Introduction à PL/SQL

Patricia Serrano Alvarado
à partir des slides de Sylvie Cazalens

PL/SQL : what for ?

- SQL langage puissant mais limité
- Plusieurs processus logiques des applications implémentés au niveau des applications et exécutés sur les postes clients
- Mais besoin d'homogénéisation et de sécurité d'accès aux BD

  => Proposition de langages procéduraux natifs

SQL/PSM

- SQL/Persistent Stored Modules
- ISO standard publié in 1996
- Extension de SQL92 avec un langage procédural
- Implémentations plus ou moins conformes
  - PostgreSQL : PL/pgSQL
  - SQL Server : Transact-SQL
  - DB2 : SQLPL
  - MySql : Stored procedure
  - Oracle : PL/SQL
  - etc.

PL/SQL

- Langage procédural pour SQL d'Oracle
- Proche de Pascal et Ada
- Facilités pour
  - Variables
  - Conditions
  - Boucles
  - Exceptions
  - Stockage de procédures

Que peut-on faire avec ?

- Code pour automatiser un traitement :
  - Processus périodiques
  - Processus ponctuels
  - Contraintes d'intégrité élaborées
  - Etc.

  - Applications
    - Auditing
    - Sécurité
    - Vérification
    - Sémantique
    - Etc.

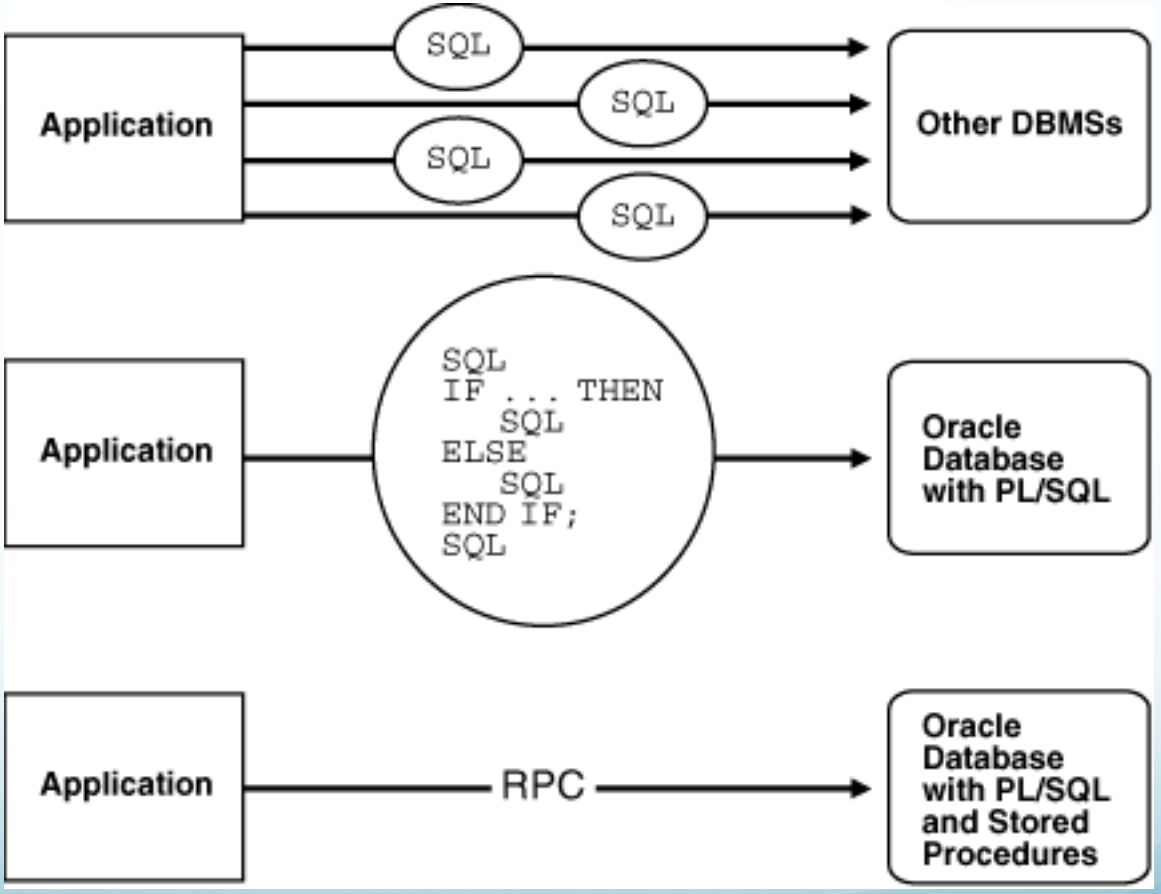
PL/SQL : avantages

- Support de SQL
- Support de programmation à objets
- Meilleure performance
- Plus grande productivité
- Entière portabilité
- Intégration forte avec Oracle
- Sécurité

Plan

- Les blocs PL/SQL
- Les blocs PL/SQL nommés :
  - Procédures et fonctions
  - Déclencheurs (triggers)
- Les packages

Amélioration de la performance



Organisation en blocs

- Un bloc PL/SQL peut être anonyme ou avoir un nom
  - bloc nommé : déclencheur, procédure, fonction, package...
  - bloc anonyme : n'a pas de nom
- Un bloc contient 3 parties
  - Déclarations (optionnel)
  - Commandes exécutables (corps)
  - Gestion des exceptions (optionnel)

Structure d'un bloc

[**DECLARE**
-- déclarations de types,
-- variables locales au bloc,
-- constantes,
-- exceptions et curseurs]
BEGIN [<nombloc>]
-- instructions PL/SQL et SQL
-- possibilité de blocs imbriqués
[**EXCEPTION**
 -- Traitement d'erreurs]
END; -- ou END <nombloc> ;

Déclarations : les types (1)

- Types simples :
  - issus de SQL : number, date, varchar2, ...
  - boolean, smallint, integer, float, real, …
  - **%type : retourne le type d'une variable ou une colonne**
- Types composés (collections)
  - record
  - table
- Type REF (pour l'objet-relationnel)

Déclarations : les types (2)

- Un type record peut être défini par :
  - Énumération de ses champs

type un_robot **is record**
(nom varchar2(25),
labo varchar2(15), …) ;

  - Par référence à la structure d'une table ou d'un curseur

<nom_curseur> <nom_table>**%rowtype**

Déclarations : les variables (1)

- De manière générale :

  <nom_variable> <type_variable> ;

- Déclaration avec une valeur au départ
  <Nom_variable> <type> **default** <valeur> ;

- Exemple

```
DECLARE
    Nom varchar2(25) ;
    Pds number default 4;
    Nom robots.nomr%type;
    Rbt un_robot ;
    Rbt1 robots%rowtype ;
```

Déclarations : les variables (2)

- Visibilité d'une variable :
  - dans le bloc où elle est déclarée et
  - dans les blocs imbriqués (sauf si redéfinie dans bloc imbriqué)

- Déclarations de constantes
  - <Nom_constante> **constant** < type> := <valeur>;

Corps d'un bloc

- Le corps peut comporter des instructions
  - d'affectation,
  - SQL : close, commit, delete, fetch, insert, lock table, open, rollback, savepoint, select, set transaction, update...
  - de contrôle (conditionnelles, répétitives),
  - de gestion de curseurs,
  - de gestion d'erreurs.

Corps : remarques générales

- Le caractère point-virgule est terminateur d'instruction

Chaque instruction est terminée par ;

- L'imbrication de blocs est possible, mais pas recommandée.

Affectation d'une variable

- Opérateur d'affectation (**:=**)

  nom := 'Pikachu' ;

  Rbt.nom := 'Pikachu' ;

- Option **into** de l'ordre select :

```
Select poids into pds
from robots
where nomr = 'Tom';
```

```
Select * into Rbt1
from robots
where nomr = 'Tom';
```

- Ordre **fetch** (avec les curseurs)

Structures conditionnelles

IF <condition> **THEN**
<instruction>; … <instruction>;

[**ELSIF** <condition**> THEN**
<instruction>; … <instruction>;]

[**ELSE**
<instruction>; … <instruction>;]

END IF;

Répétitive simple

- Répète indéfiniment une séquence d'instructions.

LOOP

<instruction>; … <instruction>;

END LOOP;

- Sorties de boucle :

IF <condition> **THEN EXIT** ; **END IF**;
EXIT WHEN <condition> ;

Répétitive « tant que »

- Répète la séquence d'instructions tant que la condition est vraie.

WHILE <condition> **LOOP**
<instruction>; … <instruction>;

END LOOP;

Répétitive « pour »

FOR <variable_boucle> **IN** [REVERSE] <borne_inf> … <borne_sup>

LOOP

<instruction>; … <instruction>;

END LOOP;

Exemple

- Programme pour faire un transfert bancaire.

Avant de permettre un débit de 500 Euros, il faut être sûr que le compte a un solde suffisant. Si c'est le cas, le programme fait le débit. Autrement le programme insère un tuple dans une table d'audit.

(exemple pris de : PL/SQL User's Guide Reference)

```
DECLARE
        acct_balance NUMBER(11,2);
        acct CONSTANT NUMBER(4) := 3;
        debit_amt  CONSTANT NUMBER(5,2) := 500.00;
BEGIN
        SELECT bal INTO acct_balance
        FROM accounts
        WHERE account_id = acct
        FOR UPDATE OF bal;

        IF acct_balance >= debit_amt THEN
                UPDATE accounts SET bal = bal - debit_amt
                WHERE account_id = acct;
        ELSE INSERT INTO temp VALUES
                (acct, acct_balance, 'Insufficient funds');
            -- insert account, current balance, and message
        END IF;
        COMMIT;
END;
```

Curseurs

- Zone de travail de l'environnement utilisateur qui contient des informations permettant l'exécution d'un ordre SQL.
- Utilisation : pour rechercher dans un nombre arbitraire de lignes (tuples) avec une instruction SELECT.
- Permet de garder en mémoire un array ou tableau.

cont.

- **Curseur implicite**: information sur la dernière instruction INSERT, UPDATE, DELETE, SELECT INTO

  (utilisation immédiate dans le programme)

- **Curseur explicite**: stockage de l'information et utilisation peu importe où dans le programme

- Attributs d'un curseur : %FOUND, %ROWCOUNT, %NOTFOUND, %ISOPEN

Exos curseur implicite

```
DELETE FROM emp
WHERE empno = my_empno;
IF SQL%FOUND THEN -- delete succeeded
      INSERT INTO new_emp VALUES (my_empno, my_ename, ...);
```

```
DELETE FROM emp
WHERE ...
IF SQL%ROWCOUNT > 10 THEN -- more than 10 rows were deleted
... END IF.
```

Gestion d’un curseur explicite

1. Déclaration
2. Ouverture
3. Traitement des lignes
4. Fermeture

Déclaration

- Déclaration obligatoire, partie « declare »
- Associe le curseur à une requête SELECT.

Syntaxe :

CURSOR <nom_curseur> **IS** <clause_select> ;

- Possibilité de paramétrer le curseur.

Ouverture

- Déclenche:
  - Allocation de mémoire pour les lignes du curseur
  - L'analyse syntaxique et sémantique du select
  - Le positionnement de verrous éventuels
- Se fait dans le corps du programme (donc après « begin »)
- Syntaxe :

OPEN <nom_curseur> ;

Traitement de lignes

- Les lignes obtenues par l'exécution de la clause « select » sont traitées une par une, par l'exécution d'un *ordre « fetch » dans une structure répétitive*.
- La valeur de chaque attribut doit être stockée dans une variable réceptrice.

FETCH <nom_curseur>
INTO <liste_variables>|<nom_de_record>;

Fermeture

- Libère la place mémoire.

- Syntaxe :

CLOSE <nom_curseur> ;

Attributs d'un curseur

- %NOTFOUND : nom_curseur%NOTFOUND
  - Vrai si le dernier fetch n'a pas ramené de ligne.
- %FOUND : nom_curseur%FOUND
  - Vrai si le dernier fetch a ramené une ligne (Faux si plus de ligne)
- %ROWCOUNT : nom_curseur%ROWCOUNT
  - Nombre de lignes traitées par l'ordre SQL, évolue à chaque ligne distribuée.
- %ISOPEN : nom_curseur%ISOPEN
  - Vrai si le curseur est ouvert (curseurs explicits)

Cont.

- Les attributs d'un curseur peuvent être utilisés comme condition de sortie de boucle.
- Exemple de curseur stockant le nom et salaire de 10 employés

Exemple

```
DECLARE
    CURSOR c1 IS SELECT last_name, salary
                 FROM employees
                 WHERE ROWNUM < 11;
    my_ename employees.last_name%TYPE;
    my_salary employees.salary%TYPE;
BEGIN
    OPEN c1;
    LOOP
        FETCH c1 INTO my_ename, my_salary;
        IF c1%FOUND THEN -- fetch succeeded
            dbms_output.put_line('Name = ' || my_ename || ', salary = ' || my_salary);
        ELSE EXIT -- fetch failed, so exit loop;
        END IF;
    END LOOP;
    CLOSE c1;
END; /
```

Répétitive « FOR » et curseur

- Spécifique à la gestion d'un curseur.
- Détermine dynamiquement le nombre de fois que la requête doit être exécutée en fonction du nombre de tuples de la requête.
- Open, fetch et close ne sont plus nécessaires car exécutés implicitement.

Exemple

```
DECLARE
    CURSOR c1 IS SELECT object_name, status
                    FROM user_objects
                    WHERE object_type = 'TABLE'
                    AND object_name NOT LIKE '%$%';
BEGIN
    FOR item IN c1 LOOP
        dbms_output.put_line('Table = ' || item.object_name || ', Status = ' || item.status);
    END LOOP;
END;
/
```

Gestion des erreurs

- Objectif : associer des traitements spécifiques aux erreurs qui surviennent lors de l'exécution d'un bloc PL/SQL.
- **1. Erreurs standards** (détectées par le moteur PL/SQL)
- 2. **Erreurs définis** par l'utilisateur/développeur.
- La traitement se fait dans la partie « EXCEPTION » du bloc PL/SQL.

1. Erreurs standards

- Fonctions SQLCODE et SQLERRM pour identifier l'exception.
- SQLCODE : code d'erreur, renvoie 0 si l'exécution s'est déroulée avec succès sinon renvoie une valeur non nulle.
- SQLERRM : message d'erreur correspondant.

Quelques erreurs standard

Exception	Oracle Error	SQLCODE Value
ACCESS_INTO_NULL	ORA-06530	-6530
CASE_NOT_FOUND	ORA-06592	-6592
COLLECTION_IS_NULL	ORA-06531	-6531
CURSOR_ALREADY_OPEN	ORA-06511	-6511
DUP_VAL_ON_INDEX	ORA-00001	-1
INVALID_CURSOR	ORA-01001	-1001
INVALID_NUMBER	ORA-01722	-1722
LOGIN_DENIED	ORA-01017	-1017
NO_DATA_FOUND	ORA-01403	+100
NOT_LOGGED_ON	ORA-01012	-1012
PROGRAM_ERROR	ORA-06501	-6501
ROWTYPE_MISMATCH	ORA-06504	-6504
SELF_IS_NULL	ORA-30625	-30625
STORAGE_ERROR	ORA-06500	-6500
SUBSCRIPT_BEYOND_COUNT	ORA-06533	-6533
SUBSCRIPT_OUTSIDE_LIMIT	ORA-06532	-6532
SYS_INVALID_ROWID	ORA-01410	-1410
TIMEOUT_ON_RESOURCE	ORA-00051	-51

Description traitement associé

exception

when <nom_erreur_1> **then**

<traitement_erreur_1> ;

...

when <nom_erreur_n> **then**

<traitement_erreur_n> ;

when others then

<traitement> ;

Exemple

```
DECLARE pe_ratio NUMBER(3,1);
BEGIN
      DELETE FROM stats
      WHERE symbol = 'XYZ';
      SELECT price / NVL(earnings, 0) INTO pe_ratio
      FROM stocks
      WHERE symbol = 'XYZ';
         INSERT INTO stats (symbol, ratio) VALUES ('XYZ', pe_ratio);

      EXCEPTION
            WHEN ZERO_DIVIDE THEN
                  NULL;
END;
/
```

2. Erreurs « utilisateur »

- Le nom de l'anomalie doit être déclaré.

declare

<nom_anomalie> **exception** ;

- Déclenchement du traitement

raise <nom_anomalie> ;

- Traitement

exception

when <nom_anomalie> **then**

Exemple

```
DECLARE
    comm_missing EXCEPTION; -- declare exception
BEGIN
     IF commission IS NULL THEN
          RAISE comm_missing; -- raise exception
     END IF;
     bonus := (salary * 0.10) + (commission * 0.15);
EXCEPTION
     WHEN comm_missing THEN ... -- process the exception
```

Procédures et fonctions PL/SQL

- Procédures et fonctions stockées sont utilisées pour enregistrer des traitements fréquemment utilisés au niveau du noyau.
- Un exemplaire stocké dans la base et exécutable en mode partagé par toutes les applications qui y font référence.

Procédures PL/SQL

CREATE PROCEDURE <nom_proc>
[(<arg1> <statut 1> <type1> … [, argumentN)] **IS**
[<decl_var_locales> ;]
BEGIN
…
[section_exception]
END [nom_procedure] ;

<Statut> ::= **IN** | **OUT** | **IN OUT**

Procédures PL/SQL

- L'ordre de création provoque la compilation du source avec stockage du code source et du pseudo code (ce dernier si pas d'erreurs de compilation).
- Suppression :

Drop procedure <nom_proc>

Exemple

```
PROCEDURE cree_client (p_nom VARCHAR2, p_ville
VARCHAR2) IS
BEGIN
   INSERT INTO clients (no_cli, nom_cli, ville_cli)
   VALUES (seq_noclient.NEXTVAL, p_nom, p_ville);
   COMMIT ;
END;
/
```

Appel procédures

- Mode interactif

execute <nom_proc> (<paramètres effectifs>) ;

EXECUTE cree_client('Toto','Nantes');

- Bloc PL/SQL

appel direct

- Java

#SQL {call}

Fonctions PL/SQL

CREATE FUNCTION <nom_fonc> [(<arg1> <statut1> <type1> … [, argumentN)] **RETURN** <type_fonc> **IS**

[<decl_var_locales>]

BEGIN

…

[section_exception]

END [<nom_fonc>];

Exemple

```
CREATE OR REPLACE FUNCTION solde (no INTEGER)
  RETURN REAL IS le_solde REAL;
BEGIN
     SELECT solde INTO le_solde FROM clients
     WHERE no_cli = no;
     RETURN le_solde;
END solde;
/

SELECT solde(1000) FROM dual ;
```

Gestion des erreurs

- Gestion de la totalité des erreurs à l'intérieur de la procédure.
- → Dans la section EXCEPTION
- Gestion des erreurs par l'environnement. La procédure génère un diagnostic d'erreur qui est transmis au programme appelant qui doit le gérer.
- → Erreur SGBD : ORA_xxxxx + message transmis à l'appelant
- → Erreur utilisateur : **raise_application_error**(<num>,< texte>)

Exemple

```
DECLARE
  num_tables NUMBER;
BEGIN
  SELECT COUNT(*) INTO num_tables
  FROM USER_TABLES;
  IF num_tables < 1000 THEN
    /* Issue your own error code (ORA-20101) with your own error message.
       Note that you do not need to qualify raise_application_error with
       DBMS_STANDARD */
    raise_application_error(-20101, 'Expecting at least 1000 tables');
  ELSE
    NULL; -- Do the rest of the processing (for the non-error case).
  END IF;
END;
/
```

Les packages

- Des fonctions, procédures, traitements d'exception ayant un lien peuvent être regroupés en package.

- Il y a des packages prédéfinis :
  - DBMS_LOB : pour gérer les « Large Objects »
  - DBMS_OUTPUT : affichage pour la mise au point des programmes PL/SQL.

SQL Dynamique

- Permet la génération de code SQL pendant l'exécution
- Utile pour
  - Programmes génériques et flexibles
  - Exécuter du code DDL (définition des données)
  - Lorsqu'au moment de la compilation on ne connaît pas le code complet SQL (le nom des tables, le prédicat du WHERE, etc.)
- Commande EXECUTE IMMEDIAT pour exécuter le code SQL dynamique

Exemples

- ```
  EXECUTE IMMEDIATE 'CREATE TABLE bonus (id NUMBER, amt
  NUMBER)';
  ```
- ```
  sql_stmt := 'INSERT INTO dept VALUES (:1, :2, :3)';

  EXECUTE IMMEDIATE sql_stmt USING dept_id, dept_name, location;
  ```
- ```
  plsql_block := 'BEGIN emp_pkg.raise_salary(:id, :amt); END;';

  EXECUTE IMMEDIATE plsql_block USING 7788, 500;
  ```
- ```
  Requete:='CREATE COMPTE etc.'

  EXECUTE IMMEDIATE (requete);
  ```

- Pour en savoir plus : code dynamique

Liens intéressants

- Oracle database documentation library.http://www.oracle.com/pls/db112/portal.all_books
  - PL/SQL https://docs.oracle.com/cd/E11882_01/appdev.112/e25519/toc.htm
- Oracle 10g PL/SQL User's Guide Reference

```
DECLARE
CURSOR cur_dept IS SELECT * FROM dept ORDER BY deptno;
CURSOR cur_emp (par_dept VARCHAR2) IS
SELECT ename, salary
FROM emp
WHERE deptno = par_dept
ORDER BY ename;

r_dept DEPT%ROWTYPE;
var_ename EMP.ENAME%TYPE;
var_salary EMP.SALARY%TYPE;
var_tot_salary NUMBER (10,2);

BEGIN
OPEN cur_dept;
LOOP

        FETCH cur_dept INTO r_dept;
        EXIT WHEN cur_dept%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE ('Department : ' || r_dept.deptno || ' - '||
        r_dept.dname);
        var_tot_salary := 0;
        OPEN cur_emp (r_dept.deptno);
        LOOP
        FETCH cur_emp INTO var_ename, var_salary;
        EXIT WHEN cur_emp%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE ('Name: ' ||var_ename || '
        Salary:'||var_salary);
        var_tot_salary := var_tot_salary + var_salary;
        END LOOP;
        CLOSE cur_emp;
        DBMS_OUTPUT.PUT_LINE ('Total Salary for Dept: ' || var_tot_salary);

END LOOP;
CLOSE cur_dept;
END;
/
```