



Le langage de bloc PL/SQL

■ Plan:

- Principes
- Sélections
- Exceptions
- Procédures et fonctions stockées
- Paquetages



Programmation d'application dans un environnement BD

- SQL: langage ensembliste
 - Ensemble de requêtes distinctes
 - Puissant
 - Déclaratif: On décrit le problème (quoi) sans dire comment il faut accéder aux données ni comment les afficher (comment → prog. impérative)...
- Programmation avancée
 - Pro*C, JDBC, PL/ SQL,...

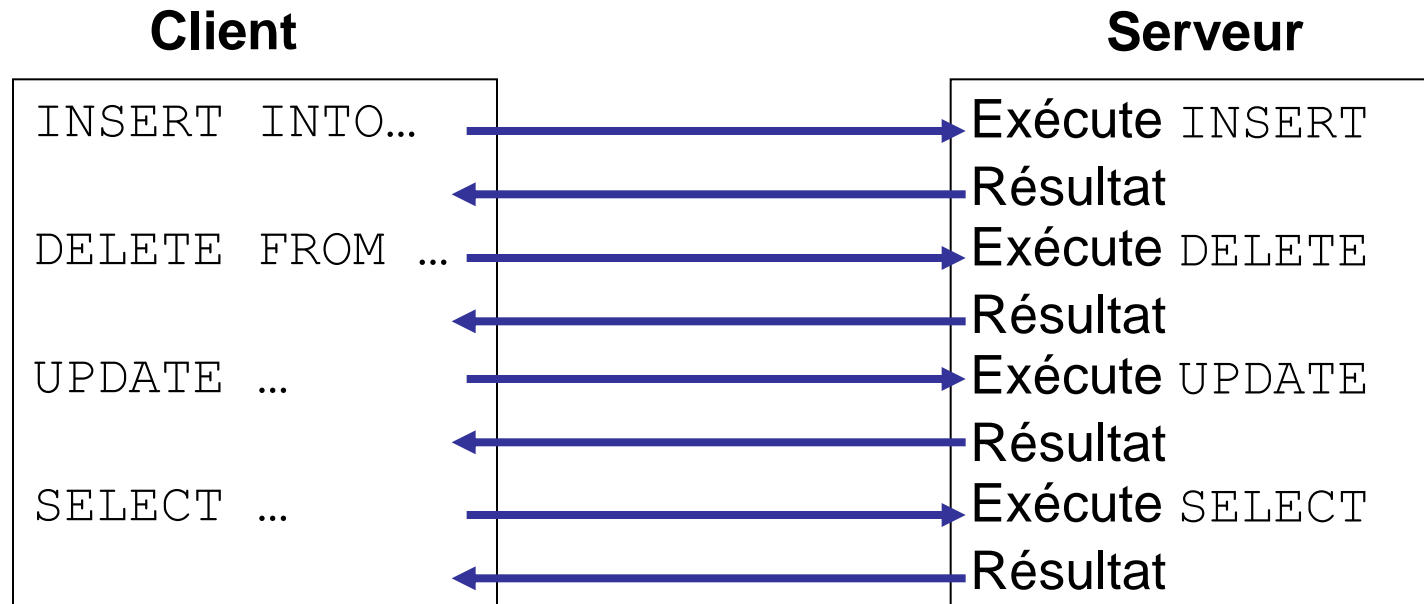


SQL – PL/SQL

- PL/SQL:
 - Sur-couche procédurale à SQL
 - Boucles, contrôles, affectations ,exceptions,...
 - Chaque programme est un bloc (BEGIN-END)
 - Programmation adaptée pour:
 - Transactions
 - Architecture Client/Serveur

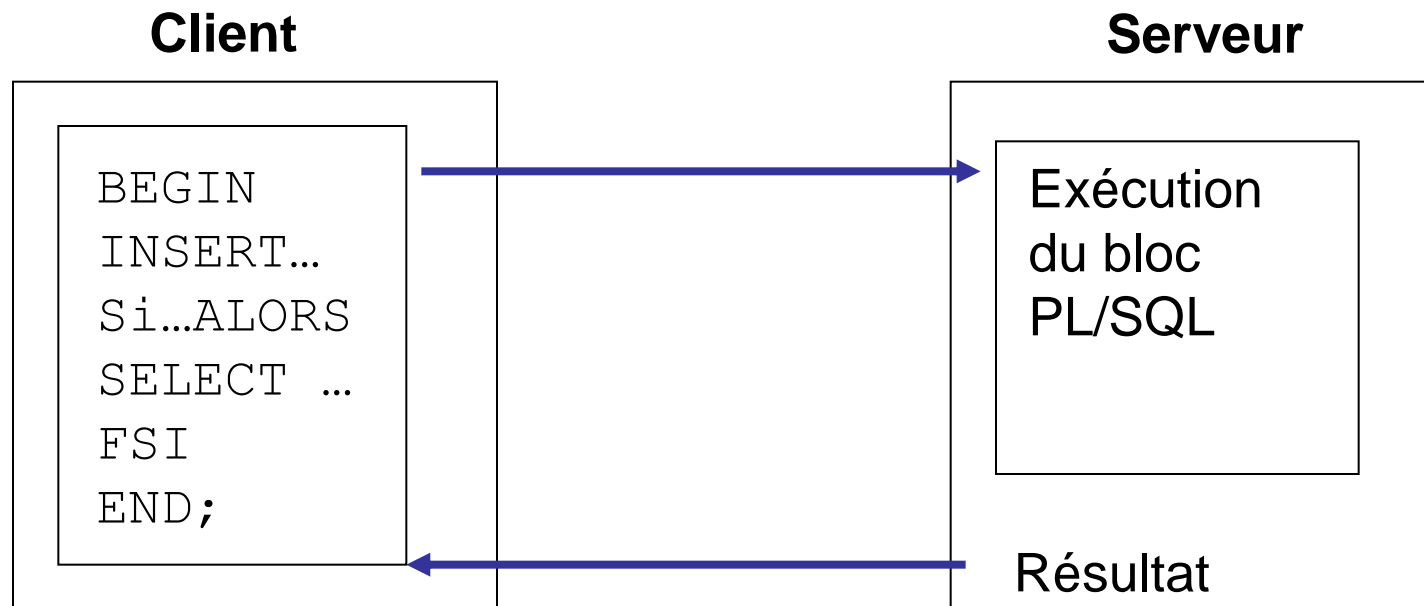
Requêtes SQL

- Chaque requête 'Client' est transmise au serveur de données pour être exécutée avec retour de résultats



Bloc PL/SQL

- Le bloc de requêtes est envoyé sur le serveur. Celui-ci exécute le bloc et renvoie un résultat final.



Format d'un bloc PL/SQL

- Section DECLARE: déclaration de

- Variables locales simples
- Variables tableaux
- Curseurs

- Section BEGIN

- Section des ordres exécutables
- Ordres SQL
- Ordres PL

- Section EXCEPTION

- Réception en cas d'erreur
- Exceptions SQL ou utilisateur

```
DECLARE
    --déclarations

BEGIN
    --exécutions

EXCEPTION
    --erreurs

END ;
/
```



Variables simples

- Variables de type SQL

```
Nbr      NUMBER (2) ;  
Nom      VARCHAR2 (30) ;
```

```
Minimum  INTEGER:=5 ;  
Salaire  NUMBER (8,2) ;
```

- Variables de type Booléen (TRUE, FALSE)

```
Fin      BOOLEAN ;  
Ok       BOOLEAN:=True ;
```

- On peut ajouter les conditions

- CONSTANT

```
Minimum CONSTANT  INTEGER:=5 ;
```

- NOT NULL

```
Debut NUMBER NOT NULL ;
```

- DEFAULT

```
Réponse BOOLEAN DEFAULT TRUE ;
```



Variables faisant référence au Dictionnaire des données

- Référence à une **colonne**

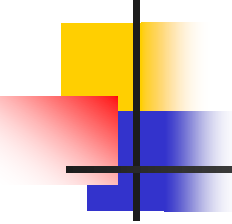
<code>vsalaire</code>	<code>Employes.salaire%TYPE;</code>
<code>Vnom</code>	<code>Etudiant.nom%TYPE;</code>
<code>Vcomm</code>	<code>vsalaire%TYPE</code>

- Référence à une **ligne**

<code>Vemploye</code>	<code>Employes%ROWTYPE;</code>
<code>Vetudiant</code>	<code>Etudiant%ROWTYPE;</code>

- Contenu d'une variable : `variable.colonne`

<code>Vemploye.adresse</code>



Variables paramétrées lues sous SQL*Plus: utilisation de &

- Variables lues par un ACCEPT ... PROMPT

Variable non déclarée

```
+ { ACCEPT param PROMPT 'Entrer la valeur :';  
  { DECLARE  
    { -- déclarations  
    { BEGIN  
      { -- travail avec le contenu de plu:  
      { -- &param si numérique  
      { -- '&param' si chaîne de caractères  
    { END;  
      { /  
+ { -- Ordre SQL
```

PL

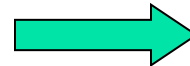
Variables en sortie sous SQL*Plus:

- Variable déclarée sous SQL*Plus, utilisée dans le bloc PL puis affichée sous SQL*Plus

Déclaration variable

+ {
PL {
+ {

```
VARIABLE i NUMBER;  
BEGIN  
    :i :=15;  
END;  
/  
PRINT i;
```



SQL> print i;

```
      I  
-----  
     15
```



Blocs imbriqués et portée d'une variable

- Les instructions peuvent être imbriquées
 - Un bloc imbriqué est considéré comme une instruction
 - La section EXCEPTION peut contenir des blocs imbriqués
- Un identifiant est visible dans les régions où on peut référencer cet identifiant
 - Un bloc voit les objets du bloc du niveau supérieur
 - Un bloc ne voit pas les objets des blocs de niveau inférieur



Blocs imbriqués et portée d'une variable

```
DECLARE
    x NUMBER;
BEGIN
    ...
    DECLARE
        y NUMBER;
    BEGIN
        ...
    END ;
    ...
END ;
/
```

Portée de y

Portée de x



Instructions PL

- Affectation (:=)
 - **A:=B;**
- Structure alternative ou conditionnelle
 - Opérateurs SQL:
<, >, ... , OR, AND, ... , BETWEEN, LIKE, IN
 - **IF..THEN ..ELSE...END IF;**

```
IF condition THEN instructions;  
    ELSIF condition THEN instructions;  
    ELSE instructions;  
END IF;
```



Structure alternative: CASE (1)

- Choix selon la valeur d'une variable

```
CASE variable
  WHEN valeur1 THEN action1;
  WHEN valeur2 THEN action2;
  ...
  ELSE action;
END CASE;
```



Structure alternative: CASE (2)

- Plusieurs choix possibles

```
CASE
    WHEN expression1 THEN action1;
    WHEN expression2 THEN action2;
    ...
    ELSE action;
END CASE;
```



Structures itératives

- LOOP

```
LOOP
    instructions;
    EXIT WHEN (condition);
END LOOP;
```

- FOR

```
FOR (indice IN [REVERSE] borne1..borne2) LOOP
    instructions;
END LOOP;
```

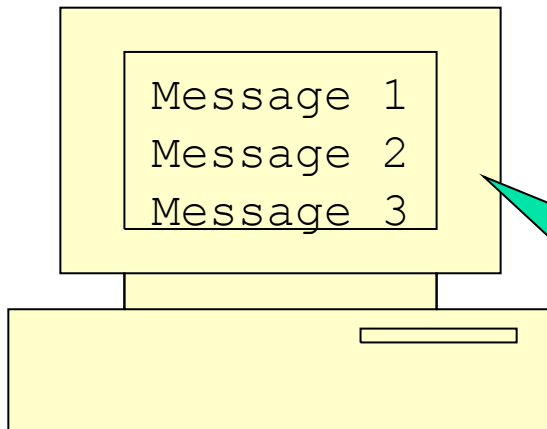
- WHILE

```
WHILE (condition) LOOP
    instructions;
END LOOP;
```


Affichage de résultats *intermédiaires* package DBMS_OUTPUT

- Messages enregistrés dans une mémoire tampon côté serveur
- La mémoire tampon est affichée sur le poste client **à la fin**

Client SQL*Plus



```
SQL> SET SERVEROUTPUT ON
```

Serveur Oracle

```
BEGIN  
DBMS_OUTPUT.PUT_LINE ( 'Message 1' ) ;  
DBMS_OUTPUT.PUT_LINE ( 'Message 2' ) ;  
DBMS_OUTPUT.PUT_LINE ( 'Message 3' ) ;  
END ;
```

Message1
Message2
Message3

Ne pas oublier de le faire coté client



La package DBMS_OUTPUT

- Ecriture dans le buffer avec saut de ligne

```
DBMS_OUTPUT.PUT_LINE (<chaîne caractères>);
```

- Ecriture dans le buffer sans saut de ligne

```
DBMS_OUTPUT.PUT (<chaîne caractères>);
```

- Opérateur de concaténation : ||

```
DBMS_OUTPUT.PUT_LINE('Affichage des ' || n || ' premiers');  
DBMS_OUTPUT.PUT_LINE ('nombres en ligne');  
FOR i IN 1..n LOOP  
    DBMS_OUTPUT.PUT (i);  
END LOOP;
```



Exemple d'un bloc PL/SQL

Sélectionner l'équipe d'un joueur donné en entrée

```
set serveroutput on
ACCEPT vnj PROMPT 'Entrer le nom d'un joueur: ';
DECLARE
    vequipe Joueur.idEq%TYPE;
    vjoueur Joueur.idjoueur%TYPE;
    vnbre_eq number;
BEGIN
    select idEq, idJoueur into vequipe, vjoueur
    from Joueur
    where nomJoueur='&vnj';
    dbms_output.put_line('l'equipe du joueur num' || vjoueur ||
                          ' est:' || vequipe);
END;
```

/



Les blocs c'est bien mais....

- L'utilisateur dispose du code et l'envoie au serveur
 - Dans le contexte de développement d'une application, on ne peut pas donner le code à l'utilisateur final!
 - Risque de modification, de plagiat..
- Le même bloc peut être exécuté plusieurs fois et on transfèrera plusieurs fois toutes les lignes du code au serveur
- Le code n'est pas compilé et donc pas optimisé



Procédures et fonctions stockées



Procédures stockées

- Programme PL/SQL stocké avec la base
- Le programme client exécute ce programme en lui passant des paramètres (par valeur)
- Si le code est bon, le SGBD conserve le programme source (`USER_SOURCE`) et le **programme compilé**
- Le programme compilé est **optimisé** en tenant compte des objets accélérateurs (`INDEX, CLUSTER, PARTITION,...`)

Procédures stockées

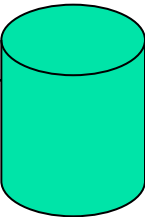
Client SQL*Plus

```
EXECUTE P (p1 , p2) ;
```

Serveur Oracle

```
PROCEDURE P (v1 , v2) AS  
BEGIN  
  Ordre SQL et PL/SQL  
  ...  
END P ;  
/
```

Retour résultats



Avantages des procédures stockées



- **Vitesse**: programme compilé et optimisé
 - Une requête SQL normale est interprétée et optimisée à chaque exécution sur le serveur (allègement du client)
- **Intégrité**: encapsulation de données
 - Droit d'exécution et plus de manipulation
- **Performance**: moins de transferts réseau
 - Plus de transfert de bloc de programme (limiter trafic sur le réseau)
 - Une procédure pour plusieurs utilisateurs (partage)
- **Abstraction**: augmentation du niveau d'abstraction des développeurs clients

Déclaration d'une procédure stockée


```
CREATE [OR REPLACE] PROCEDURE <nom_procedure>
[(variable1 type1,...,variablen typen [OUT])] is
...
-- déclarations des variables et
-- curseurs utilisés dans le corps de la procédure
BEGIN
...
-- instructions SQL ou PL/SQL
EXCEPTION
...
END;
/
```




Fonctions stockées

- Idem procédure mais ne retourne qu'un seul résultat
- Même structure d'ensemble qu'une procédure
- Utilisation du mot clé `RETURN` pour retourner le résultat

```
CREATE [OR REPLACE] FUNCTION <nom_fonction>
[(variable1 type1,...,variablen typen )]
RETURN type_resultat AS
-- déclarations des variables, curseurs, exceptions
BEGIN
-- instructions SQL ou PL/SQL
RETURN(variable);
EXCEPTION
...
END;
/
```



1 ou plusieurs
RETURN



Sélections multi-lignes: CURSEUR

- Deux types de sélections
 - Sélections mono-lignes
 - SELECT ... INTO
 - Sélections multi-lignes
 - Curseurs
 - Renvoi de plusieurs lignes
- Exemple:** Ecrire un bloc PL/SQL qui affiche pour un match donné (no du match rentré au clavier par l'utilisateur) les équipes ayant participé au match (nom des clubs) ainsi que par équipe, les joueurs ayant marqué des points.

Sélection mono-ligne

SELECT INTO

- Toute valeur de colonne est rangée dans une variable avec INTO

```
SELECT nom, adresse, tel INTO vnom, vadresse, vtel
FROM Etudiant WHERE ine=356;
```

```
SELECT nom, adresse, libDip INTO vnom, vadresse, vdip
FROM Etudiant e, Diplome d
WHERE ine=356
AND e.idDip=d.idDip;
```

- Les variables auront préalablement été déclarées:

```
vnom Etudiant.nom%TYPE;      -- préférable à VARCHAR2
vadresse Etudiant.adresse%TYPE;
vtel Etudiant.tel%TYPE;
Vdip Diplôme.libDip%TYPE;
```

Sélection mono-ligne

SELECT INTO

- Variable ROWTYPE (ligne)

```
SELECT * INTO vretud FROM Etudiant  
WHERE ine=356;  
...  
DBMS_OUTPUT.PUT_LINE('Nom etudiant : '||vretud.nom)
```

Où la déclaration sera:

```
vretud Etudiant%ROWTYPE;
```

Sélection multi-lignes: les curseurs



- Curseurs obligatoires pour sélectionner plusieurs lignes
- La curseur contient en permanence l'adresse de la ligne courante
- Un curseur peut être paramétré
 - (Devoir personnel: TP4 Bis)



Gestion des curseurs

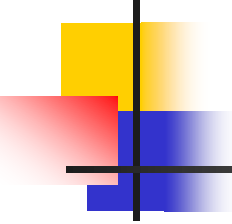
Il existe plusieurs façons de gérer les curseurs

- De façon manuelle
 - Déclaration du curseur
 - OPEN, FETCH, CLOSE
- De façon semi-automatique
 - Déclaration d'un curseur
 - On ne gère pas l'ouverture et la fermeture
- De façon automatique
 - Le curseur n'est pas déclaré
 - Il n'est variable que dans la boucle qui le parcourt



Utilisation des curseurs de façon manuelle

- Déclaration du curseur: **DECLARE**
 - Ordre SQL sans exécution
- Ouverture du curseur: **OPEN**
 - SQL 'monte' les lignes sélectionnées en mémoire
 - Verrouillage préventif possible
- Sélection d'une ligne : **FETCH**
 - Chaque FETCH ramène une ligne dans le programme client
 - Tant qu'il y a des lignes en mémoire, FETCH
- Fermeture du curseur: **CLOSE**
 - Récupération de l'espace mémoire



Variables système des curseurs

- Curseur%**FOUND**
 - Variable booléenne
 - Curseur toujours 'ouvert' (encore des lignes)
- Curseur%**NOTFOUND**
 - Opposé au précédent
 - Curseur 'fermé' (plus de lignes)
- Curseur%**ROWCOUNT**
 - Variable NUMBER
 - Nombre de lignes déjà retournées par le curseur
- Curseur%**ISOPEN**
 - Booléen : curseur ouvert?

Gestion manuelle d'un curseur

```
CREATE OR REPLACE PROCEDURE... Declaration
CURSOR c1 IS SELECT nom, moyenne FROM Etudiant;
Vnom          Etudiant.nom%TYPE;
Vmoyenne      Etudiant.moyenne%TYPE;
e1,e2 NUMBER;
BEGIN
    OPEN c1; Ouverture
    FETCH c1 INTO vnom, vmoyenne; //lecture
    WHILE c1%FOUND LOOP
        IF vmoyenne < 10 THEN e1:=e1+1;
            INSERT INTO liste_refus VALUES (vnom);
        ELSE e2:=e2+1;
            INSERT INTO liste_accept VALUES (vnom);
        END IF;
        FETCH c1 INTO vnom, vmoyenne; //lecture
    END LOOP;
    CLOSE c1; Fermeture
    DBMS_OUTPUT.PUT_LINE(e2 || ' reçus' );
    DBMS_OUTPUT.PUT_LINE(e1 || ' non reçus' );
    COMMIT;

END;
/
```

Exemple: les joueurs ayant participé à un match donné et le nombre de matchs joués

```
ACCEPT vm PROMPT 'Entrer le no du match';
Declare
....
cursor C1 is (select J.idjoueur, j.nomJoueur
               from Joueur J, Joue Jo
               where Jo.idMatch=&vm
               and J.idjoueur=JO.idjoueur and Jo.pointsmarques>0);
begin

dbms_output.put_line('Pour le match N '||&vm);

OPEN C1;
FETCH C1 into vidjoueur, vnomjoueur;
WHILE C1%FOUND loop
    dbms_output.put_line('joueur participant: '||vidjoueur||'nom: '||vnomjoueur);
    select count(*)-1 into nb from Joue
    where Joue.idjoueur=vidjoueur and pointsmarques>0;
    dbms_output.put_line('nombre de matchs joues: '||nb);
FETCH C1 into vidjoueur, vnomjoueur;
end loop;
close C1;
end;
/
```

Gestion semi- automatique des curseurs

```
DECLARE OR REPLACE PROCEDURE LISTE_ETUDIANTS IS

CURSOR c1 IS SELECT nom, moyenne FROM Etudiant;
-- pas de déclaration de variables de réception
e1,e2 NUMBER;
BEGIN
    -- pas d'ouverture du curseur
    -- pas de fetch
    FOR c1_ligne IN c1 LOOP
        IF c1_ligne.moyenne < 10 THEN e1:=e1+1;
            INSERT INTO liste_refus VALUES (c1_ligne.nom);
        ELSE e2:=e2+1;
            INSERT INTO liste_accept VALUES (c1_ligne.nom);
        END IF;
    END LOOP;
    -- pas de close
    DBMS_OUTPUT.PUT_LINE(e2 || ' reçus');
    DBMS_OUTPUT.PUT_LINE(e1 || ' colles');
    COMMIT;

END;
/
```

Variable STRUCT
de réception

Gestion automatique des curseurs

```
DECLARE OR REPLACE PROCEDURE LISTE_ETUDIANTS IS
-- pas de déclaration de curseur
e1,e2 NUMBER;
BEGIN
  FOR c1_ligne IN (SELECT nom, moyenne FROM Etudiant) LOOP
    IF c1_ligne.moyenne < 10 THEN e1:=e1+1;
      INSERT INTO liste_refus VALUES (c1_ligne.nom);
    ELSE e2:=e2+1;
      INSERT INTO liste_accept VALUES (c1_ligne.nom);
    END IF;
  END LOOP;

  DBMS_OUTPUT.PUT_LINE(e2 || ' reçus');
  DBMS_OUTPUT.PUT_LINE(e1 || ' colles');
  COMMIT;
END;
/
```

Variable STRUCT
de réception



Exemple

- Ecrire un bloc PL/SQL qui affiche pour un acteur donné (idacteur entré au clavier par l'utilisateur) le nom et le prénom de l'acteur ainsi que les titres et noms des auteurs des pièces dans lesquels il joue.

AUTEURS (IDAUTEUR, PRENOMAUTEUR, NOMAUTEUR, NAISSANCE, DECES, NATIONALITE)

ACTEURS (IDACTEUR, PRENOMACTEUR, NOMACTEUR, #IDTHEATRE)

PIECES(IDPIECE, TITRE, TYPE, #IDTHEATRE, #IDAUTEUR, #IDMS,#IDTD, #IDTC)

JOUE (#IDACTEUR, #IDPIECE)

```
ACCEPT va PROMPT 'Entrer le no d l'acteur';
```

```
Declare
```

```
vnoma Acteurs.nomacteur%Type;
```

```
vprea Acteurs.prenomacteur%Type;
```

```
begin
```

```
select nomacteur,prenomacteur into vnoma,vprea From Acteurs where idacteur=&va;
```

```
dbms_output.put_line('le nom et prenom de l'acteur N '||&va||'
```

```
son: '||vnoma||' et '||vprea);
```

```
for ligne in (select P.titre, A.nomauteur from Pieces P,Joue J, Auteurs A
```

```
where J.idacteur=&va and J.idpiece=P.idpiece and P.idauteur=A.idAuteur) loop
```

```
dbms_output.put_line('titre de piece auquel il a participé: '||ligne.titre||' dont auteur est: '||ligne.nomauteur);
```

```
end loop;
```

```
End;
```

```
/
```



Curseurs paramétrés

Exemple:

lors de l'affichage du nombre des autres acteurs qui jouent dans la pièce s'affiche également les noms et prénoms de ces acteurs.

.....

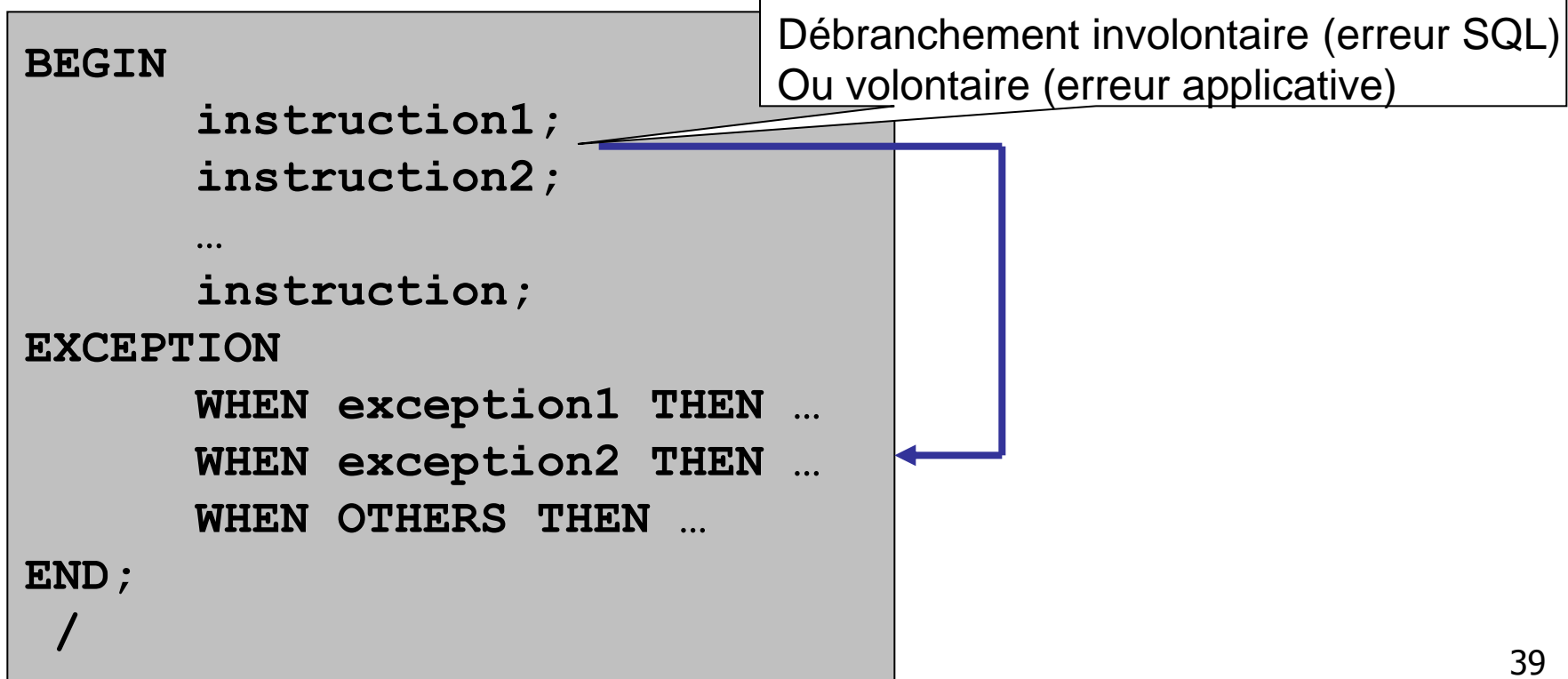
```
cursor C1 is      (select P.idpiece,P.titre,P.idauteur
                   from Pieces P,Joue J
                   where J.idacteur=&va and J.idpiece=P.idpiece);
```

```
cursor C2 (vapièce Pieces.idpiece%Type) is
               (select A.nomacteur,A.prenomacteur
                from Acteurs A,Joue J
                where J.idpiece=vapièce
                   and A.idacteur=J.idacteur
                   and j.idacteur!=&va);
```

Gestion des exceptions

Principe

- Toute erreur (SQL ou applicative) entraîne automatiquement un débranchement vers le paragraphe `EXCEPTION`





Deux types d'exceptions

■ Exceptions SQL (**déclenchement automatique**)

- Déjà définies (pas de déclaration)
 - DUP_VAL_ON_INDEX
 - NO_DATA_FOUND
 - OTHERS
 - TOO_MANY_ROWS
 - ZERO_DIVIDE
 - INVALID_CURSOR
- Non définies ou provenant de blocs, procédures ou triggers appelés ou déclenchés par les instructions du bloc courant
 - Déclaration obligatoire avec le n° d'erreur (SQLCODE)

```
Nomerreur EXCEPTION;  
PRAGMA EXCEPTION_INIT(Nomerreur, no erreur);
```

- Exceptions applicatives
 - Déclaration sans n° erreur + **déclenchement de l'erreur**

```
Nomerreur EXCEPTION;
```

```
RAISE Nomerreur;
```


Exemple de gestion d'exception (1)

```
DECLARE
Tropemprunt EXCEPTION;
i NUMBER;
BEGIN
    i:=1;
    SELECT ...
    i:=2;
    SELECT...
    IF ... THEN RAISE tropemprunt;
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        IF i=1 THEN ...
        ELSE ...
        END IF;
    WHEN tropemprunt THEN ...
    WHEN OTHERS THEN ...
END ;
/
```

Déclenchement
de l'erreur

Déclenchement
automatique

Exemple de gestion d'exception (2)

```
DECLARE
Enfant_sans_parent EXCEPTION;
PRAGMA EXCEPTION_INIT(enfant_sans_parent,-2291);

BEGIN
    INSERT INTO fils VALUES (...);
EXCEPTION
    WHEN enfant_sans_parent THEN ...
    WHEN OTHERS THEN
        dbms_output.put_line(SQLCODE||'-'||SQLERRM);
END;
/
```

Déclenchement automatique

Permettra ensuite de déclarer l'erreur
Avec PRAGMA EXCEPTION_INIT



Exemple: inscription d'un étudiant

```
CREATE PROCEDURE inscription (pid Etudiant.idEtu%TYPE,  
                             pnom Etudiant.nom%TYPE, pdip Diplôme.idDip%TYPE) AS  
CURSOR uv_ins IS SELECT c.iduv AS uv FROM Composition c  
                  WHERE c.idDip=pdip;  
  
BEGIN  
DBMS_OUTPUT.PUT_LINE('Debut inscription de '|| pnom);  
INSERT INTO Etudiant VALUES (pid, pnom,,pdip);  
FOR uv_1 IN uv_ins LOOP  
    INSERT INTO Inscrire VALUES (pid,uv_1.uv);  
END LOOP;  
DBMS_OUTPUT.PUT_LINE('Transaction réussie');  
COMMIT;  
EXCEPTION  
  
...  
END;  
/
```



Exemple avec retour de valeurs

Suppression d'un étudiant (1)

```
CREATE PROCEDURE suppression (pidEtu NUMBER,  
    retour OUT NUMBER) AS  
Inscriptions EXCEPTION;  
PRAGMA_EXCEPTION_INIT(inscriptions,-2292);  
Vnom etudiant.nom%TYPE;  
BEGIN  
SELECT nom INTO vnom FROM Etudiant WHERE idEtu=pidEtu;  
DELETE FROM Etudiant WHERE idEtu=pidEtu;  
DBMS_OUTPUT.PUT_LINE('Etudiant `|| vnom ||' supprimé');  
COMMIT;  
Retour :=0;
```

Exemple avec retour de valeurs

Suppression d'un étudiant (2)

```
EXCEPTION
WHEN NO_DATA_FOUND THEN
DBMS_OUTPUT.PUT_LINE('Etudiant \' || pidEtu || ' inconnu');
Retour :=1;
WHEN incriptions THEN
DBMS_OUTPUT.PUT_LINE('Encore des inscriptions');
Retour :=2;
...
WHEN OTHERS THEN
DBMS_OUTPUT.PUT_LINE(SQLCODE || '      ' || SQLERRM);
Retour:=9;
END;
/
```



Exemple Fonction stockée

```
CREATE OR REPLACE FUNCTION moy_points_marques
(eqj joueur.ideq%TYPE)
RETURN NUMBER is
Moyenne_points_marques NUMBER(4,2);
BEGIN
SELECT AVG(totalpoints) INTO moyenne_points_marques
FROM Joueur WHERE ideq=eqj;
RETURN (moyenne_points_marques);
END;
/
```



Appel des procédures et fonctions

- A partir d'une requête SQL normale (pour les fonctions)

```
SELECT nomjoueur FROM Joueur  
WHERE totalpoints > moy_points_marques('e1');
```

- D'un programme PL/SQL (pour les fonctions et procédures)

```
BEGIN  
...  
IF moy_points_marques(equipe)>20 THEN ...  
END;
```

- D'une procédure stockée ou une autre fonction stockée
- D'un programme externe comme Pro*C



Appel des procédures et fonctions

- A partir de SQL PLUS (pour les procédures) directement

```
EXECUTE inscription(1,3);
```

En récupérant les paramètres

```
VARIABLE ret NUMBER  
ACCEPT vnom PROMPT 'Entrer le nom : '  
...  
EXECUTE inscription('&vnom', '&vdip', :ret);  
PRINT ret
```

Attention: si le parametre est de type 'varchar' il faut le mettre entre `` EXECUTE maproc('&vnom');



Paquetages

- Ensemble de programmes ayant un lien logique entre eux
 - Exemple: Package Etudiant qui peut regrouper tous les programmes écrits sur les étudiants
- Structure
 - Partie visible ou **spécification**
 - Interface accessible au programme appelant
 - Ne contient que les déclarations des procédures ou fonctions **publiques**
 - Variables globales et session
 - Curseurs globaux
 - Partie cachée ou **body**
 - Corps des procédures ou des fonctions citées dans la partie spécification
 - Nouvelles procédures ou fonctions **privées** accessibles uniquement par des procédures ou fonctions du paquetage

Exemple

: partie spécification



```
-- Partie spécification
```

```
CREATE [OR REPLACE] PACKAGE nom_package AS  
Procedure Procedure1(liste des paramètres);
```

```
...
```

```
Function Fonction1(liste des paramètres)  
                                return TYPE;
```

```
...
```

```
Variable_globale1 type1;
```

```
...
```

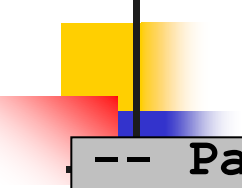
```
CURSOR Curseur_global1 IS...
```

```
...
```

```
END nom_package;
```

```
/
```


Exemple: partie body



```
-- Partie body

CREATE [OR REPLACE] PACKAGE BODY nom_package AS
Procedure Procedure1(liste des paramètres) IS
...
BEGIN
...
END Procedure1;
Function Fonction1(liste des paramètres)
                        RETURN type IS
...
BEGIN
...
RETURN (...);
END Fonction2;
END nom_package;
/
```

Exemple: package Etudiant (1)



```
CREATE PACKAGE Etudiant AS
-- Procédure publique inscription
PROCEDURE inscription (pidet Etudiant.idEtu%TYPE,
                      pnom Etudiant.nom%TYPE, pdip Diplome.idDip%TYPE) ;
-- Procédure publique suppression
PROCEDURE suppression(pidet Etudiant.idEtu%TYPE) ;
END Etudiant;
/

CREATE PACKAGE BODY Etudiant AS
PROCEDURE Inscription (pidet Etudiant.idEtu%TYPE ,
                      pnom Etudiant.nom%TYPE, pdip Diplôme.idDip%TYPE) IS
CURSOR uv_ins IS SELECT c.iduv AS uv FROM composition c
WHERE c.idDip=pdip;
BEGIN
```

Exemple: package Etudiant (2)

```
INSERT INTO Etudiant VALUES (pidet,pnom, pdip);
FOR uv_1 IN uv_ins LOOP
    INSERT INTO inscrire VALUES (pidet,uv_1.uv);
END LOOP;
DBMS_OUTPUT.PUT_LINE('Transaction réussie');
COMMIT;
EXCEPTION ...
END inscription;

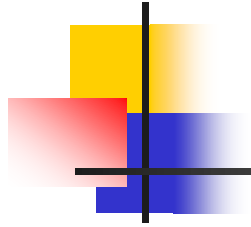
-- fonction privée inscrit_uv
FUNCTION inscrit_uv(pidet Etudiant.idEtu%TYPE)
    RETURN BOOLEAN IS
    Nbre_ins NUMBER;
BEGIN
    SELECT COUNT(*) INTO nbre_ins FROM inscrire
    WHERE idetu=pidet;
    IF nbre_ins > 0 THEN RETURN(TRUE) ELSE RETURN(FALSE);
END IF;
END inscrit_uv;
```

Exemple: package Etudiant (3)

```
PROCEDURE suppression (pidet Etudiant.idEtu%TYPE) AS
BEGIN
  If inscrit_uv(pidet) THEN
    DBMS_OUTPUT.PUT_LINE('Cet étudiant est inscrit à des UV');
    DBMS_OUTPUT.PUT_LINE('Impossible de le supprimer);
  ELSE
    DELETE FROM Etudiant WHERE idetu= pidet;
    DBMS_OUTPUT.PUT_LINE('Etudiant supprimé);
    COMMIT;
  END IF;
END suppression;
END Etudiant;
/
```

- Appel (! Uniquement programmes publics):

```
Etudiant.inscription(nom,dip) ;
```



FIN