SQL dans un langage de programmation

DEVELOPEMENT

N. Hadj-Rabia

DEVELOPPEMENT

- 1 TRANSACTIONS.
- 2 ARCHITECTURE EN COUCHE.
- 3 COUCHE DONNEES.
- 4 PL/SQL
- 5 PROCEDURES ET PACKAGES.

1 TRANSACTIONS

Nous l'avons vu, le maintien de l'intégrité nécessite souvent que plusieurs opérations élémentaires soient réalisées. C'est le cas pour les contraintes qui mettent en jeu plusieurs tables. Ceci nous a conduit, pour les contraintes que l'on peut décrire, à les déclarer "**deferred**". En effet il y a nécessairement un moment ou une contrainte est "violée". Il faut donc que toutes les opérations élémentaires soient réussies ou que rien ne soit fait.

1 TRANSACTIONS

- **1.1 DEFINITION.**

On appellera **transaction** un ensemble (minimal) d'**opérations élémentaire**s qui, sous réserve de commencer avec des données cohérentes (conformes à l'invariant), se termine normalement avec des données cohérentes, durant l'exécution de la transaction.

Les données peuvent passer par un **état incohérent**.

1 TRANSACTIONS

- **1.2 GESTION DES TRANSACTIONS.**

Le système de gestion des transactions doit assurer la règle du **"tout ou rien"**:

- Dés qu'une transaction est terminée, une autre débute automatiquement.
- Une transaction se terminera par une "**confirmation**" si toutes les opérations élémentaires ont été exécutées avec succès (les données sont alors cohérentes), par une "**annulation**" sinon.

1.2 TRANSACTIONS - GESTION

COMMIT; fin d'une transaction et confirmation des opérations élémentaires, les données sont modifiées.

ROLLBACK; fin d'une transaction et annulation des opérations élémentaires, les données sont remise dans l'état du début de la transaction.

Le découpage des traitements en transaction, le choix à la fin de la confirmation ou de l'annulation sont de la **responsabilité du programmeur**.

1.2 TRANSACTIONS - GESTION

• Les contraintes "**deferred**" sont vérifiées au moment du **COMMIT**.

• Si une contrainte "**deffered**" n'est pas respectée au moment du COMMIT, **c'est le COMMIT qui échoue** et il y a alors exécution automatique de **ROLLBACK**.

services; **employes**;

FK_chef; **FK_affect**; FK_**chef_nuserv deferred**;

transaction pour créer un service:

insert into services values (**3**,'achat',**99**);

insert into employe values(**99**,…,**3**) ; ;

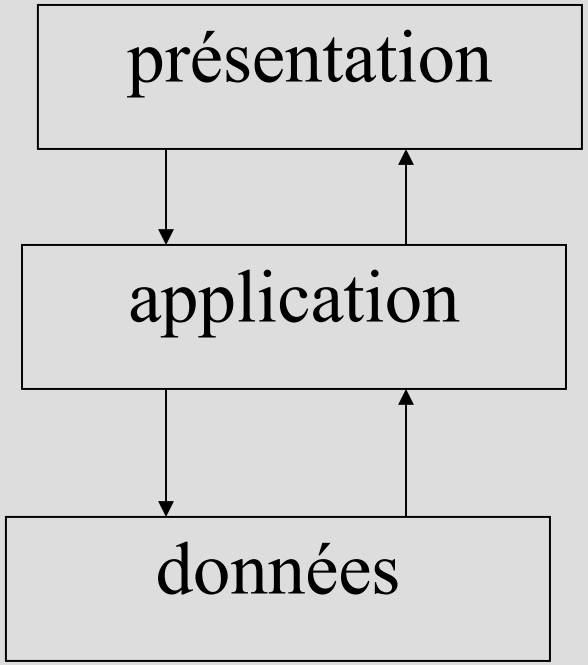
commit;

exception (s'exécute si l'une des instructions échoue)

rollback;

2 ARCHITECTURE EN COUCHE

- **TROIS COUCHES (TROIS TIERS, N-TIERS).**



Relation "**client/serveur**" entre les couches.

Chaque couche rend des "**services**" à la couche supérieure.

Les couches sont indépendantes.

2 ARCHITECTURE EN COUCHE

- **COUCHE DONNEES.**

Fournit des "**services**" d'interrogation et/ou de traitement des données stockées.

En assurant:

L'intégrité des données.

L'utilisation partagée des "**services**".

Un service = une transaction.

Peut retourner un résultat ou lever une exception.

2 ARCHITECTURE EN COUCHE

- **COUCHE APPLICATION.**

  Fournit des "**services**" qui réalisent les opérations sur les données significatives du point de vue des utilisateurs (embaucher un employé ce n'est pas seulement créer la ligne correspondante). Pour cela organise des appels aux "**services**" de la couche données (on parle de "**logique métie**r").

  Reçoit les **résultats** ou les **exceptions**, les exploite et **retourne un résultat** ou **lève une exception.**

2 ARCHITECTURE EN COUCHE

- **COUCHE PRESENTATION.**

Fournit à l'utilisateur **final un interface** qui lui permet de faire appel aux "**services**" de la couche application, de fournir les données nécessaires, et d'en obtenir le résultat ou une exception.

3 COUCHE DONNEES

Recevoir des données de la couche application.

Retourner des données à la couche application.

Lever des exceptions en fournissant les diagnostics, pour la couche application

La couche application doit être capable de recevoir et d'interpréter les exceptions levées par la couche donnée.

Il faut un "**mapping**" de sorte que les données soient vues comme un "ensemble" par la couche donnée (par SQL), et un "tableau" ou une "liste avec accès séquentiel" par la couche application (**par un langage algorithmiqu**e).

4 PL/SQL

- 4.1 DECLARATIONS.
- 4.2 AFFECTATIONS.
- 4.3 STRUCTURES DE CONTROLE.
- 4.4 UTILISATION DE COLLECTION.
- 4.5 LEVER UNE EXCEPTION
- 4.6 TRAITEMENT DES EXCEPTIONS.

4 PL/SQL

• PL/SQL est un langage procédural dans lequel SQL est "naturellement intégré". C'est un langage propre à ORACLE.

• On y trouvera, la déclaration de **variables**, de **types**, des "structures de traitement" **alternatives** et **répétitives**, le traitement des **exceptions** (HANDLER).

4 PL/SQL

• PL/SQL est un langage à structure de bloc:

[**DECLARE** *déclarations des variables, des types, des curseurs et des exceptions*]

BEGIN

instructions PL/SQL et SQL

[**EXCEPTION**

traitements des exceptions]

END;

[contenu] Le contenu entre crochets **n'est pas obligatoire**

4 PL/SQL

- **4.1 DECLARATIONS.**
  - **VARIABLE.**

nom_variable type [:= *val* | DEFAULT *val*];

pour réutiliser une **description** on a :

***nom_colonne*%TYPE**

Exemple : **Employe.salaire%TYPE**

BOOLEAN {**true** | **false**}

declare

Nom varchar2(80) := ’Dupont’;

emp_trouve boolean;

Salaire number(6,2) := 1200.50; /* constante de 6 chiffres dont 2 chiffres après la virgule*/

. . .

begin . . . **end**;

4.1 PL/SQL - DECLARATIONS

– **TYPE RECORD.**

TYPE *nom_type* IS RECORD (*déclaration des champs)*

pour réutiliser une description on a :

nom_table%**ROWTYPE** //Un seul tuple

Exemple : Employe%ROWTYPE

– **EXCEPTION.**

nom_exception **EXCEPTION**;

Declare

employe_rec EMPLOYE%ROWTYPE;

max_sal EMPLOYE.salaire%TYPE;

begin
....
end;

4.1 PL/SQL - DECLARATIONS

- **CURSEUR, REFERENCE CURSEUR.**
  Il faut donner la possibilité de traiter le **résultat d'un SELECT** (un ensemble) de manière algorithmique, ligne par ligne. Ceci est réalisé par la notion de CURSEUR.

- **CURSOR** *nom_curseur* **IS SELECT** .....

  TYPE *nom_type* IS REF CURSOR
  [RETURN *nom_record*]

  sert à envoyer le **résultat** d'un **SELECT** … à un programme écrit dans un langage algorithmique.

```
DECLARE
CURSOR SalCur IS
SELECT * FROM EMP WHERE SAL<6000.00;
emp_tuple EMPLOYE%ROWTYPE;//un seul enregistrement
BEGIN
OPEN SalCur;
LOOP
        FETCH SalCur INTO emp_tuple;

        UPDATE EMPLOYE SET SAL=6500.00
                WHERE numempl=emp_tuple.numempl;

        EXIT WHEN SalCur%NOTFOUND;

END LOOP;
END;
```

```
DECLARE

CURSOR SalCur IS
SELECT * FROM EMPLOYE WHERE SAL<6000.00;
emp_tuple EMPLOYE%ROWTYPE;

BEGIN
FOR employe IN SalCur
        LOOP
        UPDATE EMPLOYE
        SET SAL=6500.00 WHERE nuempl=emp_tuple.nuempl;
        END LOOP;
END;
```

4.1 PL/SQL - DECLARATIONS

– **TYPE COLLECTION.**

TYPE *nom_type* IS VARRAY (*max*) OF *type_scalaire*

nom_collec.count, *nom_collec*.first, *nom_collec*.last, ...

nom_collec(*index*)

sert à recevoir des données (un ensemble) d' un programme écrit dans un langage algorithmique.

```
DECLARE
TYPE LES_NUEMPL IS VARRAY(10) OF
 NUEMPL%TYPE;

COLL LES_NUEMPL;
BEGIN
...
FORALL IND IN COLL.FIRST..COLL.LAST
DELETE FROM EMPLOYE
WHERE NUEMPL = COLL(IND);
```

4 PL/SQL

- **4.2 AFFECTATIONS.**
  - **AFFECTATIONS A UNE VARIABLE DE TYPE SCALAIRE OU RECORD.**

nom_variable := *valeur/expression*;

SELECT … INTO {*nom_variable*,}.. FROM …;

Il n'y a pas d'erreur si l'affectation est réalisé

lève les exceptions prédéfinies :
NO_DATA_FOUND : 0 enregistrement trouvé

et **TOO_MANY_ROWS :** 2 ou plusieurs enregistrements trouvés.

```
declare
employe_rec EMPLOYE%ROWTYPE;
max_sal EMPLOYE.salaire%TYPE;

begin
select numepl, nomempl, hebdo, affect
        into employe_rec  from EMPLOYE where nuempl = 99;

select max(salaire) into max_sal from EMPLOYE;
. . .
end;
```

4.2 PL/SQL - AFFECTATIONS

INSERT/DELETE/UPDATE … RETURNING {*nom_colonne*,}.. INTO {*nom_variable*,}.. ;

DECLARE

LE_CHEF **EMPLOYE.NUEMPL%TYPE**;

BEGIN

DELETE FROM SERVICE WHERE NUSERV = 5

RETURNING CHEF INTO **LE_CHE**F;

….

Affectation du chef dans la variable **LE_CHEF** avant la suppression du service 5

4.2 PL/SQL - AFFECTATIONS

– AFFECTATION A UNE REFERENCE CURSEUR.

OPEN {*variable_curseur*} **FOR** SELECT …;

```
DECLARE
TYPE CUR_EMPL IS REF CURSOR;
LISTE_EMPL CUR_EMPL;
BEGIN
OPEN LISTE_EMPL FOR SELECT * FROM  EMPLOYE;
...
```

4 PL/SQL

- **4.3 STRUCTURES DE CONTRÔLE.**

IF *condition* THEN *instructions*

[{ELSIF *condition* THEN *instructions* }..]

[ELSE *instructions*]

END IF;

GOTO *label*; <<*label*>>

NULL;

toute sorte de "boucle".

4.3 PL/SQL - CONTROLE

Les instructions INSERT, DELETE, UPDATE, (pas OPEN … FOR …) positionnent des "**attributs**" **internes** à oracle et qui peuvent être testés par

sql%found : Le tuple existe

sql%notfound (BOOLEAN) : le tuple **n'existe pas**

sql%rowcount : **nombre** de tuples affectés par les instructions insert, delete ou update

4 PL/SQL

4.4. UTILISATION DE COLLECTIONS.

FORALL *index* IN ***debut***..***fin*** *instr_sql*;

instr_sql est un insert, delete ou update qui doit faire référence à une "collection" indexée par *index*.

Ce n'est pas une "boucle" l'instruction sql interprète la "collection" comme un ensemble.

Positionne une VARRAY d'attribut **sql%bulk_rowcount**

4.4 PL/SQL - COLLECTIONS

```
DECLARE
TYPE LES_NUEMPL IS VARRAY(10)
                        OF EMPLOYE.NUEMPL%TYPE;
COLL LES_NUEMPL;
BEGIN
...
FORALL IND IN COLL.FIRST..COLL.LAST
DELETE FROM EMPLOYE  WHERE NUEMPL = COLL(IND);
...
```

4 PL/SQL

• 4.5 LEVER UNE EXCEPTION.

Des **exceptions** sont levées par les instructions en cas d'échec. L'exécution du bloc est interrompue le contrôle est passé au handler (bloc EXCEPTION) qui doit traiter l'exception levée

SELECT … INTO … lève des exceptions prédéfinies.

INSERT/DELETE/UPDATE … lève des exceptions liées au viol des contraintes d'intégrité décrites. Elles sont identifiées par un numéro **SQLCODE** et associées à un message **SQLERRM**.

OPEN … FOR … ne lève pas d'exception.

4.5 PL/SQL - EXCEPTIONS

Des exceptions peuvent être levées par le programme.

RAISE *nom_exception*;

Lève une exception déclarée, l'exécution du bloc est interrompue le contrôle est passée au handler (bloc EXCEPTION) qui doit traiter l'exception levée.

4.5 PL/SQL - EXCEPTIONS

RAISE_APPLICATION_ERROR (*numero*,'*message d erreur*');

Lève une exception "erreur de l'application". L'exécution du bloc est interrompue le contrôle est passé au programme appelant (l'application) qui reçoit une exception ORACLE avec numéro et le message.

Les numéros autorisés par ORACLE sont dans la plage :

-20000, …, **-20999**.

DECLARE

mon_exception **EXCEPTION**;

BEGIN

SELECT … INTO… (exception prédéfinie)

INSERT …. (**exception SQLCODE**)

…….

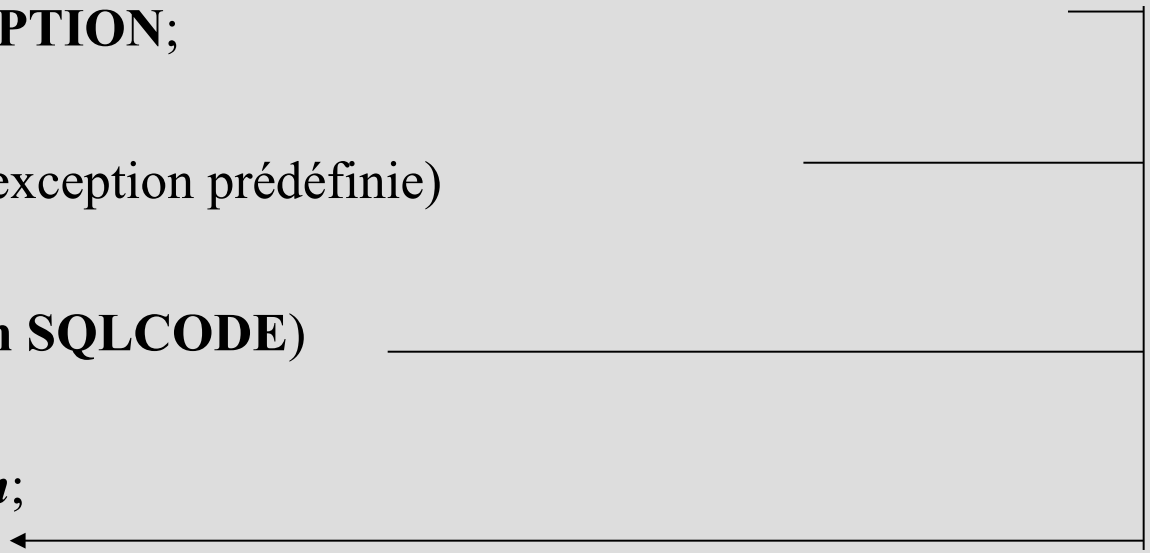
RAISE *mon_exception*;

…….

EXCEPTION

traitement des exceptions

END;



Le "bloc" se termine "normalement"

l'appelant reçoit un signal fin avec "SUCCESS"

BEGIN

…….....

RAISE_APPLICATION_ERROR (*num*, '*message*');

…….....

EXCEPTION

…….....

END;

Le "bloc" se termine "anormalement" l'appelant reçoit l'erreur levée, numéro et message, il doit traiter cette erreur.

Le "bloc" se comporte comme une instruction qui échoue à cause du viol d'une contrainte d'intégrité.

4 PL/SQL

- **4.6 TRAITEMENT DES EXCEPTIONS.**

WHEN {*exception prédéfini* | *nom_exception*}

THEN *instructions*;

WHEN OTHERS THEN *instructions*;

Pour les exceptions autres que prédéfinis ou déclarées, on teste **SQLCODE** dans WHEN OTHERS … .

Une exception non traitée est transmise à l'appelant.

Du HANDLER on ne peut pas retourner au corps du bloc.

4.6 PL/SQL - EXCEPTIONS

SQLCODE = -00001 viol d'une contrainte UNIQUE ou Primary Key

SQLCODE = -01400 viol d'une contrainte NOT NULL.

SQLCODE = -02290 viol d'une contrainte CHECK.

SQLCODE = -02291 viol d'une contrainte REFERENCES.(parent not found)

SQLCODE = -02292 viol d'une contrainte REFERENCES.(child found)

SQLCODE = -02091 échec d'un COMMIT.

Les erreurs qui proviennent des ***Trigger*** :

SQLCODE=-20XXX entre -20000 et -20999

```
DECLARE
PAS_TRAV EXCEPTION;
BEGIN
DELETE FROM TRAV WHERE NUPROJ = 127
AND NUEMPL= 36;
IF SQL%ROWCOUNT = 0 (SQL%NOTFOUND)
        THEN RAISE PAS_TRAV;
END IF;
COMMIT;
EXCEPTION
WHEN PAS_TRAV THEN  ROLLBACK;
 RAISE_APPLICATION_ERROR (-20001,'l'enregistrement n exsite pas') ;
WHEN OTHERS THEN
IF SQLCODE = -02091 THEN ROLLBACK;//erreur du commit
RAISE_APPLICATION_ERROR (-20002, l'employe est responsable du projet) ;
ELSE ROLLBACK;
RAISE_APPLICATION_ERROR –20003, 'erreur innaendue'||SQLCODE)
END;
```

5 PROCEDURE - PACKAGE

On peut dans PL/SQL écrire des PROCEDURES paramètrées. Cela va nous permettre de réaliser les "**services**" de la "couche donnée". Pour nous une procédure ce sera toujours au moins une "**transaction**" (exceptionnellement deux) et un traitement des exceptions qui fournira des diagnostics à la "couche application".

Les procédures peuvent être regroupées en PACKAGE.

5 PROCEDURE - PACKAGE

- **5.1 MANIPULATION DES PROCEDURES ET PACKAGES.**

CREATE [OR REPLACE] PROCEDURE *nom_procedure* [({*arg* [in|out|in out] *type*,}…] [IS|AS] *pl/sql_sous_prog*;

DROP PROCEDURE *nom_procedure;*

CREATE [OR REPLACE] PACKAGE *nom_package* [IS|AS] *pl/sql_package_spec*;

(déclarations communes, **signatures** des procédures)

5.1 PROCEDURE - MANIPULATION

CREATE [OR REPLACE] BODY *nom_package* [IS| AS] *pl/sql_package_body*;

(**source des procédures**)

DROP PACKAGE [BODY] *nom_package*;

CALL [*nom_package.*]*nom_procedure* ({*arg,*}…);

Dans PL/SQL

[*nom_package.*] *nom_procedure* ({*arg,*}…);

On passe d'une logique de "**précondition**" à une logique d'"**exception**".

procédure changer_chef :

entrée : *le_serv* ,*le_chef*

sortie :

exception : pas de service, pas d'employe, employe pas du service, **deja** le chef.

opération :

UPDATE SERVICE SET CHEF = *le_chef*

WHERE NUSERV = *le_serv*;

Comment les exceptions sont levées :

pas de service : attribut **sql%notfound**.

pas d'employé : viol de la contrainte de référence

employé pas du service : viol de la contrainte de référence

déjà le chef : il faut faire la vérification, peut se faire par l'absence de mise a jour, comme pas de service.

Les contraintes de référence étant "**deferred**" pas d'employé et employé pas du service vont correspondre à la même exception.

```
CREATE  PROCEDURE  CHANGER_CHEF
(LE_SERV IN NUMBER, LE_CHEF IN NUMBER) IS
PAS_SERV_OU_CHEF EXCEPTION;
BEGIN
UPDATE SERVICE SET CHEF=LE_CHEF WHERE NUSERV = LE_SERV
AND CHEF != LE_CHEF;
IF SQL%NOTFOUND THEN RAISE
PAS_SERV_OU_CHEF;
END IF;
COMMIT;
```

EXCEPTION

WHEN **PAS_SERV_OU_CHEF** THEN ROLLBACK;

RAISE_APPLICATION_ERROR (-20005,'Le service n existe pas ou l employe en est deja le chef');

WHEN OTHERS

IF **SQLCODE** = -02091 THEN ROLLBACK;//erreur lors du **commit**

RAISE_APPLICATION_ERROR (-20006,'l employe n existe pas ou n est pas du service');

ELSE ROLLBACK ;

RAISE_APPLICATION_ERROR (-20xxx,'erreur inattendue' || SQLCODE);

END IF;

Le type des paramètres est "générique".

Pas de "DECLARE"

La déclaration de PAS_SERV_OU_CHEF **EXCEPTION** et l'utilisation de RAISE PAS_SERV_OU_CHEF fait que toutes les exceptions sont traitées par le handler qui lui lève les exceptions pour l'application.

Dans le handler on traite les exceptions déclarées. Pour les autres (viol de contraintes d'intégrité, exception levée par un trigger) on reçoit un sqlcode que l'on teste pour établir le diagnostic.

Le **WHEN OTHERS** correspond à toutes les exceptions, il faut prévoir l'"erreur inattendue" pour transmettre ces exceptions à l'application.

Le **RAISE_APPLICATION_ERROR** est terminal, il faut donc faire le **ROLLBACK** avant pour terminer la transaction en annulant.

Le ROLLBACK annulant la transaction, il ne peut être fait globalement une fois au début. En effet quand on fait un ROLLBACK on "oublie" ce qui s'est passé pendant la transaction, et donc on "oublie" aussi le SQLCODE et on ne peut plus faire de diagnostics.

Procédure supprimer_service :

entrée : *le_serv , le_nouv_serv*

sortie :

exception : pas de service, pas de nouveau service pour l'affectation du chef, il y a des employés dans le service.

opération : On doit tenir compte de l'ordre temporel.

On récupérera le chef par un "returning" lors de la suppression de *le_serv*.

Il existe un trigger **par exemple** qui supprime les concerne quand un service est supprimé.

DELETE FROM SERVICE WHERE NUSERV = *le_serv* RETURNING CHEF INTO *le_chef*;

la contrainte de référence doit être "deferred".

UPDATE EMPLOYE SET AFFECT = *le_nouv_serv* WHERE NUEMPL = *le_chef*;

Pas de service : attribut **sql%notfound**.

Pas de nouveau service pour l'affectation du chef : viol de la contrainte de référence.

Il y a des employés dans le service : viol de la contrainte de référence.

Il s'agit de la même contrainte de référence il n'y aura qu'une exception. De plus cette contrainte est "**deferred**".

Pour l'insertion dans concerne, je fais en sorte qu'il ne provoque jamais l'erreur "normale" (clé en double) car cela interrompt l'exécution du bloc puis il y a exécution du handler.

```
CREATE PROCEDURE SUPPRIMER_SERV
(LE_SERV IN NUMBER, LE_NOUV_SERV IN NUMBER) IS
PAS_SERV EXCEPTION;
LE_CHEF EMPLOYE.NUEMPL%TYPE;
BEGIN
DELETE FROM SERVICE WHERE NUSERV = LE_SERV
RETURNING CHEF INTO LE_CHEF;
IF SQL%NOTFOUND THEN RAISE PAS_SERV;
UPDATE EMPLOYE SET AFFECT = le_nouv_serv WHERE NUEMPL = le_chef;
END IF;
COMMIT;
```

```
EXCEPTION
WHEN PAS_SERV THEN ROLLBACK;
RAISE_APPLICATION_ERROR (-20xxx,'pas de service a supprimer');
WHEN OTHERS THEN
IF SQLCODE =  -02091 THEN ROLLBACK;
RAISE_APPLICATION_ERROR (-20xxx,'il reste des employes affectes, ou
le service d affectation n existe pas');
ELSE ROLLBACK;
RAISE_APPLICATION_ERROR (-20xxx,'erreur inattendue' ||SQLCODE);
END IF;
END;
```

procédure lire_employe

entrée : *le_serv* (0 pour tous)

sortie : *liste_empl*

exception : si le client fournit un service qui n' existe pas, il reçoit une liste vide, sinon il y a au moins un employé, le chef.

opération :

OPEN *liste_empl* **FOR** SELECT * FROM EMPLOYE

WHERE AFFECT = *le_serv*;

```
TYPE CUR_EMPLOYE IS REF CURSOR;

CREATE PROCEDURE LIRE_EMPLOYE
(LE_SERV IN NUMBER,  LISTE_EMPL OUT CUR_EMPLOYE) IS
BEGIN
IF LE_SERV = 0 THEN
OPEN LISTE_EMPL FOR SELECT * FROM  EMPLOYE;
ELSE
OPEN LISTE_EMPL FOR SELECT * FROM EMPLOYE WHERE
                        AFFECT = LE_SERV;
END IF;
COMMIT;
END;
```

```
EXCEPTION
WHEN OTHERS THEN
        ROLLBACK;
        RAISE_APPLICATION_ERROR
(-20xxx,'erreur inattendue'|| SUBSTR(SQLERRM,1,200));
 END;
```

La variable **SQLERRM** contient le message d'erreur fournit par Oracle. SUBSTR est une fonction qui extrait une sous chaîne (ici les 200 premiers caractères) d'une chaîne de caractères. L'opérateur || est la concaténation de chaînes de caractères.

5.2 Package

CREATE **PACKAGE** EXEMPLE IS

TYPE **CUR_EMPLOYE** IS REF CURSOR;

éventuellement les EXCEPTION

éventuellement des variables (*pour les messages d'erreur par ex*).

PROCEDURE LIRE_EMPLOYE (LE_SERV IN NUMBER,
LISTE_EMPL OUT **CUR_EMPLOYE**);

la signature des autres procédures.

PROCEDURE P2(paramètres) ;

PROCEDURE P3(paramètres) ; …..

END;

CREATE **PACKAGE BODY EXEMPLE** IS

PROCEDURE LISTE_EMPLOYE (LE_SERV IN NUMBER, LISTE_EMPL OUT CUR_EMPLOYE) IS

BEGIN

…

END;

les autres procédures dont la signature est dans la partie spécification. Elles héritent des déclaration qui sont dans la partie spécification.

END;

5.3 UTILISATION PARTAGEE

- La gestion du partage se fait au niveau de la "transaction".

Transaction READ WRITE : exclusion mutuelle.

Transaction READ ONLY : lecture cohérente.

SET TRANSACTION [READ WRITE | READ ONLY];

toujours la première instruction de la transaction.

5.3 PROCEDURE - PARTAGE

- **Transaction READ WRITE :**

En principe pas de SELECT …
exceptionnellement : SELECT … FOR UPDATE;
(le SELECT fonctionne alors pour l'exclusion mutuelle
comme INSERT, DELETE, UPDATE)

Oracle lève une EXCEPTION en cas d'INTERBLOCAGE (DEADLOCK) :

SQLCODE = -00060

SQLERRM = 'deadlock detected while waiting for resource'

```
CREATE PROCEDURE SUPPRIMER_SERV
(LE_SERV IN NUMBER, LE_NOUV_SERV IN
 NUMBER) IS
PAS_SERV EXCEPTION;
LE_CHEF EMPLOYE.NUEMPL%TYPE;
BEGIN
SET TRANSACTION READ WRITE;
DELETE FROM SERVICE WHERE NUSERV =
LE_SERV RETURNING CHEF INTO LE_CHEF;
IF SQL%NOTFOUND THEN RAISE PAS_SERV;
END IF;
…..
COMMIT;
EXCEPTION
….
END ;
```

6.3 PROCEDURE - PARTAGE

- **Transaction READ ONLY :**

Grâce à une gestion sophistiquée du "journal des images avant" lit toujours les données dans l'état cohérent du début de la transaction, il n'y a pas d'EXCEPTION particulière.

```
CREATE PROCEDURE LIRE_EMPLOYE
(LE_SERV IN NUMBER,
LISTE_EMPL OUT CUR_EMPLOYE) IS
BEGIN
SET TRANSACTION READ ONLY;
IF LE_SERV = 0 THEN
OPEN LISTE_EMPL FOR SELECT * FROM
 EMPLOYE;
ELSE
OPEN LISTE_EMPL FOR SELECT * FROM
 EMPLOYE WHERE AFFECT = LE_SERV;
END IF;
COMMIT;
```