



Réponses aux ateliers PL/SQL

Atelier 1.1 Présentation de l'environnement

Questions



1. Une table peut-elle avoir plusieurs clés primaires ?

Réponse : Une table ne peut comporter qu'une seule clé primaire, même lorsque celle-ci est constituée d'une combinaison de plusieurs colonnes.

2. Une table peut-elle avoir une contrainte unique si elle possède déjà une clé primaire ?

Réponse : Il est possible de spécifier une contrainte unique pour une colonne de clé non primaire afin de garantir que toutes les valeurs de cette colonne seront uniques.

3. Une table qui possède une clé étrangère est-elle une table enfant ou une table parent ?

Réponse : Elle est définie dans des tables enfant et assure qu'un enregistrement parent a été créé avant un enregistrement enfant et que l'enregistrement enfant sera supprimé avant l'enregistrement parent.

4. Que signifie LMD ?

Réponse : Le Langage de Manipulation de Données et de modules, ou LMD (en anglais DML), pour déclarer les procédures d'exploitation et les appels à utiliser dans les programmes.

5. Que signifie LDD ?

Réponse : Le Langage de Définition de Données ou LDD (en anglais DDL), à utiliser pour déclarer les structures logiques de données et leurs contraintes d'intégrité.

6. Quels sont les types d'instructions qui ne peuvent être exécutés en PL/SQL ?

Réponse : Le langage PL/SQL ne comporte pas d'instructions du Langage de Définition de Données « ALTER », « CREATE », « RENAME » et d'instructions de contrôle comme « GRANT » et « REVOKE ».

7. Quels sont les avantages du langage PL/SQL par rapport au SQL ?

Réponse : Le langage PL/SQL combine la puissance de manipulation des données du SQL avec la puissance de traitement d'un langage procédural. Il offre de nombreux avantages : support de la programmation orientée objet, très bonnes performances, portabilité, facilité de programmation, parfaite intégration à Oracle et à Java.

8. Pour configurer le client, lequel de ces fichiers utilisez-vous ?

- A. init.ora
- B. sqlnet.ora
- C. listener.ora
- D. tnsnames.ora

Réponse : D

9. Quel est le répertoire où se trouvent les fichiers de configuration ?

- A. %ORACLE_HOME%\admin\network

B. %ORACLE_HOME%\network\admin

C. %ORACLE_HOME%\net90\admin

Réponse : B

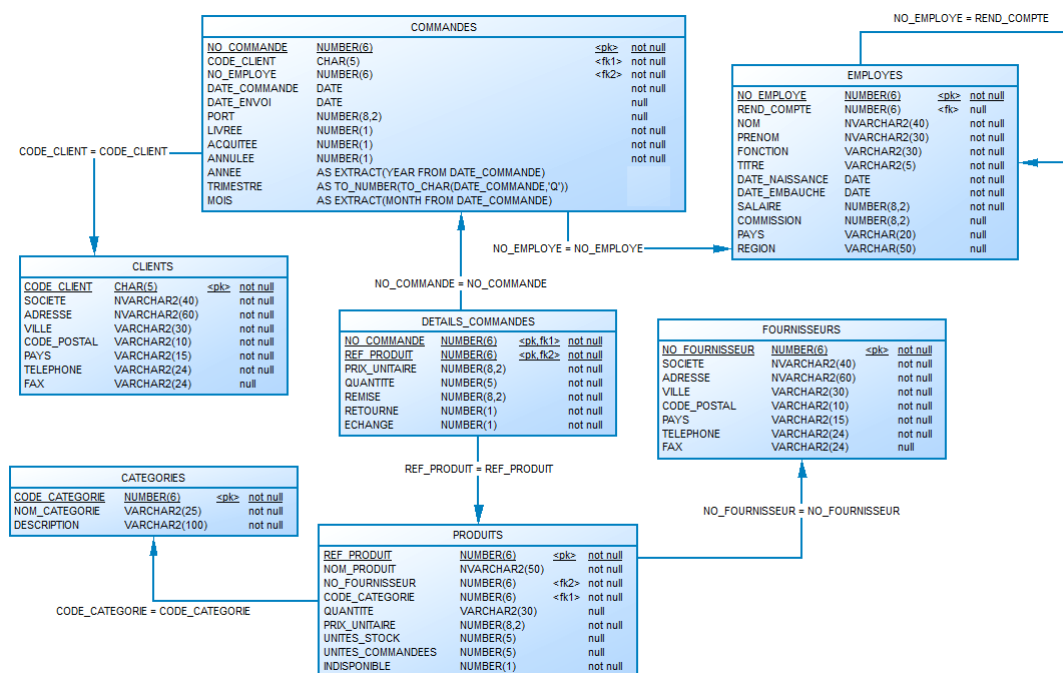
Exercice n° 1 Installation

Installez Oracle XE sur votre machine en tenant compte de votre système d'exploitation.

Exercice n° 2 Les tables utilisées pour les ateliers

Sachant que le symbole <pk> signifie clé primaire et <fk> la clé étrangère.

Quelles sont les tables en relation parent enfant ?



Réponse :

Parent

CATEGORIES

FOURNISSEURS

PRODUITS

COMMANDES

CLIENTS

EMPLOYES

EMPLOYES

Enfant

PRODUITS

PRODUITS

DETAILS_COMMANDES

DETAILS_COMMANDES

COMMANDES

COMMANDES

EMPLOYES

Atelier 1.2 Les outils SQL*Plus



Questions

1. Quel est l'outil que vous retrouvez sur chaque serveur de base de données installée ?
A. SQL*Plus.
B. iSQL*Plus.
C. SQL*Plus Worksheet
D. Oracle Enterprise Manager.

Réponse : A

2. SQL*Plus est-il un langage ou un environnement ?

Réponse : Un environnement

3. Pour utiliser iSQL*Plus sur une machine distante, avez-vous besoin d'installer le client Oracle ?

Réponse : Non

4. Quelle est la commande qui vous permet de vous connecter ?

Réponse : CONNECT

5. Dans la syntaxe de démarrage de SQL*Plus, pouvez-vous lancer l'exécution d'un script ?

Réponse : OUI

6. Quelle est la commande qui vous permet de stocker dans un fichier tout ce qui est affiché à l'écran ?

Réponse : SPOOL

7. Dans l'environnement SQL*Plus, peut-on exécuter des commandes du système d'exploitation ?

Réponse : OUI

8. Citez trois types de paramètres de mise en forme des résultats des requêtes.

Réponse : LINSEIZE, PAGESIZE, FEEDBACK

9. Quelle est la commande qui vous permet de décrire la structure d'une vue ?

Réponse : DESC

Exercice n° 1 Préparer le poste de développement

Installez le schéma des exemples pour les ateliers en respectant la démarche suivante :

```
C:\>dir Oracle11gSQL_PLSQL.zip
```

```
Le volume dans le lecteur C n'a pas de nom.
```

```
Le numéro de série du volume est BC79-154D
```

```
Répertoire de C:\
```

```

12/08/2011  20:34          6 787 143 Oracle11gSQL_PLSQL.zip

C:\>unzip Oracle11gSQL_PLSQL.zip
Archive:  Oracle11gSQL_PLSQL.zip
  creating: Oracle11gSQL_PLSQL/
  inflating: Oracle11gSQL_PLSQL/DeleteEnvStagiaireXE.sql
  inflating: Oracle11gSQL_PLSQL/InitEnvEtoileXE.sql
  inflating: Oracle11gSQL_PLSQL/InitEnvStagiaireXE.sql
  creating: Oracle11gSQL_PLSQL/stagiaire/
  inflating: Oracle11gSQL_PLSQL/stagiaire/CATEGORIES.DAT
  inflating: Oracle11gSQL_PLSQL/stagiaire/CLIENTS.DAT
  inflating: Oracle11gSQL_PLSQL/stagiaire/COMMANDES.DAT
  inflating: Oracle11gSQL_PLSQL/stagiaire/COMMANDES_2009.DAT
  inflating: Oracle11gSQL_PLSQL/stagiaire/DETAILS_COMMANDES.DAT
  inflating: Oracle11gSQL_PLSQL/stagiaire/DETAILS_COMMANDES_2009.DAT
  inflating: Oracle11gSQL_PLSQL/stagiaire/DIM_TEMPS.DAT
  inflating: Oracle11gSQL_PLSQL/stagiaire/EMPLOYES.DAT
  inflating: Oracle11gSQL_PLSQL/stagiaire/FOURNISSEURS.DAT
  inflating: Oracle11gSQL_PLSQL/stagiaire/PRODUITS.DAT
  inflating: Oracle11gSQL_PLSQL/stagiaire/STATISTIQUES.DAT

C:\>cd Oracle11gSQL_PLSQL

C:\Oracle11gSQL_PLSQL>dir

Répertoire de C:\Oracle11gSQL_PLSQL

12/08/2011  21:52      <REP>          .
12/08/2011  21:52      <REP>          ..
12/08/2011  21:52                  550 DeleteEnvStagiaireXE.sql
12/08/2011  21:51                  1 725 InitEnvEtoileXE.sql
12/08/2011  20:33                 30 681 InitEnvStagiaireXE.sql
12/08/2011  20:33      <REP>          stagiaire

C:\Oracle11gSQL_PLSQL>sqlplus /nolog @InitEnvStagiaireXE.sql

```

Téléchargez et Installez l'outil SQL Developer.

Exercice n° 2 Connexion

Démarrez SQL*Plus, en ligne de commande, avec le nom d'utilisateur du schéma exemples « **STAGIAIRE** » et son mot de passe « **PWD** ».

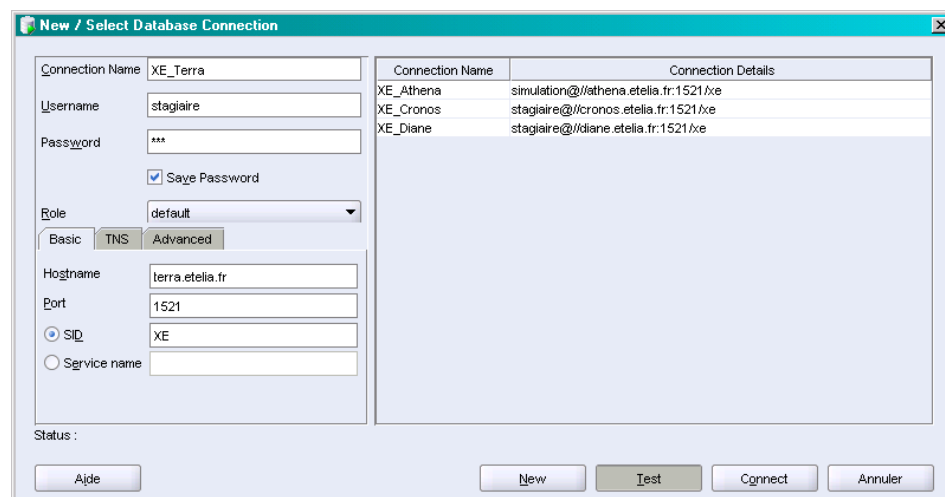
```

C:\>sqlplus stagiaire/pwd
...
C:\>sqlplus stagiaire/pwd@//diane.etelia.fr:1521/XE
...

C:\>sqlplus /nolog
SQL> CONNECT STAGIAIRE/PWD
ou
SQL> stagiaire/pwd@//diane.etelia.fr:1521/XE
SQL> show user
USER est "STAGIAIRE"

```

Démarrez SQL Developer et paramétrez la connexion à la base de données.



Exercice n° 3 Environnement SQL*Plus

En utilisant SQL*Plus en ligne de commande, redirigez les sorties vers un fichier et exécutez les commandes suivantes :

- Décrivez la table « **COMMANDES** » ;

```
C:>sqlplus stagiaire/pwd
SQL> SPOOL C:\Exercice2.lst
SQL> DESC COMMANDES
```

Nom	NULL ?	Type
NO_COMMANDE	NOT NULL	NUMBER (6)
CODE_CLIENT	NOT NULL	CHAR (5)
NO_EMPLOYE	NOT NULL	NUMBER (6)
DATE_COMMANDE	NOT NULL	DATE
DATE_ENVOI		DATE
PORT		NUMBER (8 , 2)

- Déconnectez-vous de la base de données sans sortir du SQL*Plus ;

```
SQL> DISC
Déconnecté de Oracle Database 11g Express Edition Release 11.2.0.1.0
- Production
```

- Décrivez de nouveau la table « **COMMANDES** ». Que remarquez-vous ?

```
SQL> DESC COMMANDES
SP2-0640: Non connecté
SP2-0641: "DESCRIBE" nécessite une connexion au serveur
```

- Connectez vous ;

```
SQL> CONNECT STAGIAIRE/PWD
ou
SQL> stagiaire/pwd@//diane.etelia.fr:1521/XE
```

- Affichez l'utilisateur courant ;

```
SQL> show user
USER est "STAGIAIRE"
```

- Arrêtez la redirection des sorties vers le fichier ;

```
SQL> SPOOL OFF
```

- Sans quitter l'environnement, listez le fichier que vous venez de créer.

```
SQL> HOST TYPE C:\Exercice2.lst
```

Exercice n°4 Générer des scripts SQL

Connectez-vous à SQL*Plus, redirigez les sorties vers le fichier « **DESC_ALL.SQL** » et exécutez les commandes suivantes :

- Interrogez la vue catalogue à l'aide de la syntaxe suivante :

```
SET PAGESIZE 0
SET ECHO OFF
SET FEEDBACK OFF
SELECT 'DESC ' || TABLE_NAME FROM CAT
WHERE TABLE_TYPE = 'TABLE' ;
```

```
SQL> SET PAGESIZE 0
SQL> SET ECHO OFF
SQL> SET FEEDBACK OFF
SQL> SPOOL C:\DESC_ALL.SQL
SQL> SELECT 'DESC ' || TABLE_NAME FROM CAT
      2 WHERE TABLE_TYPE = 'TABLE' ;
```

- Maintenant vous pouvez arrêter la redirection des sorties vers le fichier et exécuter le script ainsi conçu.

```
SQL> SPOOL OFF
SQL> @C:\DESC_ALL.SQL
```

Atelier 2.1 Bases du langage PL/SQL

Questions



1. Quelles sont les sections qui font partie d'un bloc ?

Réponse : Les parties d'un bloc PL/SQL sont : « **DECLARE** », « **BEGIN** » et « **EXCEPTION** ».

2. Quel est le rôle de la section « **DECLARE** » ?

Réponse : La section DECLARE contient les définitions des variables.

3. Quelles sont les syntaxes incorrectes ?

- A. `declare begin NULL;begin NULL;begin NULL;end;end;end;`
- B. `declare NULL;begin NULL;begin NULL;end;end;end;`
- C. `declare begin NULL;begin NULL;begin NULL;end;end;`
- D. `declare begin NULL;begin begin NULL;end;end;end;`
- E. `declare begin NULL;begin NULL;begin NULL;end;NULL;end;NULL;end;`

Réponse : B, C

4. Quel est le symbole de fin d'instruction en PL/SQL ?

- A. .
- B. :
- C. ;
- D. !

Réponse : B, C

5. Quelles sont les syntaxes qui représentent des commentaires en PL/SQL ?

- A. `/* Commentaire */`
- B. `-- Commentaire --`
- C. `' Commentaire '`
- D. `" Commentaire "`

Réponse : A, B

6. Quelle est la signification la syntaxe suivante :

« **PRAGMA AUTONOMOUS_TRANSACTION** » ?

Réponse : Le mot clé « **PRAGMA** » signifie que le reste de l'ordre PL/SQL est une directive de compilation. « **AUTONOMOUS_TRANSACTION** » indique au compilateur que le bloc s'exécute dans une transaction indépendante

Exercice n°1 La présentation du PL/SQL

Créez un bloc PL/SQL qui affiche la description suivante :

Utilisateur : STAGIAIRE aujourd'hui est le 17 juillet 2006

```
SQL> set serveroutput on size 1000000
SQL> begin
  2     dbms_output.put_line( 'Utilisateur : '||user||
  3                           ' aujourd'hui est le '||
  4                           TO_CHAR(SYSDATE,'fmdd month yyyy'));
  5 end;
  6 /
```

Utilisateur : STAGIAIRE aujourd'hui est le 12 août 2011

Procédure PL/SQL terminée avec succès.

Retrouvez le script créé pour l'Atelier 13 dans l'exercice 2, la mise à jour du modèle étoile permettent d'alimenter les quatre tables DIM_EMPLOYES, DIM_PRODUITS, DIM_CLIENTS et à la fin INDICATEURS. Utilisez ce script pour créer un bloc PL/SQL qui effectue la mise à jour.

```
SQL> begin
  2     MERGE INTO DIM_CLIENTS CIBLE
  3     USING ( SELECT CODE_CLIENT,SOCIETE,VILLE,PAYS
  4             ...
  60             SOURCE.QUANTITE,SOURCE.PRIX_UNITAIRE);
  61 end;
  62 /
```

Procédure PL/SQL terminée avec succès.

Utilisant les propriétés d'un bloc PL/SQL, vous devez effectuer la série des opérations suivantes :

- Augmenter les salaires des représentants de 10%.
- Insérer une nouvelle catégorie de produits avec le nom et la description suivante : 'Produits cosmétiques'. Faites en sorte que l'insertion soit permanente.
- Annuler la modification de la table EMPLOYES.
- Vérifier que la nouvelle catégorie soit toujours en place.

```
SQL> SELECT NOM, SALAIRE FROM EMPLOYES
  2 WHERE FONCTION LIKE 'Rep%';
```

NOM	SALAIRE
Peacock	2856
...	

```
SQL> UPDATE EMPLOYES SET SALAIRE = SALAIRE * 1.1
  2 WHERE FONCTION LIKE 'Rep%';
```

6 ligne(s) mise(s) à jour.

```
SQL> SELECT NOM, SALAIRE FROM EMPLOYES
  2 WHERE FONCTION LIKE 'Rep%';
```

NOM	SALAIRE
Peacock	3141,6
...	

```
SQL> declare
  2      pragma autonomous_transaction;
  3  begin
  4      INSERT INTO CATEGORIES VALUES
  5          ( 9,'Produits cosmétiques','Produits cosmétiques');
  6      COMMIT;
  7  end;
  8  /
```

Procédure PL/SQL terminée avec succès.

```
SQL> ROLLBACK;
```

Annulation (rollback) effectuée.

```
SQL> SELECT NOM, SALAIRE FROM EMPLOYES
  2  WHERE FONCTION LIKE 'Rep%';
```

NOM	SALAIRE
Peacock	2856
...	

```
SQL> SELECT NOM_CATEGORIE FROM CATEGORIES
  2  WHERE CODE_CATEGORIE = 9;
```

NOM_CATEGORIE
Produits cosmétiques

Atelier 3.1 Les variables



Questions

1. Quelles sont les déclarations invalides ?
 - A. `nom_varA NUMBER(8) DEFAULT 10 ;`
 - B. `nom_var1, nom_var2 DATE;`
 - C. `nom_var VARCHAR2(20) NOT NULL ;`
 - D. `nom_var BOOLEAN := 1;`
 - E. `nom_var BINARY_INTEGER;`
 - F. `2nom_var BINARY_INTEGER;`
 - G. `a$nom_varG DATE := '01/01/2006';`
 - H. `B#a$nom_var DATE NOT NULL := SYSDATE;`
 - I. `nom_varI NUMBER(3) := 123.45678;`
 - J. `nom_var NUMBER(3) := 1234.5678;`
 - K. `nom_varK CONSTANT NUMBER(12,3) := 1234.5678;`

Réponse : B, C, D, F, J

2. Quel est le résultat de la requête suivante ?

```
SQL> declare
2   utilisateur varchar2(50) := '1 : ' || USER;
3   begin
4     declare
5       utilisateur varchar2(50) := '2 : ' || USER;
6     begin
7       declare
8         utilisateur varchar2(50) := '3 : ' || USER;
9       begin
10        dbms_output.put_line( utilisateur);
11      end;
12    end;
13  end;
14  /
```

- A. '1 :STAGIAIRE'
- B. '2 :STAGIAIRE'
- C. '3 :STAGIAIRE'

Réponse : C

3. Quelles sont les syntaxes correctes ?
 - A. `declare v_1 NUMBER(8,2) := 2500;`
`begin v_1 = v_1 * 2; end;`
 - B. `declare v_1 date;`
`begin v_1 := sysdate; end;`

- C. declare v_1 constant date;
begin v_1 := sysdate; end;
- D. declare v_1 constant date := sysdate;
begin null; end;
- E. declare v_1 NUMBER := v_2; begin null; end;

Réponse : B, D

Exercice n°1 La déclaration des variables

Créez un bloc PL/SQL dans lequel vous déclarez les variables de la question 24.1-1 les points : A, G, I, K. Affichez les informations stockées dans ces variables.

```
SQL> set serveroutput on size 5000
SQL> declare
  2   nom_varA          NUMBER(8) DEFAULT 10 ;
  3   a$nom_varG        DATE := '01/07/2011';
  4   nom_varI          NUMBER(3) := 123.45678;
  5   nom_varK  CONSTANT NUMBER(12,3) := 1234.5678;
  6   begin
  7     dbms_output.put_line( 'nom_varA   : ' || nom_varA );
  8     dbms_output.put_line( 'a$nom_varG : ' || a$nom_varG );
  9     dbms_output.put_line( 'nom_varI   : ' || nom_varI );
 10     dbms_output.put_line( 'nom_varK   : ' || nom_varK );
 11   end;
 12   /
nom_varA      :10
a$nom_varG    :01/07/2011
nom_varI      :123
nom_varK      :1234,568
```

Procédure PL/SQL terminée avec succès.

Déclarez une variable de liaison de type « **VARCHAR2** ». Créez un premier bloc qui alimente la variable avec la valeur de l'utilisateur courant concaténée avec la date du jour. Créez un deuxième bloc qui affiche la variable.

```
SQL> VARIABLE utilisateur varchar2(255)
SQL> begin
  2   :utilisateur := user || ' ' || sysdate;
  3   end;
  4   /
SQL> begin
  2   dbms_output.put_line(:utilisateur);
  3   end;
  4   /
SQL> PRINT utilisateur
```

Atelier 3.2 Les variables

Questions



1. Quelles sont les déclarations invalides ?
 - A. `declare SUBTYPE Numeral IS NUMBER(1,0); v_1 Numeral; begin v_1 := 1; end;`
 - B. `declare SUBTYPE v_1 IS TIMESTAMP; begin v_1:= SYSTIMESTAMP; end;`
 - C. `declare v_1 DIM_TEMPS.JOUR%TYPE; begin v_1:= SYSDATE; end;`
 - D. `declare v_1 DIM_TEMPS%ROWTYPE; begin v_1.JOUR:= SYSDATE; end;`
 - E. `declare v_1 DIM_TEMPS%ROWTYPE; begin v_1:= SYSDATE; end;`
 - F. `declare TYPE var IS VARRAY(3) OF NVARCHAR2(30); v_1 var:= var('BIZOI', 'FABER'); begin null; end;`
 - G. `declare TYPE var IS RECORD (A VARCHAR2(3) := 'AA', B VARCHAR2(3) := 'BB'); v_1 var; begin null; end;`
 - H. `declare TYPE var IS TABLE OF DATE INDEX BY BINARY_INTEGER; v_1 var; begin v_1(1):=sysdate; end;`
 - I. `declare v_1 DIM_TEMPS.JOUR%TYPE := ADD_MONTHS(TRUNC(SYSDATE, 'MONTH'), 1); begin null; end;`
 - J. `declare v_1 CLIENTS%ROWTYPE; begin v_1.CODE_CLIENT := 'AA'; v_1.SOCIETE := 'BB'; end;`
 - K. `declare TYPE var IS TABLE OF DATE INDEX BY VARCHAR2(2); v_1 var; begin null; end;`

Réponse : B, E

2. Quelles est le type de retour de chaque expression suivante :

A. `256*2 + EXTRACT(YEAR FROM SYSDATE)`

Réponse : NUMBER

B. `1024 || SYSDATE || USER`

Réponse : VARCHAR2

C. `SYSDATE > '01/07/2006'`

Réponse : BOOLEAN

D. `2.5*2.5/0f + 10`

Réponse : BINARY_FLOAT_INFINITY(Infini)

E. `INSTR('QUANTITE', 'T')*256`

Réponse : NUMBER

F. `SYSDATE - ROUND(TRUNC(MOD(1600,10), -1), 2)`

Réponse : DATE

G. 2.5D*256+10

Réponse : BINARY_DOUBLE

H. 2.5f/0||USER

Réponse : VARCHAR2

I. SYSDATE + 3070 + 2.5f

Réponse : DATE

Exercice n°1 Les variables composées

A partir des syntaxes de la question 24.2-1 écrivez les blocs suivants :

- L'option D, remplacez 'SYSDATE' par l'option I de la question 24.2.2. Initialisez tous les champs de l'enregistrement utilisant la date déjà affectée 'v_1.JOUR' et insérez les dans la table 'DIM_TEMPS' en validant la transaction directement dans le bloc.

```
SQL> SELECT ROUND(MAX(JOUR) - SYSDATE) FROM DIM_TEMPS;
```

```
ROUND(MAX(JOUR)-SYSDATE)
-----
                        3062
```

```
SQL> declare --D --I
2   v_1 DIM_TEMPS%ROWTYPE;
3   begin
4   v_1.JOUR      := SYSDATE + 3070 + 2.5f;
5   v_1.SEMaine  := TO_CHAR(v_1.JOUR, 'IW');
6   v_1.MOIS     := TO_CHAR(v_1.JOUR, 'Month');
7   v_1.MOIS_N   := TO_CHAR(v_1.JOUR, 'MM');
8   v_1.TRIMESTRE:= TO_CHAR(v_1.JOUR, 'Q');
9   v_1.ANNEE    := TO_CHAR(v_1.JOUR, 'YYYY');
10  INSERT INTO DIM_TEMPS VALUES v_1;
12  COMMIT;
13  end;
14  /
```

Procédure PL/SQL terminée avec succès.

```
SQL> SELECT * FROM DIM_TEMPS WHERE JOUR > SYSDATE + 3070;
```

JOUR	SEMAINE	MOIS	MOIS_N	TRIMESTRE	ANNEE
10/01/2020	2	Janvier	1	1	2020

- Modifiez les blocs des options F, G, I de la question 24.2-1 pour permettre l'affichage des variables déclarées.

```
SQL> declare --F
2   TYPE var IS VARRAY(3) OF NVARCHAR2(30);
3   v_1 var:= var('BIZOI','FABER');
4   begin
5   dbms_output.put_line( v_1(1)||' '||v_1(2));
6   end;
7   /
```

BIZOÏ FABER

Procédure PL/SQL terminée avec succès.

```
SQL> declare --G
2     TYPE var IS RECORD ( A VARCHAR2(3) := 'AA',
3                           B VARCHAR2(3) := 'BB');
4     v_1 var;
5     begin
6         dbms_output.put_line( v_1.A || ' ' || v_1.B);
7     end;
8     /
```

AA BB

Procédure PL/SQL terminée avec succès.

```
SQL> declare --I
2     v_1 DIM_TEMPS.JOUR%TYPE := ADD_MONTHS(
3                                     TRUNC(SYSDATE, 'MONTH'), 1);
4     begin
5         dbms_output.put_line( v_1);
6     end;
7     /
```

01/09/2011

Procédure PL/SQL terminée avec succès.

- Modifiez le bloc de l'option H de la question 24.2-1, pour permettre d'alimenter le premier poste du tableau avec la date du jour et le deuxième poste avec le lendemain. Affichez les deux postes du tableau.

```
SQL> declare
2     TYPE var IS TABLE OF DATE INDEX BY VARCHAR2(2);
3     v_1 var;
4     begin
5         v_1('A'):=sysdate;
6         v_1('B'):=sysdate+1;
7         dbms_output.put_line( v_1('A') || ' ' || v_1('B'));
8     end;
9     /
```

12/08/2011 13/08/2011

Procédure PL/SQL terminée avec succès.

Atelier 4.1 Les ordres SQL dans PL/SQL

Questions



1. Sachant que les expressions doivent remplacer les trois points dans le bloc suivant, quelles sont les expressions invalides ?

```
declare
  v_1 EMPLOYES%ROWTYPE;
  TYPE TAB IS TABLE OF EMPLOYES%ROWTYPE
                                INDEX BY BINARY_INTEGER;
  t_1 TAB;
begin
  SELECT * INTO v_1 FROM EMPLOYES WHERE NO_EMPLOYE = 5;

end;
/
```

- A. SELECT * INTO v_1 FROM EMPLOYES WHERE NO_EMPLOYE = 5;
- B. UPDATE EMPLOYES SET ROW = v_1 WHERE NO_EMPLOYE = 5;
- C. SELECT count(*) INTO v_1 FROM EMPLOYES WHERE 1 = 2;
- D. SELECT * INTO v_1 FROM EMPLOYES WHERE 1 = 2;
- E. SELECT * INTO v_1 FROM EMPLOYES;
- F. SELECT * BULK COLLECT INTO t_1 FROM EMPLOYES;
- G. v_1.NO_EMPLOYE:=100;INSERT INTO EMPLOYES VALUES v_1;
- H. v_1.NO_EMPLOYE:=100;INSERT INTO EMPLOYES
VALUES (v_1.NO_EMPLOYE, v_1.REND_COMPTE, v_1.NOM,
v_1.PRENOM, v_1.FONCTION, v_1.TITRE, v_1.DATE_NAISSANCE,
v_1.DATE_EMBAUCHE, v_1.SALAIRE, v_1.COMMISSION);

Réponse : C, D, E

2. Sachant que les variables ont été déclarées auparavant, qu'elles sont du bon type et au bon endroit, quels sont les ordres de mise à jour incorrects ?
- A. INSERT INTO CATEGORIES VALUES (9,'Fruits',
'Fruits') RETURNING ROWID INTO v_rowid;
 - B. INSERT INTO CATEGORIES VALUES (9,'Fruits',
'Fruits') RETURNING * INTO v_cat;
 - C. UPDATE COMMANDES SET PORT = PORT * 1.05
RETURNING NO_COMMANDE BULK COLLECT INTO v_comm;
 - D. DELETE CATEGORIES WHERE ROWID = v_1
RETURNING CODE_CATEGORIE INTO v_cat;
 - E. UPDATE CATEGORIES SET NOM_CATEGORIE = DESCRIPTION
WHERE ROWID = v_1 RETURNING NOM_CATEGORIE
INTO v_cat;
 - F. UPDATE COMMANDES SET PORT = PORT * 1.05
RETURNING NO_COMMANDE INTO v_comm;

G. DELETE INDICATEURS RETURNING ROWID
BULK COLLECT INTO v_rowid;

H. DELETE CATEGORIES RETURNING CODE_CATEGORIE
INTO v_cat;

Réponse : B, F, H

Exercice n°1 Les ordres SQL dans PL/SQL

Créez le bloc PL/SQL qui permet d'effectuer les opérations :

- Effacez les enregistrements des commandes de l'année 2009.

```
SQL> begin
2   DELETE DETAILS_COMMANDES
3   WHERE NO_COMMANDE IN
4         ( SELECT NO_COMMANDE FROM COMMANDES
5             WHERE ANNEE = 2009);
6   dbms_output.put_line('Enregistrements effacés : '
7                           || SQL%ROWCOUNT);
8   DELETE COMMANDES WHERE ANNEE = 2009;
9   dbms_output.put_line('Enregistrements effacés : '
10                          || SQL%ROWCOUNT);
11  COMMIT;
12 end;
13 /
```

- Affichez le client, l'adresse et le numéro de téléphone du client qui a le CODE_CLIENT='PARIS'. Effacez les enregistrements du client dans la table INDICATEURS.

```
SQL> declare
2   v_client CLIENTS%ROWTYPE;
3   begin
4   SELECT * INTO v_client FROM CLIENTS
5   WHERE CODE_CLIENT = 'PARIS';
6   dbms_output.put_line( 'Client      : ' || v_client.SOCIETE);
7   dbms_output.put_line( 'Adresse     : ' || v_client.ADRESSE);
8   dbms_output.put_line( 'Téléphone   : ' || v_client.TELEPHONE);
9   DELETE INDICATEURS WHERE CODE_CLIENT = v_client.CODE_CLIENT;
10  dbms_output.put_line('Enregistrements effacés : '
11                          || SQL%ROWCOUNT);
12  COMMIT;
13 end;
14 /
Client      :Paris spécialités
Adresse     :265, boulevard Charonne
Téléphone   :01.42.34.22.66
Enregistrements effacés : 0
```

- Modifiez le produit numéro 8 en le rendant disponible 'INDISPONIBLE' := 0' et rajoutant 200 unités en stock. Affichez le nom du fournisseur et le nom de la catégorie de ce produit. Effacez les enregistrements du produit dans la table INDICATEURS.

```
SQL> declare
2   v_produit  PRODUITS%ROWTYPE;
3   v_nom_four FOURNISSEURS.SOCIETE%TYPE;
```

```

4   v_nom_cat  CATEGORIES.NOM_CATEGORIE%TYPE;
5   begin
6     SELECT * INTO v_produit FROM PRODUITS
7     WHERE REF_PRODUIT = 8;
8     v_produit.INDISPONIBLE := 0;
9     v_produit.UNITES_STOCK := 200;
10    UPDATE PRODUITS SET ROW = v_produit WHERE REF_PRODUIT = 8;
11    SELECT SOCIETE, NOM_CATEGORIE INTO v_nom_four, v_nom_cat
12    FROM PRODUITS JOIN FOURNISSEURS
13    USING( NO_FOURNISSEUR) JOIN CATEGORIES USING(CODE_CATEGORIE )
14    WHERE REF_PRODUIT = 8;
15    dbms_output.put_line( 'Fournisseur   : ' || v_nom_four);
16    dbms_output.put_line( 'Nom Catégorie : ' || v_nom_cat);
17    DELETE INDICATEURS WHERE REF_PRODUIT = 8;
18    dbms_output.put_line('Enregistrements effacés : '
19                        || SQL%ROWCOUNT);
20    COMMIT;
21  end;
22  /
Fournisseur   : Grandma Kelly's Homestead
Nom Catégorie : Condiments
Enregistrements effacés : 13

Procédure PL/SQL terminée avec succès.

```

- Affichez les deux employés encadrés par 'Buchanan'. Augmentez les frais de port de '10%' pour toutes les commandes passées par ces deux employés dans l'année '2011', la modification doit être faite dans la table INDICATEURS. Affichez mensuellement pour l'année '2011' les cumuls des frais de port et des quantités.

```

SQL> declare
2   TYPE t_employe IS TABLE OF EMPLOYES%ROWTYPE
3                               INDEX BY BINARY_INTEGER;
4   TYPE t_emp  IS TABLE OF DIM_EMPLOYES.EMPLOYE%TYPE
5                               INDEX BY BINARY_INTEGER;
6   TYPE t_mois IS TABLE OF DIM_TEMPS.MOIS%TYPE
7                               INDEX BY BINARY_INTEGER;
8   TYPE t_port IS TABLE OF INDICATEURS.PORT%TYPE
9                               INDEX BY BINARY_INTEGER;
10  TYPE t_qant IS TABLE OF INDICATEURS.QUANTITE%TYPE
11                               INDEX BY BINARY_INTEGER;
12  v_employe t_employe;
13  v_emp      t_emp;
14  v_mois      t_mois;
15  v_sum_port t_port;
16  v_sum_qant t_qant;
17  begin
18    SELECT * BULK COLLECT INTO v_employe FROM EMPLOYES
19    WHERE REND_COMPTE = ( SELECT NO_EMPLOYE
20                        FROM EMPLOYES
21                        WHERE NOM = 'Buchanan' );
22    dbms_output.put_line( 'Employé : ' || v_employe(1).NOM
23                        || ' ' || v_employe(1).NOM);
24    dbms_output.put_line( 'Employé : ' || v_employe(2).NOM

```

```

25         || ' ' || v_employe(2).NOM);
26     UPDATE INDICATEURS SET PORT = PORT * 1.1
27     WHERE NO_EMPLOYE in ( v_employe(1).NO_EMPLOYE,
28         v_employe(2).NO_EMPLOYE ) AND
29         EXTRACT ( YEAR FROM JOUR) = 2011;
30     dbms_output.put_line('Enregistrements modifies : '
31         || SQL%ROWCOUNT);
32     SELECT EMPLOYE, TO_CHAR(JOUR,'FMMonth'),
33         SUM(PORT), SUM(QUANTITE)
34     BULK COLLECT INTO v_emp, v_mois, v_sum_port, v_sum_qant
35     FROM INDICATEURS NATURAL JOIN DIM_EMPLOYES
36     WHERE NO_EMPLOYE in ( 6, 9 ) AND
37         EXTRACT ( YEAR FROM JOUR) = 2011
38     GROUP BY EMPLOYE, TO_CHAR(JOUR,'FMMonth'),
39         EXTRACT ( MONTH FROM JOUR)
40     ORDER BY EMPLOYE, EXTRACT ( MONTH FROM JOUR);
41     COMMIT;
42     dbms_output.put_line( 'Enregistrement modifie : ' ||
43         v_emp(1) || ' ' || v_mois(1) || ' ' ||
44         TO_CHAR(v_sum_port(1), '999G999D00U') || ' ' ||
45         TO_CHAR(v_sum_qant(1), '9G999'));
46     dbms_output.put_line( 'Enregistrement modifie : ' ||
47         v_emp(2) || ' ' || v_mois(2) || ' ' ||
48         TO_CHAR(v_sum_port(2), '999G999D00U') || ' ' ||
49         TO_CHAR(v_sum_qant(2), '9G999'));
50     dbms_output.put_line( 'Enregistrement modifie : ' ||
51         v_emp(3) || ' ' || v_mois(3) || ' ' ||
52         TO_CHAR(v_sum_port(3), '999G999D00U') || ' ' ||
53         TO_CHAR(v_sum_qant(3), '9G999'));
54     -- ...
55 end;
56 /

```

Employé : Dodsworth Dodsworth
Employé : Suyama Suyama
Enregistrements modifies : 90

Enregistrement modifie : Dodsworth Anne Janvier	4 998,47€	237
Enregistrement modifie : Dodsworth Anne Février	7 153,60€	297
Enregistrement modifie : Dodsworth Anne Mars	1 694,46€	317

Procédure PL/SQL terminée avec succès.

Atelier 4.2 Les ordres SQL dans PL/SQL

Questions



1. Quel est l'affichage suite à l'exécution de ce bloc ? Argumentez votre réponse.

```
SQL> declare
2   v_sql_dynamique VARCHAR2(200) := 'CREATE TABLE SAV_CAT AS ' || '
3       SELECT * FROM CATEGORIES WHERE 1=2' ;
4   v_count NUMBER(5);
5 begin
6   EXECUTE IMMEDIATE v_sql_dynamique;
7   SELECT COUNT(*) INTO v_count FROM CATEGORIES;
8   dbms_output.put_line( 'Enregistrements : ' || v_count);
9   INSERT INTO SAV_CAT SELECT * FROM CATEGORIES
10  SELECT COUNT(*) INTO v_count FROM SAV_CAT;
11  dbms_output.put_line( 'Enregistrements : ' || v_count);
12 end;
13 /
```

- A. Enregistrements : 9 9
- B. Enregistrements : 9 0
- C. Enregistrements : 9
- D. Enregistrements :
- E. ERREUR à la ligne 10 : ...

Réponse : E. L'instruction est créée au moment de l'exécution, le compilateur PL/SQL ne peut pas effectuer la liaison des identifiants (la table SAV_CAT) dans l'instruction, ce qui n'autorise pas la compilation du bloc. Vous pouvez utiliser le SQL dynamique pour créer une table mais par la suite, dans le bloc, vous ne pouvez pas utiliser de SQL statique pour manipuler les enregistrements, il échouerait puisque la table n'existerait que lors de l'exécution du bloc.

2. Pour laquelle de ces exécutions, 'v_sql' ne peut pas être un bloc PL/SQL ?
- A. EXECUTE IMMEDIATE v_sql USING v_1, v_2
RETURNING BULK COLLECT INTO v_tab;
 - B. EXECUTE IMMEDIATE v_sql USING
IN v_1, IN v_2, OUT v_3;
 - C. EXECUTE IMMEDIATE v_sql USING v_1, v_2;
 - D. EXECUTE IMMEDIATE v_sql;

Réponse : A

Exercice n°1 Les ordres SQL dynamiques

Pour des besoins d'analyse, on a besoin d'une table pour recenser toutes les ventes, créée chaque jour. La structure de la table est identique à celle de la table VENTES_CLIENTS_2011. Elle doit avoir le nom fourni par l'expression suivante :

```
'VENTES_' || TO_CHAR(SYSDATE, 'YYYYMMDD')
```

Une fois créée, vous devez l'alimenter avec les enregistrements des ventes de l'année '2011'.

Octroyez les privilèges de lecture pour tous les utilisateurs de la base et créez un synonyme public, avec le même nom, pour cette table.

```
SQL> declare
2     v_nom_table VARCHAR2(32) := 'VENTES_' ||
3         TO_CHAR(SYSDATE, 'YYYYMMDD');
4     v_sql_dynamique VARCHAR2(3000) :=
5         'CREATE TABLE ' || v_nom_table ||
6         ' AS SELECT * FROM VENTES_CLIENTS_2011 WHERE 1=2' ;
7 begin
8     dbms_output.put_line( v_sql_dynamique);
9     EXECUTE IMMEDIATE v_sql_dynamique;
10    v_sql_dynamique := 'INSERT INTO ' || v_nom_table ||
11        ' SELECT EXTRACT( MONTH FROM DATE_COMMANDE) MOIS, ' ||
12        'CODE_CLIENT, '
13        'SUM(QUANTITE*PRIX_UNITAIRE) VENTE, '
14        'SUM(QUANTITE*PRIX_UNITAIRE*REMISE) REMISE '
15        'FROM COMMANDES NATURAL JOIN DETAILS_COMMANDES '
16        'WHERE EXTRACT( YEAR FROM DATE_COMMANDE) = 2011 '
17        'GROUP BY EXTRACT ( MONTH FROM DATE_COMMANDE), '
18        'CODE_CLIENT';
19    dbms_output.put_line( v_sql_dynamique);
20    EXECUTE IMMEDIATE v_sql_dynamique;
21    v_sql_dynamique := 'GRANT SELECT ON STAGIAIRE.' ||
22        v_nom_table || ' TO PUBLIC';
23    dbms_output.put_line( v_sql_dynamique);
24    EXECUTE IMMEDIATE v_sql_dynamique;
25    v_sql_dynamique := 'CREATE PUBLIC SYNONYM ' || v_nom_table ||
26        ' FOR ' || v_nom_table;
27    dbms_output.put_line( v_sql_dynamique);
28    EXECUTE IMMEDIATE v_sql_dynamique;
29 end;
30 /
```

Atelier 5.1 Les structures de contrôle

Questions



1. Quelles sont les instructions de contrôle structurellement invalides ?
 - A. `if CONDITION then EXPRESSION end if;`
 - B. `if CONDITION then EXPRESSION elsif CONDITION then EXPRESSION else EXPRESSION end if;`
 - C. `if CONDITION then EXPRESSION else if CONDITION then EXPRESSION else EXPRESSION end if; end if;`
 - D. `if CONDITION then EXPRESSION else if CONDITION then EXPRESSION else EXPRESSION end if;`
 - E. `if CONDITION then EXPRESSION else EXPRESSION endif;`
 - F. `case EXPRESSION when 1 then EXPRESSION when 2 then EXPRESSION else EXPRESSION end case;`
 - G. `case EXPRESSION when 1 then EXPRESSION when 2 then EXPRESSION else EXPRESSION endcase;`
 - H. `case EXPRESSION when CONDITION then EXPRESSION else EXPRESSION end case;`
 - I. `case when CONDITION then EXPRESSION when CONDITION then EXPRESSION else EXPRESSION end case;`
 - J. `case when CONDITION then EXPRESSION when 1 then EXPRESSION else EXPRESSION end case;`

Réponse : D, E, G, H, J

2. Quelles sont les instructions de contrôle structurellement invalides ?
 - A. `while CONDITION loop`
`CONDITION:= NOT CONDITION; end loop;`
 - B. `while CONDITION`
`CONDITION:= NOT CONDITION; end loop;`
 - C. `while CONDITION loop`
`CONDITION:= NOT CONDITION; endloop;`
 - D. `loop exit; end loop;`
 - E. `whileloop exit; end loop;`
 - F. `loop exit; when CONDITION; end loop;`
 - G. `loop exit when CONDITION; end loop;`
 - H. `<<B01>>loop exit B01 when CONDITION; end loop;`
 - I. `for i in 1..3 loop NULL; end loop;`
 - J. `for i in 1 3 loop NULL; end loop;`
 - K. `for i in 1..3 NULL; end loop;`
 - L. `for i in 1..3 loop NULL; endloop;`

M. forall i in 1..3 ORDRE_DML;

N. forall i in 1..3 loop ORDRE_DML; end loop;

Réponse : B, C, E, J, K, L, N

Exercice n°1 Les structures conditionnelles

Créez le bloc PL/SQL qui permet d'effectuer les opérations :

- Pour les commandes de l'année '2011' augmentez les frais de port de '10%' pour tous les clients étrangers et diminuez les frais de port de '5%' pour les clients français. Contrôlez le nombre des enregistrements modifiés et si vous avez modifié des enregistrements, validez la transaction.

```
SQL> begin
2     UPDATE ( SELECT PAYS, PORT
3               FROM COMMANDES NATURAL JOIN CLIENTS
4               WHERE EXTRACT ( YEAR FROM DATE_COMMANDE) = 2011 )
5     SET PORT = PORT*( CASE PAYS WHEN 'France'
6                       THEN 1.1 ELSE .95 END);
7     if SQL%FOUND then
8         dbms_output.put_line('Enregistrements modifiés : '
9                               || SQL%ROWCOUNT);
10    COMMIT;
11    end if;
12 end;
13 /
```

- Augmentez le salaire de l'employé numéro 3 si le salaire de l'employé est inférieur à la moyenne des salaires des employés qui ont la même FONCTION. Modifiez également la commission du même employé si la commission est inférieure à la moyenne des commissions des employés qui ont la même FONCTION, on lui attribue la moyenne comme commission. Contrôlez le nombre des enregistrements modifiés et si vous avez modifié des enregistrements, validez la transaction.

```
SQL> declare
2     v_salaire          EMPLOYES.SALAIRE%TYPE;
3     v_commission       EMPLOYES.COMMISSION%TYPE;
4     v_avg_salaire       EMPLOYES.SALAIRE%TYPE;
5     v_avg_commission    EMPLOYES.COMMISSION%TYPE;
6     begin
7     SELECT SALAIRE, COMMISSION, SAL_AVG, COM_AVG
8     INTO v_salaire, v_commission,
9          v_avg_salaire, v_avg_commission
10    FROM EMPLOYES A,( SELECT FONCTION,
11                        AVG(SALAIRE) SAL_AVG,
12                        AVG(COMMISSION) COM_AVG
13                      FROM EMPLOYES
14                      GROUP BY FONCTION) B
15    WHERE A.FONCTION = B.FONCTION AND
16          A.NO_EMPLOYE = 3;
17    if v_salaire < v_avg_salaire then
18        v_salaire := v_salaire*1.1;
19    end if;
20    if v_commission < v_avg_commission then
21        v_commission := v_avg_commission;
```

```

22     end if;
23     UPDATE EMPLOYES SET
24         SALAIRE      = v_salaire,
25         COMMISSION   = v_commission
26     WHERE NO_EMPLOYE = 3;
27     if SQL%FOUND then
28         dbms_output.put_line('Enregistrements modifiés : '
29                               || SQL%ROWCOUNT);
30     COMMIT;
31     end if;
32 end;
33 /

```

Enregistrements modifiés : 1

Procédure PL/SQL terminée avec succès.

Exercice n°2 Les structures itératives

Créez le bloc PL/SQL qui permet d'effectuer les opérations :

- Affichez les chiffres de 1 à 10 comme dans le modèle suivant :

```

Le numéro 1 est impair
Le numéro 2 est pair
Le numéro 3 est impair
Le numéro 4 est pair
Le numéro 5 est impair
Le numéro 6 est pair
Le numéro 7 est impair
Le numéro 8 est pair
Le numéro 9 est impair
Le numéro 10 est pair

```

```

SQL> begin
2   for i in 1..10 loop
3       if mod( i, 2) = 0 then
4           dbms_output.put_line( 'Le numéro ' || i || ' est pair');
5       else
6           dbms_output.put_line( 'Le numéro ' || i || ' est impair');
7       end if;
8   end loop;
9   end;
10  /

```

- Déclarez un tableau de type NUMBER de dix postes, et deux boucles : une qui affecte le tableau avec les valeurs de 1 à 9 et une autre qui affiche le tableau à partir du dernier élément affecté.

```

SQL> declare
2   TYPE mon_type_tableau IS TABLE OF NUMBER
3       INDEX BY BINARY_INTEGER;
4   mon_tableau mon_type_tableau;
5   begin
6       for i in 1..9 loop
7           mon_tableau(i) := i;

```



```

8      end loop;
9      for v_compteur in reverse 1..9 loop
10         dbms_output.put_line( 'L'élément : ' ||
11                                mon_tableau(v_compteur));
12      end loop;
13  end;
14  /
L'élément : 9
L'élément : 8
L'élément : 7
L'élément : 6
L'élément : 5
L'élément : 4
L'élément : 3
L'élément : 2
L'élément : 1

```

Procédure PL/SQL terminée avec succès.

- Augmentez les salaires de '10%' pour tous les employés encadrés par 'Buchanan'. Augmentez la remise accordée par ces employés de '1%' ('REMISE + .01') pour toutes les commandes de l'année '2011'. Effacez tous les enregistrements de leurs commandes de la table INDICATEURS.

```

SQL> declare
2      TYPE t_no_emp IS TABLE OF EMPLOYES.NO_EMPLOYE%TYPE
3          INDEX BY BINARY_INTEGER;
4      v_no_emp t_no_emp;
5      nb_enreg SIMPLE_INTEGER := 0;
6  begin
7      UPDATE EMPLOYES
8          SET SALAIRE = SALAIRE * 1.1
9      WHERE REND_COMPTE = ( SELECT NO_EMPLOYE FROM EMPLOYES
10                             WHERE NOM = 'Buchanan' ) AND
11          NO_EMPLOYE <> REND_COMPTE
12      RETURNING NO_EMPLOYE BULK COLLECT INTO v_no_emp;
13      nb_enreg := SQL%ROWCOUNT;
14      dbms_output.put_line('Enregistrements modifiés : '
15                             || SQL%ROWCOUNT);
16      forall i in 1..v_no_emp.count
17          UPDATE DETAILS_COMMANDES SET REMISE = REMISE + 0.01
18          WHERE NO_COMMANDE in (SELECT NO_COMMANDE FROM COMMANDES
19                                 WHERE NO_EMPLOYE = v_no_emp(i)
20                                 AND ANNEE      = 2011);
21      dbms_output.put_line('Enregistrements modifiés : '
22                             || SQL%ROWCOUNT);
23      nb_enreg := nb_enreg + SQL%ROWCOUNT;
24      forall i in 1..v_no_emp.count
25          DELETE INDICATEURS WHERE NO_EMPLOYE = v_no_emp(i);
26      dbms_output.put_line('Enregistrements modifiés : '
27                             || SQL%ROWCOUNT);
28      nb_enreg := nb_enreg + SQL%ROWCOUNT;
29      if nb_enreg > 0 then
30          COMMIT;
31      end if;

```

```

32 end;
33 /

```

- Augmentez le prix unitaire de '10%' des produits de la catégorie 3. Mettez à jour les prix unitaires des produits pour les commandes de l'année '2011'.

```

SQL> declare
2     TYPE t_ref_prod IS TABLE OF PRODUITS.REF_PRODUIT%TYPE
3         INDEX BY BINARY_INTEGER;
4     TYPE t_prix      IS TABLE OF PRODUITS.PRIX_UNITAIRE%TYPE
5         INDEX BY BINARY_INTEGER;
6     v_ref_prod t_ref_prod;
7     v_prix      t_prix;
8     nb_enreg SIMPLE_INTEGER := 0;
9 begin
10    UPDATE PRODUITS
11        SET PRIX_UNITAIRE = PRIX_UNITAIRE * 1.1
12    WHERE CODE_CATEGORIE = 3
13    RETURNING REF_PRODUIT, PRIX_UNITAIRE
14    BULK COLLECT INTO v_ref_prod, v_prix;
15    dbms_output.put_line('Enregistrements modifiés : '
16                          || SQL%ROWCOUNT);
17    nb_enreg := SQL%ROWCOUNT;
18    forall i in 1..v_ref_prod.count
19        UPDATE DETAILS_COMMANDES SET PRIX_UNITAIRE = v_prix(i)
20        WHERE REF_PRODUIT in v_ref_prod(i)
21        AND NO_COMMANDE in
22            ( SELECT NO_COMMANDE FROM COMMANDES
23              WHERE ANNEE = 2011);
24    dbms_output.put_line('Enregistrements modifiés : '
25                          || SQL%ROWCOUNT);
26    nb_enreg := nb_enreg + SQL%ROWCOUNT;
27    if nb_enreg > 0 then
28        null;--COMMIT;
29    end if;
30 end;
31 /

```

Atelier 6.1 Les curseurs

Questions



```

SQL> declare
2     CURSOR c_1 IS SELECT * FROM PRODUITS;
3     v_1 c_1%ROWTYPE;
4 begin
5     /*A*/if c_1%FOUND then
6         dbms_output.put_line('/*A*/ %FOUND'); end if;
7     /*B*/if c_1%ISOPEN then
8         dbms_output.put_line('/*B*/ %ISOPEN'); end if;
9     /*C*/if c_1%NOTFOUND then
10        dbms_output.put_line('/*C*/ %NOTFOUND');end if;
11    /*D*/if c_1%ROWCOUNT >0 then
12        dbms_output.put_line('/*D*/ %ISOPEN'); end if;
13        open c_1;
14    /*E*/if c_1%FOUND then
15        dbms_output.put_line('/*E*/ %FOUND'); end if;
16    /*F*/if c_1%ISOPEN then
17        dbms_output.put_line('/*F*/ %ISOPEN'); end if;
18    /*G*/if c_1%NOTFOUND then
19        dbms_output.put_line('/*G*/ %NOTFOUND');end if;
20    /*H*/if c_1%ROWCOUNT = 0 then
21        dbms_output.put_line('/*H*/ %ROWCOUNT'); end if;
22        loop
23            fetch c_1 into v_1;
24            exit when c_1%NOTFOUND;
25        end loop;
26    /*I*/if c_1%FOUND then
27        dbms_output.put_line('/*I*/ %FOUND'); end if;
28    /*J*/if c_1%ISOPEN then
29        dbms_output.put_line('/*J*/ %ISOPEN'); end if;
30    /*K*/if c_1%NOTFOUND then
31        dbms_output.put_line('/*K*/ %NOTFOUND');end if;
32    /*L*/if c_1%ROWCOUNT >0 then
33        dbms_output.put_line('/*L*/ %ROWCOUNT'); end if;
34        close c_1;
35    /*M*/if c_1%FOUND then
36        dbms_output.put_line('/*M*/ %FOUND'); end if;
37    /*N*/if c_1%ISOPEN then
38        dbms_output.put_line('/*N*/ %ISOPEN'); end if;
39    /*O*/if c_1%NOTFOUND then
40        dbms_output.put_line('/*O*/ %NOTFOUND');end if;
41    /*P*/if c_1%ROWCOUNT >0 then
42        dbms_output.put_line('/*P*/ %ISOPEN'); end if;
43 end;
44 /

```

1. Quelles sont les instructions qui génèrent une erreur due à une lecture des attributs du curseur ?

Réponse : A, C, D, M, O, P

2. Si on efface les instructions qui génèrent une erreur, quelles sont les autres instructions qui valident la condition et affichent la chaîne avec leur lettre et leur attribut ?

Réponse : F, H, J, K, L

Exercice n°1 Les curseurs explicites

Créez le bloc PL/SQL qui permet d'effectuer les opérations :

- À l'aide d'un curseur explicite, insérez les enregistrements correspondants dans la table du modèle étoile QUANTITES_CLIENTS. Avant d'insérer, effacez tous les enregistrements.

```
SQL> declare
2     CURSOR c_quantite_clients IS
3         SELECT EXTRACT ( YEAR  FROM DATE_COMMANDE) ANNEE,
4                EXTRACT ( MONTH FROM DATE_COMMANDE) MOIS,
5                CODE_CLIENT,
6                SUM(QUANTITE) QUANTITE,
7                SUM(PORT) PORT
8         FROM   COMMANDES NATURAL JOIN DETAILS_COMMANDES
9         GROUP BY EXTRACT ( YEAR  FROM DATE_COMMANDE),
10                EXTRACT ( MONTH FROM DATE_COMMANDE),
11                CODE_CLIENT;
12     v_quantite_clients c_quantite_clients%ROWTYPE;
13 begin
14     delete QUANTITES_CLIENTS;
15     open c_quantite_clients;
16     if c_quantite_clients%ISOPEN then
17         loop
18             fetch c_quantite_clients into v_quantite_clients;
19             exit when c_quantite_clients%NOTFOUND;
20             INSERT INTO QUANTITES_CLIENTS VALUES v_quantite_clients;
21         end loop;
22     end if;
23     close c_quantite_clients;
24     COMMIT;
25 end;
26 /
```

- À l'aide d'un curseur explicite, insérez les enregistrements correspondants dans la table du modèle étoile VENTES_MOIS uniquement pour l'année passée en argument au curseur. Le bloc doit utiliser une variable de substitution pour alimenter une variable PL/SQL. Avant d'insérer, effacez tous les enregistrements de l'année qui a été passée en argument.

```
SQL> declare
2     CURSOR c_ventes_mois ( a_annee  NUMBER := 1998) IS
3         SELECT EXTRACT ( YEAR  FROM DATE_COMMANDE) ANNEE,
4                EXTRACT ( MONTH FROM DATE_COMMANDE) MOIS,
5                SUM(QUANTITE*PRIX_UNITAIRE) VENTE,
6                SUM(QUANTITE*PRIX_UNITAIRE*REMISE) REMISE
```

```

7          FROM  COMMANDES NATURAL JOIN DETAILS_COMMANDES
8          WHERE EXTRACT ( YEAR  FROM DATE_COMMANDE) = a_annee
9          GROUP BY EXTRACT ( YEAR  FROM DATE_COMMANDE),
10                 EXTRACT ( MONTH FROM DATE_COMMANDE);
11      v_ventes_mois c_ventes_mois%ROWTYPE;
12      v_annee  NUMBER(4) := &annee_du_calcul;
13  begin
14      delete VENTES_MOIS WHERE ANNEE = v_annee;
15      open c_ventes_mois(v_annee);
16      if c_ventes_mois%ISOPEN then
17          loop
18              fetch c_ventes_mois into v_ventes_mois;
19              exit when c_ventes_mois%NOTFOUND;
20              INSERT INTO VENTES_MOIS VALUES v_ventes_mois;
21          end loop;
22      end if;
23      close c_ventes_mois;
24      COMMIT;
25  end;
26  /

```

- À l'aide d'un curseur explicite, affichez l'employé, la fonction à partir de la table du modèle étoile DIM_EMPLOYES.

```

SQL> declare
2      CURSOR c_emp IS SELECT EMPLOYE,FONCTION FROM DIM_EMPLOYES;
3      TYPE t_emp  IS TABLE OF c_emp%ROWTYPE;
4      v_emp t_emp;
5  begin
6      open c_emp;
7      fetch c_emp bulk collect into v_emp;
8      for i in 1..v_emp.count
9      loop
10         dbms_output.put_line( 'Employé : '||v_emp(i).EMPLOYE||
11                                ' -- Fonction :'||v_emp(i).FONCTION);
12      end loop;
13      close c_emp;
14  end;
15  /

```

Atelier 6.2 Les curseurs

Exercice n°1 Les boucles et les curseurs

Créez le bloc PL/SQL qui permet d'effectuer les opérations :

- À l'aide d'un curseur et de la boucle « **FOR** », affichez les clients, leur ville et leur pays à partir de la table du modèle étoile DIM_CLIENTS.

```
SQL> set serveroutput on size 1000000
SQL> begin
  2   for i in ( SELECT CLIENT, VILLE ,PAYS FROM DIM_CLIENTS)
  3   loop
  4       dbms_output.put_line( 'Client : '||i.CLIENT||
  5           ' -- Localisé :'||i.VILLE||' '||i.PAYS);
  6   end loop;
  7 end;
  8 /
```

- À l'aide d'un curseur et de la boucle « **FOR** », insérez les enregistrements correspondants dans la table du modèle étoile VENTES_CLIENTS uniquement pour l'année passée en argument au curseur. Le bloc doit utiliser une variable de substitution pour alimenter une variable PL/SQL. Avant d'insérer, effacez tous les enregistrements de l'année qui a été passée en argument.

```
SQL> declare
  2   v_annee  NUMBER(4) := &annee_du_calcul;
  3   begin
  4   DELETE VENTES_CLIENTS WHERE ANNEE = v_annee;
  5   for i in ( SELECT EXTRACT ( YEAR  FROM DATE_COMMANDE) ANNEE,
  6               EXTRACT ( MONTH FROM DATE_COMMANDE) MOIS,
  7               CODE_CLIENT,
  8               SUM(QUANTITE*PRIX_UNITAIRE) VENTE,
  9               SUM(QUANTITE*PRIX_UNITAIRE*REMISE) REMISE
 10   FROM   COMMANDES NATURAL JOIN DETAILS_COMMANDES
 11   WHERE  EXTRACT( YEAR  FROM DATE_COMMANDE) = v_annee
 12   GROUP BY EXTRACT( YEAR  FROM DATE_COMMANDE),
 13            EXTRACT( MONTH FROM DATE_COMMANDE),
 14            CODE_CLIENT)
 15   loop
 16       INSERT INTO VENTES_CLIENTS VALUES i;
 17   end loop;
 18   COMMIT;
 19 end;
 20 /
```

- En utilisant deux curseurs déclarés directement dans la boucle « **FOR** », affichez les clients et commandes pour les clients qui payent un port supérieur à trois fois la moyenne des commandes pour la même année.

```
SQL> BEGIN
  2   for v_clients in ( SELECT * FROM CLIENTS ) loop
  3       dbms_output.put_line('Client -----'||v_clients.SOCIETE );
  4       for v_commandes in
  5           ( SELECT NO_COMMANDE FROM COMMANDES A
```

```

6          WHERE CODE_CLIENT = v_clients.CODE_CLIENT AND
7          PORT > 3*( SELECT AVG(PORT) FROM COMMANDES B
8                     WHERE TRUNC(A.DATE_COMMANDE,'year') =
9                           TRUNC(B.DATE_COMMANDE,'year'))
10      loop
11          dbms_output.put_line('---' || v_commandes.NO_COMMANDE );
12      end loop;
13  end loop;
14  END;
15  /

```

Exercice n°2 Les curseurs en mise à jour

Créez le bloc PL/SQL qui permet de mettre les frais de port à zéro pour toutes les commandes de clients qui habitent dans la même ville que les fournisseurs des produits achetés pour uniquement l'année passée en argument au curseur. Les modifications sont effectuées dans la table COMMANDES du modèle relationnel mais en même temps vous devez effacer les enregistrements de ces commandes dans la table INDICATEURS du modèle étoile.

```

SQL> declare
2      CURSOR c_commandes (a_annee NUMBER)
3      IS
4          SELECT B.NO_COMMANDE, B.PORT
5          FROM CLIENTS A,
6               COMMANDES B,
7               DETAILS_COMMANDES C,
8               PRODUITS D, FOURNISSEURS E
9          WHERE EXTRACT(YEAR FROM B.DATE_COMMANDE) = a_annee AND
10                A.VILLE = E.VILLE AND
11                A.CODE_CLIENT = B.CODE_CLIENT AND
12                B.NO_COMMANDE = C.NO_COMMANDE AND
13                C.REF_PRODUIT = D.REF_PRODUIT AND
14                D.NO_FOURNISSEUR = E.NO_FOURNISSEUR
15          FOR UPDATE OF B.PORT;
16      v_annee NUMBER(4) := &annee_du_calcul;
17  begin
18      for v_commandes in c_commandes(v_annee)
19      loop
20          UPDATE COMMANDES SET PORT = 0 WHERE CURRENT OF c_commandes;
21          DELETE INDICATEURS
22              WHERE NO_COMMANDE = v_commandes.NO_COMMANDE;
23          dbms_output.put_line( 'Numéro de commande effacé : ' ||
24                                v_commandes.NO_COMMANDE );
25      end loop;
26      COMMIT;
27  end;
28  /

```

Exercice n°3 Les variables curseurs

Créez le bloc PL/SQL qui permet d'afficher à partir du modèle étoile l'une des tables :

– VENTES_CLIENTS_2009,

- VENTES_CLIENTS_2010 ou
- VENTES_CLIENTS_2011

Dynamiquement, suivant l'année passée en argument, vous testez que la table existe et vous affichez tous les enregistrements de la table. Si la table n'existe pas, vous affichez tous les enregistrements de la table VENTES_CLIENTS pour l'année qui a été passée en argument.

```
SQL> declare
  2   TYPE curs_aff IS REF CURSOR;
  3   c_aff          curs_aff;
  4   v_aff          VENTES_CLIENTS%ROWTYPE;
  5   v_annee        NUMBER(4)      := &annee_du_calcul;
  6   v_SQL_SELECT   varchar2(2000)
  7                   := ' MOIS, CODE_CLIENT, VENTE, REMISE FROM ';
  8   v_table         varchar2(2000);
  9   begin
 10     for i in ( SELECT TABLE_NAME FROM USER_TABLES
 11               WHERE TABLE_NAME = 'VENTES_CLIENTS_' || v_annee)
 12     loop
 13       v_table := i.TABLE_NAME;
 14     end loop;
 15     if length(v_table) > 0 then
 16       v_SQL_SELECT := ' SELECT ' || v_annee || ' ANNEE, ' ||
 17                      v_SQL_SELECT || v_table;
 18     else
 19       v_SQL_SELECT := ' SELECT ANNEE, ' || v_SQL_SELECT ||
 20                      'VENTES_CLIENTS';
 21     end if;
 22     open c_aff FOR v_SQL_SELECT;
 23     loop
 24       fetch c_aff into v_aff;
 25       exit when c_aff%NOTFOUND;
 26       dbms_output.put_line( v_aff.ANNEE || ' ' || v_aff.MOIS || ' ' ||
 27                             v_aff.CODE_CLIENT || ' ' || v_aff.VENTE || ' ' || v_aff.REMISE);
 28     end loop;
 29     close c_aff;
 30   end;
 31   /
```


Atelier 7.1 Les exceptions

Questions



```

SQL> declare
2      EXCEPTION_1 EXCEPTION;
3      EXCEPTION_2 EXCEPTION;
4      EXCEPTION_3 EXCEPTION;
5      EXCEPTION_4 EXCEPTION;
6      EXCEPTION_5 EXCEPTION;
7  begin --bloc 1
8      begin --bloc 2
9          begin --bloc 3
10             begin --bloc 4
11                 begin --bloc 5
12                     RAISE EXCEPTION_5;
13                     dbms_output.put_line( 'Suite traitements bloc 5. ');
14                     exception--exception bloc 5
15                     when EXCEPTION_5 then
16                         dbms_output.put_line( 'Exception EXCEPTION_5 ');
17                         RAISE EXCEPTION_4;
18                     end;--end bloc 5
19                     dbms_output.put_line( 'Suite traitements bloc 4. ');
20                     exception--exception bloc 4
21                     when EXCEPTION_4 then
22                         dbms_output.put_line( 'Exception EXCEPTION_4 ');
23                         RAISE EXCEPTION_3;
24                     end;--end bloc 4
25                     dbms_output.put_line( 'Suite traitements bloc 3. ');
26                     exception--exception bloc 3
27                     when EXCEPTION_3 then
28                         dbms_output.put_line( 'Exception EXCEPTION_3 ');
29                         RAISE EXCEPTION_2;
30                     end;--end bloc 3
31                     dbms_output.put_line( 'Suite traitements bloc 2. ');
32                     exception--exception bloc 2
33                     when EXCEPTION_2 then
34                         dbms_output.put_line( 'Exception EXCEPTION_2 ');
35                         RAISE EXCEPTION_1;
36                     end;--end bloc 2
37                     dbms_output.put_line( 'Suite traitements bloc 1. ');
38                     exception--exception bloc 1
39                     when EXCEPTION_1 then
40                         dbms_output.put_line( 'Exception EXCEPTION_1 ');
41                     when OTHERS then
42                         dbms_output.put_line( 'Une autre erreur. ');
43                     end;--end bloc 1
44 /

```

1. Quel est l'affichage effectué par le bloc suite au traitement ?

A.

```
Exception EXCEPTION_5
Suite traitements bloc 4.
Exception EXCEPTION_4
Suite traitements bloc 3.
Exception EXCEPTION_3
Suite traitements bloc 2.
Exception EXCEPTION_2
Suite traitements bloc 1.
Exception EXCEPTION_1
```

B.

```
Exception EXCEPTION_5
Suite traitements bloc 4.
Suite traitements bloc 3.
Suite traitements bloc 2.
Suite traitements bloc 1.
```

C.

```
Exception EXCEPTION_5
Exception EXCEPTION_4
Exception EXCEPTION_3
Exception EXCEPTION_2
Exception EXCEPTION_1
```

D.

```
Suite traitements bloc 5.
Suite traitements bloc 4.
Suite traitements bloc 3.
Suite traitements bloc 2.
Suite traitements bloc 1.
```

E.

Une autre erreur.

Réponse : C

Exercice n°1 La gestion des exceptions

Créez le bloc PL/SQL qui permet d'effectuer les opérations :

- Récupérez dans trois variables PL/SQL les valeurs saisies à l'aide des variables de substitution. Les variables représentent le code d'une catégorie, le numéro d'un fournisseur et la référence d'un produit qui doivent être effacés. Ainsi vous effacez un enregistrement dans la table CATEGORIES, un enregistrement de la table FOURNISSEURS et un enregistrement de la table PRODUITS. Il faut enchaîner les traitements dans plusieurs blocs de sorte que si une de ces commandes n'aboutit pas, les suivantes seront exécutées quand même.

```
SQL> begin
2   begin
3     DELETE CATEGORIES WHERE CODE_CATEGORIE = &CODE_CATEGORIE;
4   exception
5     when OTHERS then dbms_output.put_line(
6       'La table CATEGORIES SQLCODE = ' || SQLCODE
7       || ' : ' || SQLERRM(SQLCODE));
8   end;
9   begin
```

```

10      DELETE FOURNISSEURS WHERE NO_FOURNISSEUR = &NO_FOURNISSEUR;
11  exception
12      when OTHERS then dbms_output.put_line(
13          'La table FOURNISSEURS SQLCODE = ' || SQLCODE
14          || ' : ' || SQLERRM(SQLCODE));
15  end;
16  begin
17      DELETE PRODUITS WHERE REF_PRODUIT = &REF_PRODUIT;
18  exception
19      when OTHERS then dbms_output.put_line(
20          'La table PRODUITS SQLCODE = ' || SQLCODE
21          || ' : ' || SQLERRM(SQLCODE));
22  end;
23  end;
24  /

```

Entrez une valeur pour code_categorie : 1

Entrez une valeur pour no_fournisseur : 1

Entrez une valeur pour ref_produit : 1

La table CATEGORIES SQLCODE = -2292 : ORA-02292: violation de contrainte

La table FOURNISSEURS SQLCODE = -2292 : ORA-02292: violation de contrainte

La table PRODUITS SQLCODE = -2292 : ORA-02292: violation de contrainte

- Ecrire le code permettant de générer et de traiter chacune des exceptions suivantes :

```

CASE_NOT_FOUND,
CURSOR_ALREADY_OPEN,
DUP_VAL_ON_INDEX,
INVALID_CURSOR,
INVALID_NUMBER,
NO_DATA_FOUND,
TOO_MANY_ROWS,
VALUE_ERROR,
ZERO_DIVIDE

```

Suivant les choix à l'exécution, par la saisie d'une variable de substitution, vous exécutez le code erroné qui passe directement à la section de l'exception choisie du programme qui affiche le nom de l'exception.

```

SQL> declare
2      TYPE t_1 IS TABLE OF NUMBER(2) INDEX BY BINARY_INTEGER;
3      CURSOR c_1 IS SELECT * FROM CAT;
4      vt_1      t_1;
5      v_1      c_1%ROWTYPE;
6      v_argument NUMBER(2) := &no_de_1_a_9;
7  begin
8      case v_argument
9      when 1 then --CASE_NOT_FOUND
10         dbms_output.put_line('SQL exécuté : case USER when ' ||
11             ''A'' then null; end case;');
12         case USER when 'A' then null; end case;
13      when 2 then --CURSOR_ALREADY_OPEN
14         dbms_output.put_line('SQL exécuté : OPEN c_1;OPEN c_1;');
15         OPEN c_1;OPEN c_1;
16      when 3 then --DUP_VAL_ON_INDEX
17         dbms_output.put_line('SQL exécuté : INSERT INTO ' ||

```

```

18         'CATEGORIES VALUES ( 1, 'a', 'b');');
19     INSERT INTO CATEGORIES VALUES ( 1, 'a', 'b');
20     when 4 then --INVALID_CURSOR
21         dbms_output.put_line('SQL exécuté : FETCH c_1 INTO v_1;');
22         FETCH c_1 INTO v_1;
23     when 5 then --INVALID_NUMBER
24         dbms_output.put_line('SQL exécuté : INSERT INTO ' ||
25             'CATEGORIES VALUES ( 'a', 'a', 'b');');
26         INSERT INTO CATEGORIES VALUES ( 'a', 'a', 'b');
27     when 6 then --NO_DATA_FOUND
28         dbms_output.put_line('SQL exécuté : v_argument ' ||
29             ':= vt_1(100);');
30         v_argument := vt_1(100);
31         -- ou
32         SELECT * INTO v_1 FROM CAT WHERE 1=2;
33     when 7 then --TOO_MANY_ROWS
34         dbms_output.put_line('SQL exécuté : SELECT * ' ||
35             'INTO v_1 FROM CAT;');
36         SELECT * INTO v_1 FROM CAT;
37     when 8 then --VALUE_ERROR
38         dbms_output.put_line('SQL exécuté : v_argument := 'a';');
39         v_argument := 'a';
40     when 9 then --ZERO_DIVIDE
41         dbms_output.put_line('SQL exécuté : v_argument ' ||
42             ':= v_argument / 0;');
43         v_argument := v_argument / 0;
44     else
45         NULL;
46     end case;
47 exception
48     when CASE_NOT_FOUND then
49         dbms_output.put_line('Exception CASE_NOT_FOUND: '
50             || SQLERRM(SQLCODE));
51     when CURSOR_ALREADY_OPEN then
52         dbms_output.put_line('Exception CURSOR_ALREADY_OPEN: '
53             || SQLERRM(SQLCODE));
54     when DUP_VAL_ON_INDEX then
55         dbms_output.put_line('Exception DUP_VAL_ON_INDEX: '
56             || SQLERRM(SQLCODE));
57     when INVALID_CURSOR then
58         dbms_output.put_line('Exception INVALID_CURSOR: '
59             || SQLERRM(SQLCODE));
60     when INVALID_NUMBER then
61         dbms_output.put_line('Exception INVALID_NUMBER: '
62             || SQLERRM(SQLCODE));
63     when NO_DATA_FOUND then
64         dbms_output.put_line('Exception NO_DATA_FOUND: '
65             || SQLERRM(SQLCODE));
66     when TOO_MANY_ROWS then
67         dbms_output.put_line('Exception TOO_MANY_ROWS: '
68             || SQLERRM(SQLCODE));
69     when VALUE_ERROR then
70         dbms_output.put_line('Exception VALUE_ERROR: '
71             || SQLERRM(SQLCODE));

```

```

72      when ZERO_DIVIDE then
73          dbms_output.put_line('Exception ZERO_DIVIDE: '
74                                || SQLERRM(SQLCODE));
75      when OTHERS then
76          dbms_output.put_line('Exception OTHERS: '
77                                || SQLERRM(SQLCODE));
78  end;
79  /

```

Exercice n°2 Les exceptions anonymes

Créez le bloc PL/SQL qui permet d'effectuer les opérations :

- A chaque fois qu'on efface une commande saisie par l'utilisateur, gérez l'exception « ORA-02292: violation de contrainte (.) d'intégrité - enregistrement fils existant », en effaçant tous les enregistrements de la table DETAILS_COMANDES correspondantes.

```

SQL> declare
2      DELETE_CASCADE_ENFANT EXCEPTION;
3      PRAGMA EXCEPTION_INIT(DELETE_CASCADE_ENFANT, -2292);
4      v_no_commande COMMANDES.NO_COMMANDE%TYPE := &no_commande;
5  begin
6      DELETE COMMANDES WHERE NO_COMMANDE = v_no_commande;
7  exception
8      when DELETE_CASCADE_ENFANT then
9          dbms_output.put_line( 'Exception : DELETE_CASCADE_ENFANT ');
10         DELETE DETAILS_COMMANDES WHERE NO_COMMANDE = v_no_commande;
11         DELETE COMMANDES WHERE NO_COMMANDE = v_no_commande;
12  end;
13  /

```

Exercice n°2 Les exceptions utilisateur

Créez le bloc PL/SQL qui permet d'effectuer les opérations :

- Modifiez le salaire d'un employé pour le NO_EMPLOYE saisi à l'exécution. Contrôlez que le salaire ne soit pas inférieur au salaire actuel ; si c'est le cas, lancez une exception.

```

SQL> declare
2      UPDATE_EMPLOYES EXCEPTION;
3      v_no_employe EMPLOYES.NO_EMPLOYE%TYPE;
4      v_salaire EMPLOYES.SALAIRE%TYPE;
5  begin
6      v_no_employe := &no_employe;
7      for emp in ( SELECT SALAIRE FROM EMPLOYES
8                    WHERE NO_EMPLOYE = v_no_employe) loop
9          v_salaire := &salaire;
10         if v_salaire < emp.salaire then
11             dbms_output.put_line( 'Le salaire actuel est :'||
12                                     emp.salaire);
13             RAISE UPDATE_EMPLOYES;
14         end if;
15     end loop;

```

```

16      UPDATE EMPLOYES SET SALAIRE = v_salaire
17      WHERE NO_EMPLOYE = v_no_employe;
18  exception
19      when UPDATE_EMPLOYES then
20          dbms_output.put_line(
21              'Exception utilisateur : UPDATE_EMPLOYES ');
22  end;
23  /

```

- Permettre de saisir les informations d'un employé et de les insérer dans la table EMPLOYES. Si l'âge de l'employé est inférieur à 18 ans, n'insérez pas et lancez une exception.

```

SQL> declare
2      UPDATE_EMPLOYES EXCEPTION;
3      v_employe EMPLOYES%ROWTYPE;
4  begin
5      v_employe.NO_EMPLOYE      := &NO_EMPLOYE;
6      v_employe.REND_COMPTE     := &REND_COMPTE;
7      v_employe.NOM             := '&NOM';
8      v_employe.PRENOM          := '&PRENOM';
9      v_employe.FONCTION        := '&FONCTION';
10     v_employe.TITRE            := '&TITRE';
11     v_employe.DATE_NAISSANCE   := '&DATE_NAISSANCE';
12     v_employe.DATE_EMBAUCHE   := '&DATE_EMBAUCHE';
13     v_employe.SALAIRE          := &SALAIRE;
14     v_employe.COMMISSION       := &COMMISSION;
15     if (SYSDATE - v_employe.DATE_NAISSANCE)/365 < 18 then
16         RAISE UPDATE_EMPLOYES;
17     end if;
18     INSERT INTO EMPLOYES VALUES v_employe;
19  exception
20      when UPDATE_EMPLOYES then
21          dbms_output.put_line(
22              'Exception utilisateur : UPDATE_EMPLOYES ');
23  end;
24  /

```

Atelier 8.1 Les sous-programmes

Questions



```
SQL> declare
2   PROCEDURE ProcedureA IS
3   begin
4       dbms_output.put_line( 'ProcedureA' );
5       ProcedureB;
6   end;
7   PROCEDURE ProcedureB IS
8   begin
9       dbms_output.put_line( 'ProcedureB' );
10  end;
11 begin
12     ProcedureA;
13 end;
14 /
```

1. Le bloc précédent peut-il être compilé ? justifiez votre réponse.

Réponse : Non. La procédure ProcedureA appelle la procédure ProcedureB, alors ProcedureB doit être déclarée avant ProcedureA de sorte que la référence à ProcedureB puisse être résolue.

2. Quelles sont les syntaxes invalides ?
 - A. PROCEDURE p1(a_1 NUMBER) IS declare a_1 := 0; end;
 - B. PROCEDURE p1(a_1 IN OUT NUMBER) IS declare a_1 := 0; end;
 - C. PROCEDURE p1(a_1 OUT NUMBER) IS declare a_1 := 0; end;
 - D. PROCEDURE p1(a_1 IN NUMBER) IS declare a_1 := 0; end;

Réponse : A, D

```
SQL> CREATE OR REPLACE PROCEDURE
2   p1( a_1 NUMBER := 0, a_2 NUMBER := 0, a_3 NUMBER := 0 ) IS
3   begin
4       dbms_output.put_line( a_1 || ' ' || a_2 || ' ' || a_3 );
5   end;
6   /
```

3. Quelles sont les syntaxes qui affichent la chaîne suivante '1 2 0' ?
 - A. exec p1;
 - B. exec p1(1,2,3);
 - C. exec p1(1,2);
 - D. exec p1(3);

- E. `exec p1(a_3 => 3,a_1 => 1);`
- F. `exec p1(a_2 => 3,a_1 => 1);`
- G. `exec p1(a_3 => 0,a_1 => 1,a_2 => 2);`
- H. `exec p1(a_1 => 3,a_2 => 2,a_3 => 1);`
- I. `exec p1(a_1 => 3,a_2 => 0,a_3 => 0);`
- J. `exec p1(a_2 => 2,a_1 => 1,a_3 => 0);`

Réponse : C, G, J

Exercice n°1 Les fonctions

Créez une fonction qui permet d'effectuer les opérations suivantes :

- A partir d'une catégorie des produits passée comme argument, la fonction doit retourner VRAI si l'enregistrement existe dans la table CATEGORIES, et FALSE dans le cas contraire.

```
SQL> CREATE OR REPLACE FUNCTION
 2 VerifieCategorie(a_code_cat CATEGORIES.CODE_CATEGORIE%TYPE)
 3 RETURN BOOLEAN
 4 AS
 5 begin
 6     for v_code_categorie in ( SELECT CODE_CATEGORIE
 7                               FROM CATEGORIES
 8                               WHERE CODE_CATEGORIE = a_code_cat)
 9     loop
10         RETURN TRUE;--Catégorie trouvé
11     end loop;
12     RETURN FALSE;--Catégorie non trouvé
13 end;
14 /
```

ou

```
SQL> CREATE OR REPLACE FUNCTION
 2 VerifieCategorie(a_code_cat IN OUT
 3                  CATEGORIES.CODE_CATEGORIE%TYPE)
 4 RETURN BOOLEAN
 5 AS
 6 begin
 7     SELECT CODE_CATEGORIE INTO a_code_cat FROM CATEGORIES
 8     WHERE CODE_CATEGORIE = a_code_cat;
 9     RETURN TRUE;--Catégorie trouvé
10 exception
11     when NO_DATA_FOUND then RETURN FALSE;--Catégorie non trouvé
12 end;
13 /
```

- A partir du numéro du fournisseur passé comme argument, la fonction doit retourner VRAI si l'enregistrement existe dans la table FOURNISSEURS, et sinon FALSE.

```
SQL> CREATE OR REPLACE FUNCTION
 2 VerifieFournisseur(a_no FOURNISSEURS.NO_FOURNISSEUR%TYPE)
 3 RETURN BOOLEAN
```



```

4 AS
5 begin
6   for v_code_categorie in ( SELECT NO_FOURNISSEUR
7                             FROM FOURNISSEURS WHERE NO_FOURNISSEUR = a_no)
8   loop
9     RETURN TRUE;--Fournisseur trouvé
10  end loop;
11  RETURN FALSE;--Fournisseur non trouvé
12 end;
13 /

```

- A partir du nom de l'employé et des deux dates passés comme arguments, retrouvez le nombre des contrats saisis dans la table COMMANDES.

```

SQL> CREATE OR REPLACE FUNCTION NombreContrats
2      ( a_nom VARCHAR2, a_debut DATE := NULL,a_fin DATE := NULL)
3  RETURN NUMBER
4  AS
5  begin
6    for i in ( SELECT count(NO_COMMANDE) NB_CONTRATS
7              FROM COMMANDES
8              WHERE ( DATE_COMMANDE BETWEEN a_debut AND a_fin OR
9                    a_debut IS NULL OR a_fin IS NULL ) AND
10                 NO_EMPLOYE = ( SELECT NO_EMPLOYE FROM EMPLOYES
11                               WHERE NOM = a_nom))
12    loop
13      RETURN i.NB_CONTRATS;
14    end loop;
15    RETURN 0;
16 end;
17 /

```

- A partir du numéro de la commande, retournez le montant de la commande.

```

SQL> CREATE OR REPLACE FUNCTION MontantCommande
2      ( a_no COMMANDES.NO_COMMANDE%TYPE)
3  RETURN NUMBER
4  AS
5      v_montant DETAILS_COMMANDES.PRIX_UNITAIRE%TYPE;
6  begin
7      SELECT SUM(QUANTITE*PRIX_UNITAIRE) INTO v_montant
8      FROM   COMMANDES NATURAL JOIN DETAILS_COMMANDES
9      GROUP BY NO_COMMANDE;
10     RETURN v_montant;
11 end;
12 /

```

- A partir du nom de la table passée comme argument et de la valeur de la clé pour un enregistrement, valeur passée sous forme de chaîne de caractères. Contrôlez que le nom de la table existe, retrouvez le nom de la colonne de clé primaire et créez une requête dynamique qui vérifie que l'enregistrement existe. La fonction doit retourner VRAI si l'enregistrement existe, sinon FALSE.

```

SQL> CREATE OR REPLACE FUNCTION
2  VerifieEnregDansLaTable( a_table VARCHAR2, a_valeur VARCHAR2 )
3  RETURN BOOLEAN
4  AS
5      v_requete          varchar(500) := 'SELECT COUNT(*) FROM ' ;

```

```

6      v_num          NUMBER;
7      v_cle          USER_CONS_COLUMNS.COLUMN_NAME%TYPE;
8      v_type         USER_TAB_COLUMNS.DATA_TYPE%TYPE;
9  begin
10     SELECT COLUMN_NAME, DATA_TYPE INTO v_cle, v_type
11     FROM USER_CONSTRAINTS NATURAL JOIN USER_CONS_COLUMNS
12          JOIN USER_TAB_COLUMNS USING( TABLE_NAME, COLUMN_NAME )
13     WHERE CONSTRAINT_TYPE = 'P' AND
14           TABLE_NAME = UPPER( a_table);
15     v_requete := v_requete || a_table || ' WHERE ' || v_cle;
16     case v_type
17     when 'NUMBER' then
18         v_requete := v_requete || ' = ' || a_valeur;
19     when 'DATE' then
20         if not REGEXP_LIKE( a_valeur,
21             '[[:digit:]]{2}\/[[:digit:]]{2}\/[[:digit:]]{4}') then
22             dbms_output.put_line('Ce n''est pas un format de ' ||
23                 'date valide. Le format est : ''DD/MM/YYYY''');
24             RETURN FALSE;
25         end if;
26         v_requete := v_requete || ' = TO_DATE(' || ''' ||
27             a_valeur || ''', 'DD/MM/YYYY')';
28     else
29         v_requete := v_requete || ' LIKE ' || ''' || a_valeur || ''';
30     end case;
31     EXECUTE IMMEDIATE v_requete INTO v_num;
32     if v_num = 1 then
33         RETURN TRUE;
34     else
35         RETURN FALSE;
36     end if;
37 exception
38     when NO_DATA_FOUND then
39         dbms_output.put_line('La table n''a pas de clé ' ||
40             'primaire ou elle n''existe pas. ');
41         RETURN FALSE;
42     when TOO_MANY_ROWS then
43         dbms_output.put_line(
44             'La table a une clé primaire multiple');
45         RETURN FALSE;
46 end VerifieEnregDansLaTable;
47 /

SQL> begin
2     if VerifieEnregDansLaTable('&nom_table','&la_cle')
3     then
4         dbms_output.put_line('Enregistrement trouvé');
5     else
6         dbms_output.put_line('Enregistrement non trouvé');
7     end if;
8 end;
9 /

```

Exercice n°2 Les procédures

Créez une procédure qui permet d'effectuer les opérations suivantes:

- A partir du numéro de commande passé en argument, retirez des unités en stocks toutes les quantités de produits de la commande. S'il n'y a pas assez des unités en stock, commandez la différence, en modifiant la valeur des unités commandées.

```
SQL> CREATE OR REPLACE PROCEDURE EnvoisDeCommande
2          ( a_no_commande COMMANDES.NO_COMMANDE%TYPE)
3  IS
4      v_quantite_stock  PRODUITS.UNITES_STOCK%TYPE;
5      CURSOR c_produit ( a_no_commande NUMBER) IS
6          SELECT NVL(A.UNITES_STOCK,0) UNITES_STOCK,
7                 NVL(A.UNITES_COMMANDEES,0) UNITES_COMMANDEES,
8                 B.QUANTITE
9          FROM PRODUITS A, DETAILS_COMMANDES B
10         WHERE  A.REF_PRODUIT = B.REF_PRODUIT AND
11                B.NO_COMMANDE = a_no_commande
12         FOR UPDATE OF A.UNITES_COMMANDEES, A.UNITES_COMMANDEES;
13 BEGIN
14     for v_p in c_produit(a_no_commande)
15     loop
16         if v_p.UNITES_STOCK < v_p.QUANTITE then
17             v_p.UNITES_COMMANDEES := v_p.UNITES_COMMANDEES +
18                                     v_p.QUANTITE - v_p.UNITES_STOCK;
19             v_p.UNITES_STOCK := 0;
20         else
21             v_p.UNITES_STOCK := v_p.UNITES_STOCK - v_p.QUANTITE;
22         end if;
23         UPDATE PRODUITS
24             SET UNITES_STOCK      = v_p.UNITES_STOCK,
25                 UNITES_COMMANDEES = v_p.UNITES_COMMANDEES
26         WHERE CURRENT OF c_produit;
27     end loop;
28     COMMIT;
29 END EnvoisDeCommande;
30 /
```

- A partir du numéro d'une catégorie de produits, contrôlez les produits en stock pour la catégorie si la quantité des unités en stock est supérieure à la moyenne de la catégorie, sinon commandez deux fois la différence arrondie au dixième supérieur.

```
SQL> CREATE OR REPLACE PROCEDURE VerifieStocksProduits
2          ( a_categorie PRODUITS.CODE_CATEGORIE%TYPE)
3  IS
4      v_quantite_stock  PRODUITS.UNITES_STOCK%TYPE;
5      CURSOR c_produit ( a_categorie NUMBER) IS
6          SELECT UNITES_STOCK,
7                 AVG( UNITES_STOCK) OVER() AVG_UNITES_STOCK ,
8                 UNITES_COMMANDEES
9          FROM PRODUITS
10         WHERE  CODE_CATEGORIE = a_categorie AND INDISPONIBLE = 0
11         FOR UPDATE OF UNITES_COMMANDEES ;
12 BEGIN
13     for v_p in c_produit(a_categorie)
14     loop
```

```

15     if v_p.UNITES_STOCK < v_p.AVG_UNITES_STOCK
16     then
17         v_p.UNITES_COMMANDEES := ROUND(( v_p.AVG_UNITES_STOCK -
18                                         v_p.UNITES_STOCK ) * 2, -1);
19         UPDATE PRODUITS
20         SET UNITES_COMMANDEES = v_p.UNITES_COMMANDEES
21         WHERE CURRENT OF c_produit;
22     end if;
23 end loop;
24 COMMIT;
25 END VerifieStocksProduits;
26 /

```

- A partir du numéro d'une catégorie de produits, validez l'arrivée des produits commandés, ainsi rajoutez aux quantités des unités en stock les quantités des unités commandées et mettez les unités commandées à zéro.

```

SQL> CREATE OR REPLACE PROCEDURE ArriveCommandeProduits
2         ( a_categorie PRODUITS.CODE_CATEGORIE%TYPE)
3 IS
4 BEGIN
5     UPDATE PRODUITS
6     SET UNITES_STOCK = UNITES_STOCK + UNITES_COMMANDEES,
7         UNITES_COMMANDEES = 0
8     WHERE CODE_CATEGORIE = a_categorie AND INDISPONIBLE = 0;
9     dbms_output.put_line('Table PRODUITS' ||
10                          'enregistrements modifies : ' || SQL%ROWCOUNT);
11     COMMIT;
12 END ArriveCommandeProduits;
13 /

```

- Saisir toutes les informations nécessaires pour insérer un produit dans la table PRODUITS. Utilisez la fonction de contrôle d'existence d'un enregistrement précédemment créée pour vérifier toutes les contraintes de clé primaire nécessaires. Créez trois procédures, la première qui a comme argument un enregistrement du même type que la table produits, la deuxième avec une liste arguments représentant les champs de la table produits et la troisième qui effectue le traitement. Les deux premières procédures doivent avoir le même nom, le traitement est effectué par la troisième. La première et la deuxième procédures appellent la troisième. Ainsi la première et la deuxième procédure ne sont qu'un moyen de diversifier les possibilités d'accès à la troisième procédure.

```

SQL> CREATE OR REPLACE PROCEDURE ControleInsereProduit
2     ( a_prod          PRODUITS%ROWTYPE)
3 AS
4     e_code_categorie  EXCEPTION;
5     e_no_fournisseur  EXCEPTION;
6     e_ref_produit     EXCEPTION;
7 BEGIN
8     if NOT VerifieEnregDansLaTable( 'CATEGORIES',
9                                     a_prod.CODE_CATEGORIE) then
10         RAISE e_code_categorie;
11     end if;
12     if NOT VerifieEnregDansLaTable( 'FOURNISSEURS',
13                                     a_prod.NO_FOURNISSEUR) then
14         RAISE e_no_fournisseur;
15     end if;

```

```

16     if NOT VerifieEnregDansLaTable( 'PRODUITS',
17                                     a_prod.REF_PRODUIT) then
18         RAISE e_ref_produit;
19     end if;
20     INSERT INTO PRODUITS VALUES  a_prod;
21 exception
22     when e_code_categorie then
23         dbms_output.put_line( 'La catégorie n''existe pas!');
24     when e_no_fournisseur then
25         dbms_output.put_line( 'Le fournisseur n''existe pas!');
26     when e_ref_produit then
27         dbms_output.put_line( 'Le produit existe déjà!');
28 END ControleInsereProduit;
29 /

```

```

SQL> CREATE OR REPLACE PROCEDURE AddProduit
2                                     ( a_prod PRODUITS%ROWTYPE)
3 AS
4 BEGIN
5     ControleInsereProduit( a_prod);
6 END AddProduit;
7 /

```

```

SQL> CREATE OR REPLACE PROCEDURE AddProduit
2      ( a_ref_produit      PRODUITS.REF_PRODUIT%TYPE,
3        a_nom_produit      PRODUITS.NOM_PRODUIT%TYPE,
4        a_no_fournisseur   PRODUITS.NO_FOURNISSEUR%TYPE,
5        a_code_categorie   PRODUITS.CODE_CATEGORIE%TYPE,
6        a_quantite         PRODUITS.QUANTITE%TYPE,
7        a_prix_unitaire    PRODUITS.PRIX_UNITAIRE%TYPE,
8        a_unites_stock     PRODUITS.UNITES_STOCK%TYPE,
9        a_unites_commandees PRODUITS.UNITES_COMMANDEES%TYPE,
10       a_indisponible      PRODUITS.INDISPONIBLE%TYPE)
11 AS
12     a_prod      PRODUITS%ROWTYPE;
13 BEGIN
14     a_prod.REF_PRODUIT      := a_ref_produit;
15     a_prod.NOM_PRODUIT     := a_nom_produit;
16     a_prod.NO_FOURNISSEUR  := a_no_fournisseur;
17     a_prod.CODE_CATEGORIE  := a_code_categorie;
18     a_prod.QUANTITE        := a_quantite;
19     a_prod.PRIX_UNITAIRE   := a_prix_unitaire;
20     a_prod.UNITES_STOCK    := a_unites_stock;
21     a_prod.UNITES_COMMANDEES := a_unites_commandees;
22     a_prod.INDISPONIBLE    := a_indisponible;
23     ControleInsereProduit( a_prod);
24 END AddProduit;
25 /

```

- Dans les quatre tables DIM_EMPLOYES, DIM_PRODUITS, DIM_CLIENTS et à la fin INDICATEURS du modèle étoile, mettez à jour les enregistrements et si tel est le cas insérer tous les enregistrements manquants.

```

SQL> CREATE OR REPLACE PROCEDURE MAJ_MODEL_ETOILE
2 AS

```

```

3 BEGIN
4 -----La table DIM_CLIENTS-----
5     MERGE INTO DIM_CLIENTS CIBLE
6         USING ( SELECT CODE_CLIENT,SOCIETE,VILLE,PAYS
7                 FROM CLIENTS ) SOURCE
8         ON ( CIBLE.CODE_CLIENT = SOURCE.CODE_CLIENT)
9     WHEN MATCHED THEN
10         UPDATE SET CIBLE.CLIENT = SOURCE.SOCIETE,
11                 CIBLE.VILLE = SOURCE.VILLE,
12                 CIBLE.PAYS = SOURCE.PAYS
13     WHEN NOT MATCHED THEN
14         INSERT ( CODE_CLIENT,CLIENT,VILLE,PAYS)
15         VALUES ( SOURCE.CODE_CLIENT,SOURCE.SOCIETE,
16                 SOURCE.VILLE,SOURCE.PAYS);
17     dbms_output.put_line('Table DIM_CLIENTS '||
18         'enregistrements modifies : '||SQL%ROWCOUNT);
19 -----La table DIM_EMPLOYES-----
20     MERGE INTO DIM_EMPLOYES CIBLE
21         USING ( SELECT NO_EMPLOYE,NOM||' '||PRENOM EMPLOYE,
22                 FONCTION
23                 FROM EMPLOYES ) SOURCE
24         ON ( CIBLE.NO_EMPLOYE = SOURCE.NO_EMPLOYE)
25     WHEN MATCHED THEN
26         UPDATE SET CIBLE.EMPLOYE = SOURCE.EMPLOYE,
27                 CIBLE.FONCTION = SOURCE.FONCTION
28     WHEN NOT MATCHED THEN
29         INSERT ( NO_EMPLOYE,EMPLOYE,FONCTION)
30         VALUES ( SOURCE.NO_EMPLOYE,SOURCE.EMPLOYE,
31                 SOURCE.FONCTION);
32     dbms_output.put_line('Table DIM_EMPLOYES '||
33         'enregistrements modifies : '||SQL%ROWCOUNT);
34 -----La table DIM_PRODUITS-----
35     MERGE INTO DIM_PRODUITS CIBLE
36         USING ( SELECT REF_PRODUIT,NOM_PRODUIT,NOM_CATEGORIE
37                 FROM PRODUITS NATURAL JOIN CATEGORIES ) SOURCE
38         ON ( CIBLE.REF_PRODUIT = SOURCE.REF_PRODUIT)
39     WHEN MATCHED THEN
40         UPDATE SET CIBLE.NOM_PRODUIT = SOURCE.NOM_PRODUIT,
41                 CIBLE.NOM_CATEGORIE = SOURCE.NOM_CATEGORIE
42     WHEN NOT MATCHED THEN
43         INSERT ( REF_PRODUIT,NOM_PRODUIT,NOM_CATEGORIE )
44         VALUES ( SOURCE.REF_PRODUIT,SOURCE.NOM_PRODUIT,
45                 SOURCE.NOM_CATEGORIE);
46     dbms_output.put_line('Table DIM_PRODUITS '||
47         'enregistrements modifies : '||SQL%ROWCOUNT);
48 -----La table INDICATEURS-----
49     MERGE INTO INDICATEURS CIBLE
50         USING ( SELECT DATE_COMMANDE, REF_PRODUIT, CODE_CLIENT,
51                 NO_EMPLOYE, NO_COMMANDE, PORT,
52                 QUANTITE, PRIX_UNITAIRE
53                 FROM EMPLOYES JOIN COMMANDES USING ( NO_EMPLOYE)
54                 JOIN DETAILS_COMMANDES
55                 USING ( NO_COMMANDE ) ) SOURCE
56         ON ( CIBLE.JOUR = SOURCE.DATE_COMMANDE AND

```

```

57         CIBLE.REF_PRODUIT = SOURCE.REF_PRODUIT    AND
58         CIBLE.CODE_CLIENT = SOURCE.CODE_CLIENT    AND
59         CIBLE.NO_EMPLOYE  = SOURCE.NO_EMPLOYE     AND
60         CIBLE.NO_COMMANDE = SOURCE.NO_COMMANDE )
61     WHEN MATCHED THEN
62         UPDATE SET CIBLE.PORT = SOURCE.PORT,
63                 CIBLE.QUANTITE = SOURCE.QUANTITE,
64                 CIBLE.PRIX_UNITAIRE = SOURCE.PRIX_UNITAIRE
65     WHEN NOT MATCHED THEN
66         INSERT ( JOUR,REF_PRODUIT,CODE_CLIENT,NO_EMPLOYE,
67                 NO_COMMANDE,PORT,QUANTITE,PRIX_UNITAIRE )
68         VALUES ( SOURCE.DATE_COMMANDE,SOURCE.REF_PRODUIT,
69                 SOURCE.CODE_CLIENT,SOURCE.NO_EMPLOYE,
70                 SOURCE.NO_COMMANDE,SOURCE.PORT,
71                 SOURCE.QUANTITE,SOURCE.PRIX_UNITAIRE);
72     dbms_output.put_line('Table INDICATEURS '||
73                         'enregistrements modifies : '||SQL%ROWCOUNT);
74     COMMIT;
75 END MAJ_MODEL_ETOILE;
76 /

```

SQL> **exec MAJ_MODEL_ETOILE;**

Table DIM_CLIENTS enregistrements modifies : 92

Table DIM_EMPLOYES enregistrements modifies : 9

Table DIM_PRODUITS enregistrements modifies : 77

Table INDICATEURS enregistrements modifies : 1776

- Effacez d'abord les enregistrements, puis insérez les dans les sept tables du modèle étoile :

```

QUANTITES_CLIENTS,
VENTES_CLIENTS,
VENTES_ANNEES,
VENTES_MOIS
VENTES_CLIENTS_2009
VENTES_CLIENTS_2010
VENTES_CLIENTS_2011

```

SQL> **CREATE OR REPLACE PROCEDURE MAJ_TABLES_AGREGATS**

```

2  AS
3  BEGIN
4      DELETE QUANTITES_CLIENTS;
5      DELETE VENTES_CLIENTS;
6      DELETE VENTES_ANNEES;
7      DELETE VENTES_MOIS;
8      DELETE VENTES_CLIENTS_2009;
9      DELETE VENTES_CLIENTS_2010;
10     DELETE VENTES_CLIENTS_2011;
11     INSERT ALL
12         WHEN G = 0 THEN
13             INTO VENTES_CLIENTS
14                 VALUES ( ANNEE, MOIS, CODE_CLIENT, VENTE, REMISE)
15             INTO QUANTITES_CLIENTS
16                 VALUES ( ANNEE, MOIS, CODE_CLIENT, QUANTITE, PORT)
17         WHEN G = 3 THEN
18             INTO VENTES_ANNEES
19                 VALUES ( ANNEE, VENTE, REMISE)

```

```

20      WHEN G = 1 THEN
21          INTO VENTES_MOIS
22              VALUES ( ANNEE, MOIS, VENTE, REMISE)
23      WHEN ANNEE = 1996 THEN
24          INTO VENTES_CLIENTS_2009
25              VALUES ( MOIS, CODE_CLIENT, VENTE, REMISE)
26      WHEN ANNEE = 1997 THEN
27          INTO VENTES_CLIENTS_2010
28              VALUES ( MOIS, CODE_CLIENT, VENTE, REMISE)
29      WHEN ANNEE = 1998 THEN
30          INTO VENTES_CLIENTS_2011
31              VALUES ( MOIS, CODE_CLIENT, VENTE, REMISE)
32      SELECT GROUPING_ID( EXTRACT ( YEAR  FROM DATE_COMMANDE),
33                          EXTRACT ( MONTH FROM DATE_COMMANDE),
34                          CODE_CLIENT ) G,
35          EXTRACT ( YEAR  FROM DATE_COMMANDE) ANNEE,
36          EXTRACT ( MONTH FROM DATE_COMMANDE) MOIS,
37          CODE_CLIENT,
38          SUM(QUANTITE*PRIX_UNITAIRE) VENTE,
39          SUM(QUANTITE*PRIX_UNITAIRE*REMISE) REMISE,
40          SUM(QUANTITE) QUANTITE,
41          SUM(PORT) PORT
42      FROM  COMMANDES NATURAL JOIN DETAILS_COMMANDES
43      GROUP BY ROLLUP
44          (EXTRACT ( YEAR  FROM DATE_COMMANDE),
45           EXTRACT ( MONTH FROM DATE_COMMANDE),
46           CODE_CLIENT);
47      dbms_output.put_line('Enregistrements insérés : '
48                          || SQL%ROWCOUNT);
49      COMMIT;
50  END MAJ_TABLES_AGREGATS;
51  /

```

```

SQL> exec MAJ_TABLES_AGREGATS;
Enregistrements insérés : 1608

```


Atelier 9.1 Les packages

Questions



```
SQL> CREATE OR REPLACE PACKAGE GererProduit
2      CURSOR c_produit RETURN PRODUITS%ROWTYPE
3      IS SELECT REF_PRODUIT, NOM_PRODUIT, NO_FOURNISSEUR,
4          CODE_CATEGORIE, QUANTITE, PRIX_UNITAIRE,
5          UNITES_STOCK, UNITES_COMMANDEES, INDISPONIBLE
6          FROM PRODUITS;
7      ...
```

1. Le package précédent peut-il être compilé ? Justifiez votre réponse.

Réponse : Non. Le curseur et le module doivent être définis dans le corps du package, ils ne sont pas complètement définis par les spécifications. Le curseur a besoin de son ordre « **SELECT** ».

```
SQL> CREATE OR REPLACE PACKAGE PkgSurchage
2  AS
3      PROCEDURE NomProcedure01( a_arg01 VARCHAR2);
4      PROCEDURE NomProcedure01( a_arg02 VARCHAR2);
5      PROCEDURE NomProcedure02( a_arg IN VARCHAR2);
6      PROCEDURE NomProcedure02( a_arg OUT VARCHAR2);
7      ...
```

2. Le package précédent peut-il être compilé ? Justifiez votre réponse.

Réponse : Oui. L'opération de compilation du package ne contrôle pas la cohérence des déclarations, ainsi il n'y a pas d'erreurs de suite aux surcharges. Les erreurs de surcharge sont essentiellement des erreurs survenues pendant l'exécution et non à la compilation du package qui elle s'est déroulée sans aucun incident.

Exercice n°1 Les packages

Créez un package pour la gestion des employés avec ces caractéristiques :

- Une fonction qui contrôle l'existence d'un employé dans la table EMPLOYES à partir du numéro de l'employé.

```
FUNCTION ControleEmploye ( a_no_employe EMPLOYES.NO_EMPLOYE%TYPE)
RETURN BOOLEAN AS
BEGIN
    for v_employe in ( SELECT NO_EMPLOYE FROM EMPLOYES
                      WHERE NO_EMPLOYE = a_no_employe) loop
        RETURN TRUE;
    end loop;
    RETURN FALSE;
END;
```

- Une procédure de suppression d'un employé.

```
PROCEDURE Supprimer(a_no_employe EMPLOYES.NO_EMPLOYE%TYPE)
IS
```

```
BEGIN
    DELETE EMPLOYES WHERE EMPLOYES.NO_EMPLOYE = a_no_employe;
    if SQL%NOTFOUND then
        raise e_Employe;
    end if;
END;
```

- Une procédure d'augmentation du salaire pour un employé. La procédure comporte deux arguments ; le premier est le numéro de l'employé, qui doit être contrôlé, et le deuxième est le montant de l'augmentation. Si le montant est égal à zéro l'employé se voit attribuer la moyenne des salaires.

```
PROCEDURE Augmenter(a_no_employe EMPLOYES.NO_EMPLOYE%TYPE,
                    a_salaire    EMPLOYES.SALAIRE%TYPE :=0 )
IS
    v_salaire EMPLOYES.SALAIRE%TYPE;
BEGIN
    SELECT SALAIRE INTO v_salaire FROM EMPLOYES
    WHERE NO_EMPLOYE = a_no_employe;

    case
    when a_salaire = 0 then
        v_salaire := v_avg_salaire;
    when a_salaire <= v_salaire then
        raise e_Salaire;
    else
        v_salaire := a_salaire;
    end case;
end;
```

- Une procédure d'insertion d'un employé dans la table EMPLOYES. Il faut contrôler que le supérieur hiérarchique existe déjà dans la table. L'âge de l'employé doit être supérieur à 18 ans. Vous pouvez utiliser une constante pour stocker l'âge minimum. Il faut également contrôler si l'employé n'existe pas déjà dans la table.

```
PROCEDURE AddEmploye
(
    a_no_employe    EMPLOYES.NO_EMPLOYE%TYPE,
    a_rend_compte   EMPLOYES.REND_COMPTE%TYPE,
    a_nom           EMPLOYES.NOM%TYPE,
    a_prenom        EMPLOYES.PRENOM%TYPE,
    a_fonction      EMPLOYES.FONCTION%TYPE,
    a_titre         EMPLOYES.TITRE%TYPE,
    a_date_naissance EMPLOYES.DATE_NAISSANCE%TYPE,
    a_date_embauche EMPLOYES.DATE_EMBAUCHE%TYPE,
    a_salaire       EMPLOYES.SALAIRE%TYPE,
    a_commission    EMPLOYES.COMMISSION%TYPE )
IS
BEGIN
    if sysdate - a_date_naissance < v_age_minim then
        raise e_Age;
    end if;
    if ControleEmploye( a_no_employe) then
        raise e_Employe;
    end if;
    if not ControleEmploye( a_rend_compte) then
        raise e_Superieur;
    end if;
```

```

if a_date_embauche > sysdate then
    raise e_Embauche;
end if;

INSERT INTO EMPLOYES ( NO_EMPLOYE, REND_COMPTE, NOM,
                      PRENOM, FONCTION, TITRE,
                      DATE_NAISSANCE, DATE_EMBAUCHE,
                      SALAIRE, COMMISSION )
VALUES ( a_no_employe, a_rend_compte, a_nom,
        a_prenom, a_fonction, a_titre,
        a_date_naissance, a_date_embauche,
        a_salaire, a_commission);
END;

```

- Pour les tests du package, créez un script SQL qui vous permette de saisir les informations pour l'ajout d'un employé, l'augmentation et la suppression.

```

-- création des valeurs à saisir
accept vs_no_employe      prompt "N° employé      : "
accept vs_rend_compte     prompt "Rend compte à : "
accept vs_nom             prompt "Nom          : "
accept vs_prenom          prompt "Prénom       : "
accept vs_fonction        prompt "Fonction     : "
accept vs_titre           prompt "Titre        : "
accept vs_date_naissance  prompt "Date de naissance : "
accept vs_date_embauche   prompt "Date embauche : "
accept vs_salaire         prompt "Salaire      : "
accept vs_commission      prompt "Commission   : "

BEGIN
    GererEmploye.AddEmploye
        ( &vs_no_employe,&vs_rend_compte,'&vs_nom',
          '&vs_prenom','&vs_fonction','&vs_titre',
          '&vs_date_naissance','&vs_date_embauche',
          &vs_salaire,&vs_commission);
EXCEPTION
    when GererEmploye.e_Salaire then
        dbms_output.put_line( 'Le salaire n''est pas valide. ');
    when GererEmploye.e_Superieur then
        dbms_output.put_line( 'Le supérieur n''est pas valide. ');
    when GererEmploye.e_Employe then
        dbms_output.put_line( 'L''employé n''est pas valide. ');
    when GererEmploye.e_Age then
        dbms_output.put_line( 'L''age n'est pas valide. ');
    when OTHERS then
        dbms_output.put_line( 'Erreur. ');
END;

```

Réponse : Le script du package est le suivant ;

```

SQL> CREATE OR REPLACE PACKAGE GererEmploye
2  AS
3      e_Age          EXCEPTION;
4      e_Embauche     EXCEPTION;
5      e_Salaire      EXCEPTION;
6      e_Superieur    EXCEPTION;
7      e_Employe      EXCEPTION;

```

```

8      v_employe      EMPLOYES%ROWTYPE;
9      FUNCTION ControleEmploye
10         ( a_no_employe EMPLOYES.NO_EMPLOYE%TYPE)
11      RETURN BOOLEAN;
12      PROCEDURE AddEmploye
13         ( a_no_employe      EMPLOYES.NO_EMPLOYE%TYPE,
14           a_rend_compte     EMPLOYES.REND_COMPTE%TYPE,
15           a_nom             EMPLOYES.NOM%TYPE,
16           a_prenom         EMPLOYES.PRENOM%TYPE,
17           a_fonction       EMPLOYES.FONCTION%TYPE,
18           a_titre          EMPLOYES.TITRE%TYPE,
19           a_date_naissance EMPLOYES.DATE_NAISSANCE%TYPE,
20           a_date_embauche  EMPLOYES.DATE_EMBAUICHE%TYPE,
21           a_salaire        EMPLOYES.SALAIRE%TYPE,
22           a_commission     EMPLOYES.COMMISSION%TYPE );
23      PROCEDURE Supprimer
24         ( a_no_employe      EMPLOYES.NO_EMPLOYE%TYPE);
25      PROCEDURE Augmenter
26         ( a_no_employe      EMPLOYES.NO_EMPLOYE%TYPE,
27           a_salaire        EMPLOYES.SALAIRE%TYPE :=0 );
28  END GererEmploye;
29  /

```

Package créé.

```

SQL> CREATE OR REPLACE PACKAGE BODY GererEmploye
2  IS
3      v_age_minim      CONSTANT NUMBER(2) := 18;
4      v_sum_salaire    EMPLOYES.SALAIRE%TYPE;
5      v_avg_salaire    EMPLOYES.SALAIRE%TYPE;
6      ----- FUNCTION ControleEmploye -----
7      FUNCTION ControleEmploye
8         ( a_no_employe EMPLOYES.NO_EMPLOYE%TYPE)
9      RETURN BOOLEAN
10     AS
11     BEGIN
12         for v_employe in ( SELECT NO_EMPLOYE FROM EMPLOYES
13                           WHERE NO_EMPLOYE = a_no_employe) loop
14             RETURN TRUE;
15         end loop;
16         RETURN FALSE;
17     END ControleEmploye;
18     ----- PROCEDURE AddEmploye -----
19     PROCEDURE AddEmploye
20         ( a_no_employe      EMPLOYES.NO_EMPLOYE%TYPE,
21           a_rend_compte     EMPLOYES.REND_COMPTE%TYPE,
22           a_nom             EMPLOYES.NOM%TYPE,
23           a_prenom         EMPLOYES.PRENOM%TYPE,
24           a_fonction       EMPLOYES.FONCTION%TYPE,
25           a_titre          EMPLOYES.TITRE%TYPE,
26           a_date_naissance EMPLOYES.DATE_NAISSANCE%TYPE,
27           a_date_embauche  EMPLOYES.DATE_EMBAUICHE%TYPE,
28           a_salaire        EMPLOYES.SALAIRE%TYPE,
29           a_commission     EMPLOYES.COMMISSION%TYPE )

```

```

30  IS
31  BEGIN
32      if sysdate - a_date_naissance < v_age_minim then
33          raise e_Age;
34      end if;
35      if ControleEmploye( a_no_employe) then
36          raise e_Employe;
37      end if;
38      if not ControleEmploye( a_rend_compte) then
39          raise e_Superieur;
40      end if;
41      if a_date_embauche > sysdate then
42          raise e_Embauche;
43      end if;
44
45      INSERT INTO EMPLOYES ( NO_EMPLOYE, REND_COMPTE, NOM,
46                          PRENOM, FONCTION, TITRE,
47                          DATE_NAISSANCE, DATE_EMBAUCHE,
48                          SALAIRE, COMMISSION )
49          VALUES ( a_no_employe, a_rend_compte, a_nom,
50                  a_prenom, a_fonction, a_titre,
51                  a_date_naissance, a_date_embauche,
52                  a_salaire, a_commission);
53  END AddEmploye;
54  ----- PROCEDURE Supprimer -----
55  PROCEDURE Supprimer
56      ( a_no_employe EMPLOYES.NO_EMPLOYE%TYPE)
57  IS
58  BEGIN
59      DELETE EMPLOYES WHERE EMPLOYES.NO_EMPLOYE = a_no_employe;
60      if SQL%NOTFOUND then
61          raise e_Employe;
62      end if;
63  END Supprimer;
64  ----- PROCEDURE Augmenter -----
65  PROCEDURE Augmenter
66      ( a_no_employe EMPLOYES.NO_EMPLOYE%TYPE,
67        a_salaire EMPLOYES.SALAIRE%TYPE :=0 )
68  IS
69      v_salaire EMPLOYES.SALAIRE%TYPE;
70  BEGIN
71      SELECT SALAIRE INTO v_salaire FROM EMPLOYES
72      WHERE NO_EMPLOYE = a_no_employe;
73
74      case
75      when a_salaire = 0 then
76          v_salaire := v_avg_salaire;
77      when a_salaire <= v_salaire then
78          raise e_Salaire;
79      else
80          v_salaire := a_salaire;
81      end case;
82  END Augmenter;
83  BEGIN

```

```
84      SELECT SUM(SALAIRE) INTO v_sum_salaire FROM EMPLOYES;  
85      SELECT AVG(SALAIRE) INTO v_avg_salaire FROM EMPLOYES;  
86  END GererEmploye;  
87  /
```

Corps de package créé.

Atelier 10.1 Les déclencheurs

Questions



1. Quels sont les déclencheurs qui peuvent être compilés ?
 - A. `CREATE OR REPLACE TRIGGER T1 BEFORE UPDATE ON CATEGORIES BEGIN :new.DESCRPTION := 'DESCRIPTION'; END T1;`
 - B. `CREATE OR REPLACE TRIGGER T2 AFTER UPDATE ON CATEGORIES BEGIN :new.DESCRPTION := 'DESCRIPTION'; END T2;`
 - C. `CREATE OR REPLACE TRIGGER T3 BEFORE UPDATE ON CATEGORIES FOR EACH ROW BEGIN :new.DESCRPTION := 'DESCRIPTION'; END T3;`
 - D. `CREATE OR REPLACE TRIGGER T4 AFTER UPDATE ON CATEGORIES FOR EACH ROW BEGIN :new.DESCRPTION := 'DESCRIPTION'; END T4;`
 - E. `CREATE OR REPLACE TRIGGER T5 BEFORE INSERT ON CATEGORIES FOR EACH ROW BEGIN :old.DESCRPTION := 'DESCRIPTION'; END T5;`
 - F. `CREATE OR REPLACE TRIGGER T6 AFTER INSERT ON CATEGORIES FOR EACH ROW BEGIN :old.DESCRPTION := 'DESCRIPTION'; END T6;`
 - G. `CREATE OR REPLACE TRIGGER T8 AFTER INSERT ON CATEGORIES FOR EACH ROW DECLARE a NUMBER; BEGIN a:= :old.DESCRPTION; END T8;`
 - H. `CREATE OR REPLACE TRIGGER T9 BEFORE INSERT ON CATEGORIES FOR EACH ROW DECLARE a NUMBER; BEGIN a:= :old.DESCRPTION; END T9;`

Réponse : C, G, H

Exercice n°1 Les déclencheurs d'instruction

Créez un déclencheur sur la table `PRODUITS` qui empêche l'insertion d'un enregistrement ou la mise à jour des produits en stock de la table `PRODUITS` si un de produits suivants '2, 4, 6, 8' a déjà un stock de 700 unités.

```
SQL> CREATE OR REPLACE TRIGGER ProduitsVolumeMaxStock
2 BEFORE UPDATE OF UNITES_STOCK OR INSERT
3 ON PRODUITS
4 DECLARE
5     v_vol_max PRODUITS.UNITES_STOCK%TYPE;
6 BEGIN
7     SELECT MAX(SUM(UNITES_STOCK)) INTO v_vol_max FROM PRODUITS
8     WHERE CODE_CATEGORIE IN (2,4,6,8)
9     GROUP BY CODE_CATEGORIE;
10    if v_vol_max > 700 then
```

```

11      RAISE_APPLICATION_ERROR ( -20000,'Le volume total d''''||
12      'une des categories : 1,3,5,7 est > 350');
13      end if;
14  END ProduitsVolumeMaxStock;
15  /

```

Déclencheur créé.

```

SQL> SELECT REF_PRODUIT, UNITES_STOCK, SUM(UNITES_STOCK) OVER ( )
2   FROM PRODUITS
3   WHERE CODE_CATEGORIE = 2;

```

REF_PRODUIT	UNITES_STOCK	SUM(UNITES_STOCK)OVER()
3	13	701
4	53	701
6	120	701
8	200	701
66	4	701
77	32	701
5	0	701
15	39	701
44	27	701
61	113	701
63	24	701
65	76	701

12 ligne(s) sélectionnée(s).

```

SQL> UPDATE PRODUITS
2   SET UNITES_STOCK = 10
3   WHERE REF_PRODUIT = 3;

```

UPDATE PRODUITS

*

ERREUR à la ligne 1 :

ORA-20000: Le volume total d'une des categories : 1,3,5,7 est > 350

ORA-06512: à "STAGIAIRE.PRODUITSVOLUMEMAXSTOCK", ligne 8

ORA-04088: erreur lors d'exécution du déclencheur

'STAGIAIRE.PRODUITSVOLUMEMAXSTOCK'

```

SQL> SELECT REF_PRODUIT, UNITES_STOCK, SUM(UNITES_STOCK) OVER ( )
2   FROM PRODUITS
3   WHERE CODE_CATEGORIE = 2;

```

REF_PRODUIT	UNITES_STOCK	SUM(UNITES_STOCK)OVER()
3	13	701
...		

Créez un déclencheur sur la table EMPLOYES qui empêche toute opération si elle ne s'effectue pendant les heures de travail.

```

SQL> CREATE OR REPLACE TRIGGER ContrainteTrancheHoraire
2   BEFORE INSERT OR UPDATE OR DELETE ON EMPLOYES
3   DECLARE

```



```

4      v_jour  NUMBER(1) := TO_CHAR(SYSDATE, 'D');
5      v_heure NUMBER(2) := TO_CHAR(SYSDATE, 'hh24');
6  BEGIN
7      dbms_output.put_line(v_jour||' '||v_heure);
8      IF v_jour > 5 OR  v_heure NOT BETWEEN 8 AND 19
9      THEN
10         RAISE_APPLICATION_ERROR ( -20000, 'L''application ne peut'
11             ||' pas être modifiée hors des heures de travail. '||
12             TO_CHAR(SYSDATE, 'day month yyyy hh24:mi:ss'));
13     END IF;
14 END ContrainteTrancheHoraire;
15 /

```

Déclencheur créé.

SQL>

SQL> UPDATE EMPLOYES SET SALAIRE = SALAIRE *1.1;

4 2

UPDATE EMPLOYES SET SALAIRE = SALAIRE *1.1

*

ERREUR à la ligne 1 :

ORA-20000: L'application ne peut pas être modifiée hors des heures de travail.

jeudi juillet 2006 02:42:06

ORA-06512: à "STAGIAIRE.CONTRAINTETRANCHEHORAIRE", ligne 8

ORA-04088: erreur lors d'exécution du déclencheur

'STAGIAIRE.CONTRAINTETRANCHEHORAIRE'

Exercice n°2 Les déclencheurs d'enregistrement

Créez une séquence qui commence avec le dernier numéro d'employé se trouvant dans la table EMPLOYES. Ensuite, créez un déclencheur qui initialise le NO_EMPLOYE avec la valeur suivante de la séquence s'il n'a pas été renseigné. Dans le même déclencheur, testez si le champ REND_COMPTE est correctement initialisé, avec une valeur d'un employé existant, sinon vous l'initialisez avec la même valeur que NO_EMPLOYE.

SQL> SET HEADING OFF

SQL> SET ECHO OFF

SQL> SET FEEDBACK OFF

SQL> SET PAGESIZE 0

SQL> SET TERM OFF

SQL> SPOOL C:\CREATE_S_EmployesID.SQL

SQL> SELECT 'CREATE SEQUENCE S_EmployesID START WITH '||

2 MAX(NO_EMPLOYE)||' INCREMENT BY 1;' '---"

3 FROM EMPLOYES;

CREATE SEQUENCE S_EmployesID START WITH 9 INCREMENT BY 1;

SQL> SELECT 'SELECT S_EmployesID.NEXTVAL FROM DUAL;' FROM DUAL;

SELECT S_EmployesID.NEXTVAL FROM DUAL;

SQL> SPOOL OFF

SQL> @C:\CREATE_S_EmployesID.SQL

SQL> SET TERM ON

SQL> SET HEADING ON

SQL> SET ECHO ON

```

SQL> SET FEEDBACK ON
SQL> SET PAGESIZE 150

SQL> CREATE OR REPLACE TRIGGER EmployesID
  2   BEFORE INSERT ON EMPLOYES
  3   FOR EACH ROW
  4   BEGIN
  5       :NEW.NO_EMPLOYE := S_EmployesID.NEXTVAL;
  6       if :NEW.REND_COMPTE IS NULL then
  7           :NEW.REND_COMPTE := :NEW.NO_EMPLOYE;
  8       end if;
  9       dbms_output.put_line('L'employé :'|':NEW.NO_EMPLOYE);
 10   END EmployesID;
 11   /

```

Déclencheur créé.

```

SQL> SELECT NO_EMPLOYE,REND_COMPTE,NOM FROM EMPLOYES
  2 WHERE REND_COMPTE IS NULL;

```

NO_EMPLOYE	REND_COMPTE	NOM
37		Giroux

```

SQL> INSERT INTO EMPLOYES ( NOM, PRENOM, FONCTION, TITRE,
  2   DATE_NAISSANCE, DATE_EMBAUCHE, SALAIRE)
  3 SELECT NOM, PRENOM, FONCTION, TITRE,
  4   DATE_NAISSANCE, DATE_EMBAUCHE, SALAIRE
  5 FROM EMPLOYES
  6 WHERE REND_COMPTE IS NULL;

```

L'employé :112

```

SQL> SELECT NO_EMPLOYE,REND_COMPTE,NOM FROM EMPLOYES
  2 WHERE NO_EMPLOYE = 112;

```

NO_EMPLOYE	REND_COMPTE	NOM
112	112	Giroux

Créez deux tables PRODUITS_POUBELLE et PRODUITS_ARCHIVE qui ont la même description que la table PRODUITS avec deux colonnes de plus, UTILISATEUR et DATE_SYSTEME. Créez un déclencheur sur la table PRODUITS qui archive dans la table PRODUITS_POUBELLE tous les produits effacés et dans la table PRODUITS_ARCHIVE tous les produits modifiés.

```

SQL> CREATE TABLE PRODUITS_POUBELLE AS
  2   SELECT REF_PRODUIT, NOM_PRODUIT, NO_FOURNISSEUR,
  3   CODE_CATEGORIE,QUANTITE, PRIX_UNITAIRE,
  4   UNITES_STOCK, UNITES_COMMANDEES,INDISPONIBLE,
  5   USER_UTILISATEUR, SYSDATE DATE_EFFECEMENT
  6   FROM PRODUITS WHERE 1=2;

```

Table créée.

```

SQL> CREATE TABLE PRODUITS_ARCHIVE AS

```

```

2      SELECT REF_PRODUIT, NOM_PRODUIT, NO_FOURNISSEUR,
3             CODE_CATEGORIE, QUANTITE, PRIX_UNITAIRE,
4             UNITES_STOCK, UNITES_COMMANDEES, INDISPONIBLE,
5             USER_UTILISATEUR, SYSDATE DATE_EFFECLEMENT
6      FROM PRODUITS WHERE 1=2;

```

Table créée.

```

SQL> CREATE OR REPLACE TRIGGER ProduitsArchive
2  BEFORE UPDATE OR DELETE OR INSERT ON PRODUITS
3  FOR EACH ROW
4  BEGIN
5      CASE
6      WHEN UPDATING THEN
7          INSERT INTO PRODUITS_ARCHIVE
8              VALUES (:OLD.REF_PRODUIT, :OLD.NOM_PRODUIT,
9                      :OLD.NO_FOURNISSEUR, :OLD.CODE_CATEGORIE,
10                     :OLD.QUANTITE, :OLD.PRIX_UNITAIRE,
11                     :OLD.UNITES_STOCK, :OLD.UNITES_COMMANDEES,
12                     :OLD.INDISPONIBLE, USER, SYSDATE);
13      WHEN DELETING THEN
14          INSERT INTO PRODUITS_POUBELLE
15              VALUES (:OLD.REF_PRODUIT, :OLD.NOM_PRODUIT,
16                      :OLD.NO_FOURNISSEUR, :OLD.CODE_CATEGORIE,
17                      :OLD.QUANTITE, :OLD.PRIX_UNITAIRE,
18                      :OLD.UNITES_STOCK, :OLD.UNITES_COMMANDEES,
19                      :OLD.INDISPONIBLE, USER, SYSDATE);
20      ELSE
21          NULL;
22      END CASE;
23  END ProduitsArchive;
24  /

```

Déclencheur créé.

```

SQL> DELETE DETAILS_COMMANDES
2  WHERE REF_PRODUIT = 1;

```

32 ligne(s) supprimée(s).

```

SQL> DELETE PRODUITS
2  WHERE REF_PRODUIT = 1;

```

1 ligne supprimée.

```

SQL> UPDATE PRODUITS
2  SET PRIX_UNITAIRE = PRIX_UNITAIRE*1.1
3  WHERE REF_PRODUIT IN (2,5,9);

```

3 ligne(s) mise(s) à jour.

```

SQL> SELECT REF_PRODUIT, NOM_PRODUIT, PRIX_UNITAIRE
2  FROM PRODUITS_ARCHIVE;

```

REF_PRODUIT	NOM_PRODUIT	PRIX_UNITAIRE
2	Chang	95
5	Chef Anton's Gumbo Mix	106
9	Mishi Kobe Niku	485

3 ligne(s) sélectionnée(s).

```
SQL> SELECT REF_PRODUIT, NOM_PRODUIT, PRIX_UNITAIRE
2 FROM PRODUITS_POUBELLE ;
```

REF_PRODUIT	NOM_PRODUIT	PRIX_UNITAIRE
1	Chai	90

1 ligne sélectionnée.

```
SQL> ROLLBACK;
```

Annulation (rollback) effectuée.

```
SQL> SELECT REF_PRODUIT, NOM_PRODUIT, PRIX_UNITAIRE
2 FROM PRODUITS_POUBELLE ;
```

aucune ligne sélectionnée

```
SQL> SELECT REF_PRODUIT, NOM_PRODUIT, PRIX_UNITAIRE
2 FROM PRODUITS_ARCHIVE;
```

aucune ligne sélectionnée

Créez une table PRODUITS_INSERT qui contient trois champs : REF_PRODUIT, UTILISATEUR et DATE_SYTEM. Modifiez le déclencheur précédemment créé pour pouvoir insérer dans la table PRODUITS_INSERT tous les REF_PRODUIT avec les informations correspondantes sur l'utilisateur et date de création.

```
SQL> CREATE TABLE PRODUITS_INSERT AS
2 SELECT REF_PRODUIT, USER UTILISATEUR,
3 SYSDATE DATE_EFFECEMENT
4 FROM PRODUITS WHERE 1=2;
```

Table créée.

```
SQL> CREATE OR REPLACE TRIGGER ProduitsArchive
2 BEFORE UPDATE OR DELETE OR INSERT ON PRODUITS
3 FOR EACH ROW
4 BEGIN
5 CASE
6 WHEN UPDATING THEN
7 INSERT INTO PRODUITS_ARCHIVE
8 VALUES (:OLD.REF_PRODUIT, :OLD.NOM_PRODUIT,
9 :OLD.NO_FOURNISSEUR, :OLD.CODE_CATEGORIE,
10 :OLD.QUANTITE, :OLD.PRIX_UNITAIRE,
11 :OLD.UNITES_STOCK, :OLD.UNITES_COMMANDEES,
12 :OLD.INDISPONIBLE, USER, SYSDATE);
```

```
13      WHEN DELETING THEN
14          INSERT INTO PRODUITS_POUBELLE
15              VALUES(:OLD.REF_PRODUIT, :OLD.NOM_PRODUIT,
16                      :OLD.NO_FOURNISSEUR, :OLD.CODE_CATEGORIE,
17                      :OLD.QUANTITE, :OLD.PRIX_UNITAIRE,
18                      :OLD.UNITES_STOCK, :OLD.UNITES_COMMANDEES,
19                      :OLD.INDISPONIBLE, USER, SYSDATE);
20      ELSE
21          INSERT INTO PRODUITS_INSERT
22              VALUES( :NEW.REF_PRODUIT, USER, SYSDATE);
23      END CASE;
24  END ProduitsArchive;
25  /
```

Déclencheur créé.

Atelier 11.1 L'approche objet



Questions

1. Quels sont les déclencheurs qui peuvent être compilés ?
 - A. CREATE OR REPLACE TYPE T1 IS OBJECT
(a1 CLIENTS.TELEPHONE%TYPE, a2 VARCHAR2(24));
 - B. CREATE OR REPLACE TYPE T1 IS OBJECT
(a1 NUMBER(1), a2 ROWID);
 - C. CREATE OR REPLACE TYPE T1 IS OBJECT
(a1 NUMBER(1));
 - D. CREATE OR REPLACE TYPE T1 IS OBJECT
(a1 LONG, a2 VARCHAR2(24));
 - E. CREATE OR REPLACE TYPE T1 IS OBJECT
(a1 NUMBER(1), a2 BOOLEAN);
 - F. CREATE OR REPLACE TYPE T1 IS OBJECT
(a1 NUMBER(1), a2 VARCHAR2(24));

Réponse : C, F

```
SQL> CREATE OR REPLACE TYPE T1 IS OBJECT
  2  ( a1 NUMBER(1), a2 NUMBER(1),
  3    CONSTRUCTOR FUNCTION T1( a1 NUMBER) RETURN SELF AS RESULT,
  4    MEMBER PROCEDURE m1);
  5  /
```

Type créé.

```
SQL> CREATE OR REPLACE TYPE BODY T1
  2  AS
  3    CONSTRUCTOR FUNCTION T1( a1 NUMBER) RETURN SELF AS RESULT IS
  4    BEGIN
  5      SELF.a1 := a1;SELF.a2 := a2; RETURN;
  6    END T1;
  7
  8    MEMBER PROCEDURE m1 IS
  9    BEGIN
 10      dbms_output.put_line('Objet T1 a1 :'||a1||' a2 :'||a2);
 11    END m1;
 12  END;
 13  /
```

Corps de type créé.

2. Pour l'objet T1 créé avec la syntaxe précédente, quels sont les blocs qui peuvent être compilés ?
 - A. DECLARE v1 T1; BEGIN v1.a1 := 1; v1.m1;END;
 - B. DECLARE v1 T1; BEGIN v1.m1; END;
 - C. DECLARE v1 T1 := T1(1,2); BEGIN v1.m1;END;

- D. DECLARE v1 T1 := T1(1); BEGIN v1.m1;END;
- E. DECLARE v1 T1; BEGIN v1 := T1(1,2);v1.m1;END;
- F. DECLARE v1 T1; BEGIN v1 := T1(1);v1.m1;END;
- G. DECLARE v1 T1; BEGIN v1 := T1(1,2,3);v1.m1;END;

Réponse : A, B, G

3. Pour les mêmes choix que la question précédente, quels sont les blocs qui affichent la chaîne suivante 'Objet T1 a1 :1 a2 :1' ?

Réponse : D, F

4. Pour les mêmes choix que la question précédente, quels sont les blocs qui affichent la chaîne suivante 'Objet T1 a1 :1 a2 :2' ?

Réponse : C, E

Exercice n°1 Les types d'objet

Créez un type 'DetCommObj' qui reprend à partir de la description de la table DETAILS_COMMANDES les colonnes suivantes :

- REF_PRODUIT
- PRIX_UNITAIRE
- QUANTITE
- REMISE

```
SQL> CREATE OR REPLACE TYPE DetCommObj IS OBJECT (
2   REF_PRODUIT      NUMBER(6),
3   PRIX_UNITAIRE    NUMBER(8,2),
4   QUANTITE         NUMBER(5),
5   REMISE           FLOAT(126)
6 );
7 /
```

Créez un type de tableau imbriqué 't_DetCommObj' qui stocke des objets de type 'DetCommObj'.

```
SQL> CREATE OR REPLACE TYPE t_DetCommObj IS TABLE OF DetCommObj;
2 /
```

Type créé.

Créez un objet 'CommandeObj' qui reprend la description complète de la table COMMANDES. L'objet doit contenir également un attribut de type tableau imbriqué 't_DetCommObj'.

```
SQL> CREATE OR REPLACE TYPE CommandeObj IS OBJECT (
2   NO_COMMANDE      NUMBER(6) ,
3   CODE_CLIENT      CHAR(5)   ,
4   NO_EMPLOYE       NUMBER(6) ,
5   DATE_COMMANDE    DATE      ,
6   DATE_ENVOI       DATE      ,
7   PORT             NUMBER(8,2),
8   ListDetCommObj   t_DetCommObj ,
9   CONSTRUCTOR      FUNCTION CommandeObj(
10                  NO_COMMANDE      NUMBER ,
```

```

11          CODE_CLIENT      CHAR      ,
12          NO_EMPLOYE       NUMBER    ,
13          DATE_COMMANDE    DATE      := SYSDATE,
14          DATE_ENVOI       DATE      := NULL,
15          PORT              NUMBER    := NULL)
16      RETURN SELF AS RESULT,
17      MEMBER PROCEDURE AfficheCommande,
18      MEMBER PROCEDURE AfficheListeProduits,
19      MEMBER PROCEDURE AjoutDeProduit(
20          REF_PRODUIT       NUMBER    ,
21          PRIX_UNITAIRE     NUMBER    ,
22          QUANTITE          NUMBER    ,
23          REMISE            FLOAT     := 0),
24      MEMBER FUNCTION MontantCommande
25      RETURN NUMBER,
26      MEMBER FUNCTION VerifieNotRefProduit(a_refprod NUMBER)
27      RETURN BOOLEAN,
28      MEMBER PROCEDURE EnvoisDeCommande
29  );
30  /

```

Type créé.

Créer un constructeur 'CommandeObj' pour permettre l'initialisation de l'objet uniquement avec les informations sur le numéro de commande, code client et numéro de l'employé. La date de la commande si elle n'est pas renseignée est la date du jour. Pour la date de l'envoi et les frais de port, il faut donner la possibilité de les renseigner si non les deux auront la valeur « **NULL** ».

Créez une méthode 'AjoutDeProduit' qui insère un détail de commande dans le tableau imbriqué.

Créez une méthode 'MontantCommande' qui renvoie le montant de la commande.

Créez une méthode 'EnvoisDeCommande' qui effectue la modification des unités en stock et des unités commandés, retirez des unités en stocks toutes les quantités de produits de la commande. S'il n'y a pas assez d'unités en stock, commandez la différence, en modifiant la valeur des unités commandées.

Pour finir, créez une méthode 'AfficheCommande' qui effectue l'affichage de l'ensemble des informations stockées dans l'objet.

```

SQL> CREATE OR REPLACE TYPE BODY CommandeObj
2  AS
3  -----
4  --          CONSTRUCTOR FUNCTION CommandeObj
5  -----
6      CONSTRUCTOR FUNCTION CommandeObj(
7          NO_COMMANDE       NUMBER    ,
8          CODE_CLIENT       CHAR      ,
9          NO_EMPLOYE        NUMBER    ,
10         DATE_COMMANDE     DATE      := SYSDATE,
11         DATE_ENVOI        DATE      := NULL,
12         PORT               NUMBER    := NULL)
13      RETURN SELF AS RESULT
14  IS
15      BEGIN

```



```

16         SELF.NO_COMMANDE      := NO_COMMANDE;
17         SELF.CODE_CLIENT      := CODE_CLIENT;
18         SELF.NO_EMPLOYE       := NO_EMPLOYE;
19         SELF.DATE_COMMANDE    := DATE_COMMANDE;
20         SELF.DATE_ENVOI       := DATE_ENVOI;
21         SELF.PORT              := PORT;
22         RETURN;
23     END CommandeObj;
24 -----
25 --             MEMBER PROCEDURE AfficheCommande
26 -----
27     MEMBER PROCEDURE AfficheCommande
28     IS
29     BEGIN
30         dbms_output.put_line('-----');
31         dbms_output.put_line('NO_COMMANDE      = ' || NO_COMMANDE);
32         dbms_output.put_line('CODE_CLIENT      = ' || CODE_CLIENT);
33         dbms_output.put_line('NO_EMPLOYE       = ' || NO_EMPLOYE);
34         dbms_output.put_line('DATE_COMMANDE = ' || DATE_COMMANDE);
35         dbms_output.put_line('DATE_ENVOI       = ' || DATE_ENVOI);
36         dbms_output.put_line('PORT              = ' || PORT);
37         dbms_output.put_line('Montant Commande = ' ||
38             LTRIM((TO_CHAR(MontantCommande,'999G999G999D00U'))));
39         dbms_output.put_line('-----');
40         AfficheListeProduits;
41     END AfficheCommande;
42 -----
43 --             MEMBER PROCEDURE AfficheListeProduits
44 -----
45     MEMBER PROCEDURE AfficheListeProduits
46     IS
47         v_indx INTEGER;
48     BEGIN
49         v_indx := ListDetCommObj.FIRST;
50         WHILE v_indx <= ListDetCommObj.LAST
51         LOOP
52             dbms_output.put_line('-----');
53             dbms_output.put_line('--REF_PRODUIT      = ' ||
54                 ListDetCommObj(v_indx).REF_PRODUIT);
55             dbms_output.put_line('-----');
56             dbms_output.put_line('--          PRIX_UNITAIRE = ' ||
57                 ListDetCommObj(v_indx).PRIX_UNITAIRE);
58             dbms_output.put_line('--          QUANTITE      = ' ||
59                 ListDetCommObj(v_indx).QUANTITE);
60             dbms_output.put_line('--          REMISE        = ' ||
61                 ListDetCommObj(v_indx).REMISE);
62             v_indx := ListDetCommObj.NEXT(v_indx);
63             dbms_output.put_line('-----');
64         END LOOP;
65     END AfficheListeProduits;
66 -----
67 --             MEMBER FUNCTION VerifieNotRefProduit
68 -----
69     MEMBER FUNCTION VerifieNotRefProduit(a_refprod NUMBER)

```

```

70     RETURN BOOLEAN
71     IS
72         v_refprod PRODUITS.REF_PRODUIT%TYPE;
73     BEGIN
74         SELECT REF_PRODUIT INTO v_refprod FROM PRODUITS
75         WHERE REF_PRODUIT = a_refprod;
76         RETURN FALSE;
77     EXCEPTION
78         when NO_DATA_FOUND then RETURN TRUE;
79     END VerifieNotRefProduit;
80 -----
81 --             MEMBER PROCEDURE AjoutDeProduit
82 -----
83     MEMBER PROCEDURE AjoutDeProduit(
84         REF_PRODUIT      NUMBER ,
85         PRIX_UNITAIRE    NUMBER ,
86         QUANTITE         NUMBER ,
87         REMISE           FLOAT  := 0)
88     IS
89     BEGIN
90         if (REF_PRODUIT IS NULL OR
91             PRIX_UNITAIRE IS NULL OR
92             QUANTITE IS NULL ) AND
93             VerifieNotRefProduit(REF_PRODUIT)
94         then
95             RETURN;
96         end if;
97         if ListDetCommObj IS NULL then
98             ListDetCommObj := t_DetCommObj(
99                 DetCommObj(REF_PRODUIT,PRIX_UNITAIRE,QUANTITE,REMISE));
100        else
101            ListDetCommObj.EXTEND;
102            ListDetCommObj(ListDetCommObj.LAST) :=
103                DetCommObj(REF_PRODUIT,PRIX_UNITAIRE,QUANTITE,REMISE);
104        end if;
105    END AjoutDeProduit;
106 -----
107 --             MEMBER FUNCTION MontantCommande
108 -----
109     MEMBER FUNCTION MontantCommande
110     RETURN NUMBER
111     IS
112         v_indx INTEGER;
113         v_montant DETAILS_COMMANDES.PRIX_UNITAIRE%TYPE := 0;
114     BEGIN
115         v_indx := ListDetCommObj.FIRST;
116         WHILE v_indx <= ListDetCommObj.LAST
117         LOOP
118             v_montant := v_montant + (
119                 ListDetCommObj(v_indx).PRIX_UNITAIRE *
120                 ListDetCommObj(v_indx).QUANTITE);
121             v_indx := ListDetCommObj.NEXT(v_indx);
122         END LOOP;
123         RETURN v_montant;

```

```

124     END MontantCommande;
125 -----
126 --             MEMBER PROCEDURE EnvoisDeCommande
127 -----
128     MEMBER PROCEDURE EnvoisDeCommande
129     IS
130         CURSOR c_prod( a_ref_prod NUMBER) IS
131             SELECT NVL(UNITES_STOCK,0) UNITES_STOCK,
132                    NVL(UNITES_COMMANDEES,0) UNITES_COMMANDEES
133             FROM PRODUITS
134             WHERE REF_PRODUIT = a_ref_prod
135             FOR UPDATE OF UNITES_COMMANDEES, UNITES_COMMANDEES;
136         v_indx INTEGER;
137     BEGIN
138         v_indx := ListDetCommObj.FIRST;
139         WHILE v_indx <= ListDetCommObj.LAST
140         LOOP
141             for v_p in c_prod(ListDetCommObj(v_indx).REF_PRODUIT)
142             loop
143                 if v_p.UNITES_STOCK < ListDetCommObj(v_indx).QUANTITE
144                 then
145                     v_p.UNITES_COMMANDEES := v_p.UNITES_COMMANDEES +
146                                                ListDetCommObj(v_indx).QUANTITE -
147                                                v_p.UNITES_STOCK;
148                     v_p.UNITES_STOCK := 0;
149                 else
150                     v_p.UNITES_STOCK := v_p.UNITES_STOCK -
151                                                ListDetCommObj(v_indx).QUANTITE;
152                 end if;
153                 UPDATE PRODUITS
154                     SET UNITES_STOCK      = v_p.UNITES_STOCK,
155                       UNITES_COMMANDEES = v_p.UNITES_COMMANDEES
156                 WHERE CURRENT OF c_prod;
157             end loop;
158             v_indx := ListDetCommObj.NEXT(v_indx);
159         END LOOP;
160         COMMIT;
161     END EnvoisDeCommande;
162 END;
163 /

```

Corps de type créé.

```

SQL> select REF_PRODUIT,NOM_PRODUIT,UNITES_STOCK,UNITES_COMMANDEES
       2  from PRODUITS
       3  WHERE REF_PRODUIT IN (1,2);

```

REF_PRODUIT	NOM_PRODUIT	UNITES_STOCK	UNITES_COMMANDEES
1	Chai	40	0
2	Chang	30	0

```

SQL> DECLARE
       2      v_CommandeObj CommandeObj := CommandeObj(1,'ALFKI',4);

```

```

3      a_refprod NUMBER;
4  BEGIN
5      v_CommandeObj.AjoutDeProduit( 1,10,10,0);
6      v_CommandeObj.AjoutDeProduit( 2,20,10,0);
7      v_CommandeObj.AfficheCommande;
8      v_CommandeObj.EnvoisDeCommande;
9  END;
10 /

```

```

-----
NO_COMMANDE      = 1
CODE_CLIENT      = ALFKI
NO_EMPLOYE       = 4
DATE_COMMANDE    = 20/07/06
DATE_ENVOI       =
PORT             =
Montant Commande = 300,00€

```

```

-----
--REF_PRODUIT    = 1

```

```

-----
--      PRIX_UNITAIRE = 10
--      QUANTITE      = 10
--      REMISE        = 0

```

```

-----
--REF_PRODUIT    = 2

```

```

-----
--      PRIX_UNITAIRE = 20
--      QUANTITE      = 10
--      REMISE        = 0

```

Procédure PL/SQL terminée avec succès.

```

SQL> select REF_PRODUIT,NOM_PRODUIT,UNITES_STOCK ,UNITES_COMMANDEES
2      from PRODUITS
3      WHERE REF_PRODUIT IN (1,2);

```

REF_PRODUIT	NOM_PRODUIT	UNITES_STOCK	UNITES_COMMANDEES
1	Chai	30	0
2	Chang	20	0

Exercice n°2 Le stockage de types d'objet

Pour pouvoir stocker des enregistrements de type 'CommandeObj' dans une table, vous devez créer la méthode « **MAP** » qui retourne la clé de la commande en occurrence NO_COMMANDE.

```

SQL> CREATE OR REPLACE TYPE CommandeObj IS OBJECT (
...
17  MAP MEMBER FUNCTION CleCommande
18      RETURN NUMBER,
...

```

```

SQL> CREATE OR REPLACE TYPE BODY CommandeObj
2   AS
...
24  -----
25  --                      MAP MEMBER FUNCTION CleCommande
26  -----
27  MAP MEMBER FUNCTION CleCommande
28      RETURN NUMBER
29      IS
30      BEGIN
31          RETURN SELF.NO_COMMANDE;
32      END CleCommande;
...

```

Créez une table 'TCommandeObj' qui reprend entièrement la description de l'objet, n'oubliez pas de mentionner l'alias pour le tableau imbriqué et la clé primaire qui reprend la colonne NO_COMMANDE.

Ecrivez un bloc PL/SQL qui permet de lire tous les enregistrements des tables COMMANDES et DETAIL_COMMANDES et les insère dans la nouvelle table 'TCommandeObj'.

```

SQL> CREATE TABLE TCommandeObj OF CommandeObj(
2   CONSTRAINT PK_TCommandeObj PRIMARY KEY (NO_COMMANDE)
3   ) NESTED TABLE ListDetCommObj STORE AS TListDetCommObj;

```

Table créée.

```

SQL> DECLARE
2   v_CommandeObj CommandeObj;
3   BEGIN
4   for comm in ( SELECT * FROM COMMANDES ) loop
5       v_CommandeObj := CommandeObj(
6           comm.NO_COMMANDE,comm.CODE_CLIENT,comm.NO_EMPLOYE,
7           comm.DATE_COMMANDE,comm.DATE_ENVOI,comm.PORT );
8   for det in ( SELECT * FROM DETAILS_COMMANDES
9               WHERE NO_COMMANDE = comm.NO_COMMANDE) loop
10      v_CommandeObj.AjoutDeProduit( det.REF_PRODUIT,
11                                     det.PRIX_UNITAIRE,det.QUANTITE,det.REMISE);
12      end loop;
13      INSERT INTO TCommandeObj VALUES v_CommandeObj;
14      end loop;
15      COMMIT;
16  END;
17  /

```

Procédure PL/SQL terminée avec succès.

```

SQL> DECLARE
2   v_CommandeObj CommandeObj;
3   BEGIN
4   SELECT VALUE(A) INTO v_CommandeObj FROM TCommandeObj A
5   WHERE A.NO_COMMANDE = 215094;
6   v_CommandeObj.AfficheCommande;
7   end;
8   /

```

```
-----  
NO_COMMANDE      = 215094  
CODE_CLIENT      = THEBI  
NO_EMPLOYE       = 79  
DATE_COMMANDE    = 07/01/2010  
DATE_ENVOI       = 19/02/2010  
PORT             = 93,2  
Montant Commande = 265 151,52€  
-----  
-----  
--REF_PRODUIT    = 1  
-----  
--      PRIX_UNITAIRE = 72,6  
--      QUANTITE      = 38  
--      REMISE        = 9,18  
-----  
-----  
--REF_PRODUIT    = 2  
-----  
--      PRIX_UNITAIRE = 51,48  
--      QUANTITE      = 142  
--      REMISE        = 7,18  
-----  
...  
  
Procédure PL/SQL terminée avec succès.
```

Atelier 12.1 Les packages intégrés

Exercice n°1 DBMS_OUTPUT

Créez deux blocs PL/SQL et en utilisant les propriétés du paramètre SERVEROUTPUT, le premier bloc retrouve le nom du premier produit et l'insère le tampon interne et le deuxième bloc utilise ce nom pour augmenter de 10% les unités en stock.

```
SQL> SELECT REF_PRODUIT, NOM_PRODUIT, UNITES_STOCK
2 FROM PRODUITS WHERE REF_PRODUIT = 1;
```

REF_PRODUIT	NOM_PRODUIT	UNITES_STOCK
1	Chai	39

```
SQL> SET SERVEROUTPUT OFF
```

```
SQL> DESC DBMSOUTPUT_LINESARRAY
```

```
DBMSOUTPUT_LINESARRAY VARRAY(2147483647) OF VARCHAR2(32767)
```

```
SQL> DECLARE
```

```
2 vnom_prod PRODUITS.NOM_PRODUIT%TYPE;
```

```
3 BEGIN
```

```
4 SELECT NOM_PRODUIT INTO vnom_prod FROM PRODUITS
```

```
5 WHERE REF_PRODUIT = 1;
```

```
6
```

```
7 DBMS_OUTPUT.ENABLE;
```

```
8 DBMS_OUTPUT.PUT_LINE( vnom_prod);
```

```
9 END;
```

```
10 /
```

Procédure PL/SQL terminée avec succès.

```
SQL> DECLARE
```

```
2 ligne varchar(32767);
```

```
3 nombre NUMBER ;
```

```
4 BEGIN
```

```
5 DBMS_OUTPUT.GET_LINE( ligne, nombre);
```

```
6 UPDATE PRODUITS SET UNITES_STOCK = UNITES_STOCK*1.1
```

```
7 WHERE NOM_PRODUIT = ligne;
```

```
8 END;
```

```
9 /
```

Procédure PL/SQL terminée avec succès.

```
SQL> SELECT REF_PRODUIT, NOM_PRODUIT, UNITES_STOCK
```

```
2 FROM PRODUITS WHERE REF_PRODUIT = 1;
```

REF_PRODUIT	NOM_PRODUIT	UNITES_STOCK
1	Chai	43

Exercice n°2 UTL_FILE

Créez un répertoire 'UTL_FILE_REPERTOIRE', l'objet de correspondance avec un répertoire physique sur le disque du serveur.

Créez un fichier et stockez tous les enregistrements des clients.

Attention si vous travaillez avec ORACLE XE, vous devez d'abord exécuter les commandes suivantes :

```
'SQLPLUS / AS SYSDBA'
@%ORACLE_HOME%\RDBMS\Admin\utlfile.sql
```

```
SQL> CREATE DIRECTORY UTL_FILE_REPERTOIRE AS
2                                     'C:\UTL_FILE_REPERTOIRE';
```

Répertoire créé.

```
SQL> GRANT READ,WRITE ON DIRECTORY UTL_FILE_REPERTOIRE TO PUBLIC;
```

Autorisation de privilèges (GRANT) acceptée.

```
SQL> DECLARE
2     v_nom_rep    VARCHAR2(255) := 'UTL_FILE_REPERTOIRE';
3     v_nom_fic    VARCHAR2(255) := 'F_CLIENTS' ||
4                                     TO_CHAR(SYSDATE, 'YYYYMMDD') || '.TXT';
5     v_tampon     VARCHAR2(32766);
6     v_fichier    UTL_FILE.FILE_TYPE;
7 BEGIN
8     v_fichier := UTL_FILE.FOPEN( v_nom_rep, v_nom_fic, 'W');
9     if UTL_FILE.IS_OPEN(v_fichier) then
10        for r_emp in ( SELECT SOCIETE, ADRESSE, VILLE,
11                        CODE_POSTAL, PAYS from CLIENTS)
12        loop
13            UTL_FILE.PUTF( v_fichier, '%s;%s;%s;%s;%s\n',
14                            r_emp.SOCIETE, r_emp.ADRESSE,
15                            r_emp.VILLE, r_emp.CODE_POSTAL, r_emp.PAYS);
16        end loop;
17        UTL_FILE.FCLOSE(v_fichier);
18    end if;
19 END;
20 /
```

Exercice n°3 DBMS_JOB

Placez les deux procédures de mise à jour du modèle étoile dans la file d'attente des travaux. La fréquence d'exécution de ces deux traitements doit être hebdomadaire.

```
SQL> DECLARE
2     v_travail_no NUMBER;
3 BEGIN
4     DBMS_JOB.SUBMIT( JOB          => v_travail_no,
5                     WHAT          => 'STAGIAIRE.MAJ_MODEL_ETOILE;',
6                     NEXT_DATE    => SYSDATE,
7                     INTERVAL     => 'SYSDATE + 1');
8     DBMS_OUTPUT.PUT_LINE( 'MAJ_MODEL_ETOILE' );
```



```

 9      DBMS_OUTPUT.PUT_LINE('Le numéro du travail '||v_travail_no);
10      DBMS_JOB.SUBMIT( JOB          =>v_travail_no,
11                      WHAT          =>'STAGIAIRE.MAJ_TABLES_AGREGATS;',
12                      NEXT_DATE     =>SYSDATE,
13                      INTERVAL      =>'SYSDATE + 1');
14      COMMIT;
15      DBMS_OUTPUT.PUT_LINE('MAJ_TABLES_AGREGATS');
16      DBMS_OUTPUT.PUT_LINE('Le numéro du travail '||v_travail_no);
17  END;
18  /
MAJ_MODEL_ETOILE
Le numéro du travail 25
MAJ_TABLES_AGREGATS
Le numéro du travail 26

Procédure PL/SQL terminée avec succès.

```

Exercice n°3 DBMS_METADATA

Créez un script dynamique qui recense la structure du schéma stagiaire.

```

SQL> HOST TYPE D:\liste_all.sql
SET ECHO OFF
SET LINESIZE 1500
SET LINESIZE 0
SET PAGES 0
SET HEAD OFF
SET FEEDBACK OFF
SET LONG 1000
SPOOL c:\liste_structure.sql
SELECT 'SELECT DBMS_METADATA.GET_DDL('' '||OBJECT_TYPE
      ||'' , '' '||OBJECT_NAME||'') from dual;'
FROM USER_OBJECTS;
SPOOL OFF
SPOOL c:\liste_structure.lst
@c:\liste_structure.sql
SPOOL OFF

SQL> HOST TYPE C:\liste_structure.sql
SELECT DBMS_METADATA.GET_DDL('FUNCTION' , 'NOMBRECONTRATS') from
dual;
SELECT DBMS_METADATA.GET_DDL('PROCEDURE' , 'MAJ_TABLES_AGREGATS')
from dual;
...

SQL> HOST TYPE C:\liste_structure.lst
CREATE OR REPLACE FUNCTION "STAGIAIRE"."NOMBRECONTRATS"
( a_nom VARCHAR2, a_debut DATE := NULL,a_fin DATE := NULL)
RETURN NUMBER
AS
begin
  for i in ( SELECT count(NO_COMMANDE) NB_CONTRATS
            FROM COMMANDES
            ...

```