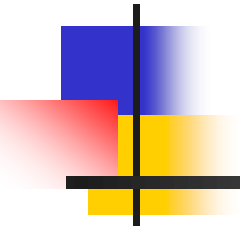
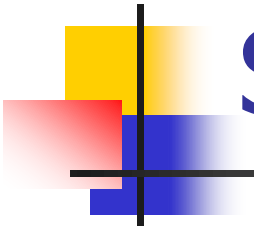


M2106 - Programmation et administration de bases de données

PL/SQL :

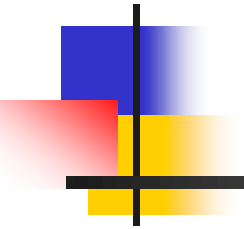
Procédures et fonctions stockées

Packages



Sommaire

- Procédures stockées
- Fonctions stockées
- Packages
- Gestion des erreurs



Procédures stockées

Définition et syntaxe

- Procédure stockée = bloc PL/SQL nommé stocké dans la base de données et exécuté à partir d'une application ou d'autres procédures stockées
- Syntaxe :

```
CREATE [OR REPLACE] PROCEDURE nom_procedure
[(paramètre {IN/OUT/IN OUT} type,…)]
IS
   -- Déclaration de variables
BEGIN
   …
[EXCEPTION]
   …
END;
/
-- Pas de bloc DECLARE. Déclaration des variables entre IS et
   BEGIN si nécessaire.
```

  - `OR REPLACE` : Remplace la description si la procédure existe
  - `Paramètre` : Variable passée en paramètre, utilisable dans le bloc
  - `IN` : Le paramètre est passé en entrée de procédure
  - `OUT` : Le paramètre est valorisé dans la procédure et renvoyé à l'environnement appelant

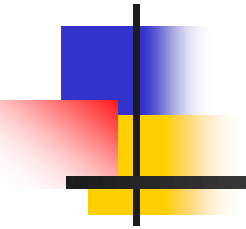
Exemple

- Procédure de suppression d' un département

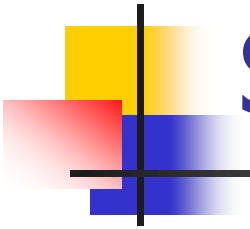
```
CREATE OR REPLACE PROCEDURE supp_dept (pNum IN dept.deptno%TYPE)
   IS
BEGIN
   DELETE FROM emp WHERE deptno=pNum;
   DELETE FROM dept WHERE deptno=pNum;
   COMMIT;
END;
/
```

- Utilisation par SQL*Plus : `EXECUTE supp_dept(10);`
- Utilisation dans un bloc PL/SQL

```
DECLARE
   vNum dept.deptno%TYPE:=10;
BEGIN
   …
   supp_dept (vNum);
   …
END;
/
```



Fonctions stockées



Syntaxe

- Syntaxe :

```
CREATE [OR REPLACE] FUNCTION nom_fonction
[(paramètre [IN] type,…)]
RETURN type
IS
   -- Déclaration de variables
BEGIN
   …
[EXCEPTION]
   …
END;
/
-- Pas de DECLARE
```

  - OR REPLACE
    - Remplace la description si la fonction existe
  - RETURN type
    - Type de la valeur retournée par la fonction

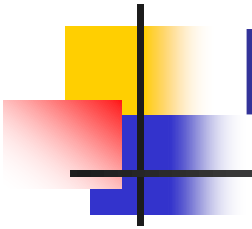
Exemple

- Fonction factorielle :

```
CREATE OR REPLACE FUNCTION factorielle (pNb IN NUMBER)
   RETURN NUMBER
IS
BEGIN
   IF pNb = 0 THEN
              RETURN 1;
   ELSE
              RETURN ((pNb * factorielle(pNb-1)));
   END IF;
END;
/
-- Ne pas mettre VARCHAR2(X) ou NUMBER (X,Y) dans les types au
   niveau des paramètres, mais seulement VARCHAR2 ou NUMBER
```

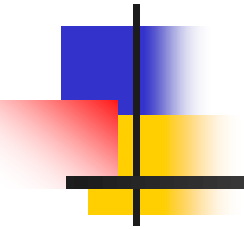
- Utilisation :

```
SELECT factorielle(5) FROM DUAL;
-- OU
DECLARE vResultat Number;
BEGIN
   vResultat := factorielle(5);
   DBMS_OUTPUT.PUT_LINE(vResultat);
END;
```

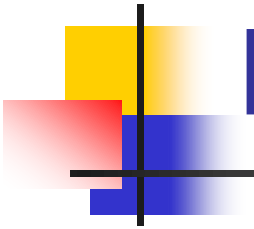


Remarques :

- La vue système `USER_SOURCE` permet de visualiser les fonctions et procédures définies par l'utilisateur.
- Pour supprimer une fonction ou procédure stockée :
  - `DROP PROCEDURE nom_procedure;`
  - `DROP FUNCTION nom_fonction;`
- Vous pouvez visualiser les erreurs de compilation de vos procédures/fonctions en utilisant la commande `show errors;` après création de la procédure ou fonction.



Packages



Package ?

- Objet du schéma qui regroupe logiquement des éléments PL/SQL liés, tels que les types de données, les fonctions, les procédures et les curseurs.
- Se divisent en deux parties : un en-tête ou spécification et un corps (body).
  - Partie spécification : décrit le contenu du package, le nom et les paramètres d'appel des fonctions et des procédures.
  - Body : code.
  - Buts séparation des spécifications et du code :
    - déployer un package sans que l'utilisateur puisse visualiser le code
    - faire évoluer simplement le code pour répondre à de nouvelles règles

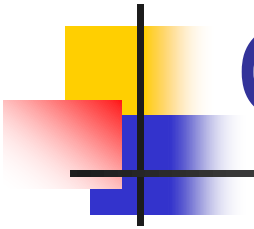
En-tête du package

- La portée de tous les éléments définis dans la spécification du package est globale pour le package.
- La spécification permet de préciser quelles seront les ressources du package utilisables par les applications.
- Toutes les informations pour savoir comment utiliser les ressources du package (paramètres d'appel, type de la valeur renvoyée) doivent être présentes dans cette spécification.
- Syntaxe

```
CREATE PACKAGE nom_package IS
-- Définition de type
-- Déclarations de variables publiques
-- Prototypes des curseurs publiques
-- Prototypes des PROCEDURES et FUNCTIONS publiques
END [nom_package];
```

En-tête du package : exemple

```
CREATE OR REPLACE PACKAGE Gestion_depts IS
    FUNCTION Creation_dept (pDeptno number, pDname
      varchar2, pLoc varchar2) RETURN Number;
    PROCEDURE Supp_dept (pDeptno number);
END Gestion_depts;
/
```



Corps du package

- Contient l'implémentation des procédures et fonctions exposées dans la partie en-tête.
- Contient également des définitions de types et des déclarations de variables dont les portées sont limitées au corps du PACKAGE.
- Peut en plus contenir des définitions locales de curseurs, variables, types, fonctions et procédures pour une utilisation à l'intérieur du package. Ces élément ne seront pas accessibles en dehors du package.
- ATTENTION à respecter ce qui était défini dans l’ en-tête dans le corps du package !

Corps du package : syntaxe

```
 CREATE PACKAGE BODY nom_package IS
-- Définitions de type locaux au package
-- Déclarations de variables locales au package
-- Implémentation des curseurs publics
-- Corps des PROCEDURES et FUNCTIONS locales au package
-- Corps des PROCEDURES et FUNCTIONS publiques

END [nom_package];
```

Corps du package : Exemple

```
CREATE OR REPLACE PACKAGE BODY Gestion_depts AS
    -- Fonction de création d'un nouveau département
    FUNCTION Creation_dept (pDeptno number, pDname varchar2, pLoc
    varchar2) RETURN Number
    IS
          vNbDepts Number(3); -- ATTENTION : Pas de bloc DECLARE
    BEGIN
          INSERT INTO dept VALUES (pDeptno, pDname, pLoc);
          COMMIT;
          SELECT COUNT(*) INTO vNbDepts FROM dept;
          Return vNbDepts;
    END;
    -- Procédure de suppression d'un département
    PROCEDURE Supp_dept (pDeptno Number)
    IS
    BEGIN
          DELETE FROM dept WHERE deptno = pDeptno;
          COMMIT;
    END;
END Gestion_depts;
/
```

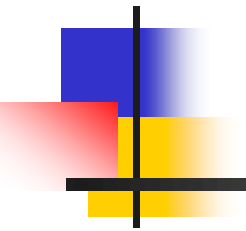
Package : Utilisation

- Les éléments d'un package (variables, procédures, fonctions) sont référencés par rapport au nom du PACKAGE à l'aide de l'opérateur " . "
- Exemples :

```
DECLARE vNb Number;
BEGIN
   vNb := Gestion_depts.Creation_dept(60, 'Marketing',
   'Paris');
   DBMS_OUTPUT.PUT_LINE('Nb de départements : ' ||
     TO_CHAR(vNb));
END;
--------------
BEGIN
   Gestion_depts.Supp_dept(60);
END;
```

Remarques :

- La vue système `USER_SOURCE` permet de visualiser les fonctions, procédures et packages définis par l'utilisateur.
- Pour supprimer un package / fonction / procédure stockée :
  - `DROP PROCEDURE nom_procedure;`
  - `DROP FUNCTION nom_fonction;`
  - `DROP PACKAGE nom_package;`
- Vous pouvez visualiser les erreurs de compilation de vos procédures/fonctions en utilisant la commande `show errors;` après création de la procédure ou fonction.

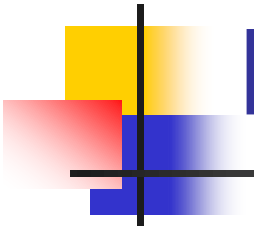


Gestion des exceptions

Traitement des erreurs

- Réalisée dans la partie `EXCEPTION` du bloc PL/SQL. Partie optionnelle, ne doit être définie que si le bloc intercepte des erreurs.
- 2 types d'erreurs :
  - les erreurs internes Oracle ou erreurs prédéfinies,
  - les anomalies dues au programme.
- Après l'exécution du code correspondant au traitement de l'exception, le bloc en cours d'exécution est terminé et l'instruction suivante à être exécutée est celle qui suit l'appel à ce bloc PL/SQL dans le bloc maître.
- La levée des exceptions ne permet pas de continuer normalement le traitement des opérations. Mais en s'appuyant sur la définition de sous-blocs au sein desquels les exceptions sont gérées, il est possible d'exécuter une suite d'instructions même si une exception est levée au cours de l'exécution d'un sous-bloc.
- Règles à respecter
  - Définir et donner un nom à chaque erreur.
  - Associer une entrée dans la section `EXCEPTION` pour chaque nom d'erreur défini dans la partie `DECLARE`
  - Définir le traitement à effectuer dans la partie `EXCEPTION`.

Erreurs prédéfinies



- Toutes les erreurs Oracle possèdent un numéro d'identification unique.
- Elles ne peuvent être interceptées dans un bloc PL/SQL que si un nom est associé au numéro de l'erreur Oracle.
- La liste de ces exceptions prédéfinies est donnée dans le slide suivant.
- Les exceptions sont traitées dans la partie `EXCEPTION` du bloc PL/SQL. À l'intérieur de cette partie, les clauses `WHEN` permettent de connaître le code à exécuter en réponse à une exception levée.

Erreurs prédéfinies

Exception prédéfinie	Erreur Oracle	Valeur de SQLCODE
ACCESS_INTO_NULL	ORA-06530	-6530
CASE_NOT_FOUND	ORA-06592	-6592
COLLECTION_IS_NULL	ORA-06531	-6531
CURSOR_ALREADY_OPEN	ORA-06511	-6511
DUP_VAL_ON_INDEX	ORA-00001	-1
INVALID_CURSOR	ORA-01001	-1001
INVALID_NUMBER	ORA-01722	-1722
LOGIN_DENIED	ORA-01017	-1017
NO_DATA_FOUND	ORA-01403	+100
NOT_LOGGED_ON	ORA-01012	-1012
PROGRAM_ERROR	ORA-06501	-6501
ROWTYPE_MISMATCH	ORA-06504	-6504
SELF_IS_NULL	ORA-30625	-30625
STORAGE_ERROR	ORA-06500	-6500
SUBSCRIPT_BEYOND_COUNT	ORA-06533	-6533
SUBSCRIPT_OUTSIDE_LIMIT	ORA-06532	-6532
SYS_INVALID_ROWID	ORA-01410	-1410
TIMEOUT_ON_RESOURCE	ORA-00051	-51
TOO_MANY_ROWS	ORA-01422	-1422
VALUE_ERROR	ORA-06502	-6502
ZERO_DIVIDE	ORA-01476	-1476

Erreurs prédéfinies : exemple

```
Declare
   vNom EMP.ename%Type;
Begin
   Select ename
   Into vNom
   From EMP
   Where hiredate < to_date('01/01/1970','DD/MM/YYYY');
Exception
   When too_many_rows then
        rollback;
   When no_data_found then
        rollback;
End;
/
```

Erreurs prédéfinies

- Le peu d'exceptions prédéfinies oblige à traiter tous les autres cas dans la clause `WHEN OTHERS` en testant le code erreur SQL. Pour cela, on utilise les fonctions `SQLCODE` et `SQLERRM`.
  - Le code erreur numérique Oracle ayant généré la plus récente erreur est récupérable en interrogeant la fonction `SQLCODE`.
  - Le libellé erreur associé est récupérable en interrogeant la fonction `SQLERRM`.

```
EXCEPTION
   WHEN NO_DATA_FOUND Then
        ...
   WHEN OTHERS THEN
     If SQLCODE = … Then …
         Elsif SQLCODE = … Then …
             ...
     End if ;
END;
/
```

Anomalies programme utilisateur

- En plus des erreurs Oracle, possibilité d'intercepter ses propres erreurs en déclarant des variables dont le type est `EXCEPTION` et en provoquant soi-même le saut dans la section de gestion des erreurs à l'aide de l'instruction `RAISE`

```
DECLARE
   eNomErreur  Exception ;
   …
Begin
   …
   IF(anomalie)
        THEN RAISE eNomErreur ;
   …
EXCEPTION
   WHEN eNomErreur Then
       (traitement);
END ;
/
=> Sortie du bloc après exécution de l'exception
```

Anomalies programme utilisateur : exemple

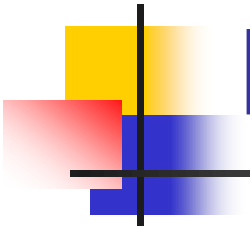
```
-- On arrête et annule les modifications si un salaire dépasse le
   plafond fixé
DECLARE
   CURSOR cEmp IS
        select * from emp;
   eDepassement EXCEPTION;
   vNvSal Number;
BEGIN
   FOR vEmp IN cEmp LOOP
        vNvSal := vEmp.sal * 2;
        IF vNvSal > 10000 THEN
          RAISE eDepassement;
        END IF ;
        UPDATE emp SET sal=vNvSal WHERE empno=vEmp.empno;
   END LOOP;
   COMMIT;
EXCEPTION
   WHEN eDepassement THEN
        Rollback;
END;
/
```

RAISE_APPLICATION_ERROR

- Possibilité de définir ses propres messages d'erreur avec la procédure `RAISE_APPLICATION_ERROR` :
  - `RAISE_APPLICATION_ERROR(numero_erreur, message[, {TRUE | FALSE}])`
  - `numero_erreur` représente un entier négatif compris entre -20000 et -20999
  - `message` représente le texte du message d'une longueur maximum de 2048 octets
  - `TRUE` indique que l'erreur est ajoutée à la pile des erreurs précédentes
  - `FALSE` indique que l'erreur remplace toutes les erreurs précédentes

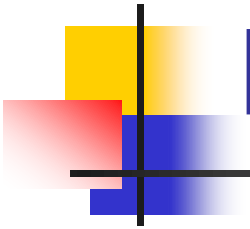
RAISE_APPLICATION_ERROR: exemple

```
DECLARE
   vNoEmp emp.empno%type;
   vNoDept dept.deptno%type;
BEGIN
   vNoDept :=10;
   select empno into vNoEmp from emp where
        deptno=vNoDept;
EXCEPTION
   WHEN NO_DATA_FOUND THEN
        -- il est possible de supprimer le département
        delete from dept where deptno=vNoDept;
   WHEN OTHERS THEN
        RAISE_APPLICATION_ERROR (-20002, 'Suppression
   impossible');
END;
/
```



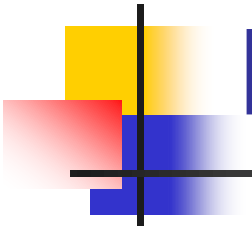
Exercices

- **RAPPEL :**
  - EMP (EMPNO, ENAME, JOB, #MGR, HIREDATE, SAL, COMM, #DEPTNO);
  - DEPT (DEPTNO, DNAME, LOC);



Exercices

1. Créer la fonction `ps_nb_emp_job` qui calcule le nombre d'employés ayant la fonction (job) passée en paramètre.
2. Créer la fonction `ps_sum_sal_job` qui calcule la somme des salaires des employés ayant la fonction (job) passée en paramètre.
3. Créer la procédure `ps_augmente` qui permet d'effectuer pour un employé (numéro) et une augmentation de salaire donnés la modification correspondante dans la table EMP. Une erreur sera levée si l'employé n'existe pas.



Exercices

4. Créer la procédure stockée `ps_insert_emp` qui permet d'insérer un employé dans la table EMP. Seront passées en paramètres toutes ses informations hormis le numéro d'employé (utiliser une séquence pour générer le numéro).
5. Créer la procédure stockée `ps_delete_emp` qui permet de supprimer un employé dont le numéro est passé en paramètre.
6. Créer un package contenant les 2 procédures/fonctions précédentes.