ED PL/SQL

(Corrigé)

Par la suite on considère que les tables utilisées par les exercices ont été déjà crées et remplies avec les données nécessaires.

Déclarations, itérations, boucles, instructions conditionnelles

Exercice 1. Soit la table suivante :
VOL(Numvol, Heure_départ, Heure_arrivée, Ville_départ, Ville_arrivée)
Écrivez un programme PL/SQL qui insère le vol AF110 partant de Paris à 21h40 et arrivant à Dublin à 23h10 (hypothèse : le vol n'est pas déjà présent dans la table).
Solution :

```
DECLARE
    v vol%ROWTYPE;
BEGIN
    v.numvol := 'AF110';
    v.heure_départ := to_date('21/11/2013 21:40', 'DD/MM/YYYY hh24:mi');
    v.heure_arrivée := to_date('21/11/2013 23:10', 'DD/MM/YYYY hh24:mi');
    v.ville_départ := 'Paris';
    v.ville_arrivée := 'Dublin';
    INSERT INTO vol VALUES v;
END;
```

Exercice 2. Soit la table RES(NO). Écrivez un bloc PL/SQL qui inséré les chiffres de 1 à 100 dans cette table.
Solution :

```
DECLARE
    nb NUMBER := 1 ;
BEGIN
    LOOP
        INSERT INTO RES
        VALUES(nb) ;
        nb = nb + 1 ;
        EXIT WHEN nb > 100 ;
    END LOOP
END
```

Exercice 3. Écrivez un bloc PL/SQL qui affiche la somme des nombres entre 1000 et 10000.
Solution :

```
DECLARE
    somme NUMBER := 0 ;
BEGIN
    FOR i IN 1000..10000 LOOP
        somme = somme + i ;
    END LOOP
    DBMS_OUTPUT.PUT_LINE('Somme = ' || somme) ;
END
```

Exercice 4. Écrivez un programme PL/SQL qui affiche le reste de la division de 17664 par 171. Ne pas utilisez la fonction MOD.

Solution :

```
DECLARE
    reste NUMBER = 17664 ;
BEGIN
    WHILE reste > 171 LOOP
        reste .= reste – 171 ;
    END LOOP
     DBMS_OUTPUT.PUT_LINE('Le reste de 17664 par 171 est ' || reste)
END
```

Exercice 5. Créez une type tableau pouvant contenir jusqu'à 50 entiers.
1. Créez une variable de ce type, faites une allocation dynamique et dimensionnez ce tableau à 20 emplacements.
2. Placez dans ce tableau la liste des 20 premiers carrés parfaits : 1, 4, 9, 16, 25, ...
3. Affichez ce tableau.

Solution :

```
DECLARE
    TYPE MTAB IS VARRAY (50) OF INTEGER ;
    t MONTAB ;
BEGIN
    t := MONTAB() ;
    t.extend(20) ;
    -- initialisation
    FOR i IN 1..20 LOOP
        t(i) := i*i ;
    END LOOP ;
    -- affichage
    FOR i IN 1..20 LOOP
        DBMS_OUTPUT.PUT_LINE('t(' || i || ') = ' || t(i)) ;
    END LOOP ;
END
```

Exercice 6. Écrire une fonction PL/SQL qui prends en entrée un nombre entier n et retourne le factoriel de ce nombre $n!$. Implémenter deux versions : itérative et récursive. La version récursive est basée sur la relation de récurrence : $n! = n \cdot [(n – 1)!]$

Solution :

```
CREATE OR REPLACE FUNCTION FACT_ITER (N INTEGER)
RETURN INTEGER
IS
      result INTEGER := 1;
BEGIN
      for i in 2..N
      loop
            result := result*i;
      end loop;
      return result;
END FACT_ITER;

CREATE OR REPLACE FUNCTION FACT_REC (N INTEGER)
RETURN INTEGER
IS
BEGIN
      IF (N < 0) THEN
```

```
            RETURN -1;
      ELSIF(N = 0) THEN
            RETURN 1;
      ELSE
            RETURN N*FACT1(N - 1);
      END IF;
END FACT_REC;
```

Curseurs, déclencheurs, relations

Exercice 7. On considère la table suivante:
PILOTE(Matricule, Nom, Ville, Age, Salaire).
Écrivez un programme PL/SQL qui calcule la moyenne des salaires des pilotes dont l'âge est entre 30 et 40 ans.

Solution :

```
DECLARE
      CURSOR curseur1 IS SELECT salaire FROM pilote
      WHERE (Age >= 30 AND Age <=40);
      salairePilote   Pilote.Salaire%TYPE;
      sommeSalaires   NUMBER(11,2) := 0;
      moyenneSalaires NUMBER(11,2);
BEGIN
      OPEN curseur1;
      LOOP
            FETCH curseur1 INTO salairePilote;
            EXIT WHEN (curseur1%NOTFOUND OR curseur1%NOTFOUND IS NULL);
            sommeSalaires := sommeSalaires + salairePilote;
      END LOOP;
      moyenneSalaires := sommeSalaires / curseur1%ROWCOUNT;
      CLOSE curseur1;
     DBMS_OUTPUT.PUT_LINE('Moyenne salaires (pilotes de 30 <E0> 40 ans) : ' ||
moyenneSalaires);
END;
```

Exercice 8. Soit la base de données suivante (simplifiée) de gestion de la mémoire d'un ordinateur :

DISQUE(nom, capacité, vitesse, fabricant);
PARTITION(nomDisque, nomPartition, taille);

Écrivez en PL/SQL le déclencheur (*trigger*) qui lors de l'insertion d'une nouvelle ligne dans la table PARTITION vérifie que la taille totale des partitions sur le disque concerné (y compris la partition qui est en cours d'être ajoutée) ne dépasse pas la capacité du disque. Si tel n'est pas le cas, l'enregistrement de la nouvelle cage ne doit pas être fait et un message doit être affiché pour indiquer cette anomalie.

Solution :

```
CREATE OR REPLACE TRIGGER VérificationDisque
        BEFORE INSERT ON PARTITION
        FOR EACH ROW      /* nécessaire pour avoir accès à :NEW */
DECLARE
        tailleTotale PARTITION.taille%TYPE = 0;
        capacitéDisque DISQUE.capacité%TYPE = 0;
BEGIN
        SELECT SUM(taille) INTO tailleTotale
                FROM PARTITION WHERE nomDisque = :NEW.nomDisque;
```

```
        SELECT capacité INTO capacitéDisque
                FROM DISQUE WHERE nom = :NEW.nomDisque;
        IF tailleTotale + :NEW.taille > capacitéDisque THEN
              RAISE_APPLICATION_ERROR(-20100, 'Pas assez d'espace disque
              pour créer la partition ' || :NEW.nomPartition);
        ENDIF
END
```

Exercice 9. Soit la relation :
EMPLOYE(ID, NOM, DEPARTEMENT, AGE, SALAIRE).
Écrivez un bloc PLSQL qui effectue une augmentation de 200 euros du salaire des employés du département 'Commercial' et qui utilise le dernier curseur implicite pour afficher le nombre d'employés affectés par ce changement.

Solution :

Les curseurs implicites sont créés par PLSQL lors de l'exécution des commandes SQL qui itèrent sur plusieurs items (INSERT, UPDATE, DELETE, SELECT, etc). Les attributs du dernier curseur implicite utilisé sont accessibles par le préfixe SQL : SQL%FOUND, SQL %NOTFOUND, SQL%ROWCOUNT

```
DECLARE
    total NUMBER(2);
BEGIN
    UPDATE EMPLOYE
    SET salaire = salaire + 200
    WHERE DEPARTEMENT = 'Commercial';
    IF SQL%NOTFOUND THEN
        DBMS_OUTPUT.PUT_LINE('Aucun salaire augmenté');
    ELSIF SQL%FOUND THEN
        total := SQL%ROWCOUNT;
        DBMS_OUTPUT.PUT_LINE( total || ' salaires ont été augmentés ');
    END IF;
END;
```

Exercice 10. Soit la relation EMPLOYE de l'exercice précédent. Écrivez un bloc PL/SQL qui affiche les noms des employés du département 'Commercial' qui sont âgés de plus de 40 ans. Utilisez un curseur implicite dans une boucle FOR.

Solution :

```
BEGIN
    FOR emp IN (SELECT  * FROM EMPLOYE WHERE AGE >= 40
            AND DEPARTEMENT = 'Commercial')
    LOOP
        DBMS_OUTPUT.PUT_LINE(emp.NOM) ;
    END LOOP
END
```

Exercice 11. Soit la relation EMPLOYE de l'exercice précédent. Écrivez une procédure PLSQL qui prends en paramètre un NUMBER (age limite) et qui affiche pour chaque département le nombre des employés qui dépassent l'age limite. Utilisez un curseur avec paramètre l'age limite.

Solution :

```
CREATE OR REPLACE PROCEDURE moyenneAge(AgeLim IN NUMBER)
IS
    CURSOR CS(Age_Limite NUMBER) IS
        SELECT DEPARTEMENT AS DNOM, COUNT(*) AS NB
        FROM EMPLOYE
        WHERE AGE > Age_Limite
        GROUP BY DEPARTEMENT ;
```

```
BEGIN
    FOR DEPT IN CS(AgeLim) LOOP
        DBMS_OUTPUT.PUT_LINE(DEPT.DNOM || '   ' || DEPT.NB)
    END FOR
END
```

Exercice 12. Soit la table suivante :

METEO(NOM_VILLE, Température, Humidité)

Écrire une fonction PL/SQL qui prends en entrée le nom d'une ville et retourne la température et l'humidité de cette ville. Gérer aussi par une exception le cas ou la ville n'existe pas.

Solution :

```
TYPE HumTemp IS RECORD(
    HUM METEO.HUMIDITY%TYPE,
    TEMP METEO.TEMPERATURE%TYPE
)
FUNCTION GET_TEMPERATURE (PVILLE IN METEO.NOM_VILLE%TYPE)
RETURN HumTemp
IS
    VAL HUMTEMP;
BEGIN
    VAL.HUM = -10000;
    VAL.TEMP = -10000;
    SELECT HUMIDITE, TEMPERATURE INTO VAL
    FROM METEO
    WHERE VILLE_NOM = PVILLE;
    RETURN VAL;
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE('Ville n\'existe pas');
        RETURN VAL;
END;
```

Exercice 13. Soit la table METEO de l'exercice précédent.

Écrire un déclencheur qui avant l'insertion d'une nouvelle ville dans la table vérifie :

a. Si la température est la plus grande de toutes les villes, afficher un message d'avertissement.

b. Si la ville existe déjà dans la table, ne pas l'insérer une nouvelle fois mais faire la mis a jour seulement.

Solution :

```
CREATE OR REPLACE TRIGGER MTRIGGER
BEFORE UPDATE ON METEO
FOR EACH ROW
DECLARE
    TMAX NUMBER;
    NB NUMBER := 0;
BEGIN
    SELECT MAX(Temperature) INTO TMAX FROM METEO ;
    SELECT COUNT(*) INTO NB FROM METEO M
    WHERE M.VILLE_NOM = :NEW.VILLE_NOM;
    IF (:NEW.TEMPERATURE > TMAX) THEN
        DBMS_OUTPUT.PUT_LINE(-20001,'Temperature MAX');
    ELSIF NB > 0 THEN
        UPDATE METEO SET TEMPERATURE =:NEW.TEMPERATURE
        WHERE VILLE_NOM = :NEW.VILLE_NOM ;
        RAISE_APPLICATION_ERROR(-20001, 'La ville existe déjà');
    END IF
END;
```

Exercice 14. On considère la base de données suivante :
COMPETITION(CODE_COMP, NOM_COMPETITION)
PARTICIPANT(NO_PART, NOM_PART, DATENAISSANCE, ADRESSE, EMAIL)
SCORE(NO_PAR, CODE_COMP, NO_JUGE, NOTE)
Écrire un bloc PLSQL qui lit à la console le nom d'une compétition et qui affiche les participants avec leur score total (la somme de tous les scores par tous les juges). Utilisez un curseur avec paramètre.

Solution :

```
ACCEPT cnom 'Nom de la compétition : ';
DECLARE
    NOMC VARCHAR2(10) := &cnom;
    CURSOR C(PNOM COMPETITION.NOM_COMP%TYPE)IS
            SELECT NOM_PART, SUM(NOTE) AS TOTAL
            FROM COMPETITION C, PARTICIPANT P, SCORE S
            WHERE C.CODE_COMP = S.CODE_COMP AND S.NO_PART = P.NO_PART
                AND C.NOM_COMP = PNOM
            GROUP BY NOM_PART;
BEGIN
    FOR I IN C(NOMC)
    LOOP
        DBMS_OUTPUT.PUT_LINE(I.NOM_PART || ' ' || I.TOTAL)
    END LOOP
END
```

Exercice 15. On considère la table COMPETITION donné dans l'exercice précédent. Écrire un déclencheur qui vérifie que le code d'une compétition commence par les lettres 'CMP' avant son l'insertion dans la table COMPETITION.

Solution :

```
CREATE OR REPLACE TRIGGER VERIFIE_CODE_COMP
BEFORE INSERT OR UPDATE ON COMPETITION
FOR EACH ROW
WHEN (:NEW.CODE_COMP NOT LIKE 'CMP%')
BEGIN
    RAISE_APPLICATION_ERROR(-20001,'COMP_CODE doit commencer par CMP');
END;
```

Exercice 16. On considère la base de données suivante :
CLIENT(CL_ID, CL_NOM, CL_ADDR, CL_VILLE, EMAILID, CONTACT_NO)
MAGAZINE(MAG_ID, MAG_NOM, PRIX_UNITE, TYPE_ABONNEMENT)
ABONNEMENT(CL_ID, MAG_ID, START_DATE, END_DATE)
Écrire une fonction PL/SQL qui retourne le nombre de clients de Dijon qui se sont abonnés au magazine « Vogue » après août 2010. S'il n'y a pas de clients qui remplissent la condition, lancer une exception utilisateur avec un message d'erreur.

Solution :

```
CREATE OR REPLACE FUNCTION NOMBRE_CLIENTS
    ( VILLE IN CLIENT.CL_VILLE%TYPE := 'DIJON',
     MAG IN MAGAZINE.MAG_NOM%TYPE := 'VOGUE')
RETURN INT
IS
    NB INTEGER;
    RECORD_NOT_EXISTS EXCEPTION;
BEGIN
    SELECT COUNT(*) INTO NB
    FROM CLIENT C JOIN ABONNEMENT A ON(C.CL_ID = A.CL_ID)
    JOIN MAGAZINE M ON (M.MAG_ID = A.MAG_ID)
    WHERE MAG_NOM = 'VOGUE' AND CL_VILLE = VILLE
    AND START_DATE > TO_DATE('31/08/2010 23:59:59', 'DD/MM/YYYY HH24:MI:SS');
```

```
    IF NB = 0 THEN
        RAISE RECORD_NOT_EXISTS;
    ELSE
        RETURN NB;
    END IF;
EXCEPTION
    WHEN RECORD_NOT_EXISTS THEN
        DBMS_OUTPUT.PUT_LINE('Pas d'abonnés dans la ville de ' || VILLE);
    RETURN 0;
END;
```

Exercice 17. Créer un déclencheur qui est lancé après chaque nouvelle commande INSERT dans la table ABONNEMENT de l'exercice précédent. Le déclencheur fait la mis a jour du nombre d'abonnements dans la table suivante :

TRACK_ABONNEMENTS(MAG_NOM MAGAZINE.MAG_NOM%TYPE,
NB_ABONN INTEGER)

On considère que la table TRACK_ABONNEMENTS contient déjà tous les magazines (ceux qui n'ont pas d'abonnement on NB_ABONN à 0).

Solution :

```
CREATE OR REPLACE TRIGGER AUDIT_CUSTOMER
AFTER INSERT ON CUSTOMER
FOR EACH  ROW
DECLARE
    NB INTEGER;
    MNOM  MAGAZINE.MAG_NOM%TYPE;
BEGIN
    IF NOT EXISTS (SELECT 1 FROM TRACK_ABONNEMENTS T
                WHERE T.MAG_NOM =:NEW.MAG_NOM) THEN
        RAISE_APPLICATION_ERROR(-20100, 'Le magazine n'existe pas !');
    ELSE
        UPDATE TRACK_ABONNEMENT SET NB_ABONN = NB + 1
        WHERE MAG_NOM = :NEW.MAG_NOM ;
    END IF
END;
```

Exercice 18. Soit les tables suivantes :

VOL(Numvol, Heure_départ, Heure_arrivée, Ville_départ, Ville_arrivée)
ESCALE(Numescale, Ville_Escale, Durée_Escale)

Écrivez un programme PL/SQL qui affiche les vols pour un tour du monde au départ de Paris avec des escales et des durées d'escale prédéfinies dans la table ESCALE. Le nombre d'escales à faire doit être demandé à l'utilisateur. Le numéro de chaque escales est donné par Numescale. Hypothèse de travail : pour chaque escale il existe un vol et un seul satisfaisant les contraintes. Par exemple, si l'utilisateur souhaite un tour du monde de quatre escales, le tour du monde proposé sera le suivant :

Paris → Escale no 1 → Escale no 2 → Escale no 3 → Escale no 4 → Paris

Solution :

```
ACCEPT nesc PROMPT 'Nombre escales : ';
DECLARE
    numEscaleCourante Escales.Numescale%TYPE;
    derniereEscale Escales.Numescale%TYPE;
    escaleCourante Escales.Ville_escale%TYPE;
    dureeEscaleCourante Escales.Duree_escale%TYPE;
    prochaineDestination Vol.Ville_arrivee%TYPE;
    numeroVol Vol.Numvol%TYPE;
    heureDepart Vol.Heure_depart%TYPE;
```

```
    destinationFinale Vol.Ville_arrivee%TYPE := 'Paris';
    nbMaxEscales NUMBER ;
BEGIN
    numEscaleCourante := 1;
    derniereEscale := &nesc;
    SELECT COUNT(*) INTO nbMaxEscales FROM ESCALES ;
    IF (derniereEscale > nbMaxEscales) THEN
        DBMS_OUTPUT.PUT_LINE('Au maximum ' || nbMaxEscales || ' escales !');
    ELSIF (dernieEscale < 1) THEN
        DBMS_OUTPUT.PUT_LINE('Au minimum 1 escale !');
    ELSE
        LOOP
            SELECT Ville_escale, Duree_escale
            INTO escaleCourante, dureeEscaleCourante
            FROM Escales WHERE (Numescale = numEscaleCourante);
            IF (numEscaleCourante = dernieEscale) THEN
                prochaineDestination := destinationFinale;
            ELSE
                SELECT Ville_escale INTO prochaineDestination
                FROM Escales
                WHERE (Numescale = numEscaleCourante + 1);
            END IF;
            SELECT Numvol, Heure_depart INTO numeroVol, heureDepart
            FROM Vol WHERE (Ville_depart = escaleCourante AND Ville_arrivee =
prochaineDestination);
            DBMS_OUTPUT.PUT_LINE('Escale a ' || escaleCourante
                                 || '. Prochaine escale: prendre le vol ' || numeroVol
                                 || ' a ' || to_char(heureDepart, 'HH24:MI')
                                 || ' dans ' || dureeEscaleCourante || ' jours.');
            numEscaleCourante := numEscaleCourante + 1;
            EXIT WHEN (numEscaleCourante > dernieEscale);
        END LOOP;
    END IF;
END;
```

Exercice 19. Modifiez le programme PL/SQL de l'exercice précédent pour qu'il fonctionne même si plusieurs vols satisfont les contraintes.

Solution :

```
ACCEPT s_nde PROMPT 'Nombre escales : ';
DECLARE
    dernieEscale Escales.Numescale%TYPE;
    nbMaxEscales NUMBER := 0;
    escaleCourante Escales.Ville_escale%TYPE;
    prochaineDestination Vol.Ville_arrivee%TYPE;
    destinationFinale Vol.Ville_arrivee%TYPE := 'Paris';
    dureeEscaleCourante Escales.Duree_escale%TYPE;
    numEscaleCourante Escales.Numescale%TYPE;
    volAPrendre Vol%ROWTYPE;

    CURSOR curseur1 IS SELECT * FROM Vol
    WHERE (ville_depart = escaleCourante AND ville_arrivee = prochaineDestination);
BEGIN
    numEscaleCourante := 1;
   dernieEscale := &s_nde;
    SELECT COUNT(*) INTO nbMaxEscales FROM escales;
    IF (dernieEscale > nbMaxEscales) THEN
        DBMS_OUTPUT.PUT_LINE('Au maximum ' || nbMaxEscales || ' escales !');
```

```
    ELSIF (dernieresEscale < 1) THEN
        DBMS_OUTPUT.PUT_LINE('Au minimum 1 escale !');
    ELSE
        <<boucleEscales>>
        LOOP
            SELECT Ville_escale, Duree_escale
            INTO escaleCourante, dureeEscaleCourante
            FROM Escales WHERE (Numescale = numEscaleCourante);
            IF numEscaleCourante = dernieresEscale THEN
                prochaineDestination := destinationFinale;
            ELSE
                SELECT Ville_escale INTO prochaineDestination
                FROM Escales
                WHERE (Numescale = numEscaleCourante + 1);
            END IF;

            OPEN curseur1;
            FETCH curseur1 INTO volAPrendre;
            IF (curseur1%NOTFOUND OR curseur1%NOTFOUND IS NULL) THEN
                DBMS_OUTPUT.PUT_LINE('Aucun vol disponible de ' || escaleCourante);
            EXIT boucleEscales;
            ELSE  -- retourner au maximum 10 propositions de vol
                WHILE (curseur1%FOUND AND curseur1%ROWCOUNT <= 10)
                LOOP
                    DBMS_OUTPUT.PUT_LINE('A partir de ' || escaleCourante || ' (durée' ||
                        dureeEscaleCourante || ' jours) prendre vol ' ||
                        volAPrendre.Numvol || ' a ' ||
                        to_char(volAPrendre.Heure_depart, 'HH24:MI'));
                    FETCH curseur1 INTO volAPrendre;
                END LOOP;
            END IF;
            CLOSE curseur1;

            numEscaleCourante := numEscaleCourante + 1;
            EXIT WHEN (numEscaleCourante > dernieresEscale);
        END LOOP;
    END IF;
EXCEPTION
    WHEN TOO_MANY_ROWS OR NO_DATA_FOUND THEN
    DBMS_OUTPUT.PUT_LINE('La table des escales est mal définie !');
END;
```

Exercice 20. On considéré la table Vol de l'exercice précédent :
VOL(Numvol, Heure_départ, Heure_arrivée, Ville_départ, Ville_arrivée)
Écrivez une procédure PL/SQL capable de faire des propositions de tours du monde, prenant en entrée la ville de départ (qui est aussi la destination finale) et deux bornes (supérieure et inférieure) pour le nombre d'escales. Dans ce cas il n'y a pas de liste prédéfinie d'escales et on ne s'intéresse pas à la durée des escales. La procédure doit afficher les vols pour chaque tour du monde proposé. Cherchez à utiliser une procédure ou fonction récursive.

Solution :

La solution est basée sur l'utilisation, dans la procédure, d'une fonction récursive qui valide les tours potentiels. Pour que les escales successives soient affichées dans le bon ordre, le programme démarre avec la destination finale. Pour chaque destination courante on se pose la question : quels vols arrivent dans la ville ou je me trouve ? La réponse à cette question désigne la liste des villes utilisées pour la descente en récursivité. On remarquera que, faute de disposer d'informations sur la localisation géographique des

villes escales, on ne peut pas s'assurer qu'une solution proposée corresponde réellement à un tour du globe. Aussi, cette solution ne permet pas d'éviter les boucles, par exemple le tour suivant est valide : Paris → Vienne → Rome → Vienne → Rome → Paris. Cette solution avec l'appel récursif dans la boucle FOR du curseur ouvrira autant de curseurs qu'il y a de niveaux de récursivité et peut donc produire un dépassement de la borne OPEN_CURSORS (paramètre de l'initialisation d'Oracle) ou même un dépassement mémoire (une exception STORAGE_ERROR sera levée).

```
CREATE OR REPLACE PROCEDURE trouverEscales(villeBase IN Vol.ville_depart%TYPE,
                            nbMinEscales IN INTEGER,
                            nbMaxEscales IN INTEGER)
IS
    FUNCTION tourValide(destCourante IN Vol.Ville_depart%TYPE,
                            villeDepart IN Vol.Ville_depart%TYPE,
                            minEscales IN INTEGER, maxEscales IN INTEGER)
    RETURN BOOLEAN
    IS
        valeurRetour BOOLEAN := FALSE;
        quelVol Vol%ROWTYPE;
        CURSOR curseurVol IS SELECT * FROM Vol
        WHERE (Ville_arrivee = destCourante);
    BEGIN
        FOR quelVol IN curseurVol
        LOOP
            IF (((quelVol.Ville_depart = villeDepart) AND (minEscales>0)) OR
              ((quelVol.Ville_depart != villeDepart) AND (maxEscales<1))) THEN NULL;
            ELSIF (quelVol.Ville_depart = villeDepart) THEN
                DBMS_OUTPUT.PUT_LINE('Tour du monde propose: ');
                DBMS_OUTPUT.PUT_LINE('    De ' || quelVol.Ville_depart ||
                    ' a ' || destCourante || ' : ' || quelVol.Numvol);
                valeurRetour := TRUE;
            ELSIF tourValide(quelVol.Ville_depart, villeDepart,
                                minEscales-1, maxEscales-1) THEN
                DBMS_OUTPUT.PUT_LINE('    De ' || quelVol.Ville_depart ||
                                    ' a ' || destCourante || ' : ' || quelVol.Numvol);
                valeurRetour := TRUE;
            END IF;
        END LOOP;
        RETURN valeurRetour;
    EXCEPTION
        WHEN OTHERS THEN RETURN FALSE;
    END tourValide;
BEGIN
    IF ((nbMinEscales > nbMaxEscales) OR (nbMaxEscales < 1)) THEN
        DBMS_OUTPUT.PUT_LINE('Vérifier contraintes pour nombre d escales');
    ELSIF tourValide(villeBase, villeBase, nbMinEscales, nbMaxEscales) THEN
        DBMS_OUTPUT.PUT_LINE('Faites votre choix !');
    ELSE
        DBMS_OUTPUT.PUT_LINE('Aucun tour valide n a été trouvé !');
    END IF;
END trouverEscales;
```

Références bibliographiques

Jérôme Gabillaud, Olivier Heurtel, *Oracle 11g - SQL, PL/SQL, SQL*Plus*, Editions ENI 2009
Joan Casteel, *Oracle 11g: PL/SQL Programming, Cengage Learning*, 2012
Michael McLaughlin, John Harper, *Oracle Database 11g PL/SQL Programming Workbook*, McGraw-Hill 2010

Ressources web :

http://gtuplsql.blogspot.fr
http://alexandre-mesle.com/enseignement/oracle/
http://docs.oracle.com/

Rappel SQL

INSERT INTO Table (...) VALUES (...)	UPDATE Table SET col1 = valeur1, col2 = valeur2 WHERE condition	SELECT attributs FROM Table WHERE condition	SELECT attributs INTO variables FROM Table WHERE condition

Rappel syntaxe PL/SQL

Bloc PLSQL :

```
DECLARE
    –  Déclarations variables, constantes, records, curseurs, exceptions etc.
BEGIN
    – Code
EXCEPTION
    – Traitement exceptions
END
```

Instructions conditionnelles :

```
IF cond1 THEN
    – code
ELSIF cond2 THEN
    – code
...
ELSE
    – code
END IF
```

```
CASE expr
WHEN val1 THEN
    – code
WHEN val2 THEN resultat2
    – code
ELSE
    – code
END CASE
```

Boucles :

```
LOOP
  – code
  EXIT WHEN condition
END LOOP
```

```
WHILE condition
LOOP
  – code
END LOOP
```

```
FOR i IN val1 .. val2 LOOP
  – code
END LOOP

FOR item IN curseur LOOP
  – code
END LOOP
```

Curseurs :

```
CURSOR C IS SELECT_STATEMENT ;
FETCH C INTO variable_record ;
FETCH C INTO liste_variables ;
CLOSE C ;
```

Procédures et fonctions :

```
CREATE OR REPLACE PROCEDURE [FUNCTION] nomProcedure(liste_paramètres)
[RETURN TYPE]
```

```
IS
  – Déclarations variables
BEGIN
     – code
     [RETURN val]
EXCEPTION
     – code
     [RETURN val]
END
```

Les paramètres peuvent être IN, OUT ou IN OUT. Exemple :
FONCTION moyenneSalaire (id_departement IN NUMBER) RETURN NUMBER IS ...

Déclencheurs :

```
CREATE OR REPLACE TRIGGER nomTrigger
[BEFORE | AFTER]
[INSERT [OR] UPDATE [OR] DELETE]
[OF  nom_attribut]
ON table
[REFERENCING OLD as O NEW as N]
[FOR EACH ROW]
[WHEN condition]
DECLARE
  – declarations
BEGIN
  – code
END
```

Attention : éviter les boucles infinies. Exemple déclencheurs A et B :

INSERT déclenche A
A exécute UPDATE
UPDATE déclenche B
B exécute INSERT
INSERT déclenche A → boucle !

Traitement exceptions :

```
EXCEPTION
WHEN exception1 THEN
  – code
WHEN exception1 THEN
  – code
WHEN exception1 THEN
  – code
WHEN OTHERS
  – code
END
```