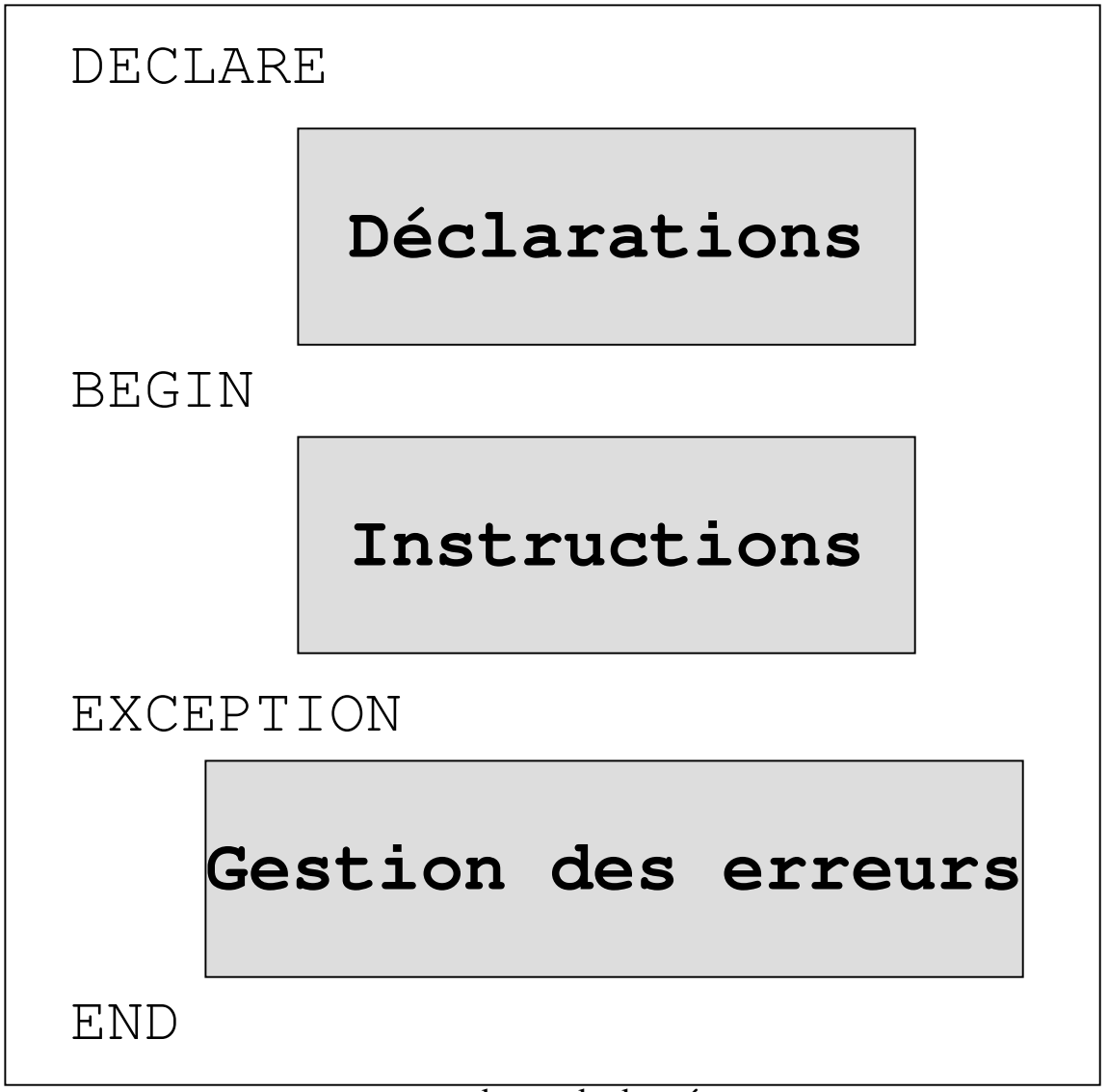
PL/SQL

- Langage procédural
- Permet de manipuler les données
- Langage commun aux outils Oracle
- Langage de programmation des événements (Forms, Base de données etc...)

Structure d'un Bloc d'instructions PL/SQL



Déclarations

- Variables et constantes
- Curseurs
- Exceptions

❶ Pas de différences entre minuscules et majuscules pour les identifiants

Types de base

- **<type>::= CHAR | BOOLEAN | DATE |**
  | NUMBER(t,d) | Varchar2(s)

  avec

  - **t** nombre total de chiffres
  - **d** nombre de chiffres après la virgule
  - **s** nombre maximum de caractères dans la chaîne

Syntaxe déclaration

- Variables
  <identifiant> <type>;
- Constantes
  <identifiant> constant <type>:=<expression>;
- Exemples
  x number(5,2);
  nom Varchar2(30);
  trouve boolean;

Instructions

- Requête SQL
- Affectation
- Instruction de contrôle conditionnel
- Instruction de contrôle itératif
- Gestion des curseurs

Syntaxe

- Les **instructions** sont séparées par des **;**
- Les commentaires se mettent entre **/*** mes commentaires ***/**
  ou après
  -- mes commentaires
- **Affectation : :=**

Conditionnelle

- **if** <condition> **then** <instructions> <suitecond>

  end if ;
- <suitecond> ::= |

  else <instructions> |

  elseif <condition> **then** <instructions> <suitecond>

Expressions de comparaison

- AND, OR, NOT
- = (égalité), <, >, <=, >=, != (différent)
  is NULL

Exemples conditionnelles

- `if (X>0) then X:=X-1; end if;`
- `if (trouve) then Z:=1; else Y:=1; end if;`
- ```
  if (Y=0) then C:='a';
  elseif (Y=1) then C:='b';
  elseif (Y=2) then C:='d';
  end if;
  ```
  ```

Traitements itératifs

- Loop
- while loop
- For loop

LOOP

LOOP
<instructions>
END LOOP;

- **Exemple:**

```
loop
 x:=x+1;
end loop;
```

Boucle infinie !

LOOP

```
LOOP
<instructions>
EXIT WHEN <condition>;
<instructions>
END LOOP;
```

- **Exemple:**

```
loop
 exit when x>10;
 x:=x+1;
end loop;
```

WHILE

```
WHILE <condition> LOOP
<instructions>
END LOOP;
```

- **Exemple**

```
while (x<10) loop
 x:=x+1;
end loop;
```

FOR LOOP

```
FOR compteur IN [REVERSE] inf..sup LOOP
<instructions>
END LOOP;
```

- **Exemple:**

```
for x in 1..10 loop
 s:=s+x;
end loop;
```

Exemple

```
Declare
 X number; Y number;
Begin
 Y:=40; X:=0; — initialisations
 while y>x loop
  x:=x+1; y:=y-1; /* ça c'est le corps
    de la boucle */
 end loop;
End;
```

Intégration des requêtes

- Requêtes retournant une seule ligne
- Requêtes retournant plusieurs lignes (curseurs)

Requête à ligne unique

- Mot clé **into**

Exemple

```
declare
MaxNum Number;
begin
 select Max(NumInter) nb into MaxNum from
Intervenant;
end;
```

Remarque

- Si la table ne contient aucune valeur MaxNum aura pour valeur NULL
- On peut utiliser la fonction NVL qui permet de remplacer une valeur NULL par autre chose

```
declare
  MaxNum Number;
  begin
   select nvl(max(NumInter),0) nb into MaxNum from
  Intervenant;
  end;
```

Requêtes à lignes multiples

Utilisation d'une structure de données appelée **CURSEUR**

- Equivalent des **recordset** d'Access
- Accès aux données de la base dans un programme
- Mises à jour "programmées" des données via PL/SQL

Exemple

- EtudiantAdm (NumEtud VarChar2(7),Nom Varchar2(30),
  Prenom Varchar2(30),Dpt Varchar2(3),
  Primary key (NumEtud));
- Etudiant(NumEtud VarChar2(7),Nom Varchar2(30),
  Prenom Varchar2(30),Groupe Varchar2(3),
  Primary key (NumEtud));
- EtudiantAdm est une table de l'administration qui contient la liste des nouveaux étudiants de l'IUT et la table Etudiant est la liste de étudiants gérée par l'IUT.

```
declare
   cursor NouveauxEtudiants is select * from etudiantAdm
   where Dpt='INF';
   UnEtud NouveauxEtudiants%rowtype;
begin
   open NouveauxEtudiants;
   loop
        fetch NouveauxEtudiants into UnEtud;
        exit when NouveauxEtudiants%notfound;
        insert into Etudiant values (UnEtud.Numetud,
   UnEtud.Nom,
                                     UnEtud.Prenom,'N');
         end loop;
  close NouveauxEtudiants;
  commit;
end;
```

Commentaires

- **cursor NouveauxEtudiants is select * from etudiant where…;**
  déclare une variable de type curseur dont le contenu sera le résultat de la requête
- **UneTable%ROW TYPE** permet de déclarer une variable de type "struct" qui contient les mêmes champs qu'une ligne de la table UneTable
- **OPEN NouveauxEtudiants** exécute la requête associée au curseur et met le résultat dans le curseur
- **fetch NouveauxEtudiants into UnEtud;**
  met la valeur de la ligne courante dans la variable UnEtud et passe à la suivante
- **UneTable%NOTFOUND** vaut vrai si le dernier fetch a échoué et faux sinon.
- **clbse NouveauxEtudiants** ferme le curseur

```
declare
   -- declaration du curseur
   cursor LesEtudiants is select NumEtud, Nom, Prenom, Groupe
                from etudiant
                order by Nom,Prenom  -- tri par nom et prenom
                for update;     -- permet les mises à jour de lignes
   UnEtudiant LesEtudiants%ROW TYPE;
   I Number;
begin
   OPEN LesEtudiants;
   I:=0;
   loop
        FETCH LesEtudiants INTO UnEtudiant;
        exit when LesEtudiants%NOTFOUND;
        I:=I+1;
        update etudiant set nom=upper(nom)
        WHERE CURRENT OF LesEtudiants; -- mise à jour du nom
   end loop;
   CLOSE LesEtudiants;
   commit;
end;
```

Exemple d'utilisation

- On récupère la liste des étudiants de 2ème année du dpt info via le logiciel de gestion des étudiants
- On veut répartir ces étudiants par groupe de 20 personnes

Solutions possibles

- Mettre à jour la colonne groupe de chaque étudiant 1 à 1
  => long et fastidieux
- Faire plusieurs requêtes "mise à jour" en SQL
  => comment faire une requête qui nous sépare les étudiants en 3 groupes?
- Utiliser un curseur

```
declare
   -- declaration du curseur
   cursor LesEtudiants is
   select NumEtud, Nom, Prenom, Groupe
   from etudiant
   order by Nom,Prenom -- tri par nom et prenom
   for update;

   UnEtudiant LesEtudiants%ROW TYPE;
   I Number;
   NbEtud Number;
   TailleGroupe Number;
```

```
begin
   select count(*) nb into NbEtud from Etudiant;
   TailleGroupe := CEIL(NbEtud / 3); -- valeur entière par
   excès
   OPEN LesEtudiants; -- ouverture du curseur
   I:=0;
   loop
    FETCH LesEtudiants INTO UnEtudiant;
    exit when LesEtudiants%NOTFOUND;
    I:=I+1;
    Update Etudiant SET Groupe=
   To_Char(CEIL(I/TailleGroupe))
    WHERE CURRENT OF LesEtudiants;
   end loop;
   CLOSE LesEtudiants;
   commit;
end;
```

PL/SQL et SQL*PLUS

- Comment stocker directement des procédures dans la base de données
- Programmation de trigger (déclencheurs) dans la base de données

Procédures stockées

- Create or replace
  <nom procedure>(paramètre) as
  <bloc pl/sql>
- Exemple
  create or replace inserer(num number) as
  begin
   insert into toto values (num);
  end;
- On peut ensuite utiliser la procédure dans un bloc pl/sql

Exemples

- Insertion d'une ligne

```
create or replace procedure insert etud(
 NumEtud VarChar2(7),Nom Varchar2(30),
 Prenom Varchar2(30)) as
begin
 insert into Etudiant values
(NumEtud,Nom,Prenom,'N');
end;
```

- Afficher des messages

```
set serveroutput on
create or replace procedure affiche mes(mes
varchar2) as
begin
dbms output.put line(mes);
end;
```

En cas de problème

« Attention : procédure créée avec erreurs de compilation »

```
> show errors

LINE/COL ERROR
-------- -----------------------------------------------------
XX/YY    message d'erreur
```

PL/SQL et les triggers

Un Trigger (Déclencheur) est une procédure qui est exécutée en réponse à un événement de manipulation des données (insertion, mise à jour, suppression)

Différents composants d'un Trigger

- L'événement qui arrive sur la base de données
- La table affectée par l'événement
- Le code à exécuter – PL/SQL

Différents types d'évènements

- INSERT
- UPDATE
- DELETE

Différents types de triggers

- Before (exécuté 1 fois)
- Before pour chaque ligne
- After (exécuté 1 fois)
- After pour chaque ligne

Syntaxe d'un Trigger

```
CREATE [OR REPLACE] TRIGGER trigger_name
{BEFORE|AFTER}
{INSERT|DELETE|UPDATE [OF column_list]}
ON nom_table
[FOR EACH ROW]
DECLARE
     declarations
BEGIN
     PL/SQL code
END;
```

Références aux valeurs de colonnes :

- :NEW
- :OLD

- INSERT : ***:new.col*** = valeur de la colonne col dans la ligne à insérer
- UPDATE : ***:old.col*** = valeur originale, ***:new.col*** = nouvelle valeur
- DELETE : ***:old.col*** = valeur dans la ligne à supprimer

Exemple génération de clés

```
CREATE OR REPLACE TRIGGER NouvCle
BEFORE INSERT ON Client FOR EACH ROW
DECLARE
      MaxNum NUMBER;
BEGIN
      SELECT MAX(NumCli)
          INTO MaxNum FROM Client;
      :NEW.NumCli := MaxNum + 1;
END ;
```

Trigger: quelques précisions

- Pour les triggers de type **before** on peut **encore agir** sur la mise à jour (replacer la valeur choisie par l'utilisateur etc...)
- Pour les triggers de type **after**, la mise à jour à été effectuée, on **ne peut plus qu'observer**

Exemple

- Pour la base de données de l'association on veut ajouter une règle de gestion impérative:
  "Aucune activité ne peut avoir plus de 20 participants"
- Mise en œuvre de cette règle par un trigger sur l'insertion et la mise à jour

```
create or replace trigger taille_groupe
before insert on participe for each row
declare
tropplein exception; NbPart number;
begin
-- récupère le nombre de participant à l'activité
select count(*) into NbPart from participe where
   intitule=:new.intitule
   and anneeparticipe=:new.anneeparticipe;
if NbPart>20 then raise tropplein; end if;
exception
when tropplein then -- provoque une erreur de type
   'application'
   RAISE_APPLICATION_ERROR(-20001,
   'Activité déjà complète');
end;
```

Gestion des exceptions

- Les exceptions provoquent l'arrêt du déroulement normal du bloc PL/SQL.
- Si un gestionnaire d'exception a été mis en place l'exception est traitée, et le programme peut éventuellement reprendre son cours
- Si aucun gestionnaire n'est mis en place l'exception provoque l'affichage d'un message d'erreur (SQL*PLUS) et l'arrêt de l'exécution du bloc.

```
declare
   cursor NouveauxEtudiants is select * from etudiant;
   E NouveauxEtudiants%rowtype;
begin
 open NouveauxEtudiants;
 loop
  fetch NouveauxEtudiants into E;
  exit when NouveauxEtudiants%notfound;
  begin
   insert into Etud2 values (E.NumEtud,E.nom,E.prenom,'','N');
  exception
    when Dup_Val_on_Index then
         update Etud2 set groupe='R' where numetud=E.Numetud;
   end;
  end loop;
  close NouveauxEtudiants;
  commit;
end;
```

Quelques exceptions prédéfinies

- Dup_val_on_index: essaid'insertion d'une ligne dont la clé existe déjà dans la table
- Invalid_number: conversion d'une chaine en nombre qui échoue
- No_data_found: aucune ligne retournée par la dernière requête
- Too_many_rows: le dernier select a retourné plus d'une ligne
- Zero_divide: division par 0
- Others: pour récupérer toutes les exceptions non traitées explicitement