATE OR REPLACE PROCEDURE problema20 IS

**CURSOR cempl IS** SELECT nom\_empl, cognom\_empl

FROM empleats order by cognom\_empl;

v\_nom VARCHAR2(20);

v\_cog empleats.cognom\_empl%type;

BEGIN

OPEN cempl;

loop

**FETCH cempl** INTO v\_nom, v\_cog;

exit when cempl%NOTFOUND;

DBMS\_OUTPUT.PUT\_LINE(v\_nom);

end loop;

CLOSE cempl;

END;

/

## 2.14.- Registres

Per facilitar-nos la declaració de variables, a PL/SQL tenim una estructura anomenada registre. Serà similar a TYPE d’altres llenguatges de programació.

S’utilitzen de la següent manera;

* DECLAREM: nom nom\_taula%rowtype
* ACCEDIM: nom.nom\_variable

Només funciona en casos de seleccionar tot (\*)

CREATE OR REPLACE PROCEDURE problema20 IS

CURSOR cempl IS SELECT \*

FROM empleats order by cognom\_empl;

rempl empleats%rowtype;

BEGIN

OPEN cempl;

loop

FETCH cempl INTO rempl;

exit when cempl%NOTFOUND;

DBMS\_OUTPUT.PUT\_LINE(rempl.nom\_empl || ' ' || rempl.cognom\_empl);

end loop;

CLOSE cempl;

END;

/

Si volem seleccionar quines variables volem, apuntarem al cursor.

El problema és quan volem fer joins o crear camps nous com avg, sum, min…

La estructura de registres també es pot aplicar a cursors.

Declarem: Nom nomCursor%Rowtype

CREATE OR REPLACE PROCEDURE problema20 IS

CURSOR cempl IS SELECT nom\_empl, cognom\_empl

FROM empleats order by cognom\_empl;

rempl cempl%rowtype;

BEGIN

OPEN cempl;

loop

FETCH cempl INTO rempl;

exit when cempl%NOTFOUND;

DBMS\_OUTPUT.PUT\_LINE(rempl.nom\_empl || ' ' || rempl.cognom\_empl);

end loop;

CLOSE cempl;

END;

/

## 2.15. Cursors i bucle for

El bucle for amb cursors segueix la sintaxis:

| FOR registre IN cursor loop **end** **loop**; |
| --- |

CREATE OR REPLACE PROCEDURE problema20 IS

CURSOR cempl IS SELECT nom\_empl, cognom\_empl

FROM empleats order by cognom\_empl;

--rempl cempl%rowtype;

BEGIN

--OPEN cempl;

for rempl in cempl

--FETCH cempl INTO rempl;

--exit when cempl%NOTFOUND;

DBMS\_OUTPUT.PUT\_LINE(rempl.nom\_empl || ' ' || rempl.cognom\_empl);

end loop;

--CLOSE cempl;

END;

/

## 

## 2.16.- Paquets

Els paquets ens permeten agrupar funcionalitats. Per definir un paquet hem de fer dues parts:

* package
* package body

### a) Package

Conté la declaració de procediments, funcions, variables, tipus, etc. que serán públics, és a dir, accessibles.

CREATE or REPLACE PACKAGE emp is

procedure borra (x number);  
 procedure alta(c number, n varchar2, a varchar2, s number);

END emp;

/

### b) Package body

Conté el cos de tots els procediments, funcions,... tant públic com privat.

Son semblants als mètodes de java.

CREATE or REPLACE PACKAGE BODY emp is

FUNCTION existe(codi number) RETURN number is

quants number;

BEGIN

select count(\*) into quants

from empleats where codi\_empl=codi;

return quants;

END;

PROCEDURE borra (x number) IS

BEGIN

delete from empleats where codi\_empl = x;

END;

PROCEDURE alta(

c number,

n varchar2,

a varchar2,

s number

) is

BEGIN

if (s >900) then

insert into empleats (codi\_empl, nom\_empl, cognom\_empl, sou)

values (c, n, a, s);

dbms\_output.put\_line('Empleat donat d''alta correctament');

else

dbms\_output.put\_line('Error. El sou ha de ser superior a 900');

end if;

END;

END emp;

/

| rant @(RUTA) → @/vagran |
| --- |

# Cargar archivos desde el vagrant

1. @(RUTA) → @/vagrant/nba\_ddl.sql
2. Cuando acabe, hay que confirmar → commit

# Para hacer pruebas de funciones

| Select function from dual; |
| --- |

**Crear usuario en Oracle y dar todos los permisos:**

| create user ari identified by ari; grant create session, resource to ari; grant all privileges to ari; |
| --- |

**RLWRAP**

Entras a vagrant, (antes de entrar a oracle)

sudo yum install rlwrap

Y para entrar: rlwrap sqlplus

**MOSTRAR TODOS LOS PROCEDIMIENTOS EXISTENTES**

| select name from all\_source where owner = 'ari'; |
| --- |

**VER UN PROCEDIMIENTO**

| select name, text from all\_source where name="FUNCION"; |
| --- |

https://pastebin.com/dgHR0WeH