Le langage PL/SQL

Myoung-Ah KANG

kang@isima.fr

Plan

-Le langage PL/SQL
-Procédures et fonctions stockées (sous-programmes)
-Groupement de procédure et packages
-Triggers (déclencheurs)

Le langage PL/SQL

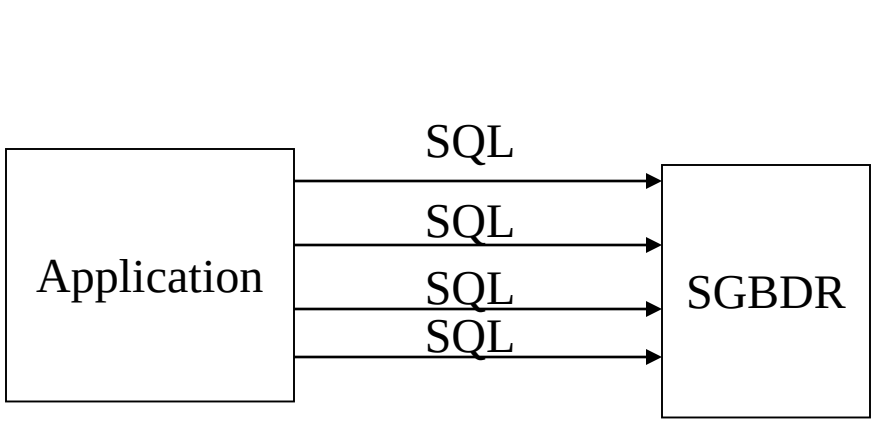
1. Introduction
2. Gestions des données
3. Instructions de contrôle
4. Curseur
5. Gestions des erreurs

1. Introduction PL/SQL

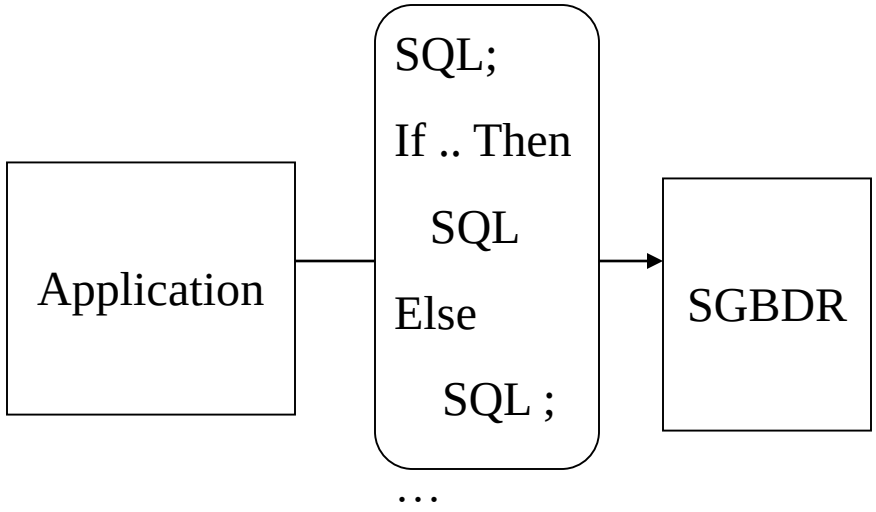
- PL/SQL permet de développer des programmes accédant une base de données ORACLE.
- PL/SQL est un langage de programmation procédurale qui étend le langage SQL. Il permet :
  - l'utilisation de la plupart des déclarions SQL (insert, delete, select, …)
  - la mise en œuvre de structure procédurales (*if…then…else,* for…loop, …)
  - la gestion des erreurs
  - l'optimisation de l'exécutions des requêtes
- Combine la puissance de manipulation des données offerte par SQL avec la puissance des langages de programmation (boucle, branchement, …)
- Code PL/SQL peut être accéder de plusieurs environnements: SQL*DBQ, SQL*FORMS, SQL*Menu, SQL*PLUS...

Environnement SQL et PL/SQL

- Dans l'environnement SQL, les ordres sont transmis au moteur SQL et exécutés les uns après les autres.
- Dans l'environnement PL/SQL, les ordres SQL et PL/SQL sont regroupé en *blocs*. Un bloc ne demande qu'un seul transfert vers le monteur, qui interprète en une seule fois l'ensemble des commandes contenues dans le bloc.



Environnement SQL



Environnement PL/SQL

Structure de bloc

[DECLARE

-- Déclarations des variables locales au bloc, constantes, …]

BEGIN

-- Instructions PL/SQL et SQL. Possibilité de blocs imbriqués

[EXCEPTION

-- Traitement des erreurs]

END ;

/

Note)

- -- : commentaire sur une ligne
- /*…*/ : commentaire sur plusieurs lignes
- lastname=LastName=LASTNAME
- Déclarations multiples non autorisées .ex), *i,j,k number ;* **- -** *Illicite*

- Le SQL admis
  - Déclaration des manipulations de données :
  - Déclarations des transactions :
  - Différents types de fonction SQL : *NUMERIC*, *CHARACTER*, *DATE*, *GROUP*, *DATA*, *CONVERSION*
  - Prédicats SQL utilisés dans la clause *WHERE* (*and*, *or*, *not*) et les opérateurs de comparaison *between*, *in*, *is null*, *like*
- Le SQL non admis
  - Les déclarations de définition de données :
  - Les déclarations de contrôle des données :

Un exemple de block PL/SQL : Achat de raquettes de tennis

```
DECLARE
qte_disp number(5);
BEGIN
      SELECT qte INTO qte_disp FROM inventaire
                  WHERE produit = ‘ RAQUETTE DE TENNIS ’
        IF qte_disp > 0 THEN  -- vérifie quantité
                   UPDATE inventaire SET qte = qte_disp – 1
                              WHERE produit = ‘ RAQUETTE DE TENNIS ’
                  INSERT INTO info_achat
                              VALUES (‘ Raquette tennis achetée ’, SYSDATE) ;
      ELSE
                  INSERT INTO info-achat
                              VALUES (‘ Stock non disponible ’, SYSDATE) ;
      END IF ;
      COMMIT ;
END ;
/
```

Environnement SQL*PLUS

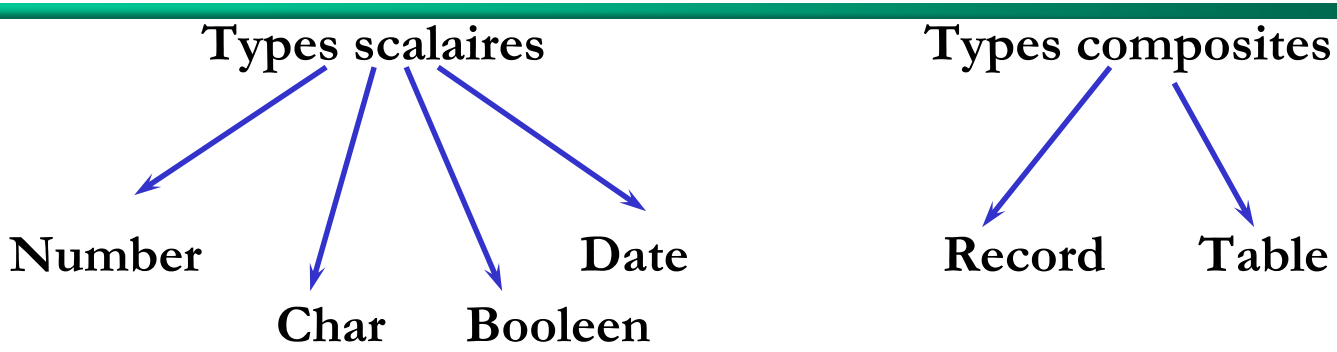
- PL/SQL ne permet pas (ou peu) d ’E/S
- Couplage avec SQL*PLUS ou un environnement de compilation (PRO*C)
- Utilisation de « votre éditeur favori » pour créer vos fichiers PL/SQL (extension .sql)
- /(slash) doit suivre chaque block PL/SQL
- Charger et exécuter ce script sous SQL*PLUS :

  SQL > GET <file> - - pour charger

  SQL > RUN - - pour exécuter

  SQL > START <file> - - pour les deux

2. Gestions des données – type de données



1. Types scalaires :

- Outre les types CHAR, NUMBER, DATE, VARCHAR2 (type natif d'Oracle), le langage PL/SQL offre les types supplémentaires suivant : BOOLEAN, INTEGER, REAL, TABLE, ROWID, etc.

- *nomtable.nomattribut%TYPE*
  - permet de déclarer des variables de type d'une colonne d'une table.

    Ex)
  - Indépendance des éventuels changements sur le schéma de relation
  - Attention à ce type d'utilisation:

    R (..., a, ...) avec une contrainte NOT NULL sur l 'attribut a

    DECLARE tempa R. a%TYPE;

    BEGIN tempa := NULL; END;

- Déclaration d'une variable : *nom_variable type*

  Ex) Taxe NUMBER (7,2); ok BOOLEAN;

2. Types composés : record (enregistrement), table (table)

2.1. Record : La déclaration d'une variable de record se fait :

– Soit par référence à une structure de table ou de curseur, en utilisation :

ex)

– Soit par une définition d'utilisateur : Déclaration en deux phases :

TYPE type-name IS RECORD
(champ1 {type1 | variable%TYPE | table.column%TYPE |
table%ROWTYPE} [NOT NULL] , ...);

Variable-name type-name;

Ex) TYPE DeptRecType IS RECORD
(nodept NUMBER(2) , nomdept dept.dnom%TYPE, …) ;
DeptRec DeptRecType;

» Identification :

2.2. Table : La déclaration en deux phases :

Phase 1) Déclaration du type table :

TYPE type-name IS TABLE OF
{column-type | variable%type | table.column%type}[NOT NULL]
INDEX BY BYNARY-INTEGER;

Phase 2) Déclaration de la variable

Variable-name type-name;

Ex) TYPE EnomTabType IS TABLE OF CHAR(10) NOT NULL
INDEX BY BINARY_INTEGER;
NomsTab EnomTabType; -- déclare la table PL/SQL

» Identification de 5iem élément du tableau : NomsTab(5)

- Le type admis dans une table est OBGLIGATOIREMENT un type scalaire.
- Tableau de tableau et tableau de record non permis (cf. record imbriqués permis donc, record de record)

Affectation de variables

- Le mot-clé DEFAUT initialise des variables sans utiliser l'opérateur d'affection.
  - Ex) taxe Number *DEFAULT* 10.50;

- Constantes
  - Ex) tax_rate *Constant* Number := 0.03;

- Affectation de variables :
  - par l'opérateur **:=**

  - par le mot réservé INTO
    - affecter à une variable le résultat d'une requête.

Exemples INTO

–Ex 1)

```
DECLARE
    U_nom pilote.nom%type ; u_sal pilote.sal%type;
BEGIN
    SELECT  nom, sal  INTO u_nom, u_sal  FROM PILOTE
    WHERE NOPILOTE = '7922' ;
END;
```

–Ex 2) affecter l'ensemble des valeurs des colonnes d'un tuple par l'exécution d'un seul ordre ;

```
DECLARE
TYPE t_emprec IS RECORD (r_nom pilote.nom%type, r_sal pilote.sal%type) ;
Emprec    t_emprec;
BEGIN
    SELECT  nom, sal  INTO emprec FROM PILOTE
    WHERE NOPILOTE = '7922' ;
END;
```

3. Instructions de contrôle

- Sélection :

```
IF condition THEN séquence_de_commandes-1;
ELSIF condition2 THEN séquence_de_commandes-2;
ELSE séquence_de_commandes-3;
END IF;
```

  - Note) Si la condition a la valeur FAUX ou la valeur NULL, les commandes ne sont pas exécuté.

EX)

```
IF trans_type = 'CR'  THEN
   UPDATE comptes SET balance = balance + credit WHERE …
ELSE
   IF nouv_bal >= min_bal THEN
                    UPDATE comptes SET balance = balance - debit
   WHERE …
    ELSE
   ….
    END IF;
END IF;
```

- Itération
  - LOOP

```
LOOP
        séquence_de_commandes
END LOOP ;
```

    - L'ordre EXIT permet de sortir d'une boucle.

```
LOOP
    … ;
    IF  condition  THEN
EXIT;
    END IF;
END LOOP;
```

```
LOOP
    … ;
    EXIT WHEN condition ;
    .. ;
END LOOP;
```

  - WHILE…LOOP

```
WHILE condition LOOP
        séquence_de_commandes;
END LOOP;
```

– FOR...LOOP

```
FOR compteur IN [REVERSE] valeur_début..valeur_fin LOOP
        séquence_de_commandes;
END LOOP;
```

  - *compteur* :
  - v*aleur_début*, *valeur_fin* : variables déclarées ou constantes
  - Le pas ne peut être égal qu'à 1.
  - L'incrémentation négative : *REVERSE*

EX)

```
SELECT COUNT(no_emp) INTO comp_emp FROM emp;
FOR i IN 1..comp_emp LOOP
...
END LOOP;
```

Ex)
Declare i number(2) ;

Begin

i := 4 ;

For i in 1..2 loop
    dbms_output.put_line (‘i : ‘ || to_char (i));
End loop;

dbms_output.put_line (‘i : ‘ ||to_char (i));

End ;
/

- Comparaisons Logiques
  - sur les nombres : =, *!*=, <, >, <=, >=
  - sur les caractères : =, *!*=, <, >, <=, >=
  - sur les dates : =, *!*=, <, >, <=, >=

  Notion de polymorphisme...

- Conversion de types
  - Conversions explicites, par utilisation de fonctions SQL telles que *to_date* , *to_char*…
  - Conversions implicites faites par le compilateur (à éviter)

Exercice) Visibilité d'une variable :
le résultat d'exécution de ce programme ?

```
DECLARE
  x  NUMBER := 0;   counter  NUMBER := 0;
BEGIN
  FOR i  IN  1..4  LOOP
            x := x + 1000;       counter := counter + 1;
            INSERT INTO temp VALUES (x, counter, 'outer loop');
            /* start an inner block */
            DECLARE
                 x  NUMBER := 0;   -- this is a local version of x
            BEGIN
                  FOR  i  IN  1..4  LOOP
                        x := x + 1; -- this increments the local x
                        counter := counter + 1;
                        INSERT INTO temp VALUES (x, counter, 'inner loop');
                 END  LOOP;
            END;
END  LOOP;
COMMIT;
END;
```

- Visibilité d'une variable :
  - Déduire à partir de ce programme les règles régissant la portée des variables dans des programmes PL/SQL.

  ➔ Une variable est utilisable dans le bloc où elle est définie ainsi que les blocs imbriqués dans le bloc de définition, sauf si elle est renommée dans un bloc interne.

- Note) Il ne faut pas faire :

```
DECLARE ename CHAR(10):= 'KING ’
BEGIN
   delete from emp where ename =ename;
END;
```

  - Puisque les noms d ’attribut sont toujours prioritaires =>

  - Portée et visibilité des variables: classique aux langages de programmation usuels.

4. Curseur

- Le curseur est une *zone de travail*. PL/SQL utilise des curseurs pour tous les accès à des information de la base de données. PL/SQL permet de nommer cette zone mémoire et de contrôler sa gestion. Il existe deux types de **curseurs**:
  - curseur explicite: c'est un curseur qui est défini par l 'utilisateur. Il permet de traiter les requêtes SQL dont le résultat est constitué de plusieurs lignes:
    - il récupère la première ligne de la réponse,
    - il garde la trace de la ligne courante

    Ceci permet à PL/SQL de traiter les lignes une à la fois.
  - curseur implicite: géré automatiquement par PL/SQL lorsqu'un curseur explicite n'a pas été déclaré.

4.1. Curseur Implicite

- Les curseurs implicites sont gérées automatiquement aux cas suivants:
  - Les ordres SELECT exécutés sous SQL*PLS
  - Les ordres SELECT donnant une seul ligne résultat avec les autres produits (PL/SQL, SQL*FORMS, PROC, etc…)
  - Ordres UPDATE, INSERT et DELETE avec tous les produits.

- PL/SQL permet d ’accéder à des informations même dans le cas d ’un **curseur implicite**
  - La récupération des informations se fait par l ’intermédiaire des **attributs du curseur implicite**
  - Le **nom** du **dernier curseur implicite** est SQL%

4.2. Curseur Explicite

- La gestion d'un curseur explicite nécessite quatre étapes ;

1. Déclaration du curseur
   - Il est déclaré dans la partie DECLARATION du BLOC PL/SQL.
   - Définir la requête SELECT et l'associer à un curseur.
2. Ouverture du curseur
   -
   - OPEN permet d'allouer un espace mémoire au curseur et de positionner des verrous éventuels.
3. Traitement des lignes
4. Fermeture du curseur
   - A la fin de traitement le curseur doit être fermé à l'aide de l'ordre CLOSE.
   - CLOSE libère la place mémoire et les verrous éventuels.

Traitement des lignes

- *Ex 1) FETCH nom_curseur INTO liste_variables ;*

```
DECLARE
      Cursor c IS SELECT nom, sal FROM pilote ;
      v_nom pilote.nom%type;
      v_sal pilote.sal%type;
BEGIN
            OPEN C;
            LOOP
               FETCH  c INTO v_nom, v_sal;
               EXIT WHEN (c%NOTFOUND) ;
               …
            END LOOP ;
            CLOSE c;
END;
```

- *Ex 2) FETCH nom_curseur INTO no_enregistrement;*

```
DECLARE
      TYPE t_emp IS RECORD
            (v_nom pilote.nom%type, v_sal pilote.sal%type) ;

      r_emp t_emp ;

      Cursor c IS SELECT nom, sal FROM pilote ;
BEGIN
            OPEN c;
            LOOP
               FETCH c INTO r_emp;
               EXIT WHEN (c%NOTFOUND) ;
               …
            END LOOP ;
            CLOSE c;
END;
```

- *Ex 3) FETCH nom_curseur INTO no_enregistrement;*

```
DECLARE
        Cursor c IS SELECT *  FROM pilote ;
        r_emp c%ROWTYPE ;

BEGIN
              OPEN  c;
              LOOP
                 FETCH  c  INTO r_emp;
                 EXIT WHEN (c%NOTFOUND) ;
                 …
              END LOOP ;
             CLOSE c;
END;
```

Curseur For…Loop

- Le curseur **For loop** est une simplification d'écriture d'un curseur explicite. Il permet à la fois d'**ouvrir** le curseur, d'exécuter des **Fetch** pour ramener chaque ligne retournée par l'ordre SQL associé et enfin de fermer le curseur quand toutes les lignes ont été traitées.

EX) DECLARE

```
        Cursor c IS SELECT  nom, sal  FROM pilote ;
        V_nom pilote.nom%type; v_sal pilote.sal%type;
BEGIN
   for rec_c in C
   loop
            v_nom := rec_c.nom ; v_sal := rec_c.sal;
   end loop ;
END;
```

4.3. Curseur Paramétré

- Permet de **réutiliser** un même curseur avec des valeurs différentes dans un **même** bloc PL/SQL.

Syntaxe :

```
Cursor nom-cur(nom-var Type [:= valeur par défaut], ...)
IS  requêt ; -- l'ordre SQL avec les pramètres P1  et P2
```

- Type: un type admis par PL/SQL ex) CHAR, NUMBER, DATE, BOOLEAN, etc..
- Ex)

```
DECLARE
    CURSOR C (psql NUMBER(7,2), pcom NUMBER(7,2))
    IS SELECT enom FROM emp
    WHERE sal > psal AND comm > pcom ;
```

- Le **passage** des valeurs des paramètres se fait à l'**ouverture** du curseur.
- Chaque paramètre réel (à l'ouverture du curseur) est associé à un seul paramètre formel (à la déclaration du curseur) selon l'un des deux modes suivants :
  - Association par position : Chaque paramètre formel est remplacé par le paramètre réel à la même position dans la liste.

    EX)

  - Association par nom : Les paramètres réels peuvent être indiqués dans un ordre quelconque en faisant apparaître la correspondance de façon explicite.

    Ex) OPEN C (pcom=> 25000, psal=> 12000);

4.4. Curseur pour la mise à jour

- Un curseur peut être utilisé pour la mise à jour d'une relation.

- L'option WHERE CURRENT OF nom_curseur pour l'ordre UPDATE ou DELETE :
  - permet de modifier ou de modifier la ligne courante dans le curseur pour UPDATE ou DELETE.
  - Déclaration du curseur : il faut préciser l'intention de l'utiliser pour la mise à jour à l'aide de l'ordre *FOR UPDATE*.

EX)

```
DECLARE
    CURSOR cur1 IS SELECT nom, sal, comm FROM pilote
    WHERE nopilot BETWWEN 1280 AND 1999 FOR UPDATE;
    v_nom pilote.nom%type;          v_sal pilote.sal%type ;
    v_comm pilote.comm%type;
BEGIN
    OPEN cur1;
    LOOP
        FETCH  cur1  INTO  v_nom,  v_sal,  v_comm;
        EXIT  WHEN  cur1%NOTFOUND;
        IF v_comm IS NULL THEN
            DELETE FROM pilote WHERE CURRENT OF cur1;
        ELSEIF v_comm > v_sal THEN
            UPDATE pilote SET sal = v_sal + v_comm,  comm=0
            WHERE CURRENT OF cur1;
        END IF;
    END LOOP;
    CLOSE cur1;
END;
```

4.5. Statut d'un curseur

- Attributs du Curseur Explicite : Le curseur explicite dispose de quatre attributs qui fournissent des informations le concernant ;
  - : évalué à TRUE si le dernier FETCH ne retourne aucune ligne (il n ’y a plus de ligne disponible).
  - : c’est le contraire du %NOTFOUND. Il est évalué à FALSE si le dernier FETCH ne retourne aucune ligne.
  - retourne le nombre de lignes, de l ’ensemble des lignes actives dans le curseur, ramenées par le fetch.
  - : évalué à TRUE si le curseur est ouvert.
- Utilisation des attributs : Il faut simplement ajouter le nom du curseur à la suite du nom de l ’attribut

  Ex) :

  If emp-curs%NOTFOUND Then …

  (emp-curs: est le nom du curseur)

- Les Attributs du Curseur Implicite : la valeur de l'attribut est relative au dernier ordre SQL exécuté avant son utilisation. ;
  - *SQL%NOTFOUND* est évalué à TRUE si :
    - un ordre INSERT, DELETE ou UPDATE, n'affecte aucune ligne.
    - un ordre SELECT, qui porte sur une seule ligne, ne retourne aucune ligne.
  - *SQL%FOUND* retourne le résultat opposé de SQL%NOTFOUND.
  - *SQL%ISOPEN*
  - *SQL%ROWCOUNT*

5. Gestions des erreurs

- Il s'agit d'affecter un traitement approprié aux erreurs. Ces erreurs sont de deux types:
  - Erreur standard détectée par le moteur PL/SQL.
  - Anomalie définie par l'utilisateur

  ➔ Définir une procédure de traitement de l'erreur dans la partie .

- Exécution du traitement d'erreurs
  - Les traitements d'erreurs sont toujours en attente d'erreur. Lorsqu'une erreur se produit et qu'elle est définie dans la section EXCEPTION, le traitement associé est exécuté.
  - Après l'exécution du traitement d'une exception, le contrôle est donné au bloc supérieur s'il existe sinon il est donné à l'environnement hôte

Gestion des erreurs standard

- Déclenchement d'une Exception

  : déclenchée automatiquement par le système

  Ex) NO_DATA_FOUND, VALUE_ERROR, LOGIN_DENIED, ZERO_DIVIDE, etc…

- Description du traitement de l'erreur

EX)

```
BEGIN
..
EXCEPTION WHEN no_data_found THEN
    .. -- traitement de l'erreur
END;
```

  - Note) la section Exception peut contenir autant de procédure WHEN .. Que l'utilisateur veut gérer d'erreurs standard et utilisateur.
  - Il est possible d'utiliser l'opérateur OR dans la clause WHEN. Ex)

Gestion des erreurs (anomalies) utilisateurs

- Gestion des erreurs utilisateurs
  - déclaration de l'anomalie :

```
DECALARE
  nom_anomalie  EXCEPTION;
```

  - traitement de l'erreur

- Déclenchement d’une Exception

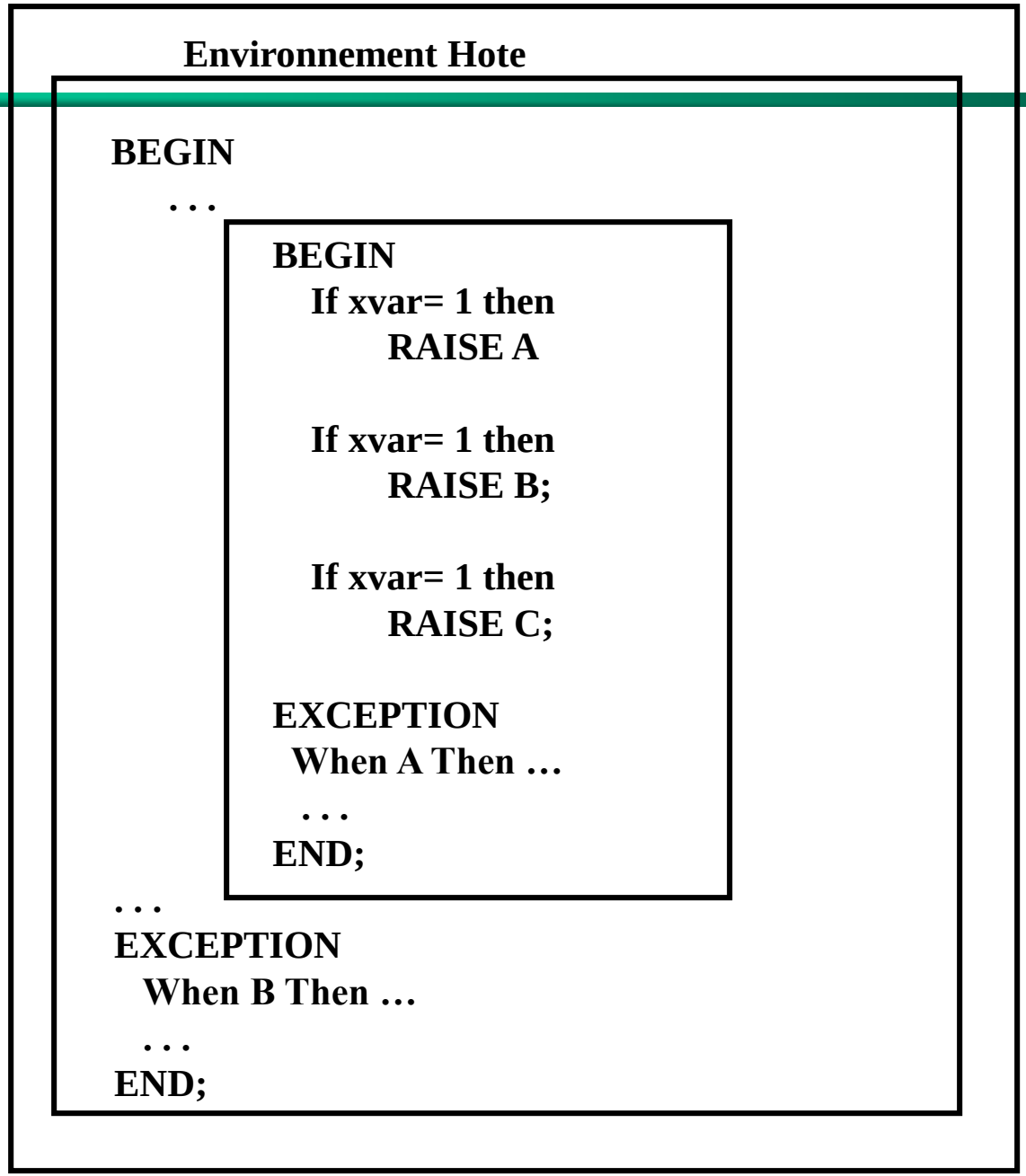
  : Déclenchée **explicitement** par l ’utilisateur par la commande *RAISE*

```
DECLARE
xnum NUMBER(3,1);
yvar NUMBER(3,1);
test1 exception;
. . .
BEGIN
. . .
If xnum > ynum Then RAISE test1;
xnum:= 15/yvar;
. . .
. . .
EXCEPTION
When test1 Then . . .;
When ZERO-DIVIDE Then . . .;

When others Then . . .;



END;
```

Propagation des Exceptions

- Si une exception est déclenchée et que son **traitement** n'est pas défini dans le BLOC courant, alors l'exécution du BLOC courant est arrêtée et l'**exception** est **propagée** au BLOC supérieur qui devient le BLOC courant

- Si le **traitement d'une exception** n'est défini dans aucun BLOC, l'exception est propagée jusqu'au dernier BLOC puis un message d 'erreur « *unhanled exception* » est renvoyé à l'environnement hôte.


Environnement Hote
BEGIN
. . .
BEGIN
If xvar= 1 then
RAISE A
If xvar= 1 then
RAISE B;
If xvar= 1 then
RAISE C;
EXCEPTION
When A Then …
. . .
END;
. . .
EXCEPTION
When B Then …
. . .
END;

Porté des Exceptions

- Si une **Exception** est déclarée dans un bloc elle est :
  - locale au BLOC
  - globale pour les SOUS-BLOCS.
- Un BLOC ne peut référencer que les **exceptions** globales ou locales (il ne peut référencer les **exceptions** de ses SOUS-BLOCS)
- Une **exception** globale peut être redéclarée dans un SOUS-BLOC (dans ce cas le SOUS-BLOC ne peut plus référencer l'**exception** globale)