

Tema 6 – Programació: Procediments, Funcions i Disparadors

6. Programació : Procediments, Funcions i Disparadors

6.1. PL/SQL

PL/SQL significa "Procedural Language extensions to the Structured Query Language" (usat per ORACLE). Bàsicament és la utilització de SQL dins d'un llenguatge procedural. Ja que SQL no té cap element de programació.

PL/SQL és un llenguatge basat en un altre llenguatge de programació anomenat ADA, el qual va ser un llenguatge dissenyat pel departament de defensa dels Estats Units.

Ada és un llenguatge que s'enfoca en l'abstracció, ocultació d'informació i altres estratègies de disseny. Gràcies a aquest disseny PL/SQL és un llenguatge molt poderós que inclou la majoria dels elements dels llenguatges procedurals (El programa es desenvolupa com una seqüència de passos que l'ordinador executa per arribar a la fi desitjada).

Els desenvolupadors han de codificar els fluxos de control de les activitats a realitzar, a més de les activitats en si.

Elements del PL/SQL:

- Àmplia varietat de tipus de dades per declarar números, cadenes de caràcters, registres, arranjaments (col·leccions a Oracle) i en les últimes versions s'ha inclòs gestionar XML.
- Estructures de control iteratives (cicles), seqüencials, condicionals (if, case, etc.).
- Maneig d'excepcions per atrapar errors.
- Reutilització de codi, com procediments, funcions, triggers, objectes (POO) i paquets.

6.2. Estructura d'un bloc PL/SQL

6.2.1. Primer contacte

Veurem la sintaxi bàsica i després crearem dues taules d'exemple, que puguem utilitzar per a fer proves:

DECLARE (opcional)

variables, cursors, excepcions definides per l'usuari

BEGIN (obligatori)

sentències **SQL**

sentències de control **PL / SQL**

EXCEPTION (opcional)

accions a realitzar quan es produeixen errors

END; (obligatori)

Així, un exemple mínim podria ser:

-- Exemple 06.02.01a

```
BEGIN
  dbms_output.put_line('Hola');
END;
```

I un exemple senzill que realment obtinguera informació de la nostra base de dades (que els seus "CREATE TABLE" i "INSERT" d'exemple tens un poc més a baix) podria ser:

-- Exemple 06.02.01b

```
DECLARE
  espaiMax NUMBER(9,3);
BEGIN
  SELECT MAX(espaiOcupatMb)
  INTO espaiMax
  FROM jocs;

  dbms_output.put_line(espaiMax);
END;
```

Detalls importants:

- La paraula DECLARE permet declarar les variables que emprarem en un bloc de codi.
- Per a cada variable s'indicarà en primer lloc el seu nom i després el seu tipus de dades (en el mateix format que s'utilitza en l'ordre CREATE TABLE).
- S'ha de posar punt i coma (;) al final d'una sentència SQL o sentència de control PL/SQL, i després de END.
- Les paraules claus de secció DECLARE, BEGIN i EXCEPTION no van seguit de punt i coma.
- L'estructura SELECT ... INTO permet bolcar a una variable el resultat d'una consulta.
- Quan un bloc és executat amb èxit sense errors de maneig i compilació, apareix el següent missatge: *PL/SQL procedure successfully completed*
- A vegades, emprarem la sentència "dbms_output.put_line" per a mostrar en pantalla el resultat d'un fragment de programa.
- Es pot tenir tota l'expressió en una sola línia, però aquest mètode no és recomanable per claredat o facilitat d'edició.

Les dades d'exemple podrien ser:

```
CREATE TABLE plataformes(
  codi CHAR(4),
  nom VARCHAR2(30),
  CONSTRAINT pk_plataformes PRIMARY KEY (codi)
);
```

```
CREATE TABLE jocs(  
  codi CHAR(5),  
  nom VARCHAR2(50),  
  descripcio VARCHAR2(1000),  
  anyLlancament NUMBER(4),  
  espaiOcupatMb NUMBER(9,3),  
  codiPlataforma CHAR(4),  
  CONSTRAINT pk_jocs PRIMARY KEY (codi),  
  CONSTRAINT fk_jocs_plataformes  
    FOREIGN KEY (codiPlataforma) REFERENCES plataformes(codi)  
);
```

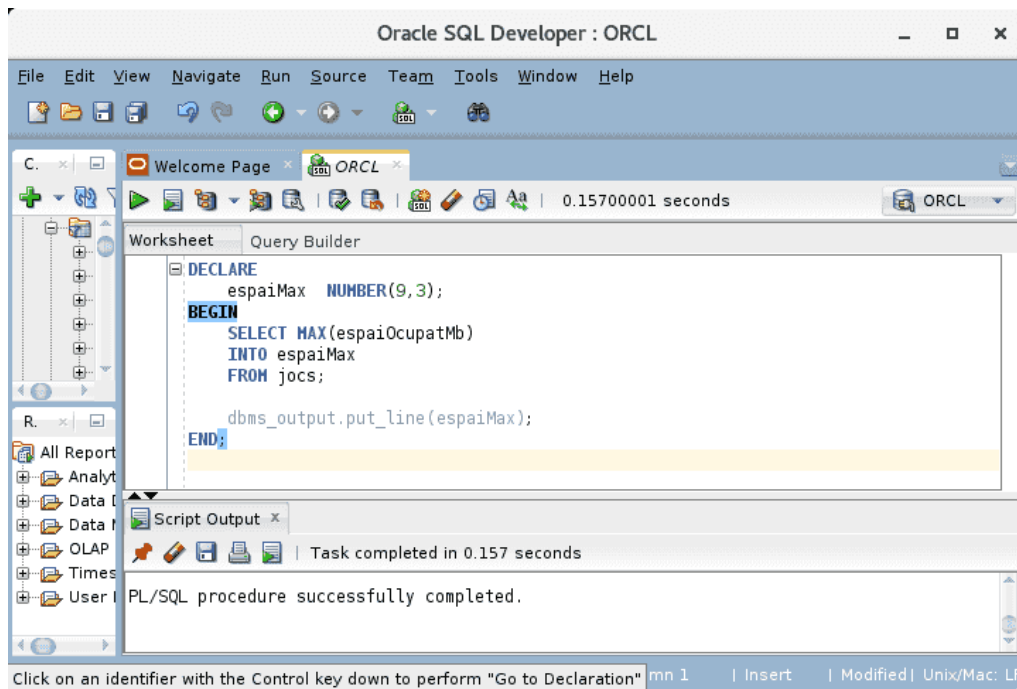
```
INSERT INTO plataformes VALUES('cpc', 'Amstrad CPC');  
INSERT INTO plataformes VALUES('pcw', 'Amstrad PCW');  
INSERT INTO plataformes VALUES('msx', 'MSX');  
INSERT INTO plataformes VALUES('spec', 'Sinclair ZX Spectrum');  
INSERT INTO plataformes VALUES('psx', 'Playstation');  
INSERT INTO plataformes VALUES('ps2', 'Playstation 2');  
INSERT INTO plataformes VALUES('ps3', 'Playstation 3');  
INSERT INTO plataformes VALUES('ps4', 'Playstation 4');  
INSERT INTO plataformes VALUES('ps5', 'Playstation 5');  
INSERT INTO plataformes VALUES('wii', 'Nintendo Wii');  
INSERT INTO plataformes VALUES('stea', 'PC + Steam');  
INSERT INTO plataformes VALUES('epic', 'PC + Epic');
```

```
INSERT INTO jocs VALUES('efre', 'Electro Freddy', NULL, 1982, 0.2, 'cpc');  
INSERT INTO jocs VALUES('mmic', 'Manic Miner', 'Plataformas sin scroll', 1983, 0.2, 'cpc');  
INSERT INTO jocs VALUES('mmiz', 'Manic Miner', 'Plataformas sin scroll', 1983, 0.2, 'spec');  
INSERT INTO jocs VALUES('aa', 'Ant Attack', NULL, 1983, 0.1, 'spec');
```

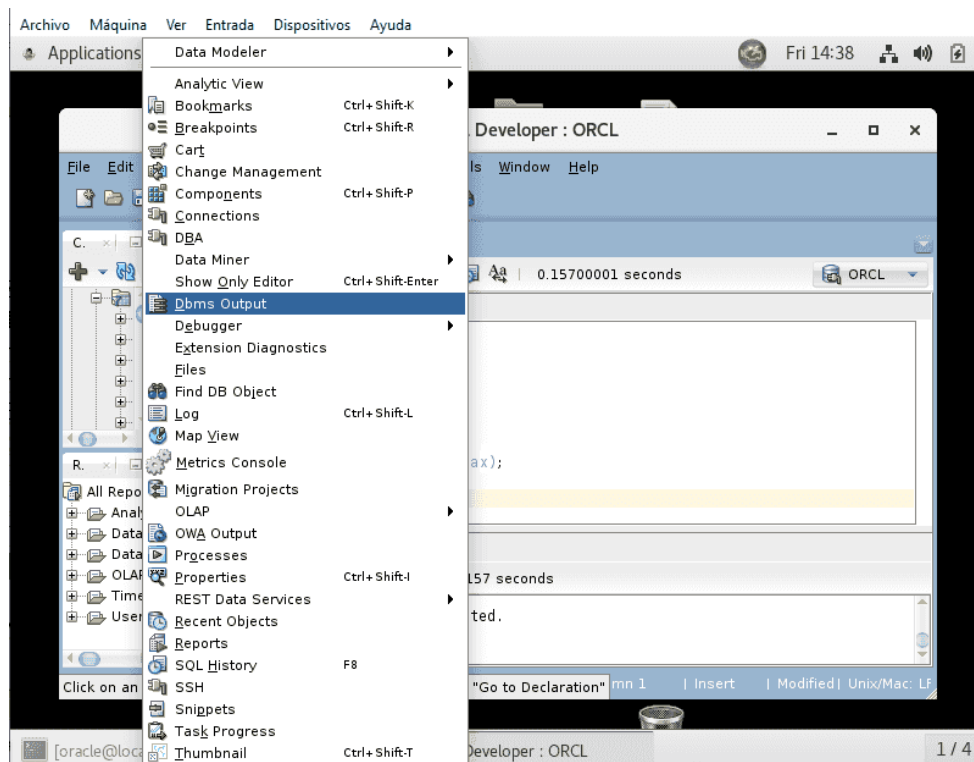
```
INSERT INTO jocs VALUES('ikaw', 'Ikari Warriors', 'Disparos, vista cenital', 1986, 0.2, 'msx');  
INSERT INTO jocs VALUES('wsr', 'Wii Sports Resort', NULL, 2009, 0, 'wii');  
INSERT INTO jocs VALUES('gt5', 'Gran Turismo 5', NULL, 2010, 0, 'ps3');  
INSERT INTO jocs VALUES('last1', 'The last of US', NULL, 2013, NULL, 'ps3');
```

```
INSERT INTO jocs VALUES('fortn', 'Fortnite', 'FPS + Battle Royale', 2017, NULL, 'epic');  
INSERT INTO jocs VALUES('aliso', 'Alien: Isolation', NULL, 2017, 35000, 'epic');  
INSERT INTO jocs VALUES('cont', 'Control', 'Aventura', 2019, NULL, 'epic');  
INSERT INTO jocs VALUES('batao', 'Batman: A.O.', NULL, 2013, 18250, 'stea');
```

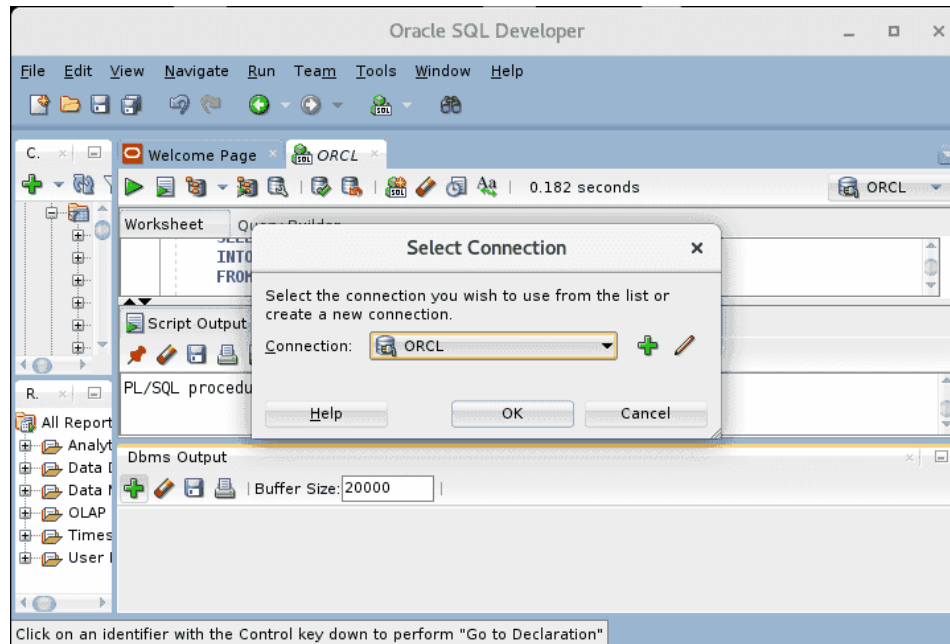
Una vegada creades aqueixes taules i bolcats aqueixes dades, el fragment de PL/SQL es podria provar des de Oracle SQL Developer, en la mateixa finestra que empràvem per a llançar consultes SQL:



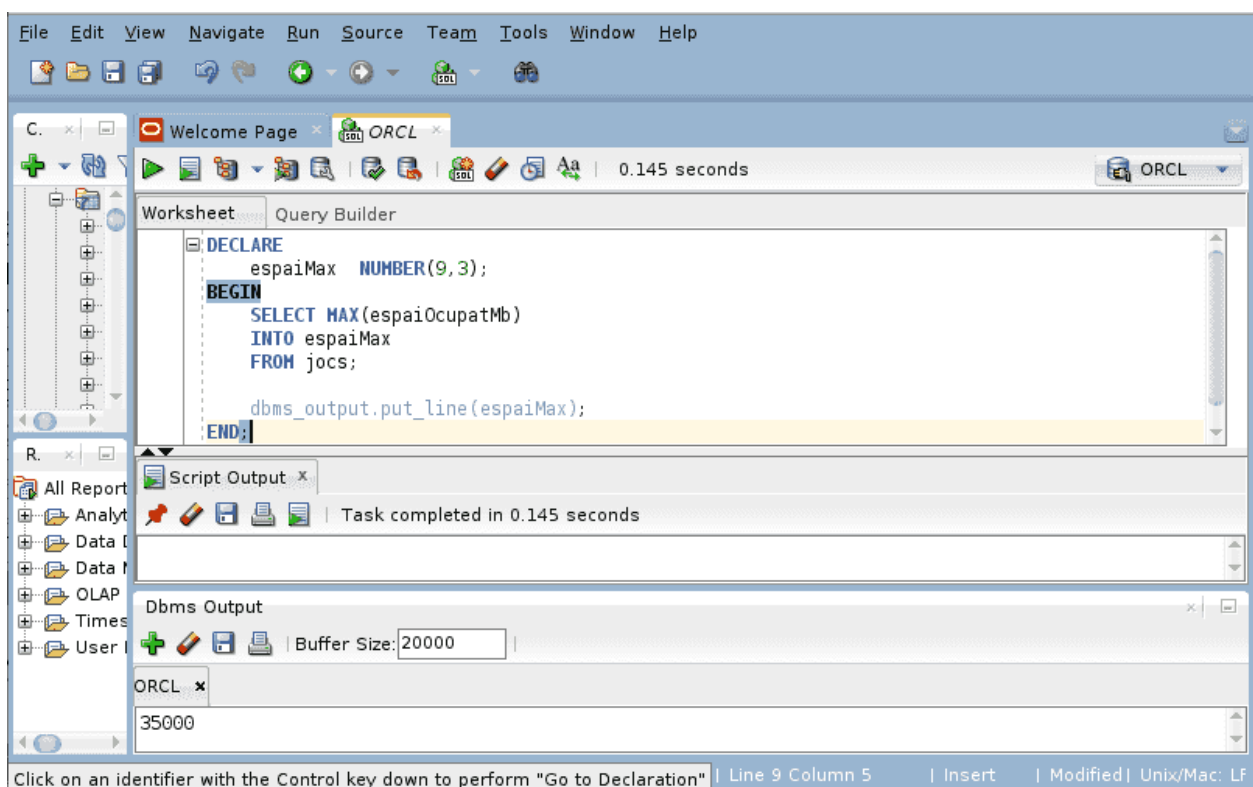
Veurem que en la finestra "Script Output" apareix la resposta "PL/SQL procedure successfully completed", però no l'eixida que esperàvem de l'ordre "dbms_output.put_line". Aqueixa eixida apareixerà en la finestra anomenada "Dbms output", que es pot demanar des del menú View:



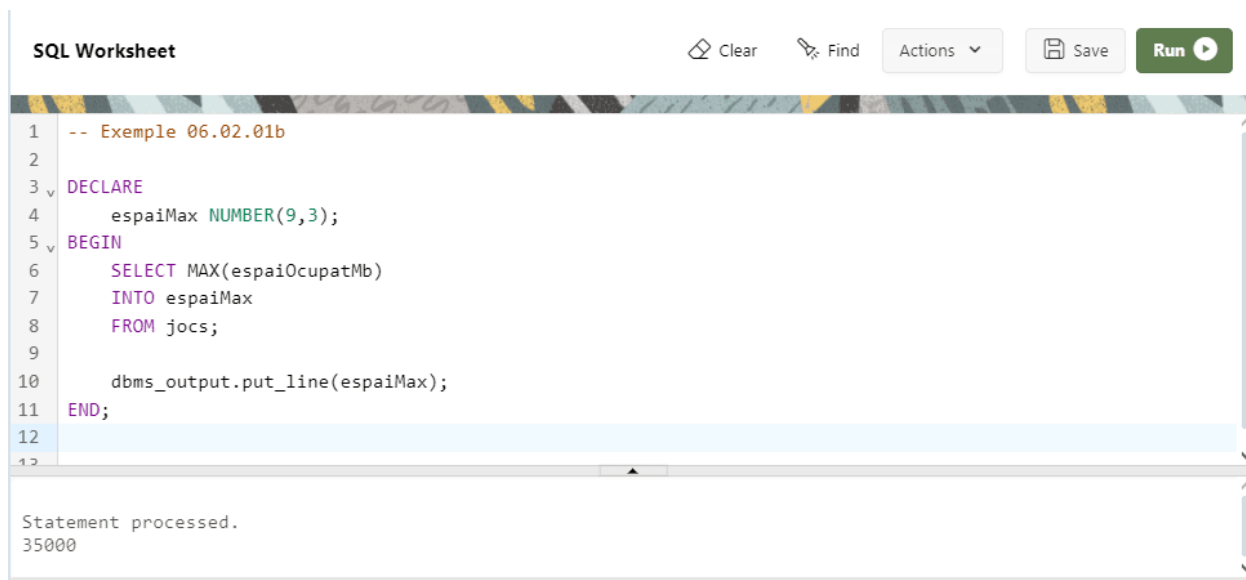
Que ens preguntarà a quina sessió volem connectar:



I a partir d'aqueix moment ja sí que podrem comprovar els resultats de "dbms_output.put_line":



En cas d'estar utilitzant la plataforma "Oracle Live", en entorn Web, podràs introduir estes comandos en la mateixa finestra habitual de "SQL Worksheet" que s'empra per als ordes SQL convencionals, i el resultat apareixerà en el panell inferior:



```
SQL Worksheet

1  -- Exemple 06.02.01b
2
3  DECLARE
4      espaiMax NUMBER(9,3);
5  BEGIN
6      SELECT MAX(espaiOcupatMb)
7      INTO espaiMax
8      FROM jocs;
9
10     dbms_output.put_line(espaiMax);
11 END;
12
13

Statement processed.
35000
```

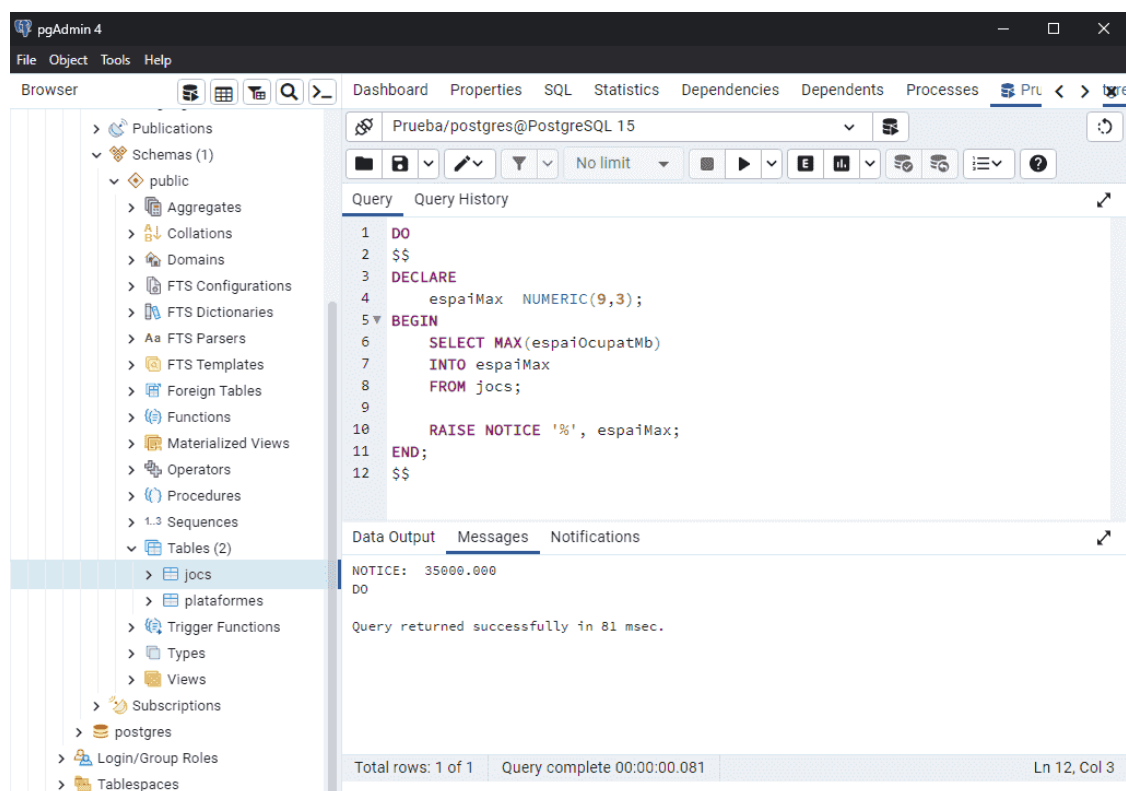
Si tens problemes per a instal·lar Oracle, pots emprar **PostgreSQL** per a la major part d'aquest tema, perquè també inclou el seu propi llenguatge procedural, anomenat PL/pgSQL, que coincideix en la majoria de les idees principals, però té algunes diferències de sintaxis. En el que afecta aquest exemple, les diferències són:

- PostgreSQL segueix els tipus de dades estàndard de SQL, no les variants de Oracle, perquè el que els camps numèrics hauran de dir-se NUMERIC (no NUMBER) i els textos seran VARCHAR (no VARCHAR2).
- L'equivalent a "dbms_output.put_line" és "RAISE NOTICE", que s'usa amb el format: RAISE NOTICE '%', dada;
- Els blocs anònims s'han de precedir amb DO i indicar-se entre delimitadors \$\$.

Amb aqueixes consideracions, el script quedaria així:

```
-- Exemple 06.02.01a PL/pgSQL
DO
$$
DECLARE
    espaiMax NUMERIC(9,3);
BEGIN
    SELECT MAX(espaiOcupatMb)
    INTO espaiMax
    FROM jocs;
```

```
RAISE NOTICE '%', espaiMax;
END;
$$
```



Exercici proposat 06.02.01a:

Basant-te en el primer exemple (06.02.01a), fes un fragment de codi que obtinga la mitjana de l'espai ocupat pels jocs de la nostra base de dades, el guarde en una variable "espaiMitja" i després ho mostre en la finestra d'eixida.

6.2.2. Tipus de blocs

Un programa PL/SQL té un o més blocs. Aquests blocs poden estar separats o niats entre ells. Per tant, un bloc pot representar una xicoteta part d'un altre bloc.

Podem utilitzar:

- Blocs anònims. Un bloc anònim és un bloc sense nom. Es declaren en el punt d'una aplicació en el qual s'executen. El nostre exemple anterior era un bloc anònim.
- Subprogrames. Són blocs de PL/SQL amb nom, que poden acceptar paràmetres i ser invocats des d'un altre punt de l'aplicació. Se'ls pot declarar com procediments o com funcions. Generalment s'usa un procediment per executar una acció i una funció per computar un valor.

Usant els components d'Oracle Developer (forms, reports, i gràfics), pots declarar procediments i funcions com a part de l'aplicació (un formulari o informe) i invocar a aquests des d'altres procediments, funcions i triggers dins d'una mateixa aplicació .

6.2.3. Declaració de variables en PL/SQL

Sintaxi:

Identificador [CONSTANT] tipus_dada [NOT NULL] [:= | DEFAULT expressió];

Exemples

```
data DATE;  
numDepartament NUMBER(2) NOT NULL := 10;  
localitat VARCHAR2(13) := 'ATLANTA';  
comisio CONSTANT NUMBER := 1400;
```

En aqueixa especificació de sintaxi:

- Identificador: es el nom de la variable.
- CONSTANT: restringeix la variable amb el propòsit que el seu valor no puga canviar; les constants han de ser inicialitzades (s'ha d'indicar forçosament el seu valor).
- Data type: pot ser un escalar, compost, referenciat o LOB.
- NOT NULL: restringeix la variable perquè siga obligatori que tinga un valor.
- := és l'operador d'assignació, que s'empra per a donar valor a una variable.
- Expr: és qualsevol expressió PL/SQL que pot ser una expressió literal, una altra variable o una expressió que involucra operadors i funcions .

Identificadors

- Poden contindre fins a 30 caràcters.
- Han de començar amb un caràcter alfabètic.
- Poden contindre números, signes de dòlar i subratllats.
- No poden contindre caràcters com a guions, barres de divisió, i espais.
- No han de tenir el mateix nom d'una columna de la taula de la base de dades.
- No poden ser paraules reservades (per exemple, una variable no es podrà dir "if").

De manera que podríem declarar una variable de text i obtindre el nom d'un dels jocs de la nostra base de dades a partir del seu codi així:

-- Exemple 06.02.03a

DECLARE

resultat VARCHAR2(50);

BEGIN

SELECT nom

INTO resultat
FROM jocs
WHERE codi='mmiz';

dbms_output.put_line(resultat);
END;

Exercici proposat 06.02.03a:

Basant-te en el segon exemple, crea un fragment de codi que obtinga el nom del joc que més ocupa de la nostra base de dades, el guarde en una variable "nomMesGran" i després ho mostre en la finestra d'eixida.

6.2.3.1. Tipus de dades més utilitzats

Alfanumèrics:

VARCHAR2(n).

Quan es declara una dada d'aquest tipus, aquest pot contenir caràcters (codificats segons el joc de caràcters de la sessió) fins a una longitud total de n posicions. La longitud màxima per a les columnes d'aquest tipus de dades és de 4000 posicions. És obligatori especificar la longitud d'una columna Varchar2.

Numèrics:

NUMBER(p, s)

On p indica el nombre total de dígit (fins a 38), s indica l'escala, o nombre de dígit a la dreta del punt decimal (fins a 127).

Si a l'emmagatzemar un valor en una columna de tipus *number* s'excedeix la precisió (nombre de dígit), Oracle dóna un error, si s'excedeix l'escala (més decimals dels que suporta) Oracle arrodoneix el valor a la capacitat de la columna.

Data:

DATE

Una columna de tipus date emmagatzema informació sobre data i hora. Per a cada dada de tipus date, Oracle emmagatzema informació sobre segle, any, mes, dia, hora, minut i segon.

Es pot especificar un valor de data mitjançant un literal (que Oracle intentarà convertir) o mitjançant la funció TO_DATE.

TO_DATE('20/04/2003 09:20','DD/MM/YYYY HH:MI')

Altres tipus:

LOB: Els tipus de dades LOB (Large Object), s'usen per emmagatzemar dades de grans dimensions, com fitxers (BFILE), text (CLOB), vídeo o imatges (BLOB).

ROWID : És una pseudocolumna que identifica de manera única a cada fila de cada taula, és una columna invisible que tota taula d'Oracle té, la crea automàticament Oracle i només es pot consultar, no apareix en els SELECT ni es pot modificar en els UPDATE, ni en els INSERT.

És el mitjà més ràpid d'accés a una fila en la taula.

Mostra informació de com s'emmagatzema una fila a la taula, ya que els seus caràcters indiquen: El segment de la base de dades on hi ha la fila, el nombre de fitxer de la tablespace relatiu que conté la fila, el bloc de dades que conté a la fila i el nombre de fila al bloc

Exemple:

-- Exemple 06.02.03b

```
DECLARE
  identificador ROWID;
  nomJoc VARCHAR2(50);
BEGIN
  SELECT ROWID
  INTO identificador
  FROM jocs
  WHERE codi='mmiz';

  dbms_output.put_line(identificador);

  SELECT nom
  INTO nomJoc
  FROM jocs
  WHERE ROWID = identificador;

  dbms_output.put_line(nomJoc);
END;
```

Boolean

El tipus booleà, només admet els valors TRUE, FALSE o NULL.

Encara que sempre el podem imitar amb un Varchar2 (1) o NUMBER, és molt útil per comparacions perquè podem posar la variable directament.

Exemple (la sentència IF es tractarà més endavant):

-- Exemple 06.02.03c

```
DECLARE
    acabat BOOLEAN := TRUE;
BEGIN
    IF acabat THEN
        dbms_output.put_line('Procés acabat');
    ELSE
        dbms_output.put_line('Procés no acabat');
    END IF;
END;
```

6.2.3.2. Atributs %TYPE i %ROWTYPE

%TYPE serveix per declarar una variable basada en:

- Altres prèviament declarades.
- Una columna de la BD.

Cal precedir %TYPE per:

- La taula i la columna de la BD.
- El nom de la variable definida amb anterioritat.

Així, una versió alternativa de l'exemple 06.02.01a, que no prefixa el tipus de dades per a la variable, sinó que el prenga del camp que s'analitzarà, podria ser:

-- Exemple 06.02.03d

```
DECLARE
    espaiMax jocs.espaiOcupatMb%TYPE;
BEGIN
    SELECT MAX(espaiOcupatMb)
    INTO espaiMax
    FROM jocs;

    dbms_output.put_line(espaiMax);
END;
```

Exercici proposat 06.02.03b:

A partir de l'exercici proposat 06.02, crea una nova versió que use els tipus de dades que realment apareixen en la taula.

Usarem **%ROWTYPE** per a tipus de dades compostes (files completes, registres...). Declara un variable d'acord amb una col·lecció de columnes en una taula de base de dades. S'indicarà **%ROWTYPE** com a sufix després del nom de la taula, com en el següent exemple:

-- Exemple 06.02.03e

```
DECLARE
  v_joc jocs%ROWTYPE;
BEGIN
  SELECT *
  INTO v_joc
  FROM jocs
  WHERE codi = 'aa';

  dbms_output.put_line(v_joc.nom);
END;
```

Com pots veure en aquest exemple, és freqüent precedir els noms de les variables amb el prefix "v_", per a evitar col·lisions amb noms de camps.

Exercici proposat 06.02.03c:

Seguint l'esquema de l'exemple 0602.03e, usa una variable amb el tipus d'una fila de la taula "jocs", obtingues les dades del registre el codi del qual és "gt5" i mostra el seu any de llançament.

Avantatges d'utilitzar **%ROWTYPE**

- El nombre de camps i tipus de dades de les columnes de base de dades no necessiten ser coneguts.
- Si canvia el nombre o tipus de camps no és necessari modificar res en el script, ja que en tindre **%rowtype** crea la variable amb el canvi efectuat.
- L'atribut declarat es pot usar per a recuperar una fila sencera amb la sentència **SELECT**.

6.2.3.3. Assignació de valors a variables

Com ja hem vist, una primera manera d'assignar valors a variables és des d'una ordre **SELECT**, emprant **INTO**:

-- Exemple 06.02.03f

```
DECLARE
  anyMax jocs.anyLlançament%TYPE;
BEGIN
  SELECT MAX(anyLlançament)
```

```
INTO anyMax
FROM jocs;

dbms_output.put_line(anyMax);
END;
```

Però també existeix la possibilitat d'assignar valors prefixats, amb l'operador ":", amb el format "identificador := expressió;", com en el següent exemple:

-- Exemple 06.02.03g

```
DECLARE
    espaiMb jocs.espaiOcupatMb%TYPE;
    espaiGb jocs.espaiOcupatMb%TYPE;
BEGIN
    SELECT espaiOcupatMb
    INTO espaiMb
    FROM jocs
    WHERE codi = 'aliso';

    espaiGb := espaiMb / 1000;

    dbms_output.put_line(espaiGb);
END;
```

Exercici proposat 06.02.03d:

Mostra les dues últimes xifres de l'any de llançament del joc el codi del qual és "last1". Primer hauràs d'obtenir la dada de l'any de llançament i després hauràs de bolcar les seues dues últimes xifres a una variable auxiliar, que finalment mostraràs en pantalla.

6.2.3.4. Dades literals

Com ja hem vist, els caràcters i les dates han d'estar tancats en cometes simples:

```
v_nom := 'Henderson';
```

Els números poden ser simples valors o notacions científiques:

```
v_num := 5;
v_num2 := 1.2e2;
```

6.2.3.5. Comentaris en codi

Com ja coneixes de SQL, existeix la possibilitat de crear comentaris d'una línia, que començaran per dos guions.

A més, en PL/SQL es poden crear comentaris de diverses línies, que es delimitaran entre els símbols `/*` i `*/`, com en aquest exemple:

-- Exemple 06.02.03h

`/* Nom d'un joc,
en concret del de codi "ikaw" */`

DECLARE

`v_nom jocs.nom%TYPE;`

BEGIN

`SELECT nom`

`INTO v_nom`

`FROM jocs`

`WHERE codi = 'ikaw';`

`dbms_output.put_line(v_nom);`

END;

6.2.3.6. Ús de funcions de SQL en PL/SQL

Les funcions de SQL es podran emprar en blocs de PL/SQL. Per exemple, podem convertir el nom d'un joc a majúscules usant la funció `UPPER`:

-- Exemple 06.02.03i

`/* Nom d'un joc en majúscules,
usant la funció UPPER */`

DECLARE

`v_nom jocs.nom%TYPE;`

`v_nomMaj jocs.nom%TYPE;`

BEGIN

`SELECT nom`

`INTO v_nom`

`FROM jocs`

`WHERE codi = 'ikaw';`

`v_nomMaj := UPPER(v_nom);`

`dbms_output.put_line(v_nomMaj);`

END;

Exercici proposat 06.02.03e:

Mostra les quatre primeres lletres de la plataforma el codi de la qual és "psx", després de convertir-les a minúscules. Primer hauràs d'obtenir el nom de la plataforma i després bolcar a una variable auxiliar el resultat de convertir-lo a minúscules i d'extraure les seues 4 primeres lletres. Inclou un comentari de diverses línies, que permeta recordar posteriorment què estaves practicant en aquest exercici.

6.2.3.7. Conversió a tipus de dades comparables

Mesclar tipus de dades pot resultar en un error i afectar el funcionament. A vegades es podrà aplicar unes certes operacions a tipus de dades "inesperades", com un SUBSTR o un LIKE a una dada de tipus data (i, en aqueix cas, s'haurà de tindre en compte la configuració regional, per a saber detalls com si l'any es mostrarà al principi o al final d'aqueixa data, i si ho farà amb 2 xifres o amb 4). Però, en general, serà desitjable convertir al tipus de dades correcte abans d'unues certes manipulacions.

Com ja hem vist en el bloc de SQL, les funcions de conversió són:

- TO_CHAR
- TO_DATE
- TO_NUMBER

Per exemple, el següent script provoca un error (una excepció), en tractar de guardar un resultat numèric en una variable de text:

-- Exemple 06.02.03j
-- Error de tipus de dades

```
DECLARE
    resultat NUMBER(10);
BEGIN
    SELECT nom
    INTO resultat
    FROM jocs
    WHERE codi='mmiz';

    dbms_output.put_line(resultat);
END;
```

L'error que obtindrem és:

ORA-06502: PL/SQL: numeric or value error: character to number conversion error

La conversió de número a text sí que és automàtica, com a mostra el següent exemple:

-- Exemple 06.02.03k
-- Conversió automàtica

```
DECLARE
    espaiMax VARCHAR2(15);
BEGIN
    SELECT MAX(espaiOcupatMb)
    INTO espaiMax
    FROM jocs;
```



```
dbms_output.put_line(espaiMax);  
END;
```

Però en Oracle no és automàtica la conversió de text a data, per la qual cosa el següent script fallarà:

```
-- Exemple 06.02.03I  
-- Conversió incorrecta de text a data
```

```
DECLARE  
  data DATE;  
BEGIN  
  data := '2023-04-13';  
  
  dbms_output.put_line(data);  
END;
```

I obtindrem com a resposta:

ORA-01861: literal does not match format string

La forma correcta seria:

```
data := TO_DATE('2023-04-13', 'YYYY-MM-DD');
```

6.2.3.8. Exercicis addicionals

1.- Determinar quins dels següent Identificadors són equivalents en PL/SQL

- Identificador1 NUMBER;
- Identificador_1 NUMBER;
- identificador1 NUMBER;
- IdEntificador_1 NUMBER;
- IDENTIFICADOR1 NUMBER;

2.- Determinar quin dels següents identificadors en PL/SQL és el vàlid:

- Primera variable VARCHAR2(40);
- end BOOLEAN;
- Una_variable VARCHAR2(40);
- Otra-variable VARCHAR2(40);

3.- Funcionaria la següent sentència en PL/SQL? En cas negatiu, proposar com resoldre-ho.

```
DECLARE  
  v_codi CHAR(5);  
  v_nom VARCHAR2(50);
```

```
v_descripcio VARCHAR2(1000);  
BEGIN  
  SELECT codi, nom, descripcio  
  INTO v_codi, v_nom, v_descripcio  
  FROM jocs;  
END;
```

4.- Indicar, del següent joc de declaracions, quines són correctes i quines no, indicant a més per què no són correctes.

```
DECLARE  
  cierto BOOLEAN:=FALSE;  
  id_externo NUMBER(4) NOT NULL;  
  cosa NUMBER:=2;  
  producto NUMBER:=2*cosa;  
  suma NUMBER:=cosa + la_otra;  
  la_otra NUMBER:=2;  
  tipo_uno NUMBER(7,2) NOT NULL:=3;  
  tipo_otro tipo_uno%TYPE;
```

5.- Es aquesta declaració correcta en PL/SQL?

```
DECLARE  
  i, j NUMBER;
```

I si la escrivim així?

```
DECLARE  
  i, j NUMBER, NUMBER;
```

6.2.4. Contacte amb la inserció i modificació de dades en PL/SQL

Tornarem més endavant amb més detalls, però és interessant conèixer ja que una ordre INSERT (i UPDATE, i DELETE) pot ser part d'un bloc PL/SQL, de manera que es podrien fer coses com aquesta:

```
-- Exemple 06.02.04a  
-- INSERT + UPDATE + DELETE
```

```
DECLARE  
  v_nom plataformes.nom%TYPE;  
BEGIN  
  INSERT INTO plataformes VALUES('ngb', 'Nintendo GameBoy');  
  
  SELECT nom INTO v_nom  
  FROM plataformes
```

```
WHERE codi = 'ngb';
```

```
dbms_output.put_line(v_nom);
```

```
UPDATE plataformes
```

```
SET nom = 'Nintendo Game Boy'
```

```
WHERE codi = 'ngb';
```

```
SELECT nom INTO v_nom
```

```
FROM plataformes
```

```
WHERE codi = 'ngb';
```

```
dbms_output.put_line(v_nom);
```

```
DELETE FROM plataformes
```

```
WHERE codi = 'ngb';
```

```
END;
```

6.3. ESTRUCTURES DE CONTROL EN PL/SQL

6.3.1. Sentència IF

La sentència IF permet comprovar si es compleixen unes certes condicions, per a poder canviar en aqueix cas l'ordre d'execució d'unes certes sentències. La seua sintaxi és:

```
IF condició THEN
    instruccions;
[ELSIF condició THEN
    instruccions;]
[ELSE
    instruccions;]
END IF;
```

Així, és possible crear seqüències com les següents:

IF-THEN-END IF

IF-THEN-ELSE-END IF

IF-THEN-ELSIF-END IF

És habitual distingir entre sentències IF simples i compostes. Les sentències simples tenen la següent sintaxi:

```
IF condició THEN sentència  
END IF;
```

Mentre que les sentències IF compostes fan servir operadors lògics com AND, OR I NOT:

```
IF condició AND condició  
THEN sentència;  
END IF;
```

Com a exemple de sentència IF més completa, podríem calcular una comissió que fora diferent segons diferents rangs de valors d'una variable:

```
IF v_start > 100 THEN  
    v_comissio := 0.2 * v_start;  
ELSIF v_start >= 50 THEN  
    v_comissio := 0.15 * v_start;  
ELSE  
    v_comissio := 0.1 * v_start  
END IF;
```

I un exemple real a partir de les nostres dades de prova podria ser informar sobre la quantitat de jocs que tenim d'una plataforma:

```
-- Exemple 06.03.01a  
-- Contacte amb IF
```

```
DECLARE  
    quantitat NUMBER(4);  
BEGIN  
    SELECT COUNT(*)  
    INTO quantitat  
    FROM jocs  
    WHERE codiPlataforma = 'ps3';  
  
    IF quantitat = 0 THEN  
        dbms_output.put_line('No tenim jocs de PS3');  
    ELSIF quantitat = 1 THEN  
        dbms_output.put_line('Tenim un joc de PS3');  
    ELSE  
        dbms_output.put_line('Tenim diversos jocs de PS3');  
    END IF;  
END;
```

Exercici proposat 06.03.01a:

A partir de les dades de la taula de jocs, crea un script que guarde en una variable la quantitat de jocs que tenim de la plataforma el codi de la qual és "ps3". Si aqueixa quantitat és major que zero, hauràs de mostrar el missatge "Tenim jocs de PS3".

Exercici proposat 06.03.01b:

A partir de les dades de la taula de jocs, crea un script que guarde en una variable la quantitat de jocs que tenim de la plataforma el codi de la qual és "ps3". En una altra variable, guarda la quantitat de jocs que tenim de la plataforma el codi de la qual és "ps4". Finalment, mostra el missatge "Tenim més jocs de PS3 que de PS4" o bé "No tenim més jocs de PS3 que de PS4", segons corresponga.

Exercici proposat 06.03.01c:

Crea una versió millorada de l'exercici anterior: a partir de les dades de la taula de jocs, crea un script que guarde en una variable la quantitat de jocs que tenim de la plataforma el codi de la qual és "ps3". En una altra variable, guarda la quantitat de jocs que tenim de la plataforma el codi de la qual és "ps4". Finalment, mostra el missatge "Tenim més jocs de PS3 que de PS4", o bé "Tenim més jocs de PS4 que de PS3", o "Tenim la mateixa quantitat de jocs de PS3 que de PS4", segons corresponga.

6.3.2. Sentència CASE

La sentència CASE avalua diferents condicions o valors fins a trobar alguna que es complisca. Es pot utilitzar amb dues sintaxis diferents. Una primera sintaxi, que recorda a la sentència "switch" dels llenguatges derivats de C (com Java, C# i JavaScript) és:

```
CASE [expressio]
WHEN [condicio1|valor1] THEN
    bloc_instruccions_1
WHEN [condicio2|valor2] THEN
    bloc_instruccions_2
...
ELSE
    bloc_instruccions_per_defecte
END CASE;
```

Un exemple del seu ús podria ser:

```
-- Exemple 06.03.02a
-- Contacte amb CASE
```

```
DECLARE
    quantitat NUMBER(4);
BEGIN
    SELECT COUNT(*)
    INTO quantitat
    FROM jocs
    WHERE codiPlataforma = 'ps5';

    CASE quantitat
        WHEN 0 THEN
            dbms_output.put_line('No tenim jocs de PS5');
        WHEN 1 THEN
            dbms_output.put_line('Tenim un joc de PS5');
        ELSE
            dbms_output.put_line('Tenim dos o més jocs de PS5 jocs');
    END CASE;
END;
```

Exercici proposat 06.03.02a:

A partir de les dades de la taula de plataformes (de jocs), crea un script que escriba "No tenim dades de plataformes basades en Playstation", "Tenim dades d'1 plataforma basada en Playstation", "Tenim dades de 2 plataformes basades en Playstation" o "Tenim dades de més de 2 plataformes basada en Playstation", segons corresponga, emprant CASE.

Com a alternativa, la sentència CASE també es pot usar **en format d'expressió**, que selecciona entre diversos valors possibles i retorna un d'ells. La sintaxi en aquest cas és:

```
CASE selector
    WHEN expression1 THEN result1
    WHEN expression2 THEN result2
    ...
    WHEN expressionN THEN resultN
    [ELSE resultN+1;]
END;
```

Un exemple d'aquest format pot ser:

```
-- Exemple 06.03.02b
-- Contacte amb CASE
```

```
DECLARE
    quantitat NUMBER(4);
    resposta VARCHAR2(30);
BEGIN
    SELECT COUNT(*)
    INTO quantitat
```

```
FROM jocs
WHERE codiPlataforma = 'ps5';

resposta := CASE quantitat
WHEN 0 THEN 'No tenim jocs de PS5'
WHEN 1 THEN 'Tenim un o dos jocs de PS5'
ELSE 'Tenim tres o més jocs de PS5'
END;

dbms_output.put_line(resposta);
END;
```

Detalls a tindre en compte en el format de CASE com a expressió:

- Els blocs WHEN no acaben en punt i coma.
- No es tanca amb END CASE.

Exercici proposat 06.03.02b:

A partir de les dades de la taula de plataformes (de jocs), crea un script que emplene una variable de text amb els valors "Zero", "Un", "Dos" o "Més de dos" segons la quantitat de jocs que tenim de la plataforma "ps4". Finalment, haurà de mostrar el valor d'aqueixa variable.

6.3.3. Taules lògiques

Les taules lògiques en SQL i PL/SQL són una ampliació de les que existeixen en altres llenguatges de programació, perquè en aquest cas s'haurà de considerar també els valors nuls:

Dato1	Dato2	AND
TRUE	TRUE	TRUE
TRUE	FALSE	FALSE
FALSE	FALSE	FALSE
TRUE	NULL	NULL
FALSE	NULL	FALSE
NULL	NULL	NULL

Dato1	Dato2	OR
-------	-------	----

TRUE	TRUE	TRUE
TRUE	FALSE	TRUE
FALSE	FALSE	FALSE
TRUE	NULL	TRUE
FALSE	NULL	NULL
NULL	NULL	NULL

Dato1	NOT
TRUE	FALSE
FALSE	TRUE
NULL	NULL

Es podria comprovar el comportament de valors com els anteriors usant un programa com aquest:

-- Exemple 06.03.03a

-- Taules lògiques

DECLARE

v1 **BOOLEAN**;

v2 **BOOLEAN**;

r **BOOLEAN**;

BEGIN

v1 := **TRUE**;

v2 := **NULL**;

r := v1 **AND** v2;

IF r **IS NULL THEN**

dbms_output.put_line('NULL');

ELSIF r **THEN**

dbms_output.put_line('TRUE');

ELSE

dbms_output.put_line('FALSE');

END IF;

END;

En el cas d'algun gestor de bases de dades, com SQLite (però no en Oracle), es poden provar de forma encara més senzilla, emprant sentències com:

SELECT FALSE AND NULL;

Exercici proposat 06.03.03a:

Quin és el valor de V_FLAG en cada cas? `v_flag := v_reorder_flag AND v_available_flag;`

V_REORDER_FLAG	V_AVAILABLE_FLAG	V_FLAG
TRUE	TRUE	?
TRUE	FALSE	?
NULL	TRUE	?
FALSE	FALSE	?

6.3.3. Estructures repetitives en PL/SQL

Un bucle és una seqüència de sentències que s'especifiquen només una vegada però que poden executar-se diverses vegades, de manera repetitiva. En PL/SQL es poden definir 3 tipus de bucles::

- Bucle WHILE. Es basa en una condició, que es comprova abans de començar el fragment repetitiu.
- Bucle LOOP. Es tracta d'accions repetitives sense condicions globals. La condició de finalització del bucle, que és opcional, se sol indicar a prop el final de la part repetitiva.
- Bucle FOR. Indica accions repetitives basant-se en un comptador. Ho emprarem quan calga donar un nombre conegut de voltes.

6.3.4.1. El bucle LOOP

És possiblement l'estructura repetitiva més senzilla de PL/SQL. La seua sintaxi és:

```
LOOP                                -- delimitador d'inici

    sentència1;                     -- sentències

    ...

    EXIT [WHEN condició]            -- condició d'eixida

END LOOP;                           -- delimitador de fi
```

En la seua sintaxi més bàsica, repeteix la seqüència de sentències que s'indique entre "LOOP" i "END LOOP;". És habitual incloure la sentència EXIT per a indicar que acabe el bucle (en cas contrari, estaríem provocant un "bucle sense fi". És d'esperar que EXIT s'invoque de manera condicional, ja siga dins d'un bloc IF o, preferiblement, usant la construcció "EXIT WHEN condició", com mostren els següents exemples.

Així es podria crear un bucle sense fi:

-- Exemple 06.03.04.01a
-- LOOP en un bucle sense fi

```
BEGIN
  LOOP
    dbms_output.put_line('A escribir mucho...');
  END LOOP;
END;
```

D'aquesta manera només es mostraria la primera línia de les dues que formen el cos del bucle:

-- Exemple 06.03.04.01b
-- LOOP + EXIT sense condició

```
BEGIN
  LOOP
    dbms_output.put_line('Primera frase...');
    EXIT;
    dbms_output.put_line('Segona frase...');
  END LOOP;
END;
```

I així es podria crear un comptador (una miqueta antinatural) d'1 a 5:

-- Exemple 06.03.04.01c
-- Contador de 1 a 5 amb LOOP + IF + EXIT

```
DECLARE
  i NUMBER;
BEGIN
  i := 1;
  LOOP
    dbms_output.put_line(i);
    IF i >= 5 THEN
      EXIT;
    END IF;
    i := i + 1;
  END LOOP;
END;
```

I aquesta variant usa una sintaxi més natural:

-- Exemple 06.03.04.01d
-- Contador de 1 a 5 amb LOOP + EXIT WHEN

```
DECLARE
  i NUMBER;
BEGIN
  i := 1;
```

```
LOOP
    dbms_output.put_line(i);
    EXIT WHEN i >= 5;
    i := i + 1;
END LOOP;
END;
```

Exercici proposat 06.03.04.01a:

Crea un bucle LOOP que mostre els números del 10 al 100, augmentant de 10 en 10. Has d'emprar la sintaxi EXIT WHEN.

Exercici proposat 06.03.04.01b:

Crea un bucle LOOP que mostre els números del 10 al 1, descomptant d'1 en 1. Has d'emprar la sintaxi EXIT WHEN.

6.3.4.2. El bucle WHILE

S'utilitza per a repetir sentències mentre una certa condició (que es comprova al principi) siga vertadera. La seua sintaxi és:

```
WHILE condició LOOP          -- La condició es avaluada al començament
    sentència1;               -- de cada iteració
    sentència2;
    ...
END LOOP;
```

Nota: Si la condició produeix un NULL, el cicle és desviat al control de la propera sentència.

Com a exemple, es podria repetir el comptador d'un a cinc anterior, aquesta vegada emprant un bucle WHILE, de la següent forma:

-- Exemple 06.03.04.02a
-- Contador de 1 a 5 amb WHILE

```
DECLARE
    i NUMBER;
BEGIN
    i := 1;
    WHILE i <= 5 LOOP
        dbms_output.put_line(i);
        i := i + 1;
```

END LOOP;
END;

Exercici proposat 06.03.04.02a:

Crea un bucle WHILE que mostre els números del 10 al 100, augmentant de 10 en 10.

Exercici proposat 06.03.04.02b:

Crea un bucle WHILE que mostre els números del 10 al 1, descomptant d'1 en 1.

6.3.4.3. El bucle FOR

S'empra quan es tracta d'un nombre fix de repeticions. La seua sintaxi és:

```
FOR índex IN [REVERSE] valor_inicial .. valor_final LOOP
    instruccions;
...
END LOOP;
```

No és necessari declarar l'índex, el sistema el farà per nosaltres de manera implícita. Pel mateix motiu, no es podrà usar l'índex fora del bucle, ja que en altres zones del programa no estarà definit.

L'índex no es pot modificar, no s'haurà d'usar com a objectiu d'una assignació.

Com a exemple, una nova versió del programa que compta d'1 a 5 podria ser:

```
-- Exemple 06.03.04.03a
-- Contador de 1 a 5 amb FOR
```

```
BEGIN
  FOR i IN 1 .. 5 LOOP
    dbms_output.put_line(i);
  END LOOP;
END;
```

Per a comptar de manera descendent, s'usarà la paraula clau REVERSE, així:

```
-- Exemple 06.03.04.03b
-- Contador de 5 a 1 amb FOR + REVERSE
```

```
BEGIN
  FOR i IN REVERSE 1 .. 5 LOOP
    dbms_output.put_line(i);
  END LOOP;
END;
```

Exercici proposat 06.03.04.03a:

Crea un bucle FOR que mostre els números del 20 al 30.

Exercici proposat 06.03.04.03b:

Crea un bucle FOR que mostre els números del 10 al 1, descomptant d'1 en 1.

Exercici proposat 06.03.04.03c:

Crea un bucle FOR que mostre els números del 20 al 200, augmentant de 10 en 10.

Nota: En el cas de **PostgreSQL**, els valors d'inici i fi d'un comptador FOR que incloga la paraula **REVERSE** hauran d'indicar-se a l'inrevés (primer el valor més alt i després el més baix). A més, PL/pgSQL inclou una paraula reservada **BY**, que permet usar increments diferents d'un, com en aquest exemple:

```
FOR i IN REVERSE 10..1 BY 2 LOOP
```

```
...
```

Com ja havíem avançat en l'apartat 6.2.4, podem fer insercions, modificacions i esborrats de dades des de PL/SQL. Per tant, podríem emprar qualsevol dels bucles que hem vist per a introduir dades repetitives, com podrien ser 5 consoles fictícies anomenades des de GamingStation 1 fins a GamingStation 5, de la següent forma:

```
-- Exemple 06.03.04.03c
-- Inserció repetitiva
DECLARE
  v_id plataformes.codi%TYPE;
  v_nom plataformes.nom%TYPE;
BEGIN
  FOR i IN 1 .. 5 LOOP
    v_id := 'gs' || i;
    v_nom := 'GamingStation' || i;
    INSERT INTO plataformes VALUES(v_id, v_nom);
  END LOOP;
END;
```

6.3.4.3. Recomanacions per a l'ús de cada tipus de bucles

- Serà recomanable emprar un bucle LOOP quan les sentències dins del bucle hagen d'executar-se almenys una vegada.

- L'estructura WHILE s'utilitzarà si la condició ha de ser avaluada al començament de cada iteració (i, per tant, potser no arriba a repetir-se mai).
- La sentència FOR es reservarà per a quan es conega la quantitat d'iteracions.

6.3.4.5. Bucles niats i etiquetes

Es poden niar cicles a múltiples nivells. Es pot niar bucles LOOP, FOR o WHILE dins de qualsevol altre.

Quan hi ha bucles anidats, la finalització del bucle interior no implica la culminació del cicle, llevat que es llance una excepció. Així i tot, és possible etiquetar cicles i eixir del cicle extern amb la sentència EXIT.

Un primer exemple senzill de bucle niat podria ser:

```
-- Exemple 06.03.04.05a
-- Bucle niat senzill
BEGIN
  FOR fila IN 1 .. 2 LOOP
    FOR columna IN 1 .. 5 LOOP
      dbms_output.put_line( fila || ', ' || columna );
    END LOOP;
  END LOOP;
END;
```

El resultat del qual seria:

```
1,1
1,2
1,3
1,4
1,5
2,1
2,2
2,3
2,4
2,5
```

I es pot forçar una eixida prematura amb EXIT així:

```
-- Exemple 06.03.04.05b
-- Bucle niat amb EXIT
BEGIN
  FOR fila IN 1 .. 2 LOOP
    FOR columna IN 1 .. 5 LOOP
      IF columna = 4 THEN
        EXIT;
      END IF;
      dbms_output.put_line( fila || ', ' || columna );
    END LOOP;
  END LOOP;
END;
```

```
END LOOP;  
END LOOP;  
END;
```

Que tindria com a resultat:

```
1,1  
1,2  
1,3  
2,1  
2,2  
2,3
```

Però es pot forçar al fet que s'isca també del bucle exterior, afegint etiquetes, de la següent forma:

```
-- Exemple 06.03.04.05c  
-- Bucle niat amb EXIT (2)  
BEGIN  
  <<exterior>>  
  FOR fila IN 1 .. 2 LOOP  
    <<interior>>  
    FOR columna IN 1 .. 5 LOOP  
      IF columna = 4 THEN  
        EXIT exterior;  
      END IF;  
      dbms_output.put_line( fila || ',' || columna );  
    END LOOP;  
  END LOOP;  
END;
```

I en aquest cas el resultat seria:

```
1,1  
1,2  
1,3
```

6.3.4.6. Exercicis de repàs d'estructures repetitives

1.- És correcta la següent sintaxi general de la sentència IF-THEN ELSE? Per què ? Com la escriuries?

```
BEGIN  
  IF condicion1 THEN  
    BEGIN  
      secuencia_de_instrucciones1;  
    ELSE  
      secuencia_de_instrucciones2;
```

```
ENDIF;  
END;
```

2.- Quin resultat ens donaria la següent comparació?

```
DECLARE  
    identificador1 VARCHAR2(10):='Hola Pepe';  
    identificador2 VARCHAR2(10):='Hola pepe';  
BEGIN  
    IF identificador1<>identificador2 THEN  
        RETURN TRUE;  
    ELSE  
        RETURN FALSE;  
    END IF;  
END;
```

3.-Indicar que errors hi ha al següent cod:

```
DECLARE  
    a NUMBER:=1;  
    b NUMBER:=6;  
    salida_bucle BOOLEAN;  
BEGIN  
    salida_bucle:='FALSE';  
    WHILE NOT salida_bucle  
    LOOP  
        BEGIN  
            IF a>=b THEN  
                salida_bucle:='TRUE';  
            ELSE  
                a:=(a+1);  
            END IF;  
        END LOOP;  
    END;
```

4- Quin valor ha de contenir la variable 'sumador' al eixir del bucle?, Per què?

```
DECLARE  
    sumador NUMBER;  
BEGIN  
    FOR i IN 1..100 LOOP  
        sumador:=sumador+i;  
    END LOOP;  
END;
```


5- Funcionaria el següent tros de codi? Per què? Com arreglar-lo?

```
DECLARE
    mi_valor NUMBER;
    cierto BOOLEAN:=FALSE;
BEGIN
    WHILE NOT cierto
    LOOP
        IF mi_valor=NULL THEN
            mi_valor:=1;
        ELSE
            mi_valor:=mi_valor+1;
        END IF;
        IF mi_valor>100 THEN cierto:=TRUE; END IF;
    EXIT WHEN cierto;
    END LOOP;
END;
```

6.- Escriure la sintaxi general d'un codi que avalue si es compleix una condició, en cas de complir-se que execute una sèrie de sentències, en cas contrari que avalue una altra, que de complir-se execute altres instruccions, si aquesta no es compleix que avalue una tercera condició... i així N vegades. En cas d'existir diverses solucions, comentar-les i escriure la òptima o més clara.

7.- Implementar en PL/SQL un bucle infinit que vaja sumant valors en una variable de tipus NUMBER.

8.- Basant-se en el bucle anterior, afegir la condició que isca quan la variable siga més gran que 10.000, usa el bucle WHILE

9.- Implementar un bucle en PL/SQL en el qual anem sumant valors a una variable mentre aquesta siga menor que 10, assegurant-nos que el bucle s'executa almenys una vegada.

10.- Implementar en PL/SQL, el codi necessari d'un programa que al final de la seua execució haja emmagatzemat en una variable anomenada 'cadena', el següent valor: cadena := '10*9*8*7*6*5*4*3*2*1'

6.4. Sentències DML en PL/SQL

Amb PL/SQL podem:

- Obtindre dades d'una sentència SELECT.
- Realitzar modificacions de dades (UPDATE) i esborrats (DELETE).
- Determinar el resultat d'aquestes operacions del Llenguatge de Manipulació de dades (DML).
- Controlar transaccions.
- ...

PL/SQL no suporta directament el Llenguatge de Definició de Dades (DDL), com CREATE TABLE, ALTER TABLE o DROP TABLE.

PL/SQL no suporta sentències del Llenguatge de Control de Dades (DCL), com GRANT o REVOKE, que estudiarem més endavant.

6.4.1. Recuperar datos con SELECT

La sintaxi de SELECT dins de PL/SQL serà:

```
SELECT select_llista  
INTO {variable_nom[, ...] ...  
      | nom_registre}  
FROM nom_taula  
WHERE condició;
```

On tenim:

- nom_taula: Especifica el nom de la taula de la base de dades.
- condició: Està composta d'un nom de columna, expressió, constant, i operadors de comparació incloent variables i constants PL/SQL.
- select_llista: És una llista de al menys una columna i pot incloure expressions SQL, files, funcions, o grups de funcions.
- variable_nom: És la variable escalar que reté el valor recuperat.
- nom_registre: És el registre PL/SQL que alberga el valor recuperat.

Alguns detalls a tindre en compte:

- S'ha d'acabar cada sentència SQL amb un punt i coma (;).
- La clàusula INTO és obligatòria en SELECT quan aquesta sentència s'insereix en PL/SQL.
- La clàusula WHERE és opcional i pot ser usada per a especificar variables d'entrada, constants, literals o expressions PL/SQL.
- S'haurà d'especificar el mateix nombre de variables en la clàusula INTO que columnes que apareixen en la clàusula SELECT. Caldrà assegurar-se que es corresponguen posicionalment i que els seus tipus de dades siguen compatibles.
- És possible utilitzar funcions d'agrupació, com SUM, en una sentència SQL dins de PL/SQL.

També és important destacar que les consultes han de retornar una i només una fila. Una consulta que retorne més d'una fila o cap genera un error. PL/SQL maneja aquests errors llançant excepcions (que tractarem més endavant) i que l'usuari podria atrapar: NO_DATA_FOUND i TOO_MANY_ROWS.

Un primer exemple, que obté 2 dades emprant una única sentència SELECT podria ser:

-- Exemple 06.04.01a

/* SELECT INTO + 2 variables (millorable) */

DECLARE

v_codi **CHAR**(4);
v_nom **VARCHAR2**(30);

BEGIN

SELECT codi, nom
INTO v_codi, v_nom
FROM plataformes
WHERE codi = 'ps3';

dbms_output.put_line(v_codi || ' - ' || v_nom);

END;

Però es podria millorar prenent el tipus de dades que realment correspon a cada columna:

-- Exemple 06.04.01b

/* SELECT INTO + 2 variables */

DECLARE

v_codi plataformes.codi % **TYPE**;
v_nom plataformes.nom % **TYPE**;

BEGIN

SELECT codi, nom
INTO v_codi, v_nom
FROM plataformes
WHERE codi = 'ps3';

dbms_output.put_line(v_codi || ' - ' || v_nom);

END;

I també podem fer servir funcions d'agregació, així:

-- Exemple 06.04.01b

/* SELECT INTO + funció d'agregació */

DECLARE

espaiMb jocs.espaiOcupatMb%**TYPE**;

BEGIN

SELECT SUM(espaiOcupatMb)

```
INTO espaiMb
FROM jocs;

dbms_output.put_line(espaiMb);
END;
```

Exercici proposat 06.04.01a:

Obtingues, amb una única sentència SELECT, el nom i l'any del joc el codi del qual és "aa". Mostra totes dues dades en pantalla.

6.4.2. Inserir dades usant PL/SQL

Un exemple d'inserció seria el següent:

```
-- Exemple 06.04.02a
-- INSERT en PL/SQL
```

```
DECLARE
  v_nom jocs.nom % TYPE := 'Fruity Frank';
BEGIN
  INSERT INTO jocs
    (codi, nom, codiPlataforma)
  VALUES('ff', v_nom, 'cpc');
END;
```

Es pot observar que ens hem ajudat d'una variable auxiliar i que no hem indicat els valors de totes les columnes, per la qual cosa algunes d'elles tindran el seu valor per defecte. També hauria sigut possible utilitzar funcions com SYSDATE (data actual), o seqüències per a crear claus primàries que s'incrementen automàticament (i que estudiarem més endavant).

No ha d'existir ambigüïtat amb identificadors i noms de columnes en la sentència SELECT. Cap identificador en la clàusula INSERT ha de ser el nom d'una columna de la base de dades.

Exercici proposat 06.04.02a:

Afig, des de PL/SQL, una nova plataforma de jocs, el codi de la qual serà "nds" i el nom de la qual serà "Nintendo DS".

També és possible inserir dades noves ajudant-nos d'un ROWTYPE, com en el següent exemple:

```
-- Exemple 06.04.02b
-- INSERT en PL/SQL, amb ROWTYPE
```

```
DECLARE
  reg_plat plataformes % ROWTYPE;
BEGIN
  reg_plat.codi := 'c64';
  reg_plat.nom := 'Commodore 64';
  INSERT INTO plataformes VALUES reg_plat;
END;
```

Exercici proposat 06.04.02b:

Afig, des de PL/SQL, una nova plataforma de jocs, usant ROWTYPE. El seu codi serà "smd" i el seu nom serà "Sega MegaDrive".

6.4.3. Introducció a les seqüències. RETURNING

Alguns gestors de bases de dades permeten crear claus primàries numèriques que s'incrementen automàticament. En versions antigues de Oracle (11 i posteriors), aquesta tasca és un poc més treballosa, però es pot aconseguir amb l'ajuda de "seqüències". El primer pas seria crear la seqüència com a tal (aquesta és un orde de SQL, no de PL/SQL):

```
CREATE SEQUENCE contadorPartides;
```

Podríem comprovar el pròxim valor d'aqueixa seqüència amb NEXTVAL. Com les sentències SELECT en Oracle (fins a la versió 22) sempre necessiten un FROM (cosa que no ocorre en altres gestors), és habitual recórrer a una taula anomenada "dual", que és part del sistema, que conté un únic registre i que s'utilitza per a casos com obtindre el següent valor d'una seqüència o bé obtindre la data del sistema. És a dir, faríem:

```
SELECT contadorPartides.NEXTVAL FROM dual;
```

Ara podríem crear una nova taula que tinguera una clau numèrica, com aquesta, de suposades partides a jocs, potser amb un altre jugador:

```
CREATE TABLE partides(
  num NUMBER(5),
  codiJoc CHAR(5),
  nomJugador VARCHAR2(50),
  CONSTRAINT pk_partides PRIMARY KEY (num),
  CONSTRAINT fk_partides_jocs
    FOREIGN KEY (codiJoc) REFERENCES jocs(codi)
);
```

I usaríem aqueix comptador com a dada auxiliar a l'hora d'inserir nous registres en aqueixa taula:

```
INSERT INTO partides  
VALUES ( contadorPartidas.NEXTVAL, 'efre', 'Héctor' );
```

Si realitzem aquesta mateixa operació des de PL/SQL, podem afegir una clàusula RETURNING per a obtindre quin valor tenia la seqüència en el moment del guardat de dades:

-- Exemple 06.04.03a

```
DECLARE  
    v_contador partides.num % TYPE;  
BEGIN  
    INSERT INTO partides  
        (num, codiJoc, nomJugador)  
    VALUES(contadorPartides.NEXTVAL, 'cont', 'Aurora')  
    RETURNING num INTO v_contador;  
  
    dbms_output.put_line('Contador ' || v_contador);  
END;
```

Amb aqueixa ordre RETURNING indiquem la columna en la qual ens volem basar ("num" en aquest cas) i la variable a la qual volem bolcar el seu valor (v_comptador).

Exercici proposat 06.04.03a:

Desitgem crear una nova taula a manera de diari, en la qual anar anotant idees que se'ns ocorreguen. Crea una taula "diari", amb dos camps: un codi numèric i un text. Crea una seqüència que permet afegir-li dades de manera correlativa. Afig una primera dada des de SQL, usant el valor d'aqueixa seqüència. Inclou una dada addicional des de PL/SQL i mostra el valor que té la seqüència en aqueix moment.

6.4.4. Actualització de dades en PL/SQL

Pot haver-hi ambigüitat en la clàusula SET d'una sentència UPDATE, perquè, encara que l'identificador en la part esquerra de l'operador d'assignació és sempre una columna de la base de dades, l'identificador de la part dreta pot ser una columna de la base de dades o una variable de PL/SQL.

També és important recordar que la clàusula WHERE s'usa per a determinar a quines files afectarà la modificació. Si no es modifica cap registre, no es considera un error, a diferència del que podria ocórrer en una sentència SELECT en PL/SQL.

Un exemple bàsic podria ser així:

```
-- Exemple 06.04.04a  
-- UPDATE en PL/SQL
```

```
DECLARE
  v_codi jocs.nom % TYPE := 'aa6';
  v_nom jocs.nom % TYPE := 'Ant Attack';
BEGIN
  INSERT INTO jocs
    (codi, nom, codiPlataforma)
    VALUES(v_codi, v_nom, 'c64');

  UPDATE jocs
    SET anyLlancament = 1984
    WHERE codi = v_codi;
END;
```

I, igual que en INSERT, també podem ajudar-nos de ROWTYPE:

```
-- Exemple 06.04.04b
-- UPDATE en PL/SQL, amb ROWTYPE
```

```
DECLARE
  reg_plat plataformes % ROWTYPE;
BEGIN
  reg_plat.codi := 'o1';
  reg_plat.nom := 'Oric';
  INSERT INTO plataformes VALUES reg_plat;

  reg_plat.nom := 'Oric 1';
  UPDATE plataformes
    SET ROW = reg_plat
    WHERE codi = 'o1';
END;
```

I també podem emprar RETURNING per a obtindre dades del registre afectat (que, com sempre, haurà de ser només un o, en cas contrari, obtindrem una excepció):

```
-- Exemple 06.04.04c
-- UPDATE en PL/SQL, amb RETURNING
```

```
DECLARE
  v_plat jocs.codiPlataforma % TYPE;
BEGIN
  UPDATE jocs
    SET nom = 'The last of us'
    WHERE codi = 'last1'
    RETURNING codiPlataforma INTO v_plat;

  dbms_output.put_line('Plataforma: ' || v_plat);
END;
```

En aquest exemple, es mostraria "ps3", que és el codi de plataforma per al joc de codi "last1".

Exercici proposat 06.04.04a:

Modifica el joc de codi "batao" perquè el seu nom passe a ser "Batman: Arkham Origins". Mostra l'any de llançament del registre que haja sigut afectat.

6.4.5. Esborrar dades en PL/SQL

La sentència DELETE elimina files no desitjades d'una taula. Si no s'utilitza una clàusula WHERE, podria arribar a esborrar-se tot el contingut de la taula (si no existeixen restriccions d'integritat). Igual que en INSERT i UPDATE, es pot emprar ROWTYPE per a donar valor a tot un registre, i RETURNING per a comprovar detalls del registre que s'ha esborrat. Per exemple, podem esborrar la plataforma de codi "o1" de la següent forma:

```
-- Exemple 06.04.05a  
-- DELETE en PL/SQL, amb ROWTYPE i RETURNING
```

DECLARE

```
reg_plat plataformes % ROWTYPE;
```

BEGIN

```
DELETE FROM plataformes
```

```
WHERE codi = 'o1'
```

```
RETURNING codi, nom INTO reg_plat;
```

```
dbms_output.put_line('Plataforma borrada: ' || reg_plat.nom);
```

END;

Exercici proposat 06.04.05a:

Esborra el joc de codi "c64" i mostra quin era el seu nom.

6.4.6. Quantitat de registres afectats

Amb freqüència, en el cas de registres esborrats i modificats, ens interessarà conèixer la quantitat de registres afectats (quantes files s'han esborrat o, segons el cas, quantes s'han modificat amb una ordre UPDATE), i això el podem aconseguir mirant el valor de SQL%ROWCOUNT. El seu valor s'ha de consultar just després de la corresponent sentència del DML, per la qual cosa és habitual bolcar-lo a una variable, per a poder utilitzar-ho més tard.

```
-- Exemple 06.04.06a  
-- SQL%ROWCOUNT
```



```
DECLARE
  modificats NUMBER(5);
BEGIN
  -- Prova de modificació
  UPDATE plataformes
    SET nom = 'Amstrad CPC i CPC+'
    WHERE codi = 'cpc';

  -- Memoritzem el resultat
  modificats := SQL%ROWCOUNT;

  -- I fem una altra operació (desfem canvis)
  UPDATE plataformes
    SET nom = 'Amstrad CPC i CPC+'
    WHERE codi = 'cpc';

  dbms_output.put_line('Actualitzats: ' || modificats);
END;
```

Exercici proposat 06.04.06a:

Esborra la plataforma de codi "c64" i comprova que realment s'ha esborrat.

6.5. Procediments i funcions

En PL/SQL es poden escriure quatre tipus de blocs de codi:

- Blocs anònims: No s'emmagatzemen en la BD. S'executen després d'escriure'ls.
- Procediments: Es queden emmagatzemats i es poden invocar. Poden rebre i retornar múltiples paràmetres. Donen una sèrie de passos i no retornen cap valor.
- Funcions: També es queden emmagatzemades i es pot invocar. Poden rebre paràmetres i retornar o valor.
- Triggers: S'emmagatzemen i s'executen automàticament quan ocorre algun esdeveniment.

6.5.1. Procediments en PL/SQL

La sintaxi bàsica d'un procediment en PL/SQL és:

```
CREATE OR REPLACE PROCEDURE [esquema].nomproc
  (nom-paràmetre {IN, OUT, IN OUT} tipus de dada, ..) {IS, AS}
```

Declaració de variables;
Declaració de constants;

Declaració de cursors;

BEGIN

Cos del subprograma PL/SQL;

EXCEPTION

Bloc d'excepcions PL/SQL;

END;

Aclariments a la sintaxi de procediments i funcions

- Nom-paràmetre: és el nom que nosaltres vulguem donar al paràmetre. Podem utilitzar múltiples paràmetres. En cas de no necessitar-los, es pot ometre els parèntesis.
- IN: especifica que el paràmetre és d'entrada i que per tant aquest paràmetre ha de tindre un valor en el moment d'invocar a la funció o procediment. Si no s'especifica res, els paràmetres per defecte són de tipus entrada.
- OUT: especifica que es tracta d'un paràmetre d'eixida. Són paràmetres el valor dels quals és retornat després de l'execució el procediment al bloc PL/SQL que el va invocar. Les funcions PL/SQL no admeten paràmetres d'eixida.
- IN OUT: Són paràmetres d'entrada i eixida alhora.

Un exemple real podria ser:

-- Exemple 06.05.01a
-- Contacte amb PROCEDURE

```
CREATE OR REPLACE PROCEDURE EscriureDestacat(cadena IN VARCHAR2) IS
BEGIN
    DBMS_OUTPUT.PUT_LINE('--- ' || cadena || ' ---');
END EscriureDestacat;
```

I aqueix procediment es podria invocar d'un script com el següent:

```
DECLARE
    missatge VARCHAR2(30) := 'Hola';
BEGIN
    EscriureDestacat(missatge);
END;
```

Per a tornar a invocar al procediment no és necessari tornar-lo a definir, es queda guardat com a part de la base de dades.

Per descomptat, el bloc anònim que el utilitza no necessàriament haurà de declarar variables, sinó que també podria ser:

BEGIN

 EscriureDestacat('Hola');

END;

I també es podria invocar directament, sense necessitat de bloc anònim, si s'empra la sentència EXECUTE:

EXECUTE EscriureDestacat('Hola');

Exercici proposat 06.05.01a:

Crea un procediment "EscriureGuions", que escriga diversos guions en l'eixida de la base de dades, tants com s'indiquen en un paràmetre numèric (d'entrada).

Exercici proposat 06.05.01b:

Crea un procediment "InserirPlataforma", que afija una nova plataforma a la base de dades de jocs i plataformes. El seu codi i el seu nom s'indicaran com a paràmetres (d'entrada).

Exercici proposat 06.05.01c:

Crea un procediment "InserirJoc", que afija un nou joc a la base de dades. Rebrà com a paràmetres el codi del joc, el nom del joc i el codi de la plataforma. Si la plataforma no existeix, es limitarà a escriure l'avís de "No existeix aqueixa plataforma", i si existeix, guardarà el joc i avisarà "Joc guardat correctament".

Exercici proposat 06.05.01d:

Crea un procediment anomenat "MostrarJocIncomplet", que mostre el nom d'un joc del qual no sabem quant espai ocupa, juntament amb el nom de la plataforma corresponent. Com encara no sabem obtindre dades de manera repetitiva, prendrà només el joc el nom del qual siga el primer alfabèticament (ho millorarem més endavant). No rebrà cap detall com a paràmetre.

6.5.2. Funcions en PL/SQL

La sintaxi d'una funció serà la següent:

```
CREATE OR REPLACE FUNCTION [esquema].nom-funcio  
  (nom-paràmetre tipus-de-dada, ..)  
  RETURN tipus-de-dada  
{IS, AS}
```

```
  Declaració de variables;  
  Declaració de constants;  
  Declaració de cursors;
```

```
BEGIN  
  Cos del subprograma PL/SQL;
```

```
EXCEPTION  
  Bloc d'excepcions PL/SQL;
```

```
END;
```

Com es pot observar, recorda molt a un procediment, però en aquesta ocasió serà necessari indicar el tipus de dades d'eixida, així com retornar-lo amb una (o diverses) sentències RETURN, com mostra el següent exemple, que calcula el factorial d'un número de forma recursiva:

```
-- Exemple 06.05.02a  
-- Contacte amb FUNCTION
```

```
CREATE OR REPLACE FUNCTION Factorial (N IN NUMBER) RETURN NUMBER  
AS  
BEGIN  
  IF N<0 THEN  
    RETURN -1;  
  ELSIF N=0 THEN  
    RETURN 1;  
  ELSE  
    RETURN N*Factorial(N-1);  
  END IF;  
END Factorial;
```

I que es podria utilitzar així:

```
DECLARE  
  v NUMBER := 5;  
  resultat NUMBER;  
BEGIN  
  resultat := Factorial(v);  
  DBMS_OUTPUT.PUT_LINE ('Factorial de ' || v || ' = ' || resultat);  
END;
```

Exercici proposat 06.05.02a:

Converteix l'exemple 06.05.01a a funció: en comptes d'escriure en l'eixida el text entre guions, retornarà aquest text entre guions. Haurà de dir-se "DestacarText". Comprova que es comporta correctament.

Exercici proposat 06.05.02b:

Crea una funció anomenada "QuantitatDeJocs", que reba com a paràmetre el nom d'una plataforma de jocs de la nostra base de dades i retorne la quantitat de jocs que tenim per a aqueixa plataforma. Prova-la des d'un script.

Exercici proposat 06.05.02c:

Crea una funció anomenada "JocMesAntic", que reba com a paràmetre el nom d'una plataforma de jocs i retorne el nom del joc més antic (comprovant el camp de l'any de llançament) que tenim per a aqueixa plataforma, o bé el text "Plataforma no trobada" si no existeix una plataforma amb aqueix nom. Comprova el seu funcionament.

Exercici proposat 06.05.02d:

Crea una funció anomenada "AnyMitja", que reba com a paràmetre el nom d'una plataforma de jocs i retorne la mitjana dels anys de llançament dels jocs que pertanyen a aqueixa plataforma. Mostra el seu valor per a la plataforma anomenada "Amstrad CPC".

Exercici proposat 06.05.02e:

Crea una funció anomenada "QuantitatDePlataformesSenseJocs", que no rebrà cap paràmetre, i retornarà la quantitat de plataformes que apareixen en la nostra base de dades, però per a les quals no tenim anotat cap joc. Com sempre, hauràs de crear també un script que permeti comprovar que el seu funcionament és correcte.

6.5.3. Creació de paquets

Els paquets serveixen per agrupar sota un mateix nom funcions i procediments. Faciliten la modularització de programes i el seu manteniment.

Els paquets consten de dos parts:

- **Especificació.** Que serveix per declarar els elements de què consta el paquet. En aquesta especificació s'indiquen els procediments, funcions i variables **públiques** del paquet.
- **Cos.** En la qual s'especifica el funcionament del paquet. Consta de la definició dels procediments indicats en l'especificació. A més, es poden declarar i definir variables i procediments **privats**.

Especificació. Sintaxi

```
CREATE [OR REPLACE] PACKAGE nomPaquet  
{IS|AS}  
  
variables, constants, cursors i excepcions públiques  
  
capçalera de procediments y funcions  
  
END nomPaquet;
```

Exemple:

```
CREATE OR REPLACE PACKAGE paquet1 IS  
  
v_cont NUMBER := 0;  
  
PROCEDURE reset_cont(v_nuevo_cont NUMBER);  
  
FUNCTION devolver_cont  
  
RETURN NUMBER;  
  
END paquet1;
```

Cos. Sintaxi

```
CREATE [OR REPLACE] PACKAGE BODY nomPaquet  
  
IS|AS  
  
variables, constants, cursors i excepcions privades  
  
cos dels procediments i funcions  
  
END nomPaquet;
```

Exemple:

```
CREATE OR REPLACE PACKAGE BODY paquet1 IS  
    PROCEDURE reset_cont(v_nuevo_cont NUMBER)
```

```
IS  
BEGIN  
    v_cont:=v_new_cont;  
END reset_cont;  
FUNCTION devolver_cont  
IS  
BEGIN  
    RETURN v_cont;  
END devolver_cont;  
END paquet1
```

Execució

Des de dins del paquet, per utilitzar una altra funció o procediment o variable dins el mateix paquet, només cal invocar-lo pel seu nom.

Si volem utilitzar un objecte d'un paquet, fora d'ell mateix, llavors s'anteposa el nom del paquet a la funció. Per exemple *paquet1.reset_cont (4)* (en l'exemple anterior).

6.5.4. Activitats de funcions i procediments

Exercici proposat 06.05.04a:

Crea una funció PL/SQL anomenada "duplicar", que retorne duplicada la quantitat rebuda com a paràmetre. Comprova el seu funcionament.

Exercici proposat 06.05.04b:

Crea una funció PL/SQL anomenada "factorial", que retorne el factorial d'un número, per exemple $\text{factorial}(5) = 5! = 1 * 2 * 3 * 4 * 5 = 120$.

Exercici proposat 06.05.04c:

Crea un procediment PL/SQL anomenat "mostrarFins", que mostre els números des de l'1 fins al valor passat com a paràmetre, cadascun en una línia.

Exercici proposat 06.05.04d:

Crea un procediment PL/SQL anomenat "mostrarNumeros", que mostre els números des d'un valor inferior fins a un valor superior que s'indicaran com a paràmetres, amb un salt de 1, cadascun en una línia.

Exercici proposat 06.05.04e:

Crea una taula "numeros", amb un únic camp, numèric, anomenat "valor", que actuarà com a clau primària. Després crear procediment PL/SQL anomenat "insertarNumeros", que inserirà en aqueixa taula els números des d'un valor inferior fins a un valor superior que s'indican com a paràmetres.

Exercici proposat 06.05.04f:

Crea una funció anomenada InsertarProducto, que reba com a paràmetre un codi de producte, una descripció i un cost del producte, i realitze una inserció en una (suposada) taula PRODUCTES si el codi de producte (PK) no existeix i que, en cas d'existir, actualitze les dades de descripció i cost. Haurà de retornar el preu de venda al públic, que resulta d'aplicar-li a aqueix preu de cost un marge comercial del 20%.

Exercici proposat 06.05.04g:

Donat aquest PL/SQL:

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

AS

```
    Precioant      NUMBER(5);  
BEGIN  
    SELECT precio_uni INTO precioant  
    FROM productos  
    WHERE cod_producto = codigoprod;  
    IF (precioant * 0.20) > ABS(precioant – nuevoprecio) then  
        UPDATE productos  
        SET precio_uni = nuevoprecio  
        WHERE cod_producto = codigoprod;  
    ELSE  
        DBMS_OUTPUT.PUT_LINE('Error, modificación superior a 20%');  
    END IF;  
EXCEPTION  
    WHEN NO_DATA_FOUND THEN  
        DBMS_OUTPUT.PUT_LINE ('No encontrado el producto ' || codigoprod);  
END modificar_precio_producto;
```

Respon a les següents preguntes:

- a) Es tracta d'una funció o d'un procediment, per què?, caldria canviar perquè fora l'altre?
- b) Quina és la capçalera del procediment?

- c) Què és el precioant?
- d) Què és el nuevoprecio?
- e) Què és el precio_uni?
- f)) Quins són els paràmetres del procediment?
- g) Què és NO_DATA_FOUND?
- h) Quin és el nom del procediment?
- i) On comença el bloc?
- j) Què fa la clàusula INTO?
- k) què fa la condició del IF?
- l) Perquè no té la clàusula DECLARE? Què té en el seu lloc?

Exercici proposat 06.05.04h:

Corregeix els errors de sintaxis en aquesta funció Aquesta funció PL/SQL retorna el número pi (3,141592653589793238462...), calculat mitjançant l'algorisme que va idear John Wallis en 1665:

```
CREATE FUNCTION piWallis(plteraciones number) number
IS
    vCont number
    vRet number
    vCont = 0
    vRet = 1
    loop
        vCont = vCont + 1;
        when vCont > plteraciones exit loop;
        if (vCont % 2) = 0
            vRet := vRet * vCont / (vCont + 1);
        else
            vRet := vRet * (vCont + 1) / vCont;
        endif;
    end loop
    return (2 * vRet);
END;
```

Exercici proposat 06.05.04i:

Introdueix totes les funcions i procediments dels exercicis anteriors en un PACKAGE anomenat FUNCIONS_PROC.

Exercici proposat 06.05.04j:

Executa almenys un procediment i una funció del nou Paquet des d'un bloc anònim.

6.6. Cursors

6.6.1. Definició de cursor

El conjunt de files resultants d'una consulta amb la sentència SELECT, pot estar composta per cap, una o diverses files, depenent de la condició que defineix la consulta.

Per a poder processar individualment des de PL/SQL cada fila d'una consulta que retorne més d'una fila, hem de definir un cursor (que és una àrea de treball de memòria) que conté les dades de les files de la taula consultada per la sentència SELECT, i ens permet accedir a cada registre d'un en un.

6.6.2. Tipus de cursors

Hi ha dos tipus:

- Implícits: Declarats implícitament per a totes les sentències del DML i SELECT de PL/SQL, consultes que retornen una sola fila.
- Explícits: Declarats i nomenats pel programador. Manipulats per sentències específiques en les instruccions executables del bloc. S'usen per a consultes que retornen més d'1 fila.

6.6.2.1. Cursors implícits

Poden procedir de:

- SELECT de una sola fila. Si no torna cap fila o retorna més d'una s'origina un error. El controlarem a la zona EXCEPTION (que estudiarem més endavant), mitjançant els controladors: NO_DATA_FOUND Y TOO_MANY_ROWS.
- Sentències del DML, de més d'una fila.

6.6.2.2. Cursors explícits

Quan tenim diverses files retornades. El seu maneig implica diversos passos, que es detallaran a continuació.

6.6.3 Passos en el maneig de cursors

1. Definir el cursor, especificant la llista de paràmetres amb els seus corresponents tipus de dades i establint la consulta amb la sentència SELECT
2. Obrir el cursor per inicialitzar, sent aquest el moment en què es realitza la consulta.
3. Llegir una fila del cursor, passant les seves dades a les variables locals definides a aquest efecte
4. Repetir el procés fila a fila fins arribar a l'última
5. Tancar el cursor una vegada s'acabe de processar la seua última fila

6.6.3.1. Declaración de Cursor

Sintaxi:

```
CURSOR      nom_cursor      IS  
  
Sentència select ;      /* sense      INTO      */
```

Exemple:

```
DECLARE  
  v_empno      emp.empno%TYPE;  
  v_ename      emp.empno%TYPE;  
  
  CURSOR      emp_cursor      IS  
  SELECT empno, ename  
  FROM emp  
  WHERE deptno = 30;  
  ...
```

6.6.3.2. Obertura del cursor

A l'obrir el cursor s'executa realment la consulta. Si la consulta no retorna cap fila no es produirà cap EXCEPTION. El format és:

```
OPEN      nom_cursor;
```

6.6.3.3. Recuperar dades del cursor

Utilitzarem la sentència FETCH, que té el següent format:

```
FETCH nom_cursor INTO [var1,var2,...] | nom_registre;
```

Aquesta ordre s'encarrega de recuperar la fila actual i introduir els valors en les variables d'eixida. Per tant, haurem d'incloure el mateix nombre de variables que de columnes que obtindrem, així com relacionar posicionalment les variables i columnes.

Generalment, la sentència FETCH es llançarà de manera repetitiva, dins d'un bucle. En cada pas, haurem de comprovar si realment hem obtingut alguna fila, emprant atributs com el %NOTFOUND, que tractarem d'ací a un moment.

6.6.3.4. Tancament del Cursor

Haurem de tancar el cursor després del processament de les files. Això desactiva el cursor i allibera la memòria reservada. Emprarem l'ordre CLOSE:

```
CLOSE nom_cursor
```

No es poden recuperar dades del cursor una vegada tancat. Haurem de tornar a obrir el cursor si fos necessari. Hi ha un màxim nombre de cursors que poden estar oberts alhora a la BD.

6.6.3.5. Exemple bàsic

Així podria ser un exemple bàsic d'obtenció d'informació amb un cursor, emprant una estructura repetitiva WHILE:

```
DECLARE
```

```
CURSOR cursorJocs IS  
  SELECT codi, nom FROM jocs ORDER BY nom;
```

```
v_nom jocs.nom % TYPE;  
v_codi jocs.codi % TYPE;
```

```
BEGIN
```

```
OPEN cursorJocs;  
FETCH cursorJocs INTO v_codi, v_nom;  
WHILE cursorJocs % FOUND LOOP
```

```
  -- Processar informació  
  dbms_output.put_line(v_codi || ' - ' || v_nom);
```

```
  FETCH cursorJocs INTO v_codi, v_nom;  
END LOOP;
```

```
CLOSE cursorJocs;  
END;
```

Exercici proposat 06.06.03a:

Crea un script PL/SQL que empre cursors per a obtenir la següent informació a partir de la nostra base de dades de jocs i plataformes: per a cadascuna de les plataformes, es mostrarà el seu nom i la quantitat de jocs que tenim emmagatzemats. En cas que aqueixa quantitat siga 0 per a alguna plataforma, no es mostrarà el número 0, sinó el text "Cap". Usa un bucle WHILE, com en l'exemple anterior.

6.6.3.6. Atributs dels cursors explícits

Ens permeten obtenir la informació d'estat sobre el cursor:

Atribut	Tipus	Descripció
%ISOPEN	Booleà	TRUE si el cursor està obert
%NOTFOUND	Booleà	TRUE si el FETCH més recent no retorna una fila
%FOUND	Booleà	TRUE si el FETCH més recent retorna una fila; complementa %NOTFOUND
%ROWCOUNT	Numeral	Avalua el nombre total de files retornades

Així, un exemple bàsic de com comprovar si un cursor està obert podria ser el següent:

```
BEGIN  
  IF NOT cursorJocs % ISOPEN THEN  
    DBMS_OUTPUT.PUT_LINE ('El cursor està tancat. Ho obrirem');  
    OPEN cursorJocs;  
  END IF;  
  
  -- Tancament del cursor  
  CLOSE cursorJocs;  
  
END;
```

I el següent seria un exemple de com usar un cursor amb %ROWTYPE:

```
DECLARE
```

```
CURSOR cursorJocs IS
  SELECT * FROM jocs ORDER BY nom;

v_reg_joc jocs % ROWTYPE;

BEGIN
  OPEN cursorJocs;
  FETCH cursorJocs INTO v_reg_joc;
  WHILE cursorJocs % FOUND LOOP

    -- Processar informació
    dbms_output.put_line(v_reg_joc.codi
      || ' - ' || v_reg_joc.nom);

    FETCH cursorJocs INTO v_reg_joc;
  END LOOP;
  CLOSE cursorJocs;
END;
```

Mentre que el següent exemple emprà %ROWCOUNT per a saber la quantitat exacta de files retornades i %NOTFOUND per a determinar quan cal continuar repetint (el que en aquesta ocasió es fa amb un bucle DO):

```
DECLARE
  CURSOR cursorJocs IS
    SELECT codi, nom FROM jocs ORDER BY nom;

  v_nom jocs.nom % TYPE;
  v_codi jocs.codi % TYPE;

BEGIN
  OPEN cursorJocs;
  LOOP
    FETCH cursorJocs INTO v_codi, v_nom;
    EXIT WHEN cursorJocs % ROWCOUNT > 10
      OR cursorJocs % NOTFOUND;
    dbms_output.put_line(v_codi || ' - ' || v_nom);
  END LOOP;
  CLOSE cursorJocs;
END;
```

Exercici proposat 06.06.03b:

Crea un script PL/SQL similar al 06.06.03a (nom de cada plataforma i quantitat de jocs o, si és zero, la paraula "Cap"), però en aquesta ocasió hauràs d'utilitzar un bucle LOOP, juntament amb la clàusula EXIT.

Exercici proposat 06.06.03c:

Crea un procediment anomenat "MostrarJocsIncomplets", que mostre el nom dels jocs dels quals no sabem quant espai ocupa, juntament amb el nom de la plataforma corresponent. No rebrà cap detall com a paràmetre. Ajuda't d'un cursor i un bucle LOOP.

Exercici proposat 06.06.03d:

Crea una variant de l'exercici 06.06.03c: un procediment anomenat "MostrarJocsIncomplets2", que mostre el nom dels jocs dels quals no sabem quant espai ocupa, juntament amb el nom de la plataforma corresponent. No rebrà cap detall com a paràmetre. En aquest cas, hauràs d'emprar un cursor i un bucle WHILE.

6.6.4. Bucles *FOR* en cursors

La sentència FOR té una sintaxi alternativa que es pot aplicar als cursors, per a recórrer-los amb una estructura molt compacta. En ella:

- L'obertura, la recuperació de dades i el tancament són implícits.
- La variable registre es declara implícitament.

La seua sintaxi bàsica és:

```
FOR nom_registre IN nom_cursor LOOP
    instrucció_1;
    instrucció_2;
    ....
END LOOP;
```

I un exemple del seu ús podria ser:

```
DECLARE
CURSOR cursorJocs IS
    SELECT codi, nom FROM jocs ORDER BY nom;

BEGIN
    FOR v_reg_joc IN cursorJocs LOOP
        dbms_output.put_line(v_reg_joc.codi
        || ' - ' || v_reg_joc.nom);
    END LOOP;
END;
```

Exercici proposat 06.06.04a:

Crea una variant dels exercicis 06.06.03c i 03d: hauràs de definir un procediment anomenat "MostrarJocsIncomplets3", que mostre el nom dels jocs dels quals no sapiem quant espai ocupa, juntament amb el nom de la plataforma corresponent. No rebrà cap detall com a paràmetre. En aquest cas, hauràs d'emprar un cursor i un bucle FOR amb la sintaxi específica per a cursors.

Exercici proposat 06.06.04b:

Crea un script PL/SQL que, ajudant-se d'un cursor i una sentència FOR, mostre el nom de cada joc juntament amb l'any de llançament, o la frase "Any desconegut" per a aquells casos en els quals desconexem aqueix detall.

També és possible emprar la sentència FOR per a recórrer una subconsulta, i en aqueix cas no és necessari declarar el cursor, com en el següent exemple:

```
BEGIN
    FOR emp_record IN (SELECT last_name, department_id FROM employees) LOOP
        IF emp_record.department_id = 80 THEN
            ...
        END LOOP; -- tancament implícit
END;
```

Exercici proposat 06.06.04c:

Crea un script PL/SQL que, ajudant-se d'una sentència FOR i una subconsulta (cursor implícit), mostre el nom de cada joc juntament amb l'any de llançament, o la frase "Any desconegut" per a aquells casos en els quals desconexem aqueix detall.

6.6.5. BULK COLLECT com alternativa als cursors

Aquesta clàusula permet realitzar un SELECT INTO, i guardar-ho en una estructura que emmagatzeme més d'una fila. Per tant, mai donarà un error TOO_MANY_ROWS. Per exemple, podem guardar-ho en una taula temporal, usant TABLE OF i ajudant-nos de ROWTYPE.

Bulk Collect serà una alternativa interessant quan veiem que els cursors consumeixen un temps excessiu i requerim més velocitat, encara que siga a cost dels recursos de sistema, ja que bolca el resultat complet d'una vegada (potser interesse usar LIMIT).

Un exemple del seu ús podria ser el següent:

```
DECLARE
    TYPE tipus_jocs IS TABLE OF jocs %ROWTYPE;
    v_regs_jocs tipus_jocs;
```



```
BEGIN
SELECT *
BULK COLLECT INTO v_regs_jocs
FROM jocs
WHERE codiPlataforma = 'cpc'
ORDER BY nom;

dbms_output.put_line('Jocs per a Amstrad CPC');
dbms_output.put_line('-----');
FOR i IN v_regs_jocs.FIRST .. v_regs_jocs.LAST
LOOP
    dbms_output.put_line(v_regs_jocs(i).codi
        || ' - ' || v_regs_jocs(i).nom);
END LOOP;
END;
```

Exercici proposat 06.06.05a:

Crea un script PL/SQL que, ajudant-se d'un BULK COLLECT, mostre el nom de cada joc juntament amb l'any de llançament, o la frase "Any desconegut" per a aquells casos en els quals desconexim aqueix detall.

6.6.6. Activitats de cursors

1. Realitza un cursor sobre la taula Bancs i sucursals, on indiqués el nom del banc i el de cada sucursal. Fes-ho amb loop simple, while i amb FOR.
2. Crea un camp a la taula Compte, que es diga "Rendible", recorre la taula Compte i si l'haver és més gran que el deure actualitzar a S del contrari a N, sumant les rendibles i les no rendibles i traient per la pantalla a la fi . Fes servir ROWID per fer-ho (identificador únic per a cada registre d'una BD).
3. Mira de fer això amb una UPDATE massiva (1 sola update).
4. Utilitza un cursor i un bucle LOOP simple per recuperar i mostrar les dades de tots els productes. Indica a la fi quants productes hi ha.
5. Utilitza el cursor FOR sense declarar-ho.
6. Fes servir ara un while
7. Donades les següents taules:

```
-- tabla para almacenar todos los alumnos de la BD
CREATE TABLE Alumnos
(numMatricula NUMBER PRIMARY KEY,
nombre VARCHAR2 (15) , apellidos VARCHAR2(30),
titulación VARCHAR2 (15), precioMatricula NUMBER);
```

```
-- tabla para los alumnos de informática  
CREATE TABLE AlumnosInf  
(IDMatricula NUMBER PRIMARY KEY,  
nombre_apellidos VARCHAR2(50),  
precio NUMBER);
```

Inserisca les següents dades de prova en la taula ALUMNOS:

numMatricula	nombre	apellidos	titulación	precioMatricula
1	Juan	Alvarez	Administrativo	1000
2	José	Jiménez	Informática	1200
3	Maria	Pérez	Administrativo	1000
4	Elena	Martínez	Informática	1200

Construïska un cursor que inserisca només els alumnes d'informàtica en la taula ALUMNOSINF, tenint en compte l'estructura d'aquesta taula. Així, per exemple, ha de tindre en compte que l'atribut nom_nombre_apellidos resulta de la concatenació dels atributs nom i cognoms. Abans de la inserció de cada tupla en la taula ALUMNOSINF, ha de mostrar per pantalla el nom i el cognom que inserirà.

8. Donades les següents taules:

```
CREATE TABLE Tabla_Departamento (  
  Num_Depart Number(2) PRIMARY KEY,  
  Nombre_Depart VARCHAR2(15) ,  
  Ubicación VARCHAR2(15),  
  Presupuesto NUMBER(10, 2),  
  Media_Salarios NUMBER(10,2),  
  Total_Salarios NUMBER (10,2));
```

```
CREATE TABLE Tabla_Empleado(  
  Num_Empleado Number(4) PRIMARY KEY,  
  Nombre_Empleado VARCHAR(25),  
  Categoría VARCHAR (10) , -- Gerente, Comercial, ...  
  Jefe Number(4),  
  Fecha Contratación DATE,  
  Salario Number(7,2),  
  Comision Number(7,2),  
  Num_Depart NUMBER(2),  
  FOREIGN KEY (Jefe) REFERENCES Tabla_Empleado,  
  FOREIGN KEY (Num_Depart) REFERENCES Tabla_Departamento) ;
```

a) Construïska un bloc que calcule el pressupost del departament per a l'any pròxim. S'emmagatzemarà el mateix en la taula Tabla_Departamento en la columna Pressupost. Cal tindre en compte les següents pujades de sou:

Gerente	+ 20%
Comercial	+ 15%

Els altres empleats que no estiguen en cap de les categories anteriors se'ls pujarà el sou un 10%.

b) Construisca un bloc que actualitze el camp `Total_Salarios` i el camp `Media_Salarios` de la taula `Tabla_Departamento`, sent el total la suma del salari de tots els empleats, igualment amb la mitjana. Per a això:

- Creu un cursor C1, que retorne tots els departaments
- Cree un cursor C2, que retorne el salari i el codi de tots els empleats del seu departament.

9. Creu una taula `NOTES` amb els atributs que estime necessaris i construisca un bloc que inserisca en la taula `NOTES` quatre files per a cada alumne matriculat, aquestes files correspondran a les tres convocatòries ordinàries i l'última per a la convocatòria extraordinària de juny. Abans d'inserir es comprovarà que no estan ja creades. Les files hauran d'inicialitzar-se a nul.

10. Creu un bloc que emmagatzeme en la taula `AUX_ARTICULOS` (ha de crear-se prèviament) un número donat dels articles amb major preu dels existents en la taula `Tabla_Articulos`. El nombre d'articles a emmagatzemar-se ha de ser donat per teclat.

11. Escriga un bloc que recupere tots els proveïdors per països. El resultat ha d'emmagatzemar-se en una nova taula `Taula_Aux` que permeti emmagatzemar dades del tipus: Proveïdor. ONCE — País: Espanya

Utilitze un cursor per a recuperar cada país de la taula `Taula_Proveïdors` i passar aquest país a un cursor que obtinga el nom dels proveïdors en ell. Una vegada que es té el nom del proveïdor i el seu país, ha d'afegir-se a la nova taula en el format especificat.

NOTA: En primer lloc, ha de crear les taules `Taula_Proveïdors` i `Taula_Aux` i inserir les mateixes dades de prova.

12. Crea un cursor que obtinga per pantalla les següents dades de vols del model `AEROPORT`: Nom aeroport d'origen, nom aeroport de destinació, director de cadascun si ho tinguera (si no, que aparega en blanc), any que s'ha realitzat i nombre de vegades que s'efectuen aqueix mateix vol per cada any.

13. Tria 2 dels exercicis (8 a l'11) i canvia'ls usant `BULK COLLECT`

6.7. Control d'excepcions

6.7.1. Contacto con las excepciones en PL/SQL

¿Qué es una excepción? Una excepció és una situació anòmla, típicament arran d'un error en temps d'execució, que requereix un tractament especial.

¿Com sorgeix? Poden tindre lloc per dos motius: produir-se un error durant l'execució d'un script PL/SQL, o bé poden ser provocades explícitament pel programador

¿Com es gestiona? És possible tractar-les amb un gestor en el propi bloc de PL/SQL, o bé es pot propagar al procés pare.

6.7.2. Gestió d'excepcions en PL/SQL

Captura d'una excepció: Quan es produeix una excepció en la secció executable, aquesta es ramifica a la zona de EXCEPTION, on es pot capturar i tractar programant el seu corresponent gestor.

Propagació d'una excepció: Quan es produeix una excepció en un bloc i no hi ha el corresponent gestor per a ella, el bloc acaba amb un error. Podrem gestionar-la en l'entorn de crida d'aquest bloc.

6.7.3. Tipus de excepcions.

Del servidor Oracle. Provocades implícitament.

- Amb nom. Ej: NO_DATA_FOUND
- Sense nom. ORA-12560

Definides per l'usuari. Provocades explícitament

- Amb nom. (es declaren prèviament)
- Sense nom. (RAISE_APPLICATION_ERROR)

6.7.4. Gestors d'excepcions. Sintaxi

```
EXCEPTION
WHEN exception1 [OR exception2...] THEN
    instruccio1;
    instruccio2;
[...
WHEN OTHERS THEN
    instruccio3;
    instruccio4;]
```

Regles per a interrompre excepcions

- EXCEPTION, inicia la secció de gestió d'excepcions.
- Es permeten diversos gestors d'excepcions.

- Només es processa un gestor d'excepcions abans de sortir del bloc.
- WHEN OTHERS és l'última clàusula

6.7.5. Excepcions predefinides per Oracle i amb nom

Excepció	Nº error	Descripció
NO_DATA_FOUND	ORA-01403	SELECT NO TORNA DADES
TOO_MANY_ROWS	ORA-01422	SELECT TORNA MÉS D'1 FILA
INVALID_CURSOR	ORA-01001	OPERACIÓ ILEGAL DE CURSOR
ZERO_DIVIDE	ORA-01476	DIVIDIR PER 0
DUP_VAL_ON_INDEX	ORA-01017	INSERIR UN REGISTRE DUPLICAT

Així, un exemple bàsic, que intente obtindre dades amb SELECT INTO i comprove les possibles excepcions (sense dades o massa dades), podria ser:

```
DECLARE
  v_nom jocs.nom % TYPE;

BEGIN
  SELECT nom
  INTO v_nom
  FROM jocs;

  dbms_output.put_line(v_nom);

EXCEPTION
  WHEN NO_DATA_FOUND THEN
    dbms_output.put_line(' No hi ha dades ');

  WHEN TOO_MANY_ROWS THEN
    dbms_output.put_line(' Massa dades ');

  WHEN OTHERS THEN
    dbms_output.put_line('Error inesperat');

END;
```

En aquest cas, s'obtindria la resposta "Massa dades", perquè hi ha diversos jocs en la nostra base de dades. Per contra, amb el mateix bloc anònim, obtindríem la resposta "No hi ha dades" si la nostra consulta intenta obtindre un joc inexistent:

```
SELECT nom  
INTO v_nom  
FROM jocs  
WHERE codi = '1';
```

Un exemple més avançat, d'excepció predefinida, podria ser:

```
DECLARE  
    // declaració i associació de l'excepció  
    e_restri_integri_emp    EXCEPTION;  
    PRAGMA EXCEPTION_INIT(e_restri_integri_emp, -2292);  
  
    v_deptno                dept.deptno%type:=10;  
BEGIN  
    DELETE FROM dept WHERE deptno = v_deptno;  
    COMMIT;  
EXCEPTION  
    WHEN e_restri_integri_emp THEN  
        dbms_output.put_line('no se puede borrar depto, existen empleados');  
END;
```

Quan es produeix una **excepció sense nom**, hi ha dos pseudovariables que emmagatzemen informació de l'error:

- SQLCODE: Nº de codi del error.
- SQLERRM: Missatge associat al nº de l'error.

Així, es poden gestionar en el gestor WHEN OTHERS:

```
WHEN OTHERS THEN  
IF SQLCODE= - 2292 THEN  
    dbms_output.put_line ('Error d'integritat referencial');  
END IF;
```

Definides per l'usuari, amb nom. Exemple

```
DECLARE
    e_invalid_product    EXCEPTION;
BEGIN
    UPDATE product
    SET descript = p_descripcion_product
    WHERE prodid = p_product_number;
    IF SQL%NOTFOUND THEN
        RAISE e_invalid_product;
    END IF;
    COMMIT;
EXCEPTION
    WHEN e_invalid_product THEN
        DBMS_OUTPUT.PUT_LINE('nº del producte invalidat');
END;
```

Funcions per atrapar excepcions

```
DECLARE

    v_error_code    NUMBER;
    v_error_message  VARCHAR2(255);
BEGIN
    ...
EXCEPTION
    ...
    WHEN OTHERS THEN
        ROLLBACK;
        v_error_code := SQLCODE;
        v_error_message := SQLERRM;
        INSERT INTO errors
        VALUES (v_error_code , v_error_message);
END;
```

6.7.6. RAISE_APPLICATION_ERROR

Procediment que permet emetre missatges d'error definits per l'usuari des de subprogrames emmagatzemats.

Només es pot cridar des d'un subprograma emmagatzemat.

S'usa en la secció executable o a la d'excepcions.

Retorna condicions d'error a l'usuari de forma consistent amb altres errors del servidor Oracle.

Sintaxi:

```
RAISE_APPLICATION_ERROR( num_error, missatge );
```

donde num_error entre -20000 y -20999

Exemple.

EXCEPTION

```
WHEN NO_DATA_FOUND THEN
  RAISE_APPLICATION_ERROR (-20201, 'manager, no es un empleo válido');
...
DELETE FROM emp
WHERE mgr = v_mgr;
IF SQL%NOTFOUND THEN
  RAISE_APPLICATION_ERROR(-20202, 'este no es un jefe válido');
END IF;
```

6.7.7. Propagant excepcions

```
DECLARE
  e_no_rows EXCEPTION;
  e_integrity EXCEPTION;
  PRAGMA EXCEPTION_INIT (e_integrity, -2292)
BEGIN
  FOR c_record IN emp_cursor LOOP
    BEGIN
      SELECT ...
      UPDATE ...
      IF SQL%NOTFOUND THEN
        RAISE e_no_rows;
      END IF;
    END;
  END LOOP;
  EXCEPTION
  WHEN e_integrity THEN ...
  WHEN e_no_rows THEN ...
END;
```

6.7.8. Activitats

1. Realitza el següent exercici 6 funcions, usant excepcions: Crea una funció anomenada PVP que prenga com a argument un codi de producte, una descripció i un cost del producte, i realitze una inserció en una taula PRODUCTES si el codi de producte (PK) no existeix i en cas d'existir actualitze les dades de descripció i cost i retorne el preu de venda al públic, que resulta d'aplicar-li a aqueix preu de cost un marge comercial del 20%).
2. Realitza l'exercici anterior donant un missatge d'error en cas que ja existisca. (Sense fer el update)
3. Realitza l'exercici anterior donant un missatge d'error en cas que no existisca. (Sense fer el insert)
4. Provoca un error perquè salte una excepció i captura-la abans d'eixir de la funció.
5. Nia totes les excepcions possibles dels exercicis anteriors en la mateixa funció.
6. Mostra el nombre de productes, de manera que indique "No hi ha productes" en comptes de "Hi ha 0 productes".
7. Crea un bloc PL/SQL en el qual inserisques un producte amb codi 3, ja existent. Fes el control d'excepcions de manera que un error de clau primària duplicada s'ignore (no s'inserirà el producte, però el bloc PL/SQL s'executarà correctament). Si intentem inserir un producte amb preu negatiu, ha d'aparèixer un error.
8. Escriu un bloc PL/SQL on uses un registre per a emmagatzemar les dades d'un producte. El bloc inserirà aqueix producte, però si el preu no és positiu generarà una excepció de tipus preu_prod_invalid, que hauràs de declarar prèviament, en comptes de realitzar la inserció.
9. Modifica l'exercici anterior, de manera que ara captures l'excepció elevada i produeixes una altra, amb SQLCODE i missatge d'error especifique, usant RAISE_APPLICATION_ERROR
10. Fes una funció que reba com a paràmetre un jugador, un equip, una data i un minut. I ha de fer: Si el jugador no existeix ha de inserir-lo. Si ja existeix ha de comprovar que l'equip donat és el mateix o donarem un error. Farem una consulta per la data, si hi ha més d'un partit o cap donarem un error. Si només hi ha una data, introduïrem un gol d'aquest jugador en el minut donat, si el jugador ha jugat el partit, en cas contrari donarem un error, a més si l'insert falla perquè ja existeix donarem un error. Finalment la funció haurà de retornar un Varchar2 dient: En cas que tot haja anat bé: "TOT HA ANAT BÉ, S'HAN FET X INSERCIONS" (sent X la quantitat de INSERT efectuades). En cas que hi haja hagut algun error: "<el missatge d'error concret. I s'han fet X INSERCIONS (Sent X la quantitat de INSERT efectuades abans que arribés a l'error), haurem de fer ROLLBACK abans de sortir en aquest cas.
11. Crea les excepcions de FK posant-li tant la del pare a fill com viceversa. I després prova que es llancen fent operacions DML entre EQUIPS-JUGADOR.
12. Crea una taula TEMP que tinga codi i descripció. I insereix un JUGADOR i: Si el jugador ja existeix, inserir a la taula TEMP un missatge d'error en la descripció. Si alguna dada de les inserides és de major longitud que l'especificada a la taula, inserir un error en la taula TEMP. Si es produeix qualsevol altre error, inserir a la taula TEMP el codi i el missatge de l'error produït.

6.8. Introducció a PL/SQL dinàmic

Consisteix en

- PL/SQL ofereix la possibilitat d'executar sentències SQL a partir de cadenes de caràcters. Per a això hem d'emprar la instrucció EXECUTE IMMEDIATE.
- Podem obtenir informació sobre nombre de files afectades per la instrucció executada per EXECUTE IMMEDIATE utilitzant SQL %ROWCOUNT.

Exemple

```
DECLARE
    ret NUMBER;
FUNCTION fn_execute RETURN NUMBER IS
    sql_str VARCHAR2(1000);
BEGIN
    sql_str := 'UPDATE DATOS SET NOMBRE = "NUEVO NOMBRE" WHERE CODIGO = 1';
    EXECUTE IMMEDIATE sql_str;
    RETURN SQL%ROWCOUNT;
END fn_execute;

BEGIN
    ret := fn_execute();
    dbms_output.put_line(TO_CHAR(ret));
END;
```

Podem a més parametritzar les nostres consultes a través de variables host. Una variable host és una variable que pertany al programa que s'està executant la sentència SQL dinàmica i que podem assignar a l'interior de la sentència SQL amb la paraula clau USING. Les variables host van precedides de dos punts:

```
DECLARE
    ret NUMBER;
FUNCTION fn_execute (nombre VARCHAR2, codigo NUMBER)
    RETURN NUMBER
IS
    sql_str VARCHAR2(1000);
BEGIN
    sql_str := 'UPDATE DATOS SET NOMBRE = :new_nombre WHERE CODIGO = :codigo';
    EXECUTE IMMEDIATE sql_str USING nombre, codigo;
    RETURN SQL%ROWCOUNT;
END fn_execute ;

BEGIN
    ret := fn_execute('Devjoker',1);
```

```
dbms_output.put_line(TO_CHAR(ret));  
END;
```

Exercicis

- 1- Crea un procediment que donat un nom i una contrasenya et cree un usuari i et done permisos.
- 2- Canvia l'exercici anterior capturant tots els errors possibles que es puguin donar.
- 3- Crea una funció dinàmicament que et demane 5 noms i 5 tipus i et cree una taula de 5 camps, la clau primària t'indicarà quin és amb un paràmetre extra. Tornarà OK.
- 4- Realitza un procediment anomenat Cerca, que et busque en la taula PRODUCTES per qualsevol dels 3 camps que té, poden buscar per un, dos o els tres. Has de fer la cerca dinàmicament per tots aquells que no se li passe a NULL a la funció, donant un error si tots es passen a NULL. El procediment haurà de retornar el resultat de la SELECT per l'eixida estàndard. TOTA la funció ha de ser dinàmica. Fes-la amb un cursor i després una segona versió amb BULK COLLECT.
5. Crea la taula usant SQL dinàmic i EXECUTE IMMEDIATE dins d'un bloc PL/SQL. Comprova que efectivament la taula va ser creada, amb una consulta simple sobre el catalogue de Oracle. Finalment, elimina (DROP) la taula
6. De nou en un bloc PL/SQL, crea una taula T2 amb un camp S alfanumèric, i inserida una fila en aqueixa taula
7. Recupera i imprimeix de nou les dades de tots els productes, ara usant SQL dinàmic.
8. Altera la taula de productes, afegint una restricció c_preu_positiu que comprove que el preu d'un producte és major que zero.

6.9. Triggers

Un trigger (disparador) és un objecte d'una base de dades, associat a una taula, i que s'activa quan es produeix un cert esdeveniment en una taula (per exemple, en el moment de guardar dades). Per tant, es poden emprar per a realitzar comprovacions de valors, per a bolcar dades en una taula auxiliar, etc.

En gestors com MySQL, la sintaxi és relativament senzilla:

```
CREATE TRIGGER nom BEFORE/AFTER INSERT/DELETE/UPDATE ON taula  
FOR EACH ROW operació
```

És a dir:

- El nom s'indica després de CREATE TRIGGER
- A continuació s'especifica si es desitja que s'active abans (BEFORE) o després (AFTER) d'un cert esdeveniment.

- Els esdeveniments habituals són les operacions INSERT, DELETE y UPDATE.
- Després de l'expressió FOREACH ROW s'indica quins passos es desitja donar en cas que s'active aqueix esdeveniment.
- En aqueix cas, ens podem ajudar de pseudoregistres OLD i NEW: usarem OLD.camp i NEW.camp per a accedir a l'antic valor d'un cert camp en cada registre i al seu nou valor després d'aqueixa operació. En concret:
 - En un UPDATE, OLD conté els valors d'abans de la modificació i NEW conté els valors després de la modificació.
 - En un INSERT, NEW conté els nous valors i OLD està buit.
 - En un *DELETE, OLD conté els valors inicials i NEW està buit.

Per exemple, es podria forçar al fet que, cada vegada que s'introdueixen les dades d'un nou joc, el codi de plataforma es convertisca a minúscules, fent:

```
CREATE TRIGGER nouJoc BEFORE INSERT ON jocs
FOR EACH ROW
SET NEW.codiPlataforma = LOWER(NEW.codiPlataforma);
```

En Oracle es complica lleugerament, pels següents detalls:

- Aqueixa "operació" ha d'indicar-se com un bloc anònim, entre BEGIN i END, i potser amb el seu bloc previ de declaració de variables.
- Les assignacions s'hauran de realitzar amb ":=" i sense la paraula "SET".
- Les "pseudovariables" NEW i OLD s'han de precedir per "dos punts" (:NEW, :OLD).

Amb aqueixes consideracions, l'equivalent a l'exemple anterior seria així:

```
CREATE OR REPLACE TRIGGER nouJoc BEFORE INSERT ON jocs
FOR EACH ROW
BEGIN
:NEW.codiPlataforma := LOWER(:NEW.codiPlataforma);
END;
```

Si ara intentem inserir un registre amb un codi de plataforma en majúscules:

```
INSERT INTO jocs VALUES('dstr', 'Death Stranding', NULL, 2019, NULL, 'EPIC');
```

No obtindrem cap missatge d'error, però en sol·licitar veure aqueix registre, podrem observar que el codi s'ha guardat en minúscules:

```
SELECT nom, codiPlataforma FROM jocs WHERE codi = 'dstr';
```

NOM	CODI

Death Stranding

epic

Com a exemple que implique dues taules, es podria emportar compte de cada esborrat creant en primer lloc una taula addicional (que continguera, per exemple, només el codi i el nom):

```
CREATE TABLE jocsEsborrats(  
  codi CHAR(5),  
  nom VARCHAR2(50)  
);
```

I anotar en ella el codi i el nom de cada joc que s'esborre, de la següent forma:

```
CREATE TRIGGER jocEsborrat AFTER DELETE ON jocs  
FOR EACH ROW  
BEGIN  
  INSERT INTO jocsEsborrats VALUES(:OLD.codi, :OLD.nom);  
END;
```

Si ara llancem una ordre d'esborrat:

```
DELETE FROM jocs WHERE codi = 'dstr';
```

Aqueix joc desapareixerà de la taula corresponent, però apareixerà en la nova taula que hem creat:

```
SELECT * FROM jocsEsborrats;
```

```
CODI NOM
```

```
-----  
dstr Death Stranding
```

El fet que es llance un bloc anònim, juntament amb el fet que existeixen variables booleanes INSERTING, UPDATING i DELETING, que permeten saber quin tipus d'operació s'ha realitzat, permeten construir accions molt complexes. Per exemple, podríem crear una taula en la qual anotàrem quan s'afeg alguna dada, es modifica o s'esborra:

```
CREATE OR REPLACE TRIGGER canviEnJocs  
AFTER INSERT OR UPDATE OR DELETE  
ON jocs  
DECLARE  
  accio logJocs.canvi % TYPE;  
BEGIN  
  IF INSERTING THEN  
    accio := 'Insertit';  
  ELSIF UPDATING THEN
```

```
accio := 'Modificat';  
ELSIF DELETING THEN  
    accio := 'Esborrat';  
END IF;  
  
INSERT INTO EVALUATIONS_LOG (dataCanvi, tipusCanvi)  
    VALUES (SYSDATE, accio);  
END;
```

Exercici proposat 06.09a:

Crea un trigger que force al fet que, quan es guarda una nova plataforma, el seu codi s'emmagatzeme en minúscules.

Exercici proposat 06.09b:

Crea un trigger que anote "Sense revisar" en el nom d'un joc, en el cas que no s'haja indicat aquest nom (sigués NULL).

Exercici proposat 06.09c:

Crea una taula de còpia de seguretat dels jocs, i fes que s'anote també en ella cada joc que s'inserisca en la taula principal.

Movido: antes estaba por delante de estructuras de control

BLOC NIATS I ABAST DE LES VARIABLES.

Els blocs de PL / SQL poden ser niats on sigui que una declaració executable sigui permesa.

Un bloc niat arriba a ser una declaració.

Una secció d'excepció pot contenir blocs niats.

Una secció d'excepció pot contenir blocs niats.

L'abast d'un identificador és la regió d'una unitat de programa des de la qual es pot referenciar l'identificador.

Abast Dels Identificadors.

Un identificador és visible a la regió on es pot referenciar l'identificador sense haver de caracteritzar aquest:

Un bloc pot buscar blocs tancats.

Un bloc no pot observar els blocs tancats.

Exemple

Esto son ejemplos de WHILE ¿demasiado avanzados?

Exemple:

DECLARE

V_country_id locations.country_id%TYPE:='CA';

V_location_id locations.location_id%TYPE;

V_city locations.city%TYPE:='Montreal';

V_counter NUMBER := 1;

BEGIN

SELECT MAX (location_id) INTO v_location_id FROM locations

WHERE country_id = vcountry_id;

WHILE v_counter <=3 LOOP

INSERT INTO locations (location_id, city, country_id)

VALUES ((v_location_id + v_counter), v_city, v_country_id);

v_counter:= v_counter+1;

END LOOP;

END;

/

Exemple 2:

DECLARE

v_contador NUMBER := 0;

BEGIN

WHILE v_contador <= 10 LOOP

INSERT INTO prueba (id, contador)

VALUES (v_id, v_contador);

v_contador := v_contador + 1;

```
END LOOP;  
  
COMMIT;  
  
END;
```

Falta fuente como imagen, diapositiva 42

Capacitar Un Identificador

La capacitació pot etiquetar un bloc tancat.

Capacitar un identificador pot usar-se el prefix de etiquetació de blocs.

DECLARE

```
    birthdate DATE;
```

BEGIN

```
    DECLARE
```

```
        birthdate DATE;
```

```
        BEGIN
```

```
            outer.birthdate := TO_DATE (03-AUG-1976, DD-MON-YYYY)
```

```
        END
```

END

VARIABLES DE SUSTITUCIÓ

Una primera forma d' "obtenir" informació per part de l'usuari per després utilitzar-la dins d'un bloc o programa PL/SQL és emprant variables de substitució.

(Exercici) Escriu en PL/SQL un bloc que presente la coneguda frase: Hola Món en majúscules.

Modifiqueu el codi anterior perquè en lloc de Hola Món, l'usuari vegi una línia de sortida amb el seu nom. Exemple: si l'usuari ingressa el nom Jose, la sortida de el programa serà Hola Jose

Solució

DECLARE

```
aux_hola varchar2(50) := 'Hola';
```

```
aux_nom varchar2(50) := '&nom_usuari';
```

```
BEGIN
```

```
DBMS_OUTPUT.put_line(upper(aux_hola)||' '||upper(aux_nom));
```

```
END;
```

El &nom_usuari és una variable de substitució, el seu valor serà sol·licitat a l'usuari quan s'executi el bloc PL/SQL i el valor ingressat serà emprat en les Sentències d'Execució

Fin de la parte movida (antes de estructuras de control)

Movid y pendiente de revisar: 6.2.3.8. Exercicis addicionals

1.- Determinar quins dels següent Identificadors són equivalents en PL/SQL

- Identificador1 NUMBER;
- Identificador_1 NUMBER;
- identificador1 NUMBER;
- IdEntificador_1 NUMBER;
- IDENTIFICADOR1 NUMBER;

2.- Determinar quin dels següents identificadors en PL/SQL és el vàlid:

- Primera variable VARCHAR2(40);
- end BOOLEAN;
- Una_variable VARCHAR2(40);
- Otra-variable VARCHAR2(40);

3.- Funcionaria la següent sentència en PL/SQL ?. En cas negatiu proposar com resoldre-ho.

```
DECLARE
```

```
nombre VARCHAR2(80);
```

```
direccion VARCHAR2(80);
```

```
tipo empleado VARCHAR2(4);
```

```
BEGIN
```

```
SELECT name, address, type
```

```
INTO nombre, direccion, tipo empleado
```

FROM emp;

END;

4- SUBTYPE s'usa per crear un nou tipus basat en un d'existent, explica que tipus tindria la variable total.

DECLARE

SUBTYPE numero1 IS NUMBER;

SUBTYPE cadena IS VARCHAR2(10);

total numero1(6,2);

5.- Indicar del següent joc de declaracions, quines són correctes i quines no, indicant a més per què no són correctes.

DECLARE

cierto BOOLEAN:=FALSE;

id_externo NUMBER(4) NOT NULL;

cosa NUMBER:=2;

producto NUMBER:=2*cosa;

suma NUMBER:=cosa + la_otra;

la_otra NUMBER:=2;

tipo_uno NUMBER(7,2) NOT NULL:=3;

tipo_otro tipo_uno%TYPE;

6.- Suposant que tenim una taula anomenada EMP, en la qual

hi ha tres camps: nombre Varchar2 (40), direccio

Varchar2 (255), telefono NUMBER (10) ... Quins valors podran

prendre les variables que definim en la següent declaració?

DECLARE

valor1 emp%ROWTYPE;

valor2 emp.nombre%TYPE;

valor3 emp.telefono%TYPE;

CURSOR c1 IS

SELECT nombre,direccion FROM emp;

valor4 c1%ROWTYPE;

valor5 emp%ROWTYPE;

7.- Basant-se en l'enunciat anterior, Seria correcta la següent assignació?

valor1:=valor5;

¿l aquesta?

valor4:=valor1;

Raonar el perquè en tots dos casos.

8.-¿ Es aquesta declaració correcta en PL/SQL?

DECLARE

i, j NUMBER;

I si la escrivim així?

DECLARE

i, j NUMBER, NUMBER;

9- Anem a crear el nostre primer bloc anònim. Demaneu 3 variables de substitució a l'usuari, una per guardar el teu nom i dos nombres, el programa ha saludar-te "Hola <el teu nom>" i una segona línia que digui "El resultat és:" juntament amb la suma dels dos nombres. Aconsegueix les dos coses pel dbms_output.

10- Escriu un bloc PL/SQL on declares una variable, inicialitzada a la data actual, que imprimeixes per pantalla. Posteriorment, demana amb una variable de substitució que t'indiquen un booleà, i llavors has de traure amb una variable numèrica l'any actual. Fes-ho de dos formes:

Si el booleà es TRUE: Usant les funcions to_number y to_char.

Si el booleà es FALSE: Usant la funció EXTRACT que es estándar.

EXTRACT(<dades a obtenir> FROM <data>)

6.6. Tipus complexos

6.6.1. Registres

Un RECORD (registre) és una estructura que està format per un o diverses dades de diferents tipus, agrupats formant una unitat lògica.

###Seguir aquí

Un ROWTYPE és un REGISTRE.

La sintaxi seria:

```
TYPE <nom de l'estructura> IS RECORD  
(  
    <camp> <tipus de dada> [NULL | NOT NULL]  
    [,<camp> <tipus de dada> [NULL | NOT NULL]]...  
);
```

Exemple RECORD

Falta fuente como imagen, diapositiva 112

Tipus Complexos. Taula Indexada i Niada

Sintaxi Taula Indexada:

```
TYPE <nom del array> IS TABLE OF <tipus de dada> [NOT NULL] INDEX BY BINARY_INTEGER;
```

Sintaxi Taula Niada:

```
TYPE <nom del array> IS TABLE OF <tipus de dada> [NOT NULL]
```

Característiques. Taula Indexada

El rang de l'índex pot ser negatiu, després el primer valor de l'índex no ha de ser necessàriament 0/1 com en altres llenguatges.

Els tipus de dades poden ser escalars o complexos.

Podem indicar com NOT NULL, en aquest cas no podríem donar-li el valor NULL a cap dels elements de l'ARRAY. Cal indicar que només existiran les posicions de l'ARRAY que li donem valor, mentre no li donem valor les posicions no són NULL per defecte, directament no existeixen, per tant si intentem accedir a aquestes posicions encara que sigui per comprovar si hi ha NULL o no, ens donarà error, l'única manera és usant funcions especials per ARRAYS que veurem a continuació.

Aquest tipus es pot usar per tornar a aplicacions externes, el RECORD no.

Exemple. Taula Indexada

Falta fuente como imagen, diapositiva 115

Taula Indexada usant RECORD

Falta fuente como imagen, diapositiva 116

Funcions Taula.

Falta fuente como imagen, diapositiva 117

FIRST: Et retorna l'índex de la posició més petita de l'array, si l'array està buit retorna NULL.

EXAMPLE: array_n.first RESULTAT: -3

LAST: Et retorna l'índex de la posició més gran de l'array, si l'array està buit retorna NULL.

EXAMPLE: array_n.last RESULTAT: 5

EXISTS (n): Et torna TRUE si existeix en l'array la posició n.

EXAMPLE: array_n.exists (2) RESULTAT: TRUE

EXAMPLE: array_n.exists (4) RESULTAT: FALSE

COUNT: Et retorna el nombre d'elements omplets de l'array.

EXAMPLE: array_n.count RESULTAT: 5

PRIOR (n): Et retorna l'índex de l'element anterior a la posició n en l'array, si hi ha buits no és necessàriament l'anterior numèricament.

EXEMPLE: array_n.prior (3) RESULTAT: 2

NEXT (n): Et retorna l'índex de l'element posterior a la posició n en l'array, si hi ha buits no és necessàriament l'anterior numèricament.

EXEMPLE: array_n.next (3) RESULTAT: 5

DELETE: Esborra tots els elements de l'array.

DELETE (n): Esborra l'element n de l'array.

EXEMPLE: array_n.DELETE (2) RESULTAT: Elimina la posició 2

DELETE (n, m): Esborra des element n a l'element m de l'array.

EXEMPLE: array_n.DELETE (1,3) RESULTAT: Elimina la posició 1,2 i 3

Característiques. Taula Niada

El rang de l'índex comença per 1.

Els tipus de dades poden ser escalars o complexos.

Podem indicar també NOT NULL, en aquest cas no podríem donar-li el valor NULL a cap dels elements de l'ARRAY.

En aquest cas estem obligats a inicialitzar l'array.

Si després volem afegir més elements haurem d'usar la funció EXTEND.

Aquest tipus es pot usar per tornar a aplicacions externes, el RECORD no.

Es pot usar un ROWTYPE d'una taula com ja veurem amb BULK COLLECT

Exemple. Taula Niada

Falta fuente como imagen, diapositiva 119

Funcions. Taules Niades

Les funcions vistes a les taules indexades funcionen perfectament amb aquestes.

A més aquest tipus d'array tenen:

EXTEND: Reserva 1 element més a l'array.

EXTEND (n): Reserva n elements més a l'array.

TRIM: Elimina l'element de la fi de la taula niada, no és que ho esborri, és que deixa d'existir.

EXEMPLE: nested_table.trim;

RESULTAT: Ara la taula niada només té 3 elements.

TRIM (n): Elimina n elements de la fi de la taula niada, des de la posició indicada fins al final.

EXEMPLE: nested_table.trim (2);

RESULTAT: Ara la taula niada només té 1 element.

EXERCICIS

1.- ¿Funcionària el següent codi?. Explicar per què i com

solucionar-ho (si vos ocorren diverses formes d'arreglar-ho, explicar-les totes).

DECLARE

base NUMBER:=100;

BEGIN

FOR dept IN 1..10 LOOP

UPDATE dept SET nom_dept=base+dept

WHERE cod_dept=base+dept;

END LOOP;

END;

2- ¿Què passaria a l'executar el següent codi?, Per què?, Com arreglar-ho? (Si hi ha diverses possibilitats, comentar-les i indicar quina seria més eficient).

```
DECLARE
```

```
    ap1_emp VARCHAR2(40):='Fernandez';
```

```
BEGIN
```

```
    DELETE FROM emp WHERE ap1_emp=ap1_emp;
```

```
    COMMIT;
```

```
END;
```

3.- És correcte el següent codi en PL/SQL ?, Per què? Nota: Ignorar el funcionament del codi (no fa res), cenyir-se exclusivament a la sintaxi vàlida en PL/SQL.

```
FOR ctr IN 1..10 LOOP
```

```
    IF NOT fin THEN
```

```
        INSERT INTO temp
```

```
        VALUES (ctr, 'Hola');
```

```
        COMMIT;
```

```
        factor:=ctr*2;
```

```
    ELSE
```

```
        ctr:=10;
```

```
    END IF;
```

```
END LOOP;
```

4.-Quin resultat provoca l'execució del següent codi PL/SQL?

```
DECLARE
```

```
    variable NUMBER:=1;
```

```
    almacenamos NUMBER:=1;
```

```
BEGIN
```

```
    FOR i IN 5..variable LOOP
```

```
        almacenamos:=almacenamos+i;

    END LOOP;

    INSERT INTO traza VALUES(TO_CHAR(almacenamos));

    COMMIT;

END;
```

5- Què dona aquest bloc?

```
DECLARE

    V_num NUMBER;

BEGIN

    SELECT COUNT(*) INTO V_num

    FROM productos;

    DBMS_OUTPUT.PUT_LINE(V_num);

END;
```

6- Crea una taula MISSATGES amb un sol camp VALOR, de tipus Varchar2 (5). Crea un bloc que inseriu 8 elements en la taula amb valors de l'1 a l'10, excepte el 4 i 5.

7- Torna a fer servir la taula anterior amb les dades ja plenes, i realitza un bucle que mani els 10 primers números. Si el nombre que ve és menor que 4 llavors has de treure un missatge d'error si ja existeix, si el nombre és més gran o igual que 4, has de inserir-lo en la taula si no existeix i actualitzar-lo sumant-li 1 si ja existeix.

8. Crea una taula PRODUCTE (CODPROD, NOMPROD, PREU), usant SQL (no facis servir un bloc PL/SQL).

a) Afegeix un producte a la taula usant una sentència insert values dins d'un bloc PL/SQL.

b) Afegeix un altre producte, ara utilitzant una llista de variables en la sentència insert.

c) Afegeix, ara un altre utilitzant un registre PL/SQL (rowtype).

d) Esborra el producte de codi més xicotet, i incrementa el preu dels altres en un 5%.

9. Obté i mostra totes les dades d'un producte (busca a partir de la clau primària), Si el producte existeix obté totes les dades del mateix en un registre PL/SQL i imprimeix-los, si no hi ha treu un missatge d'error.

10- Fes un bloc PL/SQL, de manera que s'indique que no hi ha productes (si hi ha 0), que hi ha pocs (si hi ha entre 1 i 3) o el nombre exacte de productes existents (si hi ha més de 3).

11- CREATE TABLE ALUMNOS (DNI VARCHAR2(9) NOT NULL, NOMBRE VARCHAR2(60),
TELEFONO NUMBER, CONSTRAINT PK_ALUMNOS PRIMARY KEY (DNI)):

Utilitzant l'script de la taula facilitada. Realitza les següents accions, NO POT FER CAP ACCIÓ QUE NO VINGUI REFLECTIDA EN AQUESTS PUNTS, ni fer servir les paraules SELECT, Varchar2 o NUMBER, però pot crear totes les variables que consideres però no pots inicialitzar-les ni assignar-les valor amb ": =" en cap moment (En acabar l'exercici pots eliminar la taula):

DDL

1. Crea la taula facilitada
2. Crea una seqüència, que comenci per 10 i que vagi de 5 en 5.

DML

En una nova finestra fes un bloc anònim que faci el següent:

3. Realitza una inserció sobre la taula ALUMNES usant aquesta seqüència.
4. Imprimeix per la sortida dbms_output la seqüència, el nom i el telèfon inserit.
5. Modifica la fila de l'alumne inserit canviant el seu nom i el seu telèfon.
6. Imprimeix la direcció i el telèfon anterior i el nou.
7. Esborra la fila inserida i imprimeix pel dbms_output tota la fila esborrada (usant 1 sola variable).
8. Òbviament en cap dels dbms_output pots posar els valors altra vegada manualment, has de fer servir sempre variables.

12- Realitza el mateix exercici anterior, però la insert en comptes de tornar la seqüència, que torni el ROWID, i usa aquest ROWID per fer l'update i el delete.

13. Realitza un bloc anònim de PL/SQL, que donat el nom d'un jugador i un any del Mundial (podeu fer servir una variable de substitució) emmagatzeme en un RECORD el nom del jugador, el lloc habitual, l'equip en què juga, els minuts que ha jugat en aquest mundial, la quantitat de partits i el nombre de gols que ha marcat en ell. No cal que tinguis en compte si el jugador existeix o no.

Quan ho hagi emmagatzemat, imprimeix-lo sencer pel dbms_output per comprovar que està correctament inserit.

14. Realitza un bloc anònim de PL/SQL, que faci el mateix que l'exercici anterior però usant un TABLE de Varchar2 (Array), però aquesta vegada no tinguis en compte l'any de Mundial i ho faci sobre el total de Mundials que el jugador hagi participat.

Quan ho hagi emmagatzemat, imprimeix-lo sencer pel dbms_output per comprovar que està correctament inserit

15. Inventa't un exercici utilitzant un array niat, utilitzant al menys 3 funcions genèriques i al menys 1 vegada EXTEND.

Contenido según la programación:

Bloques de código anónimos

Tipos de datos

Operadores y expresiones

Entrada y salida para la depuración

Estructuras de Control

Palabras reservadas.

Variables del sistema y variables de usuario.

Paquetes

Estructuras de control de flujo. Alternativas. Bucles.

Herramientas para creación de guiones; procedimientos de ejecución. Procedimientos almacenados.

Funciones de usuario.

Variables locales y globales

Estructuras funcionales: procedimientos y funciones

Acceso a la BD con cursores

Excepciones en PL/SQL

Tipos: predefinidas, definidas por el usuario.

Tratamiento de excepciones

Eventos, Disparadores o Triggers

Funciones de tratamiento de cursores. ? APIS para lenguajes externos.

###