

# 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.

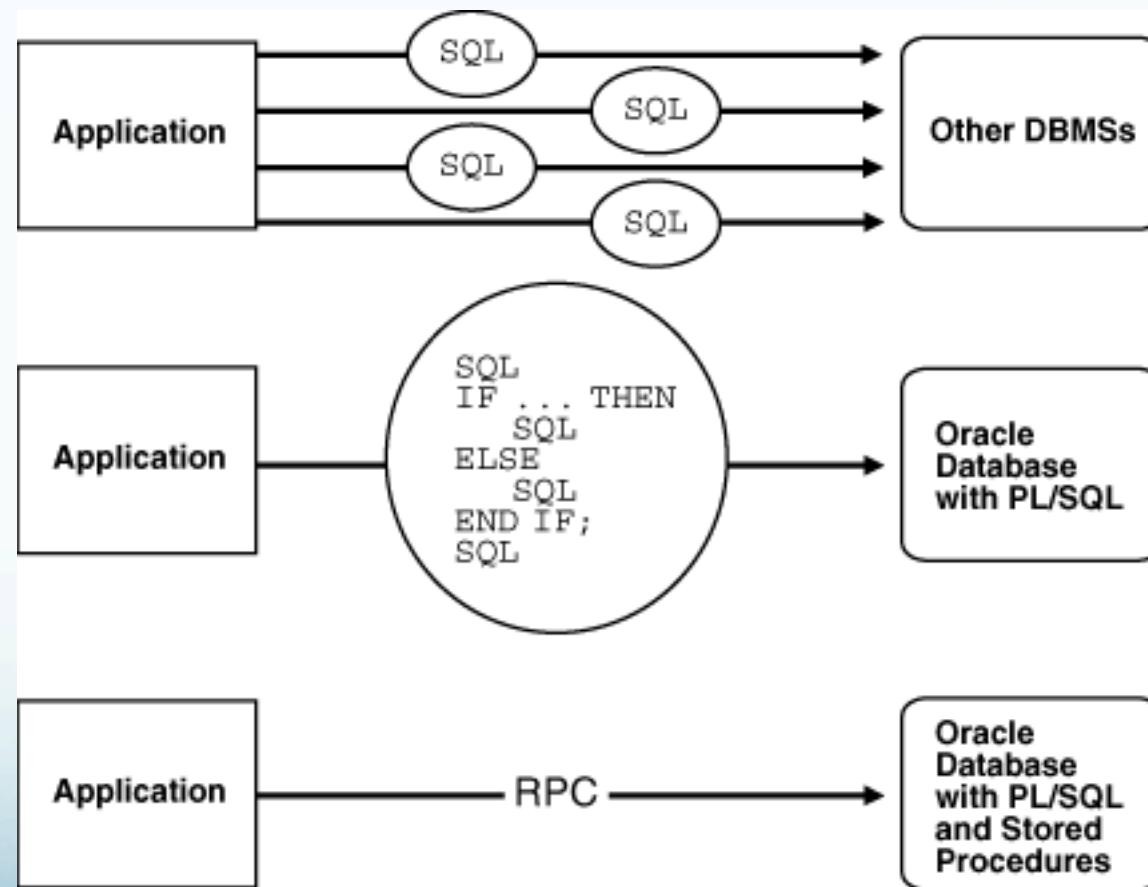
# PL/SQL : avantages

- 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 des erreurs]

**END;** /\* ou END <nombloc> ; \*/

# Déclarations : les types (1)

- Types simples :
  - issus de SQL : number, date, varchar2, ...
  - boolean, smallint, integer, float, real, ...
  - **%type**
- 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 varchar(25),
```

```
labo varchar(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> ;`
- Exemples
  - Nom varchar(25) ;
  - Pds number ;
  - 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\_variable> **constant** < type > := <valeur>;

# Corps

- 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 des 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’ ;

- Ordre **fetch** (avec les curseurs)
- Option **into** de l'ordre select.

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

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

# Exemple

- Commande PL/SQL qui, pour un nom de robot donné, calcule le total des points qu'il a obtenus et insère ce résultat, avec le nom du robot et la date courante dans une table de nom « bilan ». Dans « bilan », il peut y avoir plusieurs tuples concernant un même robot (mais les dates sont différentes)

# 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;**
- Sortie 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 sur 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)

# Curseurs

- Zone de travail de l'environnement utilisateur qui contient des informations permettant l'exécution d'un ordre SQL.  
texte source de l'ordre SQL, « traduction » de l'ordre, tampon correspondant à une ligne résultat,...

- Utilisation : pour rechercher dans un nombre arbitraire de lignes (tuples) avec une instruction SELECT.

# Curseurs

- **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

# Gestion d'un curseur explicite

- Déclaration
- Ouverture
- Traitement des lignes
- Fermeture

# Déclaration

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

Syntaxe :

**CURSOR** <nom curseur> **IS** <clause\_select> ;

- Possibilité de paramétriser 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 (1)

- `%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)

# Attributs d'un curseur (2)

- Les attributs d'un curseur peuvent être utilisés comme condition de sortie de boucle.
- Exple : Commande qui calcule pour chaque robot le total de ses points et insère pour chacun, dans une table bilan, la date courante, le nom du robot, le total des points.

# Attributs d'un curseur (2)

- 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

# 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

# Répétitive « FOR » et curseur

- Declare
- cursor c1 is
- select num, nom from clients;
- ...
- Begin
- For res in c1 loop
- Insert into Tabl values(res.num, res.nom);
- End loop;

# Gestion des erreurs

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

# 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> ;

# 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**

# 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>**

# Appel procédures

- Mode interactif

**execute <nom\_proc> (<paramètres effectifs>) ;**

- Bloc PL/SQL

appel direct

- Java

#SQL {call}

# Exemple

- Procédure PL/SQL ayant pour paramètre d'entrée le nom d'un robot, calculant le total des points qu'il a obtenus et insérant ce résultat, avec le nom du robot et la date courante dans une table de nom « bilan ».

# 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 Bilan des points d'un Robot

# 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>)**

# 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
- 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;
- Pour en savoir plus : [code dynamique](#)

# Liens intéressants

- Oracle database documentation library.  
[http://www.oracle.com/pls/db112/portal.all\\_books](http://www.oracle.com/pls/db112/portal.all_books)
- [PL/SQL User's Guide Reference](#)