PL/SQL

Agenda

- Introduction au langage PLSQL
- La clause DECLARE
- La clause BEGIN
- La clause EXCEPTION
- Curseurs
- Blocks imbriqués
- Paquetages et sous programmes

Outil de base de données approprié.

- Oracle XE au moins
- SQL Plus
- SQL Developer
  - TNSname.ora pour (hostname, port, SID)
- Installation de HR

Introduction à PLSQL

- Comprendre PL/SQL
- Avantages de PL/SQL
- Avantages en matière de performance
- Deployement de Programme PL/SQL
- Structure d’un programme PLSQL
- La Syntaxe de langage

Comprendre PL/SQL

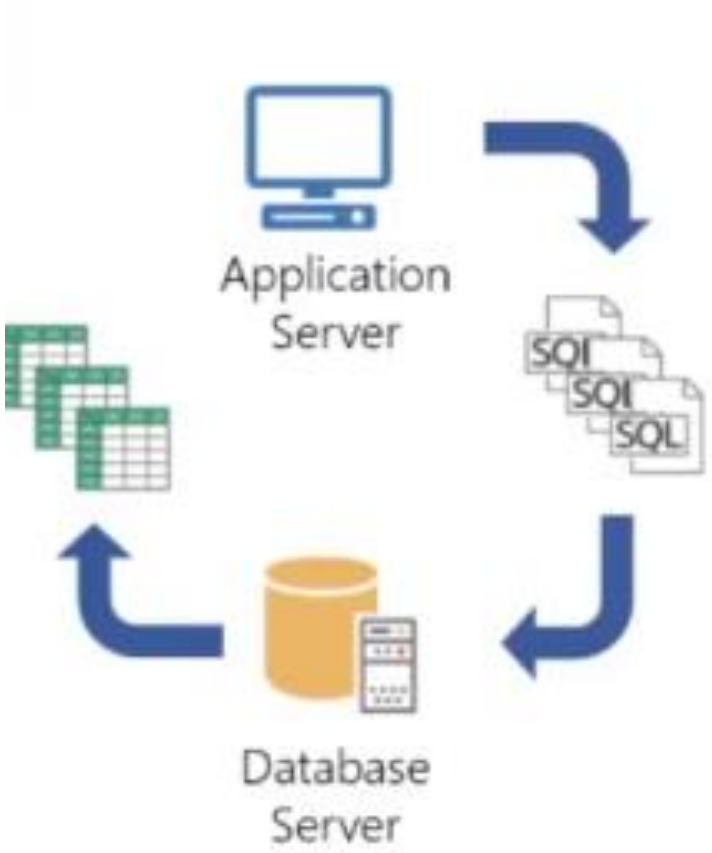
- SQL est declarative
  - Il faut specifier de quelle données on a besoin. Pas comment rechercher les donnés
  - Avantage – simple
  - Inconveniant: le comment est spécifié occasionnelement
- PL/SQL combine
  - Logique de la programmation procedurale
  - Caractéristiques declarative

Avantages de PL/SQL

- Portabilié
  - Textuel
  - OS indépendant
  - Est execute par la base de données Oracle. Non le client
- Simplicité
  - Pas besoin d’apprendre une syntaxe complexe

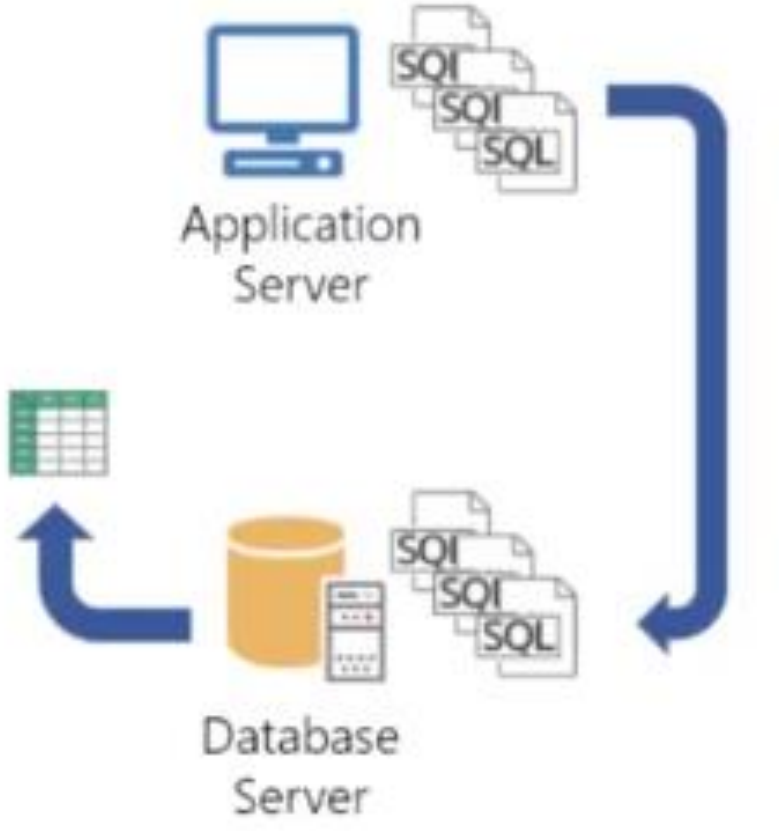
Performance

- Sans PL/SQL



Performance

- Avec PL/SQL



Deployement de programme PL/SQL

- Différente situations sont possibles
  - Script
  - Embedded within a Program (C , COBOL)
  - Strored Program (Trigger, etc..)
  - Oracle Object (Unité de programme dans des objets Oracle)

Exemple

```
 1  DECLARE
 2
 3      x_salary employee.salary%TYPE;
 4
 5  BEGIN
 6
 7      select salary
 8      into x_salary
 9      from employee
10      where ssn=&enter_ssn;
11
12      --Output the result
13      DBMS_OUTPUT.PUT_LINE('Salary is ' || x_salary);
14
15  EXCEPTION
16      --Output when no records are returned
17      WHEN no_data_found THEN
18              DBMS_OUTPUT.PUT_LINE ('No employee found');
19
20      WHEN others THEN
21              DBMS_OUTPUT.PUT_LINE('Error encountered, but cause unknown');
22
23* END;
```

Serveroutput

Fin introduction

Structure des blocs PL/SQL

- Les instructions PL/SQL sont continues dans des unités appelées Blocks
- Certains sont optionnel
- DECLARE : declaration des variables
- **BEGIN** : marque le début du programme
- **La logique du programme ainsi que du LMD SQL**
- EXCEPTION : marque le debut du traitement des exceptions
- **END: marque la fin du programme**

Regles syntaxiques

- Commenter le code
  - Comme en SQL “--“ pour les commentaire en une ligne
  - Comme en Java pour les commentaires multilignes¨¨¨/* ……;*/

Regles syntaxiques

- Une seule instruction PL/SQL par ligne
- Chaque instruction doit se terminer par un point virgule (;)
- Les LABEL ne se termine pas par (;) : BEGIN, DECLARE, etc..

Regles syntaxiques

```
DECLARE

BEGIN
        FOR I IN 1..1000 LOOP
                INSERT INTO employee (ssn, name)
                VALUES (900000000 + I, 'John Doe');
        END LOOP;
        COMMIT;

EXCEPTION
        WHEN OTHERS THEN
                ROLLBACK;

END;
/
```

SQL developer

- Exemple precédant:
  - Commentaires une ligne et plusieurs lignes sur les différents blocks

SQL DML et SQL DDL

- SQL DML
  - -INSERT, UPDATE et DELETE
- DDL
  - CREATE , DROP : pas possible en PL/SQL

La clause DECLARE

- Types de données
- Declaration des types simple
- Declaration des types complexe
  - %TYPE
  - %ROWTYPE
  - Type RECORD
  - User Defined Types (Types definit par l’utilisateur)

Les Objets de la Clause DECLARE

- Variables
- Boolean
- Constant
- Record
- Table
- User-defined-type

```
DECLARE
        --Define variables
        EmpLname            VARCHAR2(20);
        EmpSalary           NUMBER(6);
        EmpBdate            DATE;

BEGIN
...
END;
```

Règle de nommanclature des objets

- 30 caractères au maximum
- Le premier caractère est une lettre
- Ne pas utiliser les mots reserves (number, char, date, etc..)
- Les caractères spéciaux :

Legal	Illegal
$	&
#	- (dash)
_ (underscore)	/
	(space)

Exemple : declarations SQL developer

Types de données disponibles

- Types de données scalaires
  - Integer

Data Type	Definition	Rules and Features
BINARY_ INTEGER or PLS_INTEGER	Signed Integer ±2G	More efficient than NUMBER. Should be used if values will be integer.
SIMPLE_ INTEGER	Signed Integer ±2G	Only available in Oracle 11g and later. Better performance than PLS_INTEGER.

Types de données disponibles

- Sous types

BINARY_INTEGER Subtype	Definition	Rules and Features
NATURAL	Range of 0 to 2G	Uses the same rules as PLS_INTEGER, but no values less than 0.
POSITIVE	Range of 1 to 2G	Uses the same rules as PLS_INTEGER, but no values less than 1.

Types de données Flottant “Flotting Point”

Data Type	Definition	Rules and Features
BINARY_ FLOAT	Binary data type for 32-bit single-precision float	Binary data types offer more precision than NUMBER(). Also supports **infinity** and **NaN**.
BINARY_ DOUBLE	Binary data type for 64-bit double precision	Perfect for scientific computing applications.
NUMBER (precision,scale)	Numbers with Precision 1 to 38 Scale -84 to 127	Can be integer or floating point, but will operate slower than the types identified above.

Type de données Caractère

- Permet les textes et les nombres

Data Type	Definition	Rules and Features
CHAR(n)	Fixed length character data from 1 to 32,767	May map to either CHAR or LONG database columns.
VARCHAR2(n)	Varying length character data from 1 to 32,767	May map to either VARCHAR2 or LONG database columns.

Types de données prédéfinis

Data Type	Definition	Rules and Features
ROWID (obsolete)	Database row ID object	Identifies a physical row in a table.
UROWID	Physical or Logical row ID	Identifies a physical or logical row in a table.
BOOLEAN	Only values of True/False/Null	Useful for conditional testing.
DATE	Date/Time from 4712 BC to 4712 AD	Same rules that apply to database column.

Types de données “Synonyms”

- Pour la portabilité avec des sources non Oracle
- Non recommandé

Data Type	Synonym
NUMBER	DEC, DECIMAL, DOUBLE PRECISION, FLOAT, INTEGER, INT, NUMERIC, REAL, SMALLINT
CHAR	CHARACTER, STRING
VARCHAR2	VARCHAR

Déclarer des types de données simples et des affectations

- Valeurs par defaut
- NOT NULL
- Valeurs CONSTANTES

Valeur par Defaut

- Utiliser ":=" pour l'affection
- Dans affectation à la declaration, la variable est NULL

```
DECLARE
        --Definitions are the same
        nEmpSalary          NUMBER(6)  := 25000;
        nEmpSal             NUMBER(6)  DEFAULT 25000;

BEGIN
...
END;
```

Variable NOT NULL

- Comme une contrainte de base de données, il est possible d'appliquer la contraite NOT NULL à une variable
  - Si PL/SQL tante d'affecter NULL à une variable, une exception est générée
- Il faut alors initialiser cette variable pour éviter une violation de contrainte.

```
DECLARE
        --Defining a NOT NULL variable
        nDefaultDNO       NUMBER(2) NOT NULL := 10;

BEGIN
...
END;
```

Declaration de constante

- Une constant est simulaire à une variable à l'exception de ne pouvoir être modifié
- On Utilise le mot clé CONSTANT avant la déclartion du type

```
DECLARE
        --Defining a constant
        nMinSalary      CONSTANT NUMBER(6)  := 20000;
        nMaxSalary      CONSTANT NUMBER(6)  := 90000;

BEGIN
...
END;
```

Exemple

- Declarer deux constant
- Declarer des varibales des différents types et les initialer
- Les afficher convenablement
- NB: “||” pour la concaténation

Déclarer les types de données complexes et les affectations

- %TYPE
- %ROWTYPE

- TABLE
- RECORD

%TYPE

- Le type de données de la variable est dynamiquement et automatiquement lié à une colonne d’une table Oracle

%ROWTYPE


DECLARE
empRec employee%ROWTYPE;
BEGIN
...
Employee table
SSN CHAR(9)
...
Sex CHAR(1)
Salary NUMBER(6)
123456789
M
30000
345345345
F
25000
888665555
M
55000

TYPE TABLE

- Pour créer des tables à une dimension
- Utile pour stocker des données
- La défintion se fera en deux étapes:
  - On declare un User-defined- table
  - On reference le User-defined type

```
DECLARE

  --Declare variables
  TYPE EmpSSNarray
    IS TABLE of employee.ssn%TYPE
    INDEX BY SIMPLE_INTEGER;

  ManagementList EmpSSNarray;
  WorkerList     EmpSSNarray;
...
```

Exemple

```
SET SERVEROUTPUT ON;

DECLARE

  --Declare the table
  TYPE EmpSSNarray
    IS TABLE OF employee.ssn%TYPE
    INDEX BY SIMPLE_INTEGER;

  --Declare variables using the table
  ManagementList   EmpSSNarray;
  WorkerList       EmpSSNarray;
```

```
BEGIN

   --Retrieve the first Supervisor
   SELECT superssn
   INTO ManagementList(1)
   FROM employee
   WHERE superssn IS NOT NULL
   AND ROWNUM <= 1;

   --Retrieve the second Supervisor
   SELECT superssn
   INTO ManagementList(2)
   FROM employee
   WHERE superssn IS NOT NULL
   AND ROWNUM <= 1
   AND superssn <> ManagementList(1);
```

```
--Retrieve the first Worker
SELECT essn
INTO WorkerList(1)
FROM works_on
WHERE hours IS NOT NULL
AND ROWNUM <= 1
AND essn NOT IN (ManagementList(1), ManagementList(2));
```

```sql
--Retrieve the second Worker
SELECT essn
INTO WorkerList(2)
FROM works_on
WHERE hours IS NOT NULL
AND ROWNUM <= 1
AND essn NOT IN (ManagementList(1), ManagementList(2), WorkerList(1));
```

Exemple

- Afficher les deux manageurs et les deux travailleurs

TYPE RECORD

- Offre plus de flexibilité que %ROWTYPE
  - %ROWTYPE doit être associé à une table existante
  - RECORD permet tout autre combinaison de types de données
- La défintion se fera en deux étapes:
  - On declare un User-defined- record, et on reference le User-defined type

```
DECLARE

  --Declare variables
  TYPE EmpRecord
    IS RECORD (ssn          employee.ssn%TYPE,
               Lname        employee.lname%TYPE,
               Bonus        NUMBER(6)
               );

  ActiveEmp       EmpRecord;
  InactiveEmp     EmpRecord;
...
```

Exemple

```
--Retrieve the Active employee detail
SELECT essn, LName, DName, 5000
INTO ActiveEmp
FROM employee, department, works_on
WHERE employee.dno = department.dnumber
AND employee.ssn = works_on.essn
AND hours = (SELECT MAX(hours) FROM works_on)
AND ROWNUM <= 1;

--Output the results
dbms_output.put_line ('Active employee name: ' || ActiveEmp.LName);
dbms_output.put_line ('Active employee department: ' || ActiveEmp.DName);
dbms_output.put_line ('Active employee bonus: $' || ActiveEmp.BonusPaymen
```

Exemple pour InactiveEmp – zero bonus

- To do (completer le code)

créer et déclarer des types définis par l'utilisateur

qu'est-ce qui peut être inclus dans la clause BEGIN

travailler avec les variables char et varchar2

Handling String Literals

travailler avec des variables numériques

travailler avec des variables de date

assigner et utiliser des variables booléens

Utiliser des opérateurs de comparaison

UTILISER LES FONCTIONS SQL

SQL DML à l'intérieur de plsql

Utiliser select

exceptions et sql embarqué

Utiliser les sequences

Logic Control and Branching

Utilisation de GOTO

LOOP

création et utilisation de la boucle indéterministe

création et utilisation de la boucle conditionnelle

UTILISATION DES BOUCLES IMBRIQUÉES

If-Then-Else

CASE Statement

quels sont les types d'exceptions

Trapping Exceptions

identifier les détails d'exception

créer des exceptions définies par l'utilisateur

Using SQLCODE and SQLERRM Objects

comprendre le curseur implicite

créer des événements définis par l'utilisateur

comprendre le concept et le but des curseurs explicites