Création de triggers PL/SQL

- **Create Trigger:** crée un nouveau déclencheur associé à la table spécifiée et exécute les actions précisées notamment dans < Nom_Trigger >, quand certains évènements surviennent.
- **Le NOM du Trigger doit être unique dans un même schéma**
- **Evènement:** Insert, Update, delete

1

Création de triggers PL/SQL

- **Définition de l'événement du trigger**
  - Elle comprend le type d'instruction SQL qui déclenche le trigger :
    - **DELETE, INSERT, UPDATE** On peut avoir une, deux ou les trois.
    - Pour **UPDATE**, on peut spécifier une liste de colonnes. Dans ce cas, le trigger ne se déclenchera que si l'instruction UPDATE porte sur l'une au moins des colonnes précisée dans la liste.
    - S'il n'y a pas de liste, le trigger est déclenché pour toute instruction UPDATE portant sur la table.

2

Création de triggers PL/SQL

- **Option BEFORE/AFTER** L'appel du déclencheur peut avoir lieu avant le traitement de l'opération ou après.
- **Option BEFORE/AFTER**
  - Elle précise le moment de l'exécution du trigger
  - **BEFORE :** le bloc action est levé avant l'exécution de l'événement
  - **AFTER :** le bloc action est levé après l'exécution de l'événement
- After: Toute modification, dont la dernière insertion, MAJ ou suppression est visible par le déclencheur.

3

Création de triggers PL/SQL

- **Types de triggers**
  - **Le type d'un trigger détermine :**
    - **Quand ORACLE déclenche le trigger,**
    - **Combien de fois** ORACLE déclenche le trigger.
  - **Le type du trigger est défini par l'utilisation de l'une ou l'autre des options suivantes :**
    - **BEFORE, AFTER, FOR EACH ROW**
    - **ORACLE propose deux types de triggers**
      - **les triggers lignes (row)** qui se déclenchent individuellement pour **chaque ligne de la table affectée par le trigger,**
      - **les triggers globaux (statement)** qui sont déclenchées une seule fois.
  - **Si l'option FOR EACH ROW est spécifiée, c'est un trigger ligne, sinon c'est un trigger global.**

Les triggers Lignes

- On peut introduire **une restriction** sur les lignes à l'aide d'une expression logique SQL: c'est la clause **WHEN** :
- Cette expression est évaluée pour chaque ligne affectée par le trigger.
- Le trigger n'est déclenché sur une ligne que si l'expression **WHEN** est vérifiée pour cette ligne.
- **Exemple:** WHEN (new.empno>0) empêchera l'exécution du trigger si la nouvelle valeur de EMPNO est 0, négative ou NULL.

Important:

- Si plusieurs déclencheurs sont définies pour **le même évènement**, ils sont déclenchés suivant l'ordre alphabétique de leurs noms.
- Select ne modifie aucune ligne donc aucun déclencheur ne peut être défini.
- Le bloc d'action
  - Il peut contenir du SQL et du PL/SQL.
  - Il est exécuté si l'instruction de déclenchement se produit et si la clause de restriction **WHEN**, le cas échéant, est évaluée à vrai.

Création de triggers PL/SQL

Les noms de corrélation

- Dans un trigger ligne, on doit pouvoir accéder aux ancienne et nouvelle valeurs de colonne de la ligne.
- Les noms de corrélation permettent de désigner ces deux valeurs : un nom pour l'ancienne et un pour la nouvelle.
  - La nouvelle valeur est appelée **:new.colonne**
  - L'ancienne valeur est appelée **:old.colonne**

- **Exemple :** IF : new.salaire < old.salaire ...
  - Si l'instruction de déclenchement du trigger est INSERT, seule la nouvelle valeur a un sens.
  - Si l'instruction de déclenchement du trigger est DELETE, seule l'ancienne valeur a un sens.

L'option REFERENCING

- Si une table s'appelle **NEW** ou **OLD**, on peut utiliser **REFERENCING** pour éviter l'ambiguïté entre le nom de la table et le nom de corrélation.

- **Exemple: soit la table New (Colonne1, colonne2)**

```
CREATE TRIGGER nomtrigger BEFORE UPDATE
ON  New REFERENCING New AS newnew
FOR EACH ROW
BEGIN
   :newnew.colonne1:= TO_CHAR(:newnew.colonne2);
END;
```


Les prédicats conditionnels

Inserting , Deleting et Updating

- Quand un trigger comporte plusieurs instructions de déclenchement (par exemple INSERT OR DELETE OR UPDATE), on peut utiliser des prédicats conditionnels (INSERTING, DELETING et UPDATING) pour exécuter des blocs de code spécifiques pour chaque instruction de déclenchement.

- Exemple: **Employes** (**NSS**, Nom, Prenom, NumService*) Service(**CodeServ**, Nom_service, NbEmp);

```
CREATE TRIGGER ... BEFORE INSERT OR UPDATE ON Employes
For each row
BEGIN ......
    IF INSERTING THEN  Update Service Set NbEmp= NbEmp+1 where CodeServ=:new. NumService    END IF;
    IF UPDATING THEN UPDATE Service SET NbEmp = NbEmp - 1 WHERE CodeServ = :old.NumService;
                     UPDATE Service SET NbEmp = NbEmp + 1 WHERE CodeServ = :new.NumService;
    END IF;
END;
```

Les prédicats conditionnels

UPDATING peut être suivi d'un nom de colonne

```
Employes (NSS, Nom, Prenom, Salaire)
set serveroutput on;   --Activer le serveur d'affichage
CREATE OR REPLACE TRIGGER update_employe_trigger
AFTER UPDATE OF salaire ON employes
FOR EACH ROW
BEGIN
  IF UPDATING ('SALAIRE') THEN
    IF (:NEW.salaire) < (:OLD.salaire) THEN
          dbms_output.put_line('Le salaire de l employe '|| :NEW.Nom|| ':  ne change pas ');
    END IF;
  END IF;
END;
/
```

10

Manuel PL/SQL

BLOC PL/SQL

```
[DECLARE ... déclaration et initialisation]
BEGIN
... instructions exécutables
[EXCEPTION ... interception des erreurs]
END;
Select * into v1
From etudiant
Where mat=10;
```

Exemple

```
DECLARE
Mot    char(5);
note   number(4,2):=10;
x      number(4) :=0;
v1 table%ROWTYPE;    type du tuple d'une table
V1 ETUDIANT%ROWTYPE;
v2 table.attribut%TYPE; type d'un attribut d'une table
v2   ETUDIANT.Mat%TYPE;
```

11

Ordre de traitement des ligne

On ne peut pas gérer l'ordre des lignes traitées par une instruction SQL.

On ne peut donc pas créer un trigger qui dépende de l'ordre dans lequel les lignes sont traitées.

Triggers en cascade

- Un trigger peut provoquer le déclenchement d'un autre trigger.
- ORACLE autorise jusqu'à 32 triggers en cascade à un moment donné

12

Activation d'un trigger

- Un trigger peut être activé ou désactivé.
- S'il est désactivé, ORACLE le stocke mais l'ignore.
- On peut désactiver un trigger si :
  - il référence un objet non disponible
  - on veut charger rapidement un volume de données important ou recharger des données déjà contrôlées.
- Par défaut, un trigger est activé dès sa création.

13

Activation d'un trigger

- Pour désactiver un trigger, on utilise l'instruction **ALTER TRIGGER** avec l'option **DISABLE** :
  ALTER TRIGGER nomtrigger **DISABLE**;
- On peut désactiver tous les triggers associés à une table avec la commande :
  ALTER TABLE nomtable **DISABLE ALL TRIGGERS**;
- A l'inverse on peut réactiver un trigger :
  ALTER TRIGGER nomtrigger **ENABLE**;
- ou tous les triggers associés à une table :
  ALTER TABLE nomtable **ENABLE ALL TRIGGERS**;

14

Recherche d'information sur les triggers

- Les définitions des triggers sont stockées dans les tables de la métabase, notamment dans les tables
  - **USER_TRIGGERS:** Cette vue contient des informations sur les triggers définis par l'utilisateur connecté
  - **ALL_TRIGGERS :** Cette vue contient des informations sur tous les triggers accessibles à l'utilisateur connecté, y compris ceux définis par d'autres utilisateurs. Cependant, seuls les objets auxquels l'utilisateur connecté a accès sont inclus dans cette vue.
  - **DBA_TRIGGERS:** Cette vue contient des informations sur tous les triggers de la base de données, quelle que soit l'utilisateur qui les a définis. Cette vue est accessible uniquement par les utilisateurs disposant du privilège DBA.

15

Gestion des exceptions

- Si une erreur se produit pendant l'exécution d'un trigger, toutes les mises à jour produites par le trigger ainsi que par l'instruction qui l'a déclenché sont défaites.
- On peut introduire des exceptions en provoquant des erreurs.
  - Une exception est une erreur générée dans une procédure PL/SQL.
  - Elle peut être prédéfinie ou définie par l'utilisateur.
  - Un bloc PL/SQL peut contenir un bloc EXCEPTION gérant les différentes erreurs possibles avec des clauses WHEN.
  - Une clause **WHEN OTHERS THEN ROLLBACK**; gère le cas des erreurs non prévues.

16

Exception prédéfinies

- **NO_DATA_FOUND** cette exception est générée quand un SELECT INTO ne retourne pas de lignes
- **DUP_VAL_ON_INDEX** tentative d'insertion d'une ligne avec une valeur déjà existante pour une colonne à index unique
- **ZERO_DIVIDE** division par zéro

17

Deux types d'exceptions

- **Exceptions SQL**
  - Déjà définies (pas de déclaration)
    - DUP_VAL_ON_INDEX
    - NO_DATA_FOUND
    - OTHERS
  - Non définies
    - Déclaration obligatoire avec le n° erreur (**sqlcode**)

```
nomerreur   EXCEPTION;
PRAGMA  EXCEPTION_INIT(nomerreur,   n°erreur);
```

18

Gestion des exceptions

```
DECLARE
  NomErreur EXCEPTION;
  PRAGMA EXCEPTION_INIT(NomErreur, -20001);
BEGIN
   ….
  RAISE_APPLICATION_ERROR(-20001, 'Une erreur s est produit lors du traitement');
  EXCEPTION
      WHEN NomErreur THEN
        DBMS_OUTPUT.PUT_LINE(' Erreur : ' || SQLCODE || ' - ' || sqlerrm );
      WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('Erreur : ' || SQLCODE || ' - ' || sqlerrm);
END;
```

19

Gestion des Exceptions

Principe:

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

```
BEGIN
    instruction1;
    instruction2;
        ……
    instructionn;
EXCEPTION
    WHEN exception1  THEN   ……..
    WHEN exception2  THEN   ……..
    WHEN OTHERS      THEN
     ……..
END;
```

Débranchement involontaire (erreur SQL) ou volontaire (erreur applicative)

20

Exemple

```
BEGIN
INSERT INTO Employes (NSS, Nom, Prenom, Date-rect)
VALUES (1001, 'Mohamed', 'Ahmed', TO_DATE('01-JAN-96', 'DD-MON-YYYY'));
UPDATE Departement SET ID_Responsable = 1001 WHERE ID_departement = 10;
DELETE FROM Employes WHERE NSS = 1002;
EXCEPTION
  WHEN NO_DATA_FOUND THEN DBMS_OUTPUT.PUT_LINE('Aucune donnée trouvée.');
  WHEN DUP_VAL_ON_INDEX THEN DBMS_OUTPUT.PUT_LINE('Violation d"index unique.');
  WHEN OTHERS THEN DBMS_OUTPUT.PUT_LINE('Une erreur s"est produite : ' || SQLERRM);
END;
```

21

Exemple 1

- Exceptions SQL
  - Exceptions applicatives
    - Déclaration sans n° erreur

```
nomerreur    EXCEPTION;
```

- Exemple

```
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 ……..
END;
```

22

Exemple2

```
set serveroutput on;
CREATE OR REPLACE TRIGGER update_employe_trigger
before UPDATE OF salaire ON employes
FOR EACH ROW
BEGIN
  IF UPDATING ('SALAIRE') THEN
     IF ((:NEW.salaire) < (:OLD.salaire)) THEN
                    RAISE_APPLICATION_ERROR(-20001, 'nouveau salaire < ancien salaire');
                    ELSE dbms_output.put_line(:NEW.Nom|| ': a un nouveau salaire');
     END IF;
  END IF;
END;
/
```

23

Langage PLSQL

24

Langage de bloc PL/SQL

- SQL : langage ensembliste
  - Ensemble de requêtes distinctes
  - Langage de 4ème génération : on décrit le résultat sans dire comment il faut accéder aux données
  - Obtention de certains résultats : encapsulation dans un langage hôte de 3ème génération

25

Langage de bloc PL/SQL

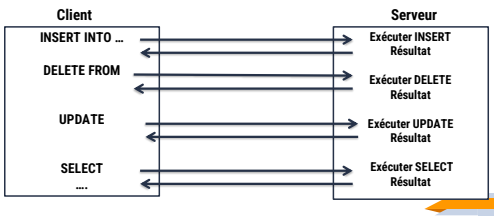
PL/SQL

- 'Procedural Language' : sur-couche procédurale à SQL, boucles, contrôles, affectations, exceptions, ….
  - Chaque programme est un bloc (BEGIN – END)
  - Programmation adaptée pour :
    - Transactions
    - Une architecture Client - Serveur

26

Requêtes SQL

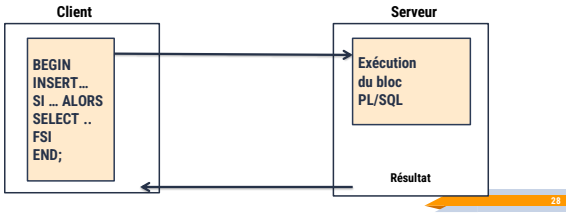
- Chaque requête 'client' est transmise au serveur de données pour être exécutée avec retour de résultats



27

Bloc PL/SQL

- Le bloc de requêtes est envoyé sur le serveur. Celui-ci exécute le bloc et renvoie 1 résultat final



28

Format d'un bloc PL/SQL

- **Section DECLARE : déclaration de**
  - Variables locales simples
  - Variables tableaux
  - cursors
- **Section BEGIN**
  - Section des ordres exécutables
  - Ordres SQL
  - Ordres PL
- **Section EXCEPTION**
  - Réception en cas d'erreur
  - Exceptions SQL ou utilisateur

```
DECLARE
--déclaration
BEGIN
-- exécutions
EXCEPTION
--erreur
END;
/
```

29

Variable simples

- Variables de type SQL

```
nbr                 NUMBER(2) ;
nom                 VARCHAR(30) ;
minimum CONSTANT    INTEGER := 5 ;
salaire             NUMBER(8,2) ;
debut               NUMBER NOT NULL ;
```

- Variables de type booléen (TRUE, FALSE, NULL)

```
fin         BOOLEAN ;
Reponse     BOOLEAN DEFAULT TRUE ;
ok          BOOLEAN := TRUE;
```

30

Variables faisant référence au dictionnaire de données

- **Référence à une colonne (table, vue)**
  - vsalaire employe.salaire%TYPE;
  - vnom etudiant.nom%TYPE;
  - Vcomm vsalaire%TYPE;
- **Référence à une ligne (table, vue)**
  - vemploye employe%ROWTYPE;
  - vetudiant etudiant%ROWTYPE;
- **Variable de type 'struct'**
- Contenu d'une variable : **variable.colonne**
  - vemploye.adresse

31

Instruction PL

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

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

32

Manuel PL/SQL

Si alors

```
IF condition
    THEN instruction;
END IF;
```

Si alors sinon

```
IF condition
    THEN instruction;
     ELSE  instruction;
END IF;
```

Imbrication de condition

```
IF condition  THEN instruction;
ELSEIF condition2  THEN instruction;
ELSEIF …
ELSE  instruction;
END IF;
```

33

Structure alternative : CASE (1)

- Choix selon la valeur d'une variable

```
CASE variable
    WHEN valeur1 THEN action1;

    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;
```


Manuel PL/SQL

```
CASE variable
   WHEN expr1 THEN instruction1;
   WHEN expr2 THEN instruction2; …
   WHEN exprN THEN instructionN;
[ELSE instructionN+1]
END CASE;
```

```
CASE
   WHEN condition1 THEN instruction1;
   WHEN conditionN THEN instruction2; …
   WHEN conditionN THEN instructionN;
[ELSE instructionN+1]
END CASE;
```

Manuel PL/SQL

Boucle POUR

```
FOR indice IN borne1.. borne2LOOP
    instructions ;
END LOOP;
```

Boucle répéter

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

Boucle tant que

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

Affichage à l'ecran

```
DBMS_OUTPUT.PUT_LINE (….|| ….|| …….);
```

37

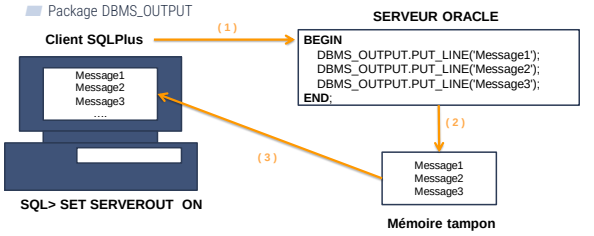
Affichage de résultats intermédiaires

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

38

Affichage de résultats intermédiaires

- Package DBMS_OUTPUT



39

Le package DBMS_OUTPUT

- **Écriture dans le buffer avec saut de ligne**
  - DBMS_OUTPUT.PUT_LINE(<chaîne caractères>);
- **Écriture dans le buffer sans saut de ligne**
  - DBMS_OUTPUT.PUT(<chaîne caractères>);
- **Écriture dans le buffer d'un saut de ligne**
  - DBMS_OUTPUT.NEW_LINE;

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

40

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 Employés WHERE NSS= matricule;
```

```
SELECT nom, adresse, nom_service INTO vnom, vadresse, vnom_serv
FROM Employés e, Service s
WHERE NSS=matricule  AND e.Num_serv=s.num_serv;
```

- Variable ROWTYPE

```
SELECT *  INTO vempl FROM Employés WHERE NSS=matricule;
…
DBMS_OUTPUT.PUT_LINE('Nom Employé : '|| vempl.nom);
```

41

Sélection multi – ligne: Les CURSEURS

- **Principe des curseurs**
  - Obligatoire pour sélectionner plusieurs lignes
  - Zone mémoire (SGA : Share Global Area) partagée pour stocker les résultats
  - Le curseur contient en permanence l'@ de la ligne courante
  - **Curseur implicite**
    - **SELECT t.* FROM** table t **WHERE** ……
    - **t** est un curseur utilisé par SQL
  - **Curseur explicite**
    - DECLARE CURSOR

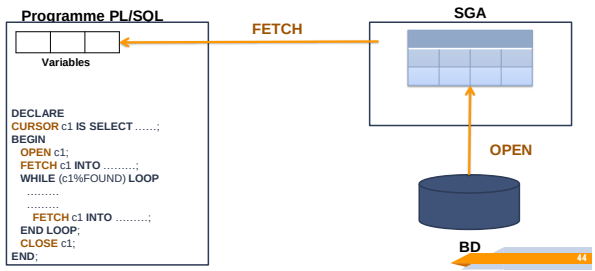
42

Curseur explicite

- **Déclaration du curseur : DECLARE**
  - Ordre SQL sans exécution
- **Ouverture du curseur : OPEN**
  - SQL 'monte' les lignes sélectionnées en SGA
- **Sélection d'une ligne : FETCH**
  - Chaque FETCH ramène une ligne dans le programme client
  - Tant que ligne en SGA : FETCH
- **Fermeture du curseur : CLOSE**
  - Récupération de l'espace mémoire en SGA

43

Traitement d'un curseur



44

Gestion classique d'un curseur

```
DECLARE
   CURSOR c1 IS SELECT nom, moyenne FROM etudiant;
   vnom   etudiant.nom%TYPE;  vmoyenne   etudiant.moyenne%TYPE;
   e1 ,e2 NUMBER;
BEGIN
   OPEN c1;
   FETCH c1 INTO vnom,vmoyenne;
   WHILE c1%FOUND LOOP
     IF vmoyenne < 10 THEN e1:=e1+1; INSERT INTO liste_refus VALUES (vnom);
     ELSE e2:=e2+1; INSERT INTO liste_recus VALUES (vnom);   END IF;
     FETCH c1 INTO vnom,vmoyenne;
  END LOOP;
  CLOSE c1;
   DBMS_OUTPUT.PUT_LINE (TO_CHAR (e2) || 'Reçus ');
   DBMS_OUTPUT.PUT_LINE (TO_CHAR (e1) || 'Refus ');
   COMMIT;
END;
```

45

Les 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%COUNT**
  - Variable number
  - Nombre de lignes déjà retournées
- **Curseur%ISOPEN**
  - Booléen : curseur ouvert ?

Procédures & fonctions Stockées

Procédures Stockées

- **Fonctions**
  - Programme (PL/SQL) stocké dans 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, …)

Procédures Stockées

- Déclaration d'une procédure stockée

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

49

Procédures Stockées

- Exemple: Inscription des étudiants

```
CREATE PROCEDURE inscription (ide varchar2(10),  pnom varchar2(30),
                              spec varchar2(30), ann_ins number)
AS
BEGIN
  DBMS_OUTPUT.PUT_LINE('Début inscription de ||pnom');
  INSERT INTO etudiant VALUES(ide,pnom,spec);
  INSERT INTO inscrire VALUES(ide,ann_ins);
  DBMS_OUTPUT.PUT_LINE('Transaction réussie');
COMMIT;
END;
/
```

50

Appel de la procédure

- A partir de sqlplus

```
ACCEPT vide PROMPT 'Entrer le matricule : '
.............
EXECUTE inscription('&ide','&vnom','&an_ins', &spec');
```

- A partir de PL/SQL

```
inscription (ide,nom,an_ins, spec);
```

51

Les fonctions stockées

- Comme une procédure mais qui ne retourne qu'un seul résultat
- Même structure d'ensemble qu'une procédure
- Utilisation du mot clé RETURN pour retourner le résultat
- Appel possible à partir de :
  - Une requête SQL normale
  - Un programme PL/SQL
  - Une procédure stockée ou une autre fonction stockée

52

Déclaration d'une fonction stockée

```
CREATE [OR REPLACE] FUNCTION nom_fonction
[(paramètre1 type1, …… , paramètren typen)]
RETURN type_résultat IS
-- déclarations de variables,curseurs et exceptions
BEGIN
30
BEGIN
-- instructions PL et SQL
RETURN(variable);
END;
/
```

1 ou plusieurs RETURN

53

Exemple de 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;
/
```

54

Appel d'une fonction

- A partir d'une requête SQL

```
SELECT moy_points_marques ('e1') FROM dual;
```

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

- A partir d'une procédure ou fonction

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