Le langage PL/SQL
2 - Compléments

Bernard ESPINASSE
Professeur à Aix-Marseille Université (AMU)
Ecole Polytechnique Universitaire de Marseille

Septembre 2015

1. Rappels de PL/SQL
2. Les Exceptions
3. Les Procédures et Fonctions
4. Les Packages
5. Les Triggers
6. Différences PL/SQL (Oracle) Vs PL/pgSQL (PostgreSQL)

Plan

Principales sources du cours

Documents :

- D. Gonzalez, Introduction à PL-pgSQL, Université Lille 3 – Charles de Gaulle.
- A. Meslé, Introduction au PL/SQL Oracle,
- D. Roegel, Le langage procédural PL/SQL, IUT Nancy 2

Présentations :

- Cours de Anne Vilnat, IUT d'Orsay.
- Cours de Laurent d'Orazio, LP TOSPI, IUT Montluçon, Université Blaise Pascal, Clermont Ferrant
- Cours de Richard Grin, Université de Nice Sophia-Antipolis
- Cours de Robert Laurini, Université de Lyon, …
- Cours de J. Razik, Université de Toulon et du Var
- …

1. Rappels PL/SQL

- **Principales caractéristiques de PL/SQL**
- **Normalisation du langage**

Pourquoi PL/SQL ?

- **SQL est un langage non procédural**
- Le développement d'application autour d'une BDR nécessite d'utiliser :
  - des **variables**
  - des **structures de contrôle** de la programmation (boucles et alternatives)

⇨Besoin d'un **langage procédural** pour lier plusieurs requêtes SQL avec des variables et dans les structures de contrôle habituelles = **L4G (langage de 4ième Génération) :**

D'où **PL/SQL** (Acronyme : ProceduraL SQL) : langage de programmation procédural et structuré pour développer des applications autour de bases de donnes relationnelles (SQL)

Normalisation du langage

- **PL/SQL** est un langage propriétaire de **Oracle**

Pas de véritable standard, la plupart des SGBD relationnels propose des **L4G** (langage de 4ième génération) spécifiques, semblables à PL/SQL :

- **PostgreSQL** propose **PL/pgSQL** très proche de PL/SQL et **PL/pgPSM**
- **MySQL** et **Mimer SQL** proposent un langage analogue dans le principe mais plus limité : **SQL/PSM (**de la norme SQL2003),
- **IBM DB2** propose un dérivé de PL/SQL : **SQL-PL**
- **Microsoft/SQL server** et **Sybase** propose **Transact-SQL (T-SQL)** développé par à l'origine par Sybase
- **…**

On a choisi ici de présenter dans ce cours le langage PL/SQL d'Oracle. Nous verrons à la fin du cours ses différences avec PL/pgSQL de PostgreSQL.

Principales caractéristiques de PL/SQL

- **PL/SQL = Extension de SQL** : des requêtes SQL cohabitent avec les structures de contrôle habituelles de la programmation structurée (blocs, alternatives, boucles)
- La syntaxe ressemble au langage **Ada**
- Un programme est constitué de **variables**, **procédures** et **fonctions**
- PL/SQL permet :
  - l'utilisation de **variables permettant l'échange d'information** entre les requêtes SQL et le reste du programme : ces variables sont de type simple et structuré dynamique (%TYPE, %ROWTYPE, etc)
  - des traitements plus complexes, notamment pour la gestion des cas particuliers et des erreurs (traitement des **exceptions**),
  - l'utilisation de **librairies standards prédéfinies** (supplied PLSQL packages, comme les RDBMS_xxx)
  - un **paramétrage** et la création **d'ordres SQL dynamiques**

Utilisation de PL/SQL

- **Le PL/SQL peut être utilisé sous 3 formes :**
  - un **bloc de code** exécuté comme **une commande SQL**, via un interpréteur standard (SQL+ ou iSQL*PLus)
  - un **fichier de commande PL/SQL**
  - un **programme stocké** (procédure, fonction, package ou trigger)
- **Ainsi PL/SQL peut être utilisé :**
  - pour l'écriture de **procédures stockées** et des **triggers** (Oracle accepte aussi le langage Java)
  - pour l'écriture de **fonctions utilisateurs** qui peuvent être utilisées dans les requêtes SQL (en plus des fonctions prédéfinies)
  - dans des **outils Oracle**, Forms et Report en particulier

Structure d'un programme PL/SQL : les blocs

- **Un programme est structuré en blocs d'instructions de 3 types :**
  - procédures anonymes
  - procédures nommées
  - fonctions nommées
- **Structure d'un bloc :**

[DECLARE]
- définitions de variables, constantes, exceptions, curseurs
BEGIN
- les instructions à exécuter (ordres SQL, instructions PL/SQL, structures de contrôles)
[EXCEPTIONS]
- la récupération des erreurs (traitement des exceptions)
END ;

- **Remarques :**
  - Un bloc peut contenir d'autres blocs
  - Seuls begin et end sont obligatoires
  - Les blocs comme les instructions se termine par un « ; »

2. Exceptions

- **Définition**
- **Gestion des exceptions**
- **Déclaration d'exceptions**
- **Exceptions prédéfinies**

Definition et gestion d'une exception

Définition : une exception est un avertissement ou une **erreur** rencontré(e) lors de l'exécution

- **Types d'exceptions :**
  - Erreur interne au SGBD (erreur système)
    - Ex : espace mémoire insuffisant
    - Oracle : SQLCODE !=0
  - Erreur causée par le programme utilisateur
    - Ex : Lors de l'affichage de tous les articles, si l'un dépasse 10000 euros, le dire et arrêter l'affichage.

Définition et gestion d'une exception utilisateur

Traitement d'une erreur utilisateur survenue dans un bloc PL/SQL :

- Définir et donner un nom à chaque erreur
- Associer le nom de l'erreur à la section EXCEPTION dans la partie DECLARE
- Associer et définir le traitement spécifique à effectuer pour l'erreur
- Exemples de traitements :
  - Notification à l'utilisateur
  - Annulation de l'opération
  - …

Déclaration d'exceptions (1)

- **Syntaxe :**

```
DECLARE
    ...
    nom_exception EXCEPTION ;
    ...
BEGIN
    …
    IF (anomalie) THEN RAISE nom_exception;
    ...
EXCEPTION
    WHEN nom_erreur THEN (traitement_1)
    [WHEN OTHERS THEN traitement_N]
END;
```

- **Remarque :**

  Sortie du bloc après exécution du traitement.

Exception utilisateur : exemple 1

Ex 1 : Récupérer les employés avec leur numéro, leur salaire et leur commission.

```
DECLARE
   pas_comm    EXCEPTION ;
   salaire     emp.sal%TYPE ;
   commi       emp.comm%TYPE ;
   numero      emp.empno%TYPE;
BEGIN
   SELECT sal, comm, empno INTO salaire, commi, numero FROM emp
       where empno := :num_emp ;
      /* num_emp fait réf. à une var. extérieure au bloc PL/SQL */
   IF commi = 0 or commi IS null THEN RAISE pas_comm ;
   ELSE
     /* ... traitement ... */ END IF ;
EXCEPTION
    WHEN pas_comm THEN
       INSERT INTO resultat VALUES (numero, salaire, 'pas de commission');
END ;
```


Exception utilisateur : exemple 2

Ex 2 : Afficher « prix élevé » si le prix d'un article dépasse à 10000 €

```
DECLARE
   CURSOR c IS SELECT * FROM produits;
   depassement EXCEPTION;
BEGIN
    FOR e IN c LOOP
        IF e.prix > 10000 THEN RAISE depassement;
        END IF;
    END LOOP;
EXCEPTION
        WHEN depassement THEN dbms_output.put_line('prix élevé');
END;
```


Exception utilisateur : exemple 3

Ex 3 :

```
DECLARE
   cpt NUMBER := 0;
   monException EXCEPTION;

BEGIN
    ...
    IF cpt < 0 THEN
        RAISE monException;
    END IF;
    ...
EXCEPTION
    WHEN monException THEN
       DBMS OUTPUT.PUT LINE('cpt ne doit pas être négatif');
    WHEN OTHERS THEN
       DBMS OUTPUT.PUT LINE('Je ne connais pas cette erreur');
END;
```

Traitement des erreurs système ORACLE (1)

Ex 3 : Afficher « prix élevé » si le prix d'un article dépasse à 10000 €

```
DECLARE
    ...
    nom_erreur EXCEPTION ;
    PRAGMA EXCEPTION_INIT(nom_erreur, code_erreur);
BEGIN
    ...
    dès que l'erreur Oracle est rencontrée, passage automatique à
    la section EXCEPTION pour réaliser le traitement approprié
    ...
EXCEPTION
    WHEN nom_erreur THEN (traitement) ;
    [WHEN OTHERS THEN (traitement) ;] ;
```

Remarque :

- Sortie du bloc après exécution du traitement
- Il existe des erreurs prédéfinies → pas besoin de les déclarer, ni de les associer à un code erreur

Traitement des erreurs système ORACLE (2)

Ex 1 :

```
DECLARE
    ...
    monException EXCEPTION
    PRAGMA EXCEPTION INIT (monException, -2400)
BEGIN
...
EXCEPTION
     WHEN monException THEN...
END;
```


Erreurs prédéfinies

- ZERO_DIVIDE : Division par 0
- INVALID_CURSOR : Tentative d'accès à un curseur non ouvert
- INVALID_NUMBER : Utilisation d'un type non numérique dans un contexte où un nombre est requis
- NO_DATA_FOUND : SELECT... INTO ne retourne aucun résultat
- NOT_LOGGED_ON : Tentative d'exécution d'opération SQL sans être connecté à Oracle
- STORAGE_ERROR : Erreur de stockage
- VALUE_ERROR : Erreur de conversion arithmétique, troncature, contrainte de taille, etc.
- CURSOR ALREADY OPEN : tentative d'ouvrir un curseur déjà ouvert
- LOGON DENIED : mauvais login/password lors de la connexion
- ROWTYPE MISMATCH : types de paramètre incompatibles
- TOO MANY ROWS : trop de lignes renvoyées par un SELECT... INTO
- …

3. Les procédures et fonctions dans PL/SQL

- **Définition**
- **Syntaxe de déclaration**
- **Exemples**

Définition d'une procédure en PL/SQL

Une **procédure** est simplement un **programme (code) PL/SQL** :

- **nommé** ou **non**
- **compilé**
- **stocké** dans la base
- **appelable** par les couches supérieures depuis :
  - un BlocPL/SQL
  - SQL*Plus

- Une **procédure anonyme** PL/SQL :
  - est un bloc « DECLARE – BEGIN – END »
  - qui peut être exécuté directement dans SQL*PLUS en tapant sa définition

- Une **procédure nommée** PL/SQL :
  - Stockée, on peut réutiliser son code en l'appelant par son nom

Déclaration d'une procédure en PL/SQL (1)

- **Syntaxe générale** :

```
CREATE [OR REPLACE] PROCEDURE [nom_procédure]
[(<liste de paramètres>)] AS I IS
   [DECLARE -- Déclarations des variables]
BEGIN
  -- Corps procédure (code PL/SQL)
[ EXCEPTION
   … ]
  -- Gestion des exceptions
END [nom_procédure];
```

- **Remarques :**
  - Pas de DECLARE : les variables sont déclarées entre IS et BEGIN
  - Si la procédure ne nécessite aucune déclaration, le code est précédé de « IS BEGIN »
  - Il peut y avoir un ou plusieurs paramètres de passage
  - Les procédures peuvent être utilisées dans d'autres procédures ou fonctions ou dans des blocs PL/SQL anonymes

Déclaration d'une procédure en PL/SQL (2)

- **Déclaration des paramètres** :

```
nom paramètre [IN   I
               OUT [NOCOPY]  I
               IN OUT [NOCOPY]] type paramètre
               [ :=  I DEFAULT expression ]
```

- **Avec :**
  - **Type_paramètre** : un type PL/SQL
  - **IN** : paramètre en entrée, non modifié par la procédure
  - **OUT** : paramètre en sortie, peut être modifié par la procédure, transmis au programme appelant
  - **IN OUT** : à la fois en entrée et en sortie
  - **par défaut** : **IN**
  - **NOCOPY** : pour passer des références et non des valeurs (mais le compilateur décide!)

Compilation et appel de procédure

- **Soit la procédure déclarée :**

```
CREATE PROCEDURE supprimer (num CHAR) IS
   BEGIN
     DELETE ... WHERE ... num ... ;
     ...
   END;
```

Paramètre : num

- **Compilation : Sous SQL*PLUS,** il faut taper une dernière ligne contenant « **/** » pour compiler la procédure

- **Appel sous SQL*Plus :**

```
SQL> EXECUTE supprimer('W');
```

- **Appel depuis PL/SQL**

```
DECLARE x char;
BEGIN . . . supprimer('W'); . . . END;
```

Exemple de procédure (1)

- **Procédure avec paramètres :**

```
CREATE PROCEDURE cree_client (p_nom VARCHAR, p_ville
VARCHAR) IS
  BEGIN
    INSERT INTO clients (no_cli, nom_cli, ville_cli)
       VALUES (seq_noclient.NEXTVAL, p_nom, p_ville);
    COMMIT ;
  END ;
```


Exemple de procédure (2)

Ex 2 : On cherche les réalisateurs qui ont joué dans un certain nombre de leur film :

```
CREATE PROCEDURE realActeursProc (nbFilms NUMBER) IS
      nbRealAct NUMBER(5);
      singulierException EXCEPTION;
BEGIN
      SELECT COUNT(distinct A.numIndividu) INTO nbRealAct
         FROM Film F, Acteur A
           WHERE A.numIndividu = realisateur
            AND F.numFilm=A.numFilm;
      IF nbRealAct > nbFilms THEN
           DBMS OUTPUT.PUT LINE(nbRealActII' réalisateurs ont joué
                                          dans plus de 'IInbFilmsII'de leurs films');
           ELSE DBMS OUTPUT.PUT LINE('Aucun réalisateur n'a joué
                                          dans plus de 'IInbFilmsII'de ses films');
      END IF;
END;
```


Définition d'une fonction en PL/SQL

Une fonction est une procédure nommée retournant une valeur

- **Syntaxe** :

```
CREATE [OR REPLACE] FUNCTION nom_fonction
[(<liste de paramètres>)]
   RETURN <Type de retour>  AS I IS
[ DECLARE -- Déclarations des variables]
BEGIN   -- Corps fonction (code PL/SQL)
    …
    RETURN valeurRetour
    …
[ EXCEPTION … ]  -- Gestion des exceptions
END [nom_fonction];
```

- **Remarques :**
  - les fonctions peuvent aussi être utilisées dans les requêtes SQL, dans d'autres procédures ou fonctions ou dans des blocs PL/SQL anonymes
- **Compilation :** sous SQL*PLUS, il faut taper une dernière ligne contenant « **/** » pour compiler la fonction.

Exemple de fonctions en PL/SQL (1)

Ex 1 : la fonction factorielle « fact » (récursivité) :

```
CREATE FUNCTION fact (n INTEGER)
     RETURN INTEGER IS
BEGIN
    IF n = 0 THEN  RETURN 1;
    ELSE
         RETURN n * fact(n – 1);
    END IF;
END;
```

Ex 2 : la fonction « euro_to_fr » :

```
CREATE or REPLACE
FUNCTION euro_to_fr(somme IN number)
   RETURN number IS
     taux constant number := 6.55957;
BEGIN
      RETURN somme * taux;
END;
```

Exemple de fonctions en PL/SQL (2)

Ex 3 : Utilisation de la fonction « euro_to_fr » dans un bloc anonyme :

```
DECLARE
  CURSOR c (p_dept integer) IS
    SELECT dept, nome, sal from emp
      WHERE dept = p_dept;
BEGIN
  FOR employe IN c(10) LOOP
    dbms_output.put_line(employe.nome
      ll ' gagne '
      ll euro_to_fr(employe.sal)
      ll ' francs');
  END LOOP;
END;
```

Utilisation de la fonction « euro_to_fr » dans une requête SQL :

```
SELECT nome, sal, euro_to_fr(sal) FROM emp;
```


Exemple de fonctions en PL/SQL (3)

Ex 4 : On cherche les réalisateurs qui ont joué dans plus de « nbFilms » de leurs films ...

```
CREATE FONCTION nbRealActeurFonc (nbFilms NUMBER)
RETURN NUMBER IS
   nbRealAct NUMBER(5) := 0 ;

BEGIN
   SELECT COUNT(distinct A.numIndividu) INTO nbRealAct
      FROM Film F, Acteur A
        WHERE A.numIndividu = realisateur
            AND F.numFilm = A.numFilm;
   RETURN nbRealAct;
END;
```


Utilisation des procédures et fonctions en PL/SQL

- **Procédures et fonctions :**

  peuvent être utilisées dans :

  - d'autres **procédures**
  - d'autres **fonctions** ou
  - dans des **blocs PL/SQL anonymes**

- **Fonctions :**

  - peuvent aussi être utilisées dans les **requêtes SQL**

Utilisation des procédures et fonctions en PL/SQL

Passage de paramètres :

- Dans la définition d'une **procédure** on indique le **type de passage** que l'on veut pour les paramètres :
  - **IN** pour le passage **par valeur**
  - **IN OUT** pour le passage **par référence**
  - **OUT** pour le passage **par référence** mais pour un **paramètre** dont la valeur n'est **pas utilisée en entrée**

- Pour les **fonctions**, seul le passage **par valeur** (**IN**) est autorisé

4. Les packages dans PL/SQL

- **Définition**
- **Déclaration**
- **Exemples**

Définition d'un package dans PL/SQL

Un package est un **module de programmes** incluant ***procédures*** et / ou ***fonctions*** **fonctionnellement dépendantes**.

- **Un package est composé de 2 parties :**
  - la **spécification** :
    - introduite par **CREATE PACKAGE**
    - liste les entêtes de procédures et fonctions contenues dans le package,
  - le **corps du package** :
    - introduit par **CREATE PACKAGE BODY**
    - contient le code effectif des procédures et fonctions déclarées précédemment.

Déclaration d'un package dans PL/SQL

Syntaxe :

```
CREATE PACKAGE <nomPaquetage> AS
    PROCEDURE <nomProcédure1> (...);
    PROCEDURE <nomProcédure2> (...);
    FUNCTION <nomFonction> (...) type;
END;

CREATE PACKAGE BODY <nomPaquetage> AS
    PROCEDURE <nomProcédure1> (...) BEGIN ... END;
    PROCEDURE <nomProcédure2> (...) BEGIN ... END;
    FUNCTION <nomFonction> (...) BEGIN ... END;
END;
```


Exemple de package dans PL/SQL

```
SQL> CREATE PACKAGE clients AS -- spécifications du package
PROCEDURE insere_client (no INTEGER, nom VARCHAR2, ...);
PROCEDURE supprime_client (no INTEGER);
...
END;
/

SQL> CREATE PACKAGE BODY clients AS -- le corps du package
PROCEDURE insere_client (no INTEGER, nom VARCHAR2, ...) IS BEGIN...
INSERT INTO clients VALUES (no, nom, ...); END;

PROCEDURE supprime_client (no INTEGER) IS
BEGIN
  DELETE FROM clients WHERE no_cli = no;
END;
...
END; -- du package
/
```

5. Les Triggers dans PL/SQL

- **Définition**
- **Déclaration**
- **Exemples**

Définition d'un Trigger dans PL/SQL

Un **trigger** est un morceau de **code PL/SQL** :
- stocké dans la base,
- déclenché lors de l'occurrence d'un événement particulier.

Syntaxe :

```
CREATE [or REPLACE] TRIGGER <nom>
{BEFORE I AFTER I INSTEAD OF } listeEvénements
ON <table>
[FOR EACH ROW]
[WHEN (...)]
        -- bloc PL/SQL
```

Avec
- listeEvénements : liste d'événements séparés par une virgule DELETE, INSERT, ou UPDATE
- Si UPDATE on peut préciser les attributs concernés (UPDATE OF listeAttributs).

Définition d'un Trigger dans PL/SQL

Ainsi les Triggers :

- permettent de synchroniser des opérations entre plusieurs tables
- peuvent être utilisés pour implémenter certaines règles de gestion (souvent les contraintes remplissent plus efficacement ce rôle)
- sont généralement déclenchés par la modification du contenu d'une table
- les ordres du LDD (CREATE, ALTER, DROP, …) et de gestion de transactions (COMMIT, SAVEPOINT,…) sont interdits dans les Triggers.

Granularité d'un Trigger : niveau ligne ou table

Le traitement spécifié dans un trigger peut se faire :

- pour **chaque ligne** concernée par l'événement
  => trigger de **niveau ligne** (**si FOR EACH ROW**)
  - pour le **WHEN condition, si condition vérifiée (vrai)** alors Trigger **déclenché pour chaque ligne**

- une seule fois pour l'ensemble des lignes concernées par l'événement :
  => trigger de **niveau table : si pas FOR EACH ROW**

Déclenchement du Trigger / Evénement

Quand le Trigger est déclenché ?

BEFORE I AFTER

- **trigger de niveau table** : déclenché avant ou après l'événement
- **trigger de niveau ligne** : exécuté avant ou après la modification de CHAQUE ligne concernée

INSTEAD OF : spécifique aux vues.

Qu'est ce qui est exécuté ?

- le corps du Trigger = bloc PL/SQL effectué quand le trigger est déclenché

de plus :

- IF INSERTING THEN ... END IF;
- IF DELETING THEN ... END IF;
- IF UPDATING THEN ... END IF;

Déclenchement du Trigger / Evénement

Quelles valeurs sont testées ?

- Dans la clause WHERE ou dans le corps, on peut se référer à la **valeur d'un attribut avant** ou **après** que soit effectuée la modification déclenchant le trigger :
  - :OLD.nomAttribut : la valeur avant la transaction UPDATE ou DELETE
  - :NEW.nomAttribut : la valeur après la transaction UPDATE ou INSERT

Exemple de Trigger dans PL/SQL (1)

- **Ex 1 :** Trigger permettant de notifier la suppression d'un client (de la table Clients) avant l'exécution de l'opération

Code PL/SQL :

```
CREATE TRIGGER avant_suppression
    BEFORE DELETE ON clients
    BEGIN
        dbms_output.put_line('suppression d'un client');
    END;
```


Exemple de Trigger dans PL/SQL (2)

- **Ex 2 :** Trigger déclenché lors d'une insertion ou d'une modification de la table client

```
-- trigger déclenché lors d'une insertion ou d'une modification de la table client
SQL> CREATE OR REPLACE TRIGGER aff_discount
BEFORE INSERT or UPDATE ON clients
FOR EACH ROW
WHEN (new.no_cli > 0)
DECLARE
    evol_discount number;
BEGIN
    evol_discount := :new.discount - :old.discount;
    DBMS_OUTPUT.PUT_LINE(' evolution : ' ll evol_discount);
END;
/
```

Remarque :

- FOR EACH ROW signale qu'une modification de 4 lignes par un seul UPDATE déclenche 4 fois le trigger.
- Si on ne souhaite qu'un seul déclenchement, on omet la clause FOR EACH

6. Différences entre PL/SQL (Oracle) et PL/pgSQL (PostgreSQL)

PL/SQL Versus PL/pgSQL

- **PL/SQL :**
  - **PL/SQL** (sigle de *Procedural Language/Structured Query Language)* est un langage **propriétaire**, créé par **Oracle**
  - Syntaxe générale PL/SQL proche de celle des langages Pascal, Ada.
  - **PL/SQL** est disponible dans **Oracle Database** (depuis la version 7), **TimesTen In-Memory Database** (depuis la version 11.2.1) et **IBM DB2** (depuis la version 9.7)

- **PL/pgSQL :**
  - **PL/pgSQL** (*Procedural Language/PostgreSQL Structured Query Language*) est un langage procédural proposé avec le SGBD libre « **PostgreSQL** »
  - **PL/pgSQL** langage est **très similaire** au PL/SQL d'Oracle, ce qui permet de porter des scripts de ou vers Oracle au prix de quelques adaptations

Différences entre PL/pgSQL et PL/SQL (1)

Source: Portage d'Oracle PL/SQL (http://docs.postgresqlfr.org/9.0/plpgsql-porting.html)

- **Les affectations, boucles, conditionnelles sont similaires.**
- **Différences majeures :**
  - La notion de **paquetage** de PL/SQL n'a pas d'équivalent dans PL/pgSQ
  - La **structure itérative FOR** peut directement itérer sur le résultat d'une requête SQL dans PL/pgSQL
  - Les **arguments des procédures et fonctions** :
    - ne peuvent pas prendre de valeurs par défaut dans PL/pgSQL
    - mais la surcharge de fonctions et de procédures est possible dans PL/pgSQL

Différences entre PL/pgSQL et PL/SQL (2)

- Comme il n'y a pas de **paquetages** dans PL/pgSQL, utilisez des « **schémas** » pour organiser vos fonctions en groupes.
- Les boucles **FOR** d'entiers en ordre inverse (REVERSE) fonctionnent différemment : PL/SQL compte du second numéro jusqu'au premier
- Les boucles **FOR** sur des requêtes (autres que des curseurs) fonctionnent aussi différemment :
  - la variable cible doit avoir été déclarée alors que PL/SQL les déclare toujours implicitement
  - l'avantage est que les valeurs des variables sont toujours accessibles à la sortie de la boucle
- Attention il existe plusieurs différences de notation pour l'utilisation des **variables curseurs** : voir documentation PL/pgSQL.