PARTIE 2 - LE LANGAGE PROCEDURAL D'ORACLE : LE LANGAGE PL/SQL

I – LES CURSEURS EN PL/SQL

Dès l'instant où on exécute une instruction SQL, il y a création d'un curseur implicite : c'est une zone de travail qui contient des informations permettant l'exécution d'un ordre SQL.

Un curseur **explicite** est un curseur décrit et généré au niveau de la procédure ainsi pour traiter une requête qui retourne plusieurs lignes, l'utilisateur doit définir un curseur qui lui permet d'extraire la totalité des lignes sélectionnées.

L'utilisation d'un curseur nécessite 4 étapes :

1. Déclaration du curseur : Section DECLARE
2. Ouverture du curseur : Section BEGIN
3. Traitement des lignes : Section BEGIN
4. Fermeture du curseur : Section BEGIN OU EXCEPTION

1. La déclaration d'un curseur

La déclaration du curseur permet de stocker l'ordre Select dans le curseur.
Le curseur se définit dans la partie Declare d'un bloc PL/Sql.

Cursor nomcurseur IS Requete_SELECT ;

```
Declare
  Cursor DEPT10 is
        select ename, sal  from emp where deptno=10 order by sal ;
Begin
... ...;
End ;
```

2. L'ouverture du curseur

L'ouverture du curseur réalise :

1. l'allocation mémoire du curseur
2. l'analyse sémantique et syntaxique de l'ordre
3. le positionnement de verrous éventuels (si select for update...)

C'est seulement à l'ouverture du curseur que la requête SQL s'éxécute.
L'ouverture du curseur se fait dans la section Begin du Bloc.

OPEN nomcurseur ;

```
Declare
  Cursor DEPT10 is
        select ename, sal   from emp where deptno=10 order by sal ;
Begin
...Open DEPT10;
.....
End ;
```

3. Traitement des lignes

Après l'exécution du Select les lignes ramenées sont traitées une par une, la valeur de chaque colonne du Select doit être stockée dans une variable réceptrice définie dans la partie Declare du bloc. Le fetch ramène une seule ligne à la fois, pour traiter n lignes il faut une boucle.

FETCH nomcurseur INTO liste_variables ou Nom_enregistrement;

```
create table resultat (nom1 char(10), sal1 number(7,2))
/
Declare         -- programme plsql_ex5.sql
  Cursor DEPT10 is    select ename, sal  from emp where deptno=20 order by sal ;
-- variables réceptrices
  nom emp.ename%TYPE;  -- Variable locale de même type que le champ ename
  salaire emp.sal%TYPE;
Begin
   Open DEPT10;
   Fetch DEPT10 into nom, salaire ;            -- Lecture 1° tuple
   WHILE  DEPT10%found
          loop              -- Tant qu'on trouve une ligne
     If salaire > 2500   then
       insert into resultat values (nom,salaire);
     end if;
     Fetch DEPT10 into nom,salaire ;          -- Lecture tuple suivant
   end loop;
   Close DEPT10;
End ;
/
select * from resultat
/
drop table resultat
/
```

```
SQL>  @ ../gautier/plsql_ex5
     Table créée.
     Procédure PL/SQL terminée avec succès.

     NOM1          SAL1
     ---------- ---------
     JONES         2975
     SCOTT         3000
     FORD          3000
     Table supprimée.
```

Attributs :	Explication :
Nomcurseur%Found	Vrai si exécution correcte de l'ordre SQL
Nomcurseur%Notfound	Vrai si exécution incorrecte de l'ordre SQL
Nomcurseur%Isopen	Vrai si curseur ouvert
Nomcurseur%Rowcount	Donne la nième ligne traitée

Les attributs d'un curseur sont des indicateurs sur l'état d'un curseur. Ils nous fournissent des informations quant à l'exécution de l'ordre. Elles sont conservées par Pl/Sql après l'exécution du curseur.

Ces attributs permettent de tester directement le résultat de l'exécution. Tous les attributs ont un nom.

4. La fermeture du curseur

Après le traitement des lignes, l'étape de fermeture permet d'effectuer la libération de la place mémoire.

CLOSE nomcurseur *Close dept10 ;*

4. Complément : Utiliser une variable de type enregistrement

1° solution : **Nom-de-variable nom_table%rowtype;**

Correspond à la déclaration d'une variable de même type que l'enregistrement (= le tuple = la ligne) de la table.

```
DECLARE
        LigFleur        FLEURS%ROWTYPE ;
        X               number(10,3) ;
BEGIN
        SELECT      *   INTO  LigFleur  from fleur where nofleur=10;  -- 1 seule ligne
         X := LigFleur.Prx *1.1 ;        -- On peut accéder à chaque champ de l'enregistrement
```

Dans un contexte curseur (résultat du select >1 tuple), l'attribut rowtype permet la déclaration implicite d'une structure dont les éléments sont d'un type identique aux colonnes ramenées par le curseur.

Dans la partie déclarative du bloc. **Cursor nomcurseur is ordre_select ;**
nom_structure nomcurseur%ROWTYPE;

Les éléments de la structure sont identifiés par : **nom_structure.nomcolonne**

La structure est renseignée par le Fetch : **Fetch nomcurseur into nom_structure;**

Au préalable afin de bien tester le programme ci-dessous, sous SQL*PLUS

```
SQL> update  pilote
          set comm = null where nopilot='1243'
create table resultat (nom1 char(35), sal1 number(8,2))
/
-- Programme Plsql_ex6.sql
Declare
  Cursor C1 is  select * from pilote where adresse='PARIS';
-- variable réceptrice
  unpilot pilote%rowtype;
  Begin
   Open C1;

   Fetch c1 into unpilot ; -- Lecture 1° tuple
   WHILE  C1%found
```

```
    loop
      If unpilot.comm is not null  then
        insert into resultat values (unpilot.nom, unpilot.sal);
      end if;
      Fetch c1 into unpilot ; -- Lecture tuple suivant
    end loop;
   Close c1;
End ;
/
select * from resultat
/
drop table resultat
/
```

2° solution : Déclarer un type enregistrement

```
TYPE nom_enregistrement IS RECORD
                ( Nom-de-champ1   type,
                  Nom-de-champ2 type,
                  .....) ;
        -- Déclaration d'une variable de ce type
      Une-Variable nom_enregistrement ;
```

```
create table resultat(nom1 char(35), sal1 number(8,2))
/
-- Programme PLSQL_EX7.sql --

DECLARE
  Type  EngPilote IS Record
        (nom_pilote     pilote.nom%type,
         revenu_pilote  pilote.sal%type);
  Unpilot EngPilote;

BEGIN
  -- Exemple d'affectation
  Unpilot.nom_pilote := 'FEDOI';

-- ou Recherche d'un pilote
-- 1 seule ligne pas de curseur
  SELECT nom,sal INTO Unpilot from pilote
           where nopilote = 1333;
   if unpilot.nom_pilote is not NULL then
           Insert into resultat
                   values(unpilot.nom_pilote, unpilot.revenu_pilote);
  end if ;
END;
/
select * from resultat
/
drop table resultat
```

5. Exercices d'application

1. Ecrire tous les programmes donnés dans cette partie et les tester. Adapter les tuples des tables afin de passer en revue les différentes possibilités

2. Ecrire le programme PLSQL_EX8.sql qui permet de retrouver tous les pilotes de Paris ayant une commission non null en déclarant le type d'enregistrement avec RECORD.

3. Vous avez ci-dessous un programme PL/SQL et les tables d'origine. Etudiez ce programme, expliquez succinctement son but et donnez les lignes affichées à la fin de son exécution ainsi que le contenu des deux tables d'origine.

Script majliv.sql.

```
rem Livraison des commandes et mise a jour des stocks

rem Creation de la table TEMOIN
create table CDELIV ( NOCDE number(6) , TEXTE char(60) )
/
rem Bloc de mise a jour
DECLARE
 cursor C_cde is  select COMMANDES.NOCDE,REFART,QTECDE
          from COMMANDES,LIGNESCDE
          where COMMANDES.ETATCDE = 'EC'
          and COMMANDES.NOCDE = LIGNESCDE.NOCDE
          order by COMMANDES.NOCDE;
 V_cde       C_cde%ROWTYPE;
 V_qtestk   ARTICLES.QTESTK%TYPE;
 V_nvqte      number(9);
 V_texte      char(60);
 V_err       BOOLEAN;
 V_cderef   COMMANDES.NOCDE%TYPE;
BEGIN
 open C_cde; -- EXECUTION DU CURSEUR
 fetch C_cde into V_cde; -- LECTURE 1ere LIGNE
 <Bcde>
 while C_cde%FOUND loop
      V_cderef := V_cde.NOCDE; -- DEBUT DE COMMANDE
      V_err := FALSE;
      V_texte := 'Probleme sur article(s) : ';
      commit;
      <Blig>
      while C_cde%FOUND and V_cde.NOCDE = V_cderef loop
           select  QTESTK into V_qtestk -- TRAITEMENT DE LIGNE
                from ARTICLES
                where REFART = V_cde.REFART;
           V_nvqte :>= V_qtestk - V_cde.QTECDE;
           if V_nvqte = 0 then -- STOCK OK
                update ARTICLES    set QTESTK = V_nvqte
                     where REFART = V_cde.REFART;
           else -- STOCK pas OK
                V_texte :=
                rtrim(V_texte)||' '||rtrim(V_cde.REFART);
                V_err := TRUE;
           end if;
          fetch C_cde into V_cde; -- LECTURE LIGNE SUIVANTE
      end loop Blig;
      if V_err then -- FIN DE COMMANDE:Validation ou Annulation
           rollback;
           insert into CDELIV values (V_cderef , V_texte);
           commit;
      else
           update COMMANDES set ETATCDE = 'LI'
                where NOCDE = V_cderef;
           V_texte := 'Commande livree completement';
           insert into CDELIV values (V_cderef, V_texte);
           commit;
      end if;
 end loop Bcde;
 close C_cde; -- FIN DE BLOCK
END;
/
rem Consultation de la table temoin
select * from CDELIV
/
drop table CDELIV
/
```

```
SQL> select * from ARTICLES;

REFART     DESIGNATION            PRIXUNITHT     QTESTK
---------- -------------------- ---------- ----------
AB03       Carpettes                   150         30
AB10       Tapis de Chine             1500          2
AB22       Tapis Persans              1250,1        40
CD50       Chaine HIFI                 735,4         1
ZZ10       Lot de planchettes         1500        105
CD21       Platine laser               500        100
ZZ01       Lot de carpettes            500        150

7 ligne(s) selectionnee(s).

SQL> select * from CDE;

     NOCDE ET REFART          QTECDE
---------- -- ---------- ----------
      1301 EC AB03                5
      1250 EC AB03                8
      1210 EC AB10                3
      1210 EC CD50                4
      1230 EC AB10               20
```

II – MODIFICATION DE DONNEES

Les modifications de données s'effectuent normalement par les instructions SQL : INSERT, UPDATE et DELETE comme nous avons pu le remarquer dans le programme ci-dessus d'exercice : majliv.sql.

PL/SQl permet la possibilité d'utiliser l'option CURRENT OF nom_curseur dans la clause WHERE des instructions UPDATE et DELETE. Cette option permet de modifier ou de supprimer la ligne distribuée par la commande FETCH. Pour utiliser cette option, il faut ajouter la clause FOR UPDATE à la fin de la définition du curseur.

```
-- Programme PLSQL_EX9.sql --

DECLARE
  Cursor C1 is
    select ename, sal from emp
    for update of sal;
  resC1 c1%rowtype;
BEGIN
  Open C1;
  Fetch C1 into resC1;
  While C1%found Loop
    If resC1.sal > 1500 then
      update emp
        set sal = sal * 1.1
        where current of c1;
    end if;
    Fetch C1 into resC1;
  end loop;
  close C1 ;
END;
/
```

Explications :

(... For update of nom_colonne)

Il faut se réserver la ligne lors de la déclaration du curseur par le positionnement d'un verrou d'intention .

(... where current of c1 ;)

Il faut spécifier que l'on veut traiter la ligne courante au Fetch par la clause :

Exercice : Au préalable sous SQL*Plus ajouter une colonne BUDGET de type number à la table DEPT. Dans le programme SQL « Exo4_plsql.sql » mettre à jour cette colonne avec la somme totale des salaires des employés du département.

Résultat à obtenir :

DEPTNO	DNAME	LOC	BUDGET
10	ACCOUNTING	NEW-YORK	8750
20	RESEARCH	DALLAS	11139,8
30	SALES	CHICAGO	9400
40	OPERATION	BOSTON	
50	INFORMATIQUE	NANTES	

III – GESTION DES ERREURS

La section EXCEPTION permet d'affecter un traitement approprié aux erreurs survenues lors de l'exécution du programme PLSQL.

On distingue 2 types d'erreur (ou d'exceptions)

- ⇨ Les erreurs internes d'Oracle
- ⇨ Les anomalies déterminées par l'utilisateur

Après exécution de la procédure d'erreur dans un programme d'un seul bloc, le programme PLSQL est terminée .

a) Les erreurs internes d'Oracle

Une erreur interne est produite quand un bloc PL/Sql viole une règle d'Oracle ou dépasse une limite dépendant du système d'exploitation.

Les noms d'erreurs fournis par Oracle sont regroupées dans ce tableau :

CURSOR_ALREADY_OPEN	STORAGE_ERROR
DUP_VAL_ON_INDEX	TIMEOUT_ON_RESOURCE
INVALID_CURSOR	TOO_MANY_ROWS
INVALID_NUMBER	TRANSACTION_BACKED_OUT
LOGIN_DENIED	VALUE_ERROR
NO_DATA_FOUND	ZERO_DIVIDE
NOT_LOGGED_ON	OTHERS
PROGRAM_ERROR	

Exemple : Utilisation des erreurs prédéfinies

```
DECLARE
   wsal emp.sal%type;

BEGIN
  select sal into wsal
    from emp;

EXCEPTION
  WHEN TOO_MANY_ROWS then ... ;
   -- gérer erreur trop de lignes
  WHEN NO_DATA_FOUND then ... ;
   -- gérer erreur pas de ligne
  WHEN OTHERS       then ... ;
   -- gérer toutes les autres erreurs
END ;
```

b) Les erreurs utilisateurs (externes)

PL/Sql permet à l'utilisateur de définir ses propres exceptions.

La gestion des anomalies utilisateur peut se faire dans un bloc PL/Sql en effectuant les opérations suivantes :

1. Nommer l'erreur (type exception) dans la partie Declare du bloc.
   DECLARE
     Nom_ano Exception;

2. Déterminer l'erreur et passer la main au traitement approprié par la commande **Raise**.
   BEGIN
     If (condition_anomalie) then raise Nom_ano ;

3. Effectuer le traitement défini dans la partie EXCEPTION du Bloc.
   EXCEPTION
     WHEN (Nom_ano) then (traitement);

Syntaxe :

```
DECLARE
  ...
  Nom_ano EXCEPTION;
BEGIN       ...
     instructions ;
     IF (condition_anomalie)
       THEN RAISE Nom_ano
     ...
EXCEPTION
  WHEN Nom_ano THEN (traitement);

END ;
```

On sort du bloc après l'exécution du traitement d'erreur.

Exemple :

```
DECLARE
   Erreur_comm exception ;
   Res_pilot pilote%rowtype ;
BEGIN
  Select * into Res_pilot From Pilote
    Where nopilot = '7100' ;
 If res_pilot.comm > res.pilot.sal Then
   Raise erreur_comm ;
 .......
EXECPTION
  When erreur_comm then
    Insert into erreur values(res_pilot.nom, ' Commission > salaire') ;
  When NO_DATA_FOUND Then
    Insert into erreur values(res_pilot.nopilot, ' non trouvé') ;
END ;
```

c) Visualiser les erreurs non prévues

Le développeur peut utiliser les fonctions propres à PL.SQL Sqlcode et Sqlerrm pour coder les erreurs Oracle en Exception.

Sqlcode : est une fonction propre à PL/Sql qui retourne une valeur numérique : le numéro (généralement négatif) de l'erreur courante.

Sqlerrm : renvoit le libellé de l'erreur courante ou
reçoit en entrée le numéro de l'erreur et renvoie en sortie le message, correspondant au code de l'erreur si spécifié, codé sur 196 octets.

Exemple : Utilisation des erreurs prédéfinies et nommées

```
Create table resultat(number, char(50)
/
DECLARE
   wsal emp.sal%type;
   sal_zero Exception;
   code number;
   lg number;
   mess char(50);

BEGIN
  select sal into wsal from emp;
  if wsal=0 then
     raise sal_zero
  end if;

EXCEPTION
   WHEN sal_zero then
   -- gérer erreur salaire
   WHEN TOO_MANY_ROWS then ... ;
   -- gérer erreur trop de lignes
  WHEN NO_DATA_FOUND then ... ;
   -- gérer erreur pas de ligne
  WHEN OTHERS       then ... ;
   -- gérer toutes les autres erreurs
   code := sqlcode;
   mess := sqlerrm ;
   lg := length(mess);
   insert into resultat values (code,lg,mess);
   commit;
END ;
```

IV– EXERCICE DE SYNTHESE

Ecrire le programme PL/SQL qui permette de saisir une nouvelle affectation.

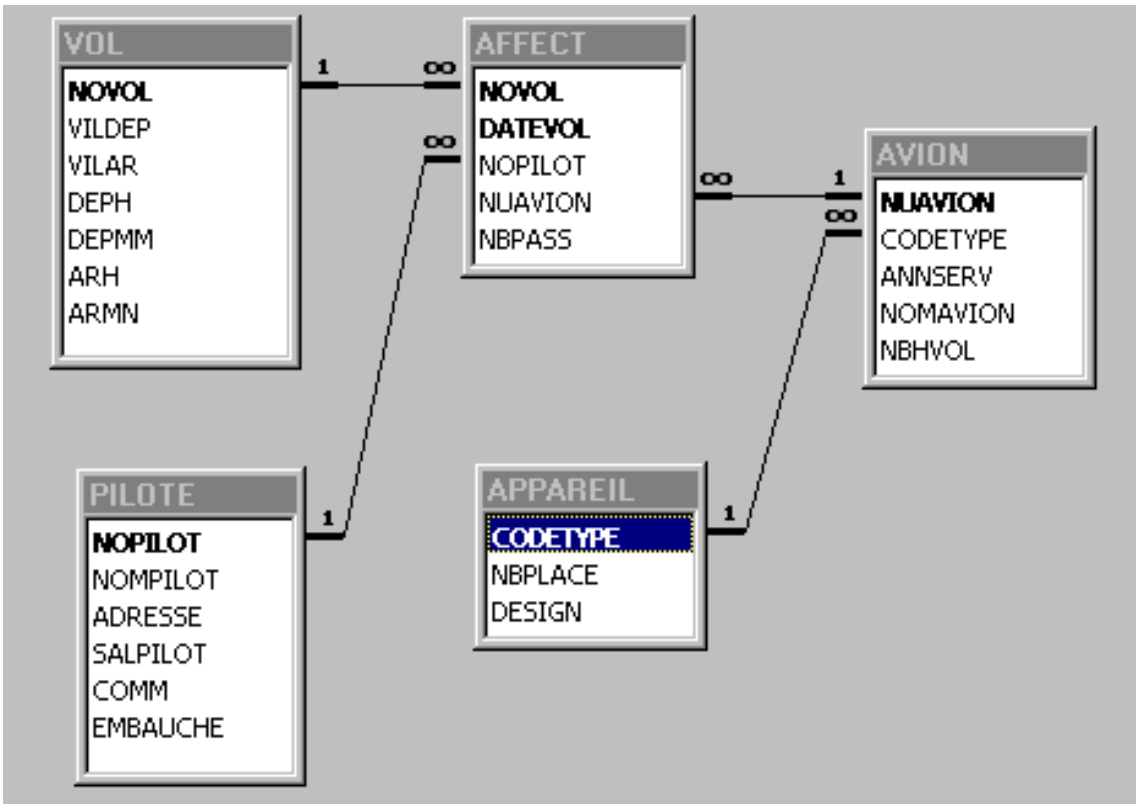
Les données à saisir sont :

- Ville de départ
- Ville d'arrivée
- Numéro d'avion
- Numéro de pilote
- Date de vol

Avant l'ajout de la nouvelle affectation, les contrôles suivants sont à effectuer :

⇨ Un pilote ne peut être affecté à un vol ayant une date de vol antérieure à sa date d'embauche.
⇨ L'avion affecté au vol doit être disponible sur l'aéroport de départ depuis au moins 6 heures .

Remarque : On suppose qu'il n'existe qu'un seul vol entre la ville de départ et la ville d'arrivée.



Vous devez créer une table erreur et une table résultat .

Vous trouverez sur la page suivante : 2 tests possibles. A vous de prévoir d'autres tests afin de passer en revue l'ensemble des possibilités d'anomalies : date embauche > Date vol etc ...

Test N° 1

PARTIE 2 - LE LANGAGE PROCEDURAL D'ORACLE : LE LANGAGE PL/SQL

I – LES CURSEURS EN PL/SQL

1. La déclaration d'un curseur
2. L'ouverture du curseur
3. Traitement des lignes
4. La fermeture du curseur
5. Complément : Utiliser une variable de type enregistrement
6. Exercices d'application

II – MODIFICATION DE DONNEES

III – GESTION DES ERREURS

a) Les erreurs internes d'Oracle

b) Les erreurs utilisateurs (externes)

c) Visualiser les erreurs non prévues

IV– EXERCICE DE SYNTHESE

Bibliographie :

Oracle 7 – Editions Laser – Roger CHAPUIS

Oracle 7 – Langages – Architecture – Administration – Eyrolles – ABDELLATIF, LIMANE et ZEROUAL

Oracle (version 7) _ Editions ENI – Manuel pratique – MEGA +

Le langage PL/SQL – Stage MAFPEN – Christian FISCHER

CORRECTION DES EXERCICES

```
SQL> @majliv

Table creee.


Procedure PL/SQL terminee avec succes.


     NOCDE TEXTE
---------- --------------------------------------------
      1210 Probleme sur article(s) : AB10 CD50
      1230 Probleme sur article(s) : AB10
      1250 Commande livree completement
      1301 Commande livree completement


Table supprimee.

SQL> select * from ARTICLES;

REFART     DESIGNATION          PRIXUNITHT     QTESTK
---------- -------------------- ---------- ----------
AB03       Carpettes                   150         17
AB10       Tapis de Chine             1500          2
AB22       Tapis Persans            1250,1         40
CD50       Chaine HIFI               735,4          1
ZZ10       Lot de planchette          1500        105
CD21       Platine laser               500        100
ZZ01       Lot de carpettes            500        150

7 ligne(s) selectionnee(s).

SQL> select * from CDE;

    NOCDE ET REFART          QTECDE
---------- -- ---------- ----------
      1301 LI AB03                5
      1250 LI AB03                8
      1210 EC AB10                3
      1210 EC CD50                4
      1230 EC AB10               20
```

-- Programme EXO4_PLSQL.sql --

```
DECLARE
  Cursor C1 is
    select * from dept
    for update of budget;
  resC1 c1%rowtype;
  tot   dept.budget%type;

BEGIN
  Open C1;
  Fetch C1 into resC1;
  While C1%found Loop
    Select sum(sal) into tot from emp
      where deptno = resC1.deptno;
      update dept
        set budget = tot
        where current of c1;
    Fetch C1 into resC1;
  end loop;
  CLOSE C1;
END;
/
select * from dept
/
```

```
SQL> @ ../gautier/exo4_plsql

Procédure PL/SQL terminée avec succès.


   DEPTNO DNAME         LOC            BUDGET
   ---------    --------------  -------------        ---------
      10 ACCOUNTING      NEW YORK       8750
      20 RESEARCH          DALLAS       11139,8
      30 SALES                 CHICAGO        9400
      40 OPERATIONS        BOSTON
      50 INFORMATIQUE   NANTES
```

Exercice de synthèse

```
Prompt "Donner la ville de départ"
Accept wvildep
Prompt "Donner la ville d'arrivée"
Accept wvilar
Prompt "Donner le numéro d'avion"
Accept wnuavion
Prompt "Donner le numéro de pilote"
Accept wnupilot
Prompt "Donner la date du vol"
Accept wdate

CREATE TABLE ERREUR(msg1 CHAR(200))
/
CREATE TABLE RESULTAT (msg2 char(200))
/
-- PROGRAMME EXO5_PLSQL.sql ---

DECLARE
   err_date exception;
   err_delai exception;
   date_emb pilote.embauche%type;
   num_vol vol.novol%type;
   date_vol affect.datevol%type;
   heure_dep vol.deph%type;
   heure_arr vol.arh%type;
   mess char(50);
   Cursor c1 is
     Select arh From vol, affect
          where affect.novol = vol.novol
          and datevol ='&wdate'
          and vilar = '&wvildep'
          and nuavion = '&wnuavion';

BEGIN /* début 1 Vérifier sur pilote */
   -- Vérification pour la date du vol --
   Select embauche into date_emb from pilote
      where nopilot = '&wnupilot';
   if '&wdate' < date_emb then
      raise err_date;
   end if;
```

```
insert into resultat
  values ('OK - la date embauche est antérieure à date vol');


BEGIN  /* début 2 sur vol */
   -- récupération du numéro de vol et de l'heure de vol
   select novol, deph into num_vol, heure_dep From vol
     where vildep = '&wvildep'
       and vilar = '&wvilar';

   mess := 'le vol ' || num_vol || ' a été trouvé';
   insert into resultat
     values (mess);

   -- Vérification pour les 6 heures de battement de l'avion --
   Open C1;
   Fetch C1 into heure_arr;

   if C1%found then
      mess := 'heure arrivée avion ' || heure_arr || ' heures';
      insert into resultat
         values (mess);

      If heure_arr + 6 >= heure_dep then
           raise err_delai;
        else
          insert into resultat
            values ('avion arrivé ce jour mais délai OK');
        end if;
   else
      insert into resultat
        values ('OK : avion non arrivé ce jour là');
   end if;

   Close C1;

   -- Tout va bien on fait l'insertion --
   Insert into affect
       values (num_vol, '&wdate','&wnupilot','&wnuavion',0);
   insert into resultat
```

```
          values ('le tuple est ajouté dans affect');

   EXCEPTION
     When NO_DATA_FOUND Then
       insert into erreur
         values ('tuple non trouvé dans la table vol');
     When ERR_DELAI Then
            insert into erreur
              values ('Le délai est trop court pour avion');

   END; /* fin de 2 */

EXCEPTION /* fin de 1 */
   When NO_DATA_FOUND Then
     insert into erreur
        values ('tuple non trouvé dans la table pilote');
   When ERR_DATE Then
     insert into erreur
        values ('Impossible date vol < date embauche');
   mess := sqlerrm;
   insert into erreur
        values (mess);
END;
/
select * from erreur
/
drop table erreur
/
select * from resultat
/
drop table resultat
/
```