

Chapitre 1

PL/SQL, Triggers et Vues

PL/SQL : Que peut-on faire avec ?

- ❖ Code pour automatiser un traitement
 - Processus périodiques
 - Gestion des exceptions
 - Contraintes d'intégrité élaborées
 - Etc.
- ❖ Applications
 - Auditing
 - Sécurité
 - Vérification

PL/SQL (Procedural Language / Structured Query Language)

- ❖ Langage Procédural pour SQL d'Oracle
- ❖ Proche de Pascal et Ada
- ❖ Facilités pour
 - Variables
 - Conditions
 - Boucles
 - Exceptions
 - Stockage de procédures
- ❖ Implémentations plus ou moins conformes
 - PostgreSQL : PL/pgSQL
 - SQL Server : Transact-SQL
 - DB2 : SQLPL
 - MySQL : Stored procedure
 - etc.

PL/SQL : Structure d'un bloc

- ❖ Un bloc contient 3 parties
 - Déclarations (optionnel)
 - Commandes exécutables (corps)
 - Gestion des exceptions (optionnel)

```
[DECLARE
-- déclarations de types,
-- variables locales au bloc,
-- constantes,
-- exceptions et curseurs]
BEGIN [<nombloc>]
-- instructions PL/SQL et SQL
-- possibilité de blocs imbriqués
[EXCEPTION -- Traitement des erreurs]
END; /* ou END <nombloc> ; */
```

PL/SQL : Déclarations

- ❖ Types
 - issus de SQL : number, date, varchar2, ...
 - boolean, integer, float, real,...
- ❖ Variables
 - De manière générale : <nom_variable> <type_variable> ;
 - Déclaration avec une valeur au départ <Nom_variable> <type> **default** <valeur> ;
- ❖ Exemples
 - Nomcli **varchar2(25)** ;
 - N **number default 2** ;
- ❖ Visibilité d'une variable
 - dans le bloc où elle est déclarée
 - dans les blocs imbriqués (sauf si redéfinie dans bloc imbriqué)
- ❖ Déclarations de constantes
 - <Nom_variable> **constant** <type> := <valeur>;

38

PL/SQL : Exemple

FOURNISSEUR (numfou **NUMBER**, nomfou **VARCHAR2(30)**);

LIVRAISON (numli **NUMBER**, numfou **NUMBER**, dateli **DATE** default sysdate);

Afficher nom du fournisseur de la livraison numéro 10 ?

```
DECLARE
    num NUMBER:= 10;
    nom VARCHAR2 (30) ;
BEGIN
    SELECT nomfou INTO nom
    FROM Fournisseur, Livraison
    WHERE numli= num and Fournisseur.numfou=Livraison.numfou;
    DBMS_OUTPUT.PUT_LINE('La livraison numéro : ' || num || ' concerne le
    fournisseur : ' || nom);
END;
```

40

PL/SQL : Corps

- ❖ Le corps peut comporter des instructions
 - d'affectation
 - SQL : delete, insert, locktable, open, rollback, savepoint, select, set transaction, update...
 - de contrôle (conditionnelles, répétitives)
 - de gestion des erreurs
- ❖ Chaque instruction est terminée par « ; »
- ❖ L'imbrication de blocs est possible, mais pas recommandée
- ❖ Affectation d'une variable
 - Opérateur d'affectation (:=)
 - ex. Produit.LIBPROD := 'Livre' ;
- ❖ Option **into** de l'ordre select

```
Select LIBPROD into designation
from Produit
where REFPROD= 2;
```

39

PL/SQL : Structures de contrôle

- ❖ Structure conditionnelle

```
IF <condition>THEN
<instruction>;...
    [ELSIF <condition> THEN <instruction>; ... <instruction>; ]
    [ELSE <instruction>; ... <instruction>
END IF;
```

- ❖ Boucle répétitive simple

```
LOOP
<instruction>; ... <instruction>;
END LOOP;
```

Sortie d'une boucle
IF <condition> **THEN EXIT** ; **END IF**;
Ou EXIT WHEN <condition> ;

- ❖ Boucle Tant que

```
WHILE <condition>
LOOP <instruction>; ... <instruction>;
END LOOP;
```

- ❖ Boucle For

```
FOR <variable_boucle> IN <borne_inf> ..<borne_sup>
LOOP
<instruction>; ... <instruction>;
END LOOP;
```

41

Structures de contrôle : Exemples

❖ Loop

```
DECLARE
    compteur NUMBER;
    somme NUMBER := 0;
    moyenne NUMBER;
BEGIN
    compteur := 1;
LOOP
    somme := somme + compteur ;
    compteur := compteur + 1;
EXIT WHEN compteur =11;
END LOOP;
moyenne := somme / 10;
DBMS_OUTPUT.PUT_LINE ('La moyenne est '||moyenne);
END;
```

42

Structures de contrôle : Exemples

❖ While

```
DECLARE
    compteur NUMBER;
    somme NUMBER := 0;
    moyenne NUMBER;
BEGIN
    compteur := 1;
WHILE compteur <= 10
LOOP
    somme := somme + compteur ;
    compteur := compteur + 1;
END LOOP;
moyenne := somme / 10;
DBMS_OUTPUT.PUT_LINE ('La moyenne est '||moyenne);
END;
```

43

Structures de contrôle : Exemples

❖ For

```
DECLARE
    compteur NUMBER;
    somme NUMBER := 0;
    moyenne NUMBER;
BEGIN
FOR compteur IN 1 .. 10 LOOP
    somme := somme + compteur ;
END LOOP;
moyenne := somme / 10;
DBMS_OUTPUT.PUT_LINE ('La moyenne est '||moyenne);
END;
```

44

Procédures PL/SQL

❖ Utilisées pour enregistrer des traitements fréquemment utilisés au niveau du noyau **sans valeur de retour**

❖ Syntaxe

```
CREATE OR REPLACE PROCEDURE /* nom */ ( /* paramètres */ )
IS
    /* déclaration des variables locales (Optionnel)*/
BEGIN
    /* instructions */
END;
```

45

Procédures PL/SQL : Exemple

```
CREATE OR REPLACE PROCEDURE compteAREbours (n NUMBER) IS
BEGIN
IF n >= 0 THEN
DBMS_OUTPUT.PUT_LINE ( n );
compteAREbours ( n - 1 );
END IF;
END;
```

❖ Invocation

- En PL/SQL, une procédure s'invoque tout simplement avec son nom
- Sous SQL+, on doit utiliser le mot-clé **CALL**. Par exemple, on invoque le compte à rebours sous SQL+ avec la commande **CALL** compteAREbours(20)

46

Fonctions PL/SQL

- ❖ Utilisées pour enregistrer des traitements fréquemment utilisés au niveau du noyau avec valeur de retour

❖ Syntaxe

```
CREATE OR REPLACE FUNCTION /* nom */ ( /* paramètres */ )
RETURN /* types */
IS
    /* déclaration des variables locales */
BEGIN
    /* instructions */
END;
```

47

Fonctions PL/SQL : Exemple

```
CREATE OR REPLACE FUNCTION Maximum( a NUMBER, b NUMBER)
RETURN NUMBER
IS
BEGIN
IF a < b THEN RETURN b ;
ELSE
RETURN a ;
END IF;
END;
```

❖ Invocation

- En PL/SQL, une fonction s'invoque tout simplement avec son nom
- Sous SQL+, On passe par une pseudo-table nommée **DUAL** de la façon suivante : **SELECT** Maximum(21, 12) **FROM** **DUAL**;

48

Triggers

- ❖ Objectif : Comment assurer une cohérence sémantique plus complexe
 - Lorsqu'une table est modifiée, générer la modification d'une autre
 - Lorsqu'une valeur est insérée, la modifier avant de la stocker dans la BD
 - Etc.
- ❖ Règle : Événement-Condition-Action (ECA)
 - Évènement
 - Insert, delete, update sur une table ou vue
 - Condition
 - Test ou prédicat logique
 - Action
 - Si la condition est satisfaite, code PL/SQL à exécuter

49

Triggers : Syntaxe

```
create [or replace] trigger [schema .] trigger
{ before | after | instead of }
{ ddl_event_clause Insert | Update | Delete
| { ddl_event [or ddl_event]...
  | database_event [or database_event]...
  }
on { [schema .] schema | database }
}
[when ( condition ) ]
{ pl/sql_block | call_procedure_statement }
```

Les attributs des tables sont accessibles à travers les variables :NEW et :OLD

« Instead of » appliqué seulement sur les vues

50

Vues

- ❖ Une vue est une requête stockée qui est interrogée comme une table
- ❖ Création d'une vue

CREATE VIEW <nom-vue> **AS** <requête définissant la vue>

- ❖ Exemple

PRODUIT (numprod **NUMBER**, nomprod **VARCHAR2**(30))

PROPOSER (numfou **NUMBER**, numprod **NUMBER**, prix **NUMBER NOT NULL**)

- Créer une vue appelée « ProduitAffaire » sur les produits dont le prix ne dépasse pas 1000DH (Attributs de la vue : numprod, nomprod, prix)

CREATE VIEW ProduitAffaire **AS**

SELECT numprod, nomprod, prix

FROM PRODUIT, PROPOSER

WHERE prix < 1000 and PRODUIT.numprod = PROPOSER.numprod;

52

Triggers : Exemple

CATALOGUE (numcat **NUMBER**, datedeb **DATE**, datefin **DATE**)

- ❖ Trigger sur la relation CATALOGUE. On empêche l'insertion ou la modification d'un tuple si la date de début de la validité du catalogue est supérieure à celle de sa fin

CREATE OR REPLACE TRIGGER CATALOGUE_bef_ins_upd_row

BEFORE INSERT OR UPDATE

ON CATALOGUE

FOR EACH ROW

DECLARE

dates_error **EXCEPTION**;

BEGIN

IF :new.datedeb > :new.datefin **THEN RAISE** dates_error;

END IF;

EXCEPTION

WHEN dates_error **THEN**

RAISE_APPLICATION_ERROR (-20001, 'Date de début est supérieure à la date de fin');

END;

:new.numcat :new.datedeb :new.datefin



Catalogue		
numcat	datedeb	datefin
...

51

Triggers sur les vues (1/4)

- ❖ Création d'une vue permettant de sélectionner les employés qui ont le job « CLERK ». EMP (empno, ename, deptno, sal, job)

CREATE OR REPLACE VIEW VW_EMP_CLERK **AS**

Select empno "Numéro", ename "Nom", deptno "Dept", sal "Salaire"

From EMP

Where job= 'CLERK';

Select * from VW_EMP_CLERK ;

Numéro	Nom	Dept	Salaire
7369	SMITH	20	880
7876	ADAMS	20	1210
7900	JAMES	30	1045
7934	MILLER	10	1430
9991	Dupontont		
9992	Duboudin		

Table EMP

empno	ename	deptno	sal	job
7369	SMITH	20	880	Clerk
.....

//6 ligne(s) sélectionnée(s).

Triggers sur les vues (2/4)

- ❖ A travers cette vue, les utilisateurs peuvent insérer des lignes dans EMP

```
Insert into VW_EMP_CLERK values( 9994, 'Yan', 20, 2500 ) ;  
Insertion dans la table EMP  
//Ligne créée
```

- ❖ Cependant, on ne peut pas voir leurs insertions car la colonne job (inutile dans ce cas) ne fait pas partie de la vue et donc de l'insertion !

```
Select * from VW_EMP_CLERK ;
```

Numéro	Nom	Dept	Salaire
7369	SMITH	20	880
7876	ADAMS	20	1210
7900	JAMES	30	1045
7934	MILLER	10	1430
9991	DUPONTONT		
9992	DUBOUDIN		

//6 ligne(s) sélectionnée(s).

Table EMP

empno	ename	deptno	sal	job
.....
.....
9994	Yan	20	2500	

54

Triggers sur les vues (3/4)

- ❖ Créer un déclencheur sur vue qui va résoudre ce problème

```
CREATE OR REPLACE TRIGGER TRG_BIR_VW_EMP_CLERK  
INSTEAD OF INSERT -- à la place de l'insertion  
ON VW_EMP_CLERK -- sur la vue VW_EMP_CLERK  
FOR EACH ROW -- pour chaque ligne  
Begin  
Insert into EMP ( empno, ename, deptno, sal, job ) -- on valorise la colonne  
JOB  
Values (:NEW.Numéro, :NEW.Nom, :NEW.Dept, :NEW.Salaire, 'CLERK' ) ;  
End ;
```

55

Triggers sur les vues (4/4)

- ❖ L'utilisateur peut désormais visualiser ses insertions

```
Insert into VW_EMP_CLERK values( 9994, 'Yan', 20, 2500 ) ;  
Insertion dans la table EMP, 1 ligne créée.
```

```
Select * from VW_EMP_CLERK ;
```

Numéro	Nom	Dept	Salaire
7369	SMITH	20	880
7876	ADAMS	20	1210
7900	JAMES	30	1045
7934	MILLER	10	1430
9991	DUPONTONT		
9992	DUBOUDIN		

9994 Yan 20 2500

//7 ligne(s) sélectionnée(s).

Table EMP

empno	ename	deptno	sal	job
.....
.....
9994	Yan	20	2500	Clerk

56

Synthèse

❖ Avantages de PL/SQL

- **Intégration** : Il est possible d'écrire des fonctions complexes de manipulation de données sans recourir à un langage externe
- **Amélioration des performances** : Le code PL/SQL est très proche du moteur Oracle. De plus pour le code stocké, les requêtes qu'il manipule sont pré-compilées, et donc son exécution est optimisée

❖ Triggers

- **Avantage** : le code est centralisé dans la base de données, et se déclenchera quel que soit l'outil utilisé pour mettre à jour ces données
- **Inconvénient** : son exécution utilise des ressources qui peuvent augmenter sensiblement les temps de traitement, notamment lors de modifications massives apportées sur une table

57