# Aide-Mémoire PL/SQL

Pas oublier SET SERVEROUTPUT ON pour activer l’affichage dans la console

**Variable :** nom type [: = initialisation] ;

Déclaration : nom\_variable [CONSTANT] Type [NOT NULL] [{DEFAULT | : =} VALEUR] ;

Substitution (Pas stocke en mémoire et non accessible à partir d’un bloc PL/SQL). &s\_nom

Liaison (Stocké en mémoire pendant la session et peut être utilisé dans plusieurs blocs).

**Traitements conditionnels**

IF condition 1 THEN CASE variable

Instruction 1 WHEN valeur1 THEN

ELSIF condition 2 Instruction 1

Instruction 2 WHEN valeur2 THEN

ELSE Instruction 2

Instruction 3 ELSE

END IF ; Instruction par défaut

END CASE;

**Traitements répétitifs**

LOOP FOR variable IN inf .. sup LOOP WHILE condition LOOP

Instruction Instruction Instruction

EXIT WHEN condition END LOOP; END LOOP;

END LOOP;

**Les structures**

Déclarer la structure

TYPE nomType IS RECORD (variable1, variable2); !!! variable = nom type

maVariable nomType ;

**Les tableaux**

Déclarer le tableau

TYPE nomTableau IS TABLE OF type valeur NOT NULL INDEX BY BYNARY\_INTEGER;

Ex:

-- Déclare une table

type salarié is record (nom varchar2(50), fonction varchar(20));

type tableau\_salarié is table of salarié not null index by binary\_integer;

tab\_salarié tableau\_salarié;

- Utilisation tab\_salarié(1).nom := ‘valeur’ ;

**Tableaux pré-dimensionnés**

TYPE <nom\_tableau> IS VARRAY(taille) OF <type\_valeur> [not null] index by {BINARY\_INTEGER};

Actions possibles sur les tableaux : EXISTS(n), COUNT, FIRST, LAST, PRIOR, NEXT(n), DELETE(n)

**Les variables basées**

De type colonne : Attribut%TYPE variable nomTable.nomColonne%TYPE variable

De type ligne : Attribut%ROWTYPE variable nomTable%ROWTYPE variable.nomColonne

**Le Select**

SELECT INTO récupère le résultat d’un select dans une variable  !!! Doit retourner un seul résultat

BULK COLLECT récupère un ensemble de lignes en une seule fois

**Mise à jour des tables**

INSERT: INSERT INTO table VALUES (,,,,) RETURNING col1 , col2 ,.. INTO var1, var2, ...;

Returning: renvoie les valeurs des champs des enregistrements mis à jours(insert, update, delete)

Il ne reste plus qu’à afficher les variables

UPDATE commandes set row = var\_commande where numero\_cde = var\_commande.numero\_cde;

Permet de faire une lise à jour de l’ensemble de la ligne sur la table.

**Curseurs**

Tous les curseurs ont des attributs que l’utilisateur peut utiliser

%ROWCOUNT : nombre de lignes traitées par le curseur

%FOUND : vrai si au moins une ligne a été traitée par la requête ou le dernier fetch

%NOTFOUND : vrai si aucune ligne n’a été traitée par la requête ou le dernier fetch

%ISOPEN : vrai si le curseur est ouvert (utile seulement pour les curseurs explicites)

Declare CURSOR nomCurseur IS requête sql FOR UPDATE [ OF col1]; OF col1 bloquera que la col1

Explicites : declare le curseur associer à une requête, ouverture dans le corps (OPEN), Défilement (FETCH), fermeture (CLOSE) ou voir boucle for

Implicites : les curseurs implicites sont tous nommés SQL : nb\_lignes := SQL%ROWCOUNT ;

Ex: OPEN emp\_cur ;

LOOP

FETCH emp\_cur INTO ligne ;

EXIT WHEN emp\_cur%NOTFOUND ;

END LOOP ;

CLOSE emp\_cur ;

Avec une boucle for : pas besoin d’ouvrir ni de fermer le curseur

For nom\_enregistrement IN nom\_curseur  
loop  
 commandes pl/sql  
end loop

Curseur paramétré : CURSOR nom (les paramètres) IS requête FOR UPDATE ;

**Procédures Stockées**

Procédure un ensemble de code PL/SQL

Un paquetage est le regroupement de plusieurs procédures et fonctions dans un objet distinct

CREATE OR REPLACE PROCEDURE nomProcédure (paramètres IN type) IS

Declaration des variable local

Begin

Instruction

END;

Oracle permet le passage de paramètres par référence. Il existe trois types de passage de paramètres :  
IN : passage par valeur OUT : aucune valeur passée, sert de valeur de retour

IN OUT : passage de paramètre par référence. Par défaut, le passage de paramètre se fait de type IN.

Appel de la procédure par son nom

**Fonction**

Une fonction va retourner une valeur

CREATE OR REPLACE FUNCTION nomFonction(paramètres) RETURN type IS

Déclaration des variables locales

BEGIN

Instructions

END ;

**Gestion des exceptions système et personnel**

Pour les personnels Déclarez l’exception et la déclencher avec RAISE monException ;

EXCEPTION

WHEN monException THEN

DBMS\_OUTPUT . PUT\_LINE ( 'Mon message d’’erreur‘) ;

WHEN OTHERS THEN

DBMS\_OUTPUT . PUT\_LINE ( 'SQLCODE = ' || SQLCODE ) ;

DBMS\_OUTPUT . PUT\_LINE ( 'SQLERRM = ' || SQLERRM ) ;

Pour tester la contrainte de suppression de clé étrangère

Déclaration

fk\_suppression\_client exception;

pragma exception\_init (fk\_suppression\_client, -2292);

**Les packages**

Un package est un ensemble de sous-programmes et de variables formé par :

* Une spécification : déclaration de variables et de sous-programmes
* Un corps : implémentation des sous-programmes

Entête :

CREATE OR REPLACE PACKAGE nomPackage IS

Declaration des variables et type , curseurs , procédures et fonctions

END nomPackage;

Corps:

CREATE OR REPLACE PACKAGE BODY nomPackage IS

Implementation

END nomPackage;

**Déclencheur / Triggers**

Pas de commit ou rollback dans un trigger, il ne possède pas d’argument, définit une action qui doit se déclencher lors d’un évènement sur la base de données (avant ou après)

CREATE [ OR REPLACE ] TRIGGER nomTrigger

{ BEFORE | AFTER | INSTEAD OF } insert/update/delete [ or <nomevenement2> ] [ CLAUSE\_REFERENCING] [ WHEN CONDITION ] [ FOR EACH ROW ]

[ DECLARE . . . ]

BEGIN

. . .

[ EXCEPTION . . . ]

END [ <nom\_trigger> ] ;

:OLD.nom\_attribut (permet d’accéder à la valeur avant modification)

:NEW.nom\_attribut (permet d’accéder à la valeur après modification)

INSERT : Pas d’accès à l’élément OLD (n’existe pas)

UPDATE : Accès possible à l’élément OLD et NEW

DELETE : Pas d’accès à l’élément NEW (n’existe plus)

Utilisable uniquement avec un déclencheur de niveau enregistrement

/\*\*\*\*\*\*\*\*\*\* Predefined exceptions \*\*\*\*\*\*\*\*\*\*/

CURSOR\_ALREADY\_OPEN exception;

pragma EXCEPTION\_INIT(CURSOR\_ALREADY\_OPEN, '-6511');

DUP\_VAL\_ON\_INDEX exception;

pragma EXCEPTION\_INIT(DUP\_VAL\_ON\_INDEX, '-0001');

TIMEOUT\_ON\_RESOURCE exception;

pragma EXCEPTION\_INIT(TIMEOUT\_ON\_RESOURCE, '-0051');

INVALID\_CURSOR exception;

pragma EXCEPTION\_INIT(INVALID\_CURSOR, '-1001');

NOT\_LOGGED\_ON exception;

pragma EXCEPTION\_INIT(NOT\_LOGGED\_ON, '-1012');

LOGIN\_DENIED exception;

pragma EXCEPTION\_INIT(LOGIN\_DENIED, '-1017');

NO\_DATA\_FOUND exception;

pragma EXCEPTION\_INIT(NO\_DATA\_FOUND, 100);

ZERO\_DIVIDE exception;

pragma EXCEPTION\_INIT(ZERO\_DIVIDE, '-1476');

INVALID\_NUMBER exception;

pragma EXCEPTION\_INIT(INVALID\_NUMBER, '-1722');

TOO\_MANY\_ROWS exception;

pragma EXCEPTION\_INIT(TOO\_MANY\_ROWS, '-1422');

STORAGE\_ERROR exception;

pragma EXCEPTION\_INIT(STORAGE\_ERROR, '-6500');

PROGRAM\_ERROR exception;

pragma EXCEPTION\_INIT(PROGRAM\_ERROR, '-6501');

VALUE\_ERROR exception;

pragma EXCEPTION\_INIT(VALUE\_ERROR, '-6502');

ACCESS\_INTO\_NULL exception;

pragma EXCEPTION\_INIT(ACCESS\_INTO\_NULL, '-6530');

COLLECTION\_IS\_NULL exception;

pragma EXCEPTION\_INIT(COLLECTION\_IS\_NULL , '-6531');

SUBSCRIPT\_OUTSIDE\_LIMIT exception;

pragma EXCEPTION\_INIT(SUBSCRIPT\_OUTSIDE\_LIMIT,'-6532');

SUBSCRIPT\_BEYOND\_COUNT exception;

pragma EXCEPTION\_INIT(SUBSCRIPT\_BEYOND\_COUNT ,'-6533');