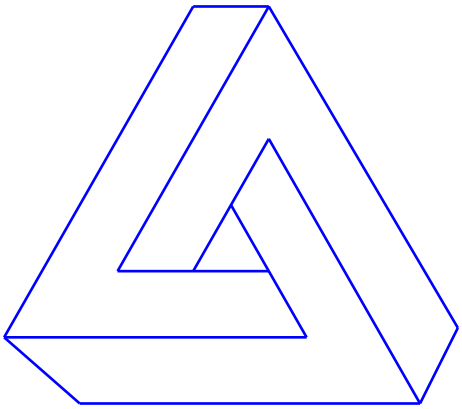
BASES DE DONNÉES

PL/SQL

Pascale BRIGOULET, Franck GLAZIOU,
Pascal LAFOURCADE et Marie-Françoise SERVAJEAN



Nom :

Prénom :

Groupe :

Table des matières

PL/SQL

PL/SQL (Procedural Language / Structured Query Language) est un langage fondé sur les paradigmes de programmation procédurale et structurée. C'est un langage propriétaire, créé par Oracle et utilisé dans le cadre de bases de données relationnelles. Il permet de combiner des requêtes SQL et des instructions procédurales (boucles, conditions...), dans le but de créer des traitements complexes destinés à être stockés sur le serveur de base de données (objets serveur), comme des procédures stockées ou des déclencheurs.

1 Fichier de commandes SQL

SQL est un utilitaire en ligne de commande d'Oracle qui permet aux utilisateurs d'exécuter interactivement des commandes SQL et PL/SQL. Il est ainsi possible de paramètrer les fichiers de commandes SQL pour avoir plus d'interaction avec l'utilisateur.

Variables. Il possible de stocker des données dans des variables. La commande `variable vnoproduit CHAR(6)` déclare une variable appellée `vnoproduit` de type chaîne de caractères de longueur 6. Afin de se souvenir qu'une variable est déclarée par le mot clé `variable` à l'extérieur d'un bloc PL/SQL, il est conseillé de préfixer les noms de ces variables par la lettre `v`. Cette commande déclare une «bind variable» utilisable dans une commande SQL. Les différents formats autorisés sont : `NUMBER`, `CHAR(n)` et `VARCHAR2(n)`.

Pour accéder à une variable il faut préfixer son nom par la caractère spécial & ainsi `&vnoproduit` permet d'accéder à la chaîne de caractères stockée dans la variable `vnoproduit`. Il est alors possible de faire des requêtes SQL qui utilisent le contenu de ces varaibles :

```
SELECT * FROM tproduit WHERE Nproduit = '&vnoproduit';
```

Interaction avec l'utilisateur. Afin d'interagir avec l'utilisateur, il existe deux types de commandes.

Affichage : Il est possible d'écrire le contenu d'une variable et d'afficher du texte.

- Pour afficher le contenu d'une variable, il faut utiliser la commande `PRINT` comme suit :
  `PRINT vnoproduit`
- Pour écrire un message visible par l'utilisateur, il faut utiliser le mot clé `prompt`. Par exemple la commande suivante affiche le texte placé après `prompt` :
  `PROMPT taper le nom du produit`
  ⚠ Attention il n'y a pas de quote.

Saisie : La seconde commande permet de saisir des données par un utilisateur. La commande suivante permet la saisie d'une variable au clavier et la déclare si elle ne l'était pas.

```
ACCEPT vnoproduit
```

2 Exécution du code PL/SQL

En SQL, les commandes sont transmises les unes après les autres et traitées séparément par le moteur SQL, comme le montre la Figure 1.

En extension procédurale de SQL (PL/SQL), les blocs de commandes sont transmis globalement au moteur SQL, comme le montre la Figure 2.

⚠ PL/SQL ne comprend pas d'instruction de saisie ou d'affichage.

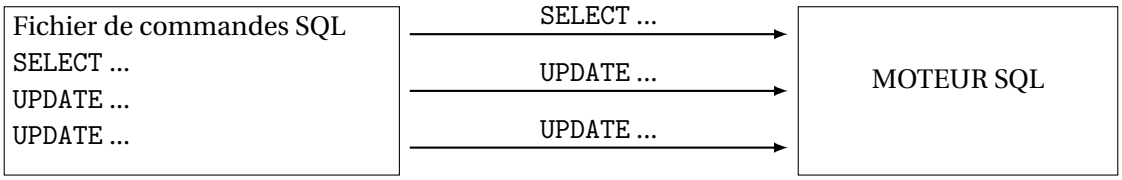


FIGURE 1

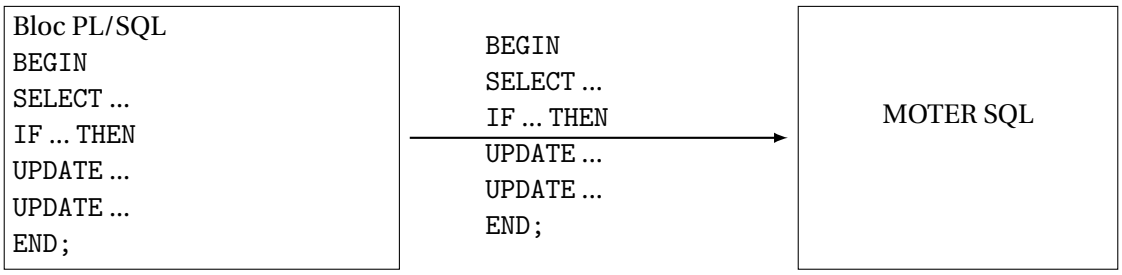


FIGURE 2 – Interaction entre PL/SQL et SQL.

3 Exemple de programme PL/SQL

Le fichier de commandes ci-dessous permet la saisie de la référence d'un produit et l'affichage de sa désignation ou d'un message d'erreur si le N° de produit n'existe pas. Ce fichier comporte trois parties :

— des commandes SQL+ pour la saisie de la référence produit,

— un bloc PL/SQL encadré dans l'exemple, il permet d'accéder à la base de données,

— des commandes SQL+ pour afficher le résultat.

```
set echo off
set verify off
set feed off

variable vnoproduit  char(4)
variable vdesignation varchar2(30)
prompt  taper la référence du produit à rechercher :
accept  vnoproduit
```

```
declare
dnoproduit char(4);

begin
dnoproduit := '&vnoproduit';
select designation into :vdesignation
from tproduit  where noproduit = dnoproduit;

exception
when no_data_found then
  :vdesignation := 'référence inconnue';
end;
```

```
.
/
print vdesignation

set verify on
set feed on
set echo on
```

⚠ Après la fin du bloc il faut impérativement mettre :

— `.` : ce point indique la fin du mode PL/SQL,

— `/` (seul sur une ligne) : cela déclenche l'exécution du bloc PL/SQL ou de l' ordre SQL stocké dans le buffer.

Les commandes `SET` positionnent des variables d'environnement pour éviter que le système n'affiche des informations « polluant » le résultat :

— `set echo off` évite que le système n'affiche la commande du bloc PL/SQL au moment où il l'exécute,

— `set verify off` évite que le système n'affiche l'opération de substitution au moment où il la fait,

— `set feed off` évite que le système n'affiche le nombre de lignes sélectionnées,

Les variables définies dans SQL par `VARIABLE` ou `ACCEPT` sont préfixées de : pour affectation, de `"&"` pour substitution.

Exercice 1. Écrire un fichier SQL qui permet d'afficher le message “Entrer votre age :”, de saisir l'âge de l'utilisateur dans la variable `vage`, d'ajouter un à cette valeur et d'afficher “Votre age plus un est 20 ans” si 19 est la valeur saisie par l'utilisateur et 20 le contenu de la variable `vage`.

4 Structure d'un bloc PL/SQL

PL/SQL n'interprète pas une commande, mais un ensemble de commandes contenu dans un "bloc" PL/SQL. Un bloc est composé de trois sections.

— Les sections `DECLARE` et `EXCEPTION` sont facultatives.

— Chaque instruction, de n'importe quelle section, est terminée par un ;

— Possibilité de placer des commentaires : `/* commentaire sur plusieurs lignes commençant par au moins un espace */`

— Il est conseillé d'utiliser `--` en début de ligne pour les commentaires.

⚠ Attention aux points virgules : en cas d'oubli, Oracle affiche un numéro de ligne attendant la suite de commande SQL.

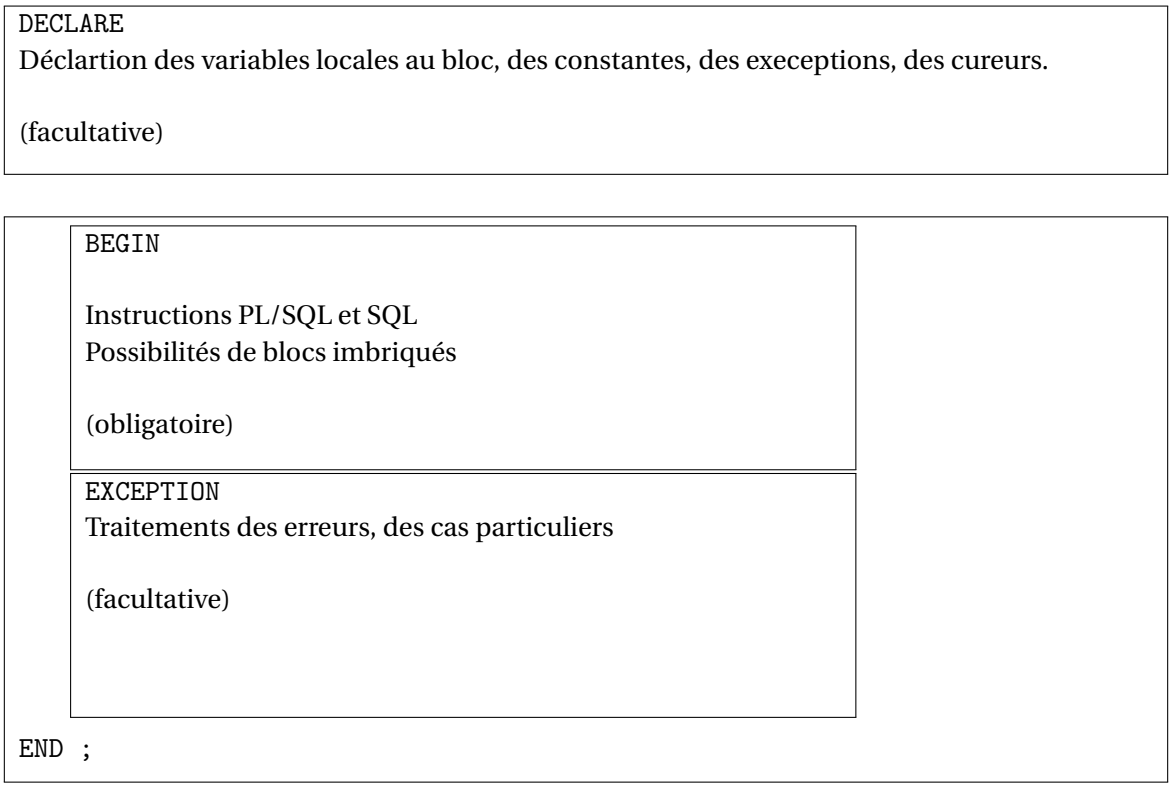


FIGURE 3 – Structure d'un programme PL/SQL.

5 Types de variables utilisés en PL/SQL

En PL/SQL les différents types possibles sont : `CHAR, NUMBER, DATE, VARCHAR2`. Les variables locales sont déclarées dans `DECLARE` et il est important de mettre le type adapté en suivant la syntaxe suivante :

```
nom_variable   type := valeur;
```

Création de variable. Il est nécessaire de choisir le type à l'initialisation lors de la déclaration et aussi sa valeur intiale comme le montre les deux exemples suivants :

— `Nom VARCHAR2(100);`

— `NB NUMBER := 1;`

— `dnoproduit tproduit.noproduit %type;`

La derenière déclaration de la variable `dnoproduit` avec le type `tproduit.noproduit %type` permet de prendre le même type que le champs `noproduit` de la table `tproduit`.

Exemple 5.1. Exemple simple d'utilisation de variable.

```
dnoproduit := '&vnoproduit';
select designation into :vdesignation
from tproduit
where  noproduit = '&vnoproduit';

select designation  into  ddesignation
from tproduit
where noproduit = dnoproduit;
```

Tableaux PL/SQL. Il est possible de créer des tableaux en PL/SQL. Pour cela il faut définir un type avec la commande `IS TABLE OF` et `INDEX BY BINARY_INTEGER`. Ensuite il faut utiliser la syntaxe adaptée pour accéder aux éléments du tableau ainsi créé.

Exemple 5.2. Exemple de création et d'utilisation d'un tableau.

```
DECLARE
TYPE tchar4 IS TABLE OF CHAR(4) INDEX BY BINARY_INTEGER;
table_noprod tchar4;
p BINARY_INTEGER;

BEGIN
p := 1 ;
table_noprod(p) := 'p001' ;
END;
.
/
```

6 Les traitements du bloc BEGIN

Dans le bloc PL/SQL, il est possible de d'affecter les résultats de `SELECT` dans des varaibles, de faire des tests et de boucles.

6.1 SELECT ... INTO ...

Les résultats monolignes des commandes `SELECT` peuvent être strockés dans des variables du mêmes types grâce au mot clé `INTO`. Par contre si un `SELECT` retourne plusieurs lignes alors il n'est pas possible de stocker ce résultat dans une variable. Pour cela il faut utiliser un *curseur* comme indiqué dans la section 8.

— Syntaxe :

```
SELECT coll, col2 INTO var1, var2
FROM table
[WHERE condition];
```

— Règle :

— La clause INTO est **obligatoire**.

— Le SELECT doit obligatoirement ramener une ligne et une seule, sinon erreur. Pour traiter un ordre SELECT qui pourrait ramener plusieurs lignes, il faut utiliser un curseur.

Les autres ordres SQL de manipulation sont inchangés et des variables peuvent être utilisées :

```
INSERT INTO table VALUES(var1, 'chaine', 123, var2);
UPDATE table SET col2 = var1 WHERE col1 = var2;
```

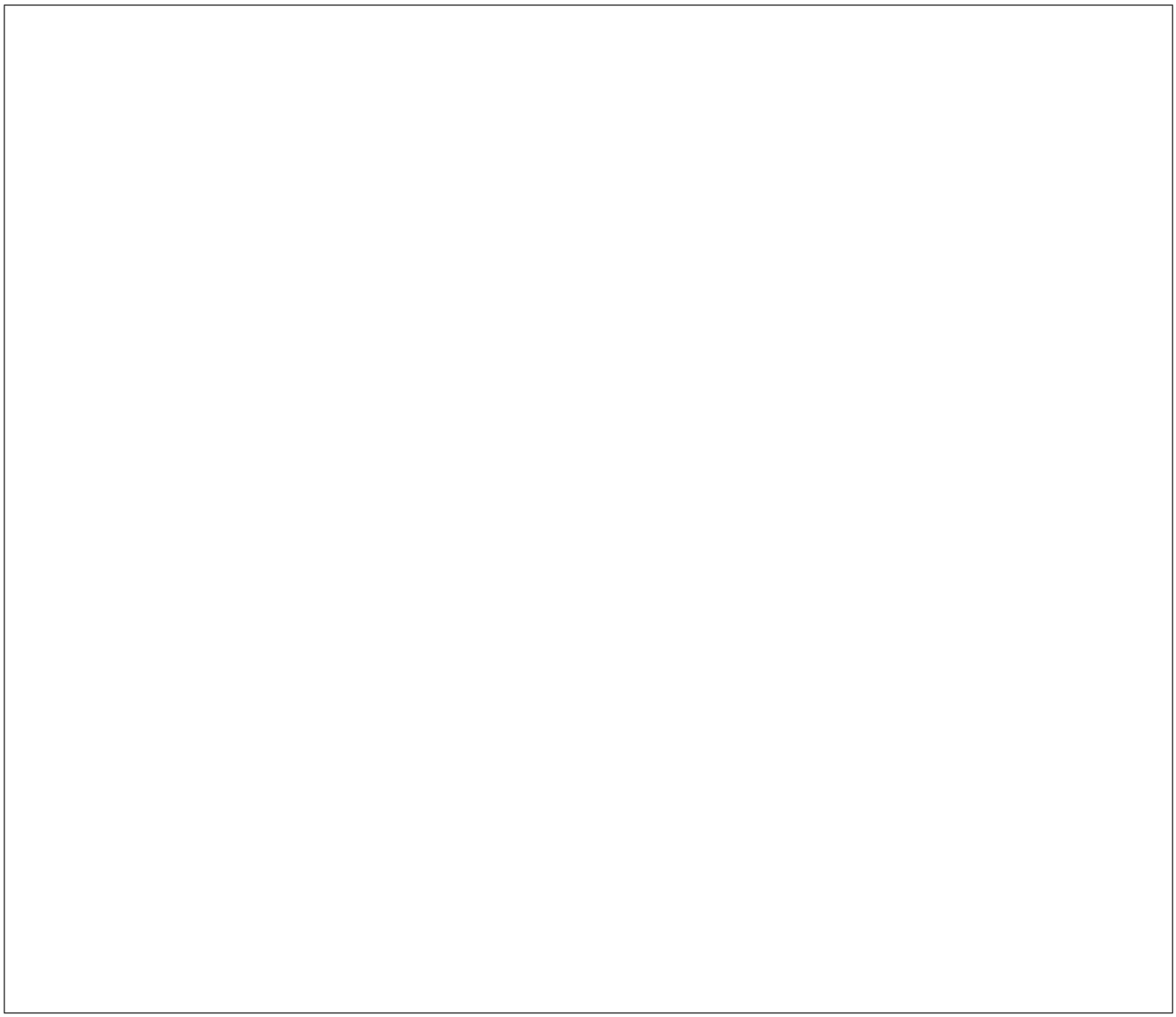
Exercice 2. Calculer le nombre de fournisseurs d'un produit entré par l'utilisateur. Le résultat sera écrit dans une table Tligne. Les tables utilisées sont données dans la Figure 4.

6.2 Traitements conditionnels IF ... THEN ... END IF;

Les opérateurs utilisés dans les conditions en PL/SQL sont les mêmes que dans SQL :
=, <, >, !=, >=, <=, IS NULL, IS NOT NULL, BETWEEN, LIKE, AND, OR, ...
La syntaxe pour écrire une condition est la suivante :

```
IF condition1 THEN traitement1;
ELSE traitement2;
END IF;
```

Exercice 3. Calculer le nombre de fournisseurs d'un produit donné. S'il n'y a pas de fournisseur, compter le nombre de produits : il doit être ≥ 1 ou 0. Écrire dans une table de Tligne soit le nombre de fournisseurs soit le message '`le produit n'existe pas`'.



6.3 Traitements répétitifs WHILE ...LOOP ...END LOOP;

La boucle WHILE : L'exécution de la boucle se fait tant que la condition de la clause de WHILE est vérifiée.

```
BEGIN
WHILE condition
LOOP
instructions;
END LOOP;
END;
```

La condition est une combinaison d'expressions au moyen d'opérateurs : <, >, =, !=, AND, OR, LIKE, ...

Exemple 6.1. Calcul de factorielle.

```
CREATE TABLE Tligne (ligne VARCHAR2(200));
VARIABLE n NUMBER
PROMPT taper n
ACCEPT n
```

```
DECLARE
fn NUMBER := 1;
i NUMBER :=1;
n NUMBER :='&n';
BEGIN
WHILE i<n
LOOP
i := i + 1;
fn := fn * i;
END LOOP;

INSERT INTO Tligne
VALUES ( 'Factorielle de '|| TO_CHAR(n)||, '  ',TO_CHAR(fn));
END;
/
SELECT * FROM Tligne;
DROP TABLEl Tligne;
```

7 Gestion des erreurs : EXCEPTION ... WHEN ... THEN

Lorsqu'une instruction se passe mal en PL/SQL, une *exception* est levée. Il est possible de spécifier une comportement adpaté dans ce cas dans la section EXCEPTION.

7.1 La section EXCEPTION

La section EXCEPTION permet d'affecter un traitement approprié aux erreurs survenues lors de l'exécution du bloc PL/SQL.

Les types d'erreurs sont les suivants :

— erreurs Oracle : elles sont prédéfinies,

— erreur programme utilisateur : à déclarer.

Exemple 7.1. Les résultats à afficher sont placés dans une table Tligne. Lorsque la requête SQL ne produit pas de résultats alors une exception est levée.

```
drop table Tligne;
create table Tligne (Tligne varchar2(150));
variable vnoproduit char(4)
prompt taper la référence du produit à rechercher :
accept vnoproduit

declare
dnoproduit char(4) ;
ddesignation varchar2(30);
dmessage varchar2(150);

begin
dnoproduit := '&vnoproduit' ;

dmessage := 'Référence inconnue ';
```

```
select designation into ddesignation From tproduit  where noproduit = dnoproduit ;

insert into Tligne values ('Désignation: ' || ddesignation);

exception
when no_data_found then
 insert into Tligne values (dmessage);
end ;
.
/
select * from Tligne;
```

Remarque 7.1. Le message d'erreur est initialisé juste avant la requête SQL qui risque de provoquer l'erreur ; la méthode est à utiliser quand plusieurs requêtes SQL sont susceptibles de déclencher la même exception.

Exercice 4. Enregistrer une livraison pour un produit donné :

- saisie du numéro de produit et de la quantité livrée,
- accès au stock du produit : exception si '`référence inconnue`',
- calcul du nouveau stock et mise à jour du stock.

Exercice 5. Enregistrer un nouveau produit d'un fournisseur :

- saisie des références du produit et du fournisseur, du prix fournisseur,
- accès au produit pour vérifier qu'il existe,
- accès au fournisseur pour vérifier qu'il existe,
- accès à la liaison produit-fournisseur pour vérifier qu'elle n'est pas déjà enregistrée (Exception à gérer),
- enregistrement de la liaison.

7.2 Erreur Oracle

Dès que l'erreur Oracle est rencontrée, passage automatique à la section `EXCEPTION` pour réaliser le traitement approprié à l'erreur. L'erreur est documentée par un code d'erreur `SQLCODE` et un message `SQLERRM`.

Syntaxe :

```
EXCEPTION
WHEN nom erreur THEN traitement;
[WHEN nom erreur THEN traitement;]
...
[WHEN OTHERS THEN traitement;]
```

Sortie du bloc après exécution du traitement.

Les principales erreurs Oracle prédéfinies sont décrires dans le tableau ci-dessous.

déclenchée par :	nom de l'erreur :	valeur correspondante du sqlcode
insert, update	DUP_VAL_ON_INDEX	-1
	INVALID_CURSOR	-1001
	INVALID_NUMBER	-1722
	LOGIN_DENIED	-1017
select	NO_DATA_FOUND	+100 (non déclenché par update ou delete, ni par select count, select sum)
	NO_LOGGED_ON	-1012
	PROGRAM_ERROR	-6501
	STORAGE_ERROR	-6500
	TIMEOUT_ON_RESOURCE	-51
select	TOO_MANY_ROWS	-1422
	VALUE_ERROR	-6502
	ZERO_DIVIDE	-1476
	OTHERS	toutes les autres erreurs non explicitement nommées, en particuliers celles concernant les contraintes d'intégrité et les conflits d'accès. Others doit être la dernière erreur de la section exception.

Exemple 7.2 (Détection de doublons sur une clé primaire.)**.** La séquence suivante crée une ligne dans la table `tproduit (noproduit char(4) primary key, ...)` et stocke les erreurs dans une table `Tligne (ligne varchar2(150))`;

```
BEGIN
INSERT INTO tproduit VALUES (dnoproduit, ddesignation ...);
EXCEPTION
WHEN DUP_VAL_ON_INDEX THEN
INSERT INTO Tligne VALUES (dnoproduit  ||  'déjà inséré');
END;
```

Exemple 7.3. Récupération du message d'erreur retourné par Oracle en cas d'exception.

⚠ Attention : SQLERRM et SQLCODE ne peuvent être utilisés directement dans une instruction SQL. Il faut les récupérer dans une variable.

```
DECLARE
dnom tproduit.designation%type;
dmess1 VARCHAR2(100);
dmess2 VARCHAR2(100);
BEGIN
dmess1 := 'Référence inconnue';
SELECT designation INTO dnom FROM tproduit
WHERE noproduit = '&vnoproduit';

EXCEPTION
WHEN OTHERS THEN
dmess2 := SQLERRM;
INSERT INTO Tligne VALUES (dmess1);
INSERT INTO Tligne VALUES (dmess2);
COMMIT;
END;
```

Exercice 6. Pour mettre en évidence l'exception TOO_MANY_ROWS, créez un fichier de commandes SQL qui :

— demande un numéro de fournisseur,

— accède aux N° des produits de ce fournisseur (par `select into`)

— retourne le message '`le fournisseur n'a pas de produit`' ou '`le fournisseur a plusieurs produits`', ou le N° du produit s'il n'en a qu'un.

Le résultat sera placé dans une table `TLIGNE (LIGNE varchar2(200))`. Placez aussi dans cette table le `SQLCODE` et le message Oracle obtenus à l'issue de `SELECT`. Modifiez le programme en remplaçant une des deux exceptions par `OTHERS`.

7.3 Erreur utilisateur : EXCEPTION, RAISE

Le traitement de l'anomalie doit être déclenché en passant dans la partie `EXCEPTION` par `RAISE`.

```
DECLARE
...
nom_erreur EXCEPTION;
...
BEGIN
...
IF anomalie
THEN RAISE nom_erreur;
...
EXCEPTION
WHEN nom_erreur THEN
traitement à effectuer en cas d'anomalie;
END;
```

Sortie du bloc après exécution du traitement.

Exemple 7.4. Accèder au stock d'un produit de Numéro donné avec Erreur Oracle quand le produit n'existe pas et Erreur utilisateur quand le stock est nul. Les anomalies sont placées dans une table `terreur(z1 varchar2(30), z2 varchar2(30))`

```
DECLARE
...
stock_nul EXCEPTION;
dstock number;
...
BEGIN
SELECT stock INTO dstock FROM tproduit WHERE noproduit = '&vnoproduit';
IF stock = 0
THEN RAISE stock_nul;
traitement quand le stock n'est pas nul...
EXCEPTION
WHEN NO_DATA_FOUND THEN
INSERT INTO terreur VALUES ('&vnoproduit', 'produit inconnu');
WHEN stock_nul THEN
INSERT INTO terreur VALUES ('&vnoproduit', 'stock nul');
END;
```

Exercice 7. Pour remplacer l'erreur Oracle par une erreur utilisateur, compter le nombre de produits ayant le numéro donné et déclencher une exception si ce nombre est nul.

8 Les curseurs

Lorsqu'une requête SELECT est susceptible de délivrer plusieurs lignes, on associe à celle ci un «curseur explicite» qui va permettre d'accéder aux lignes du résultat. L'algorithme est analogue au traitement d'un fichier séquentiel.

8.1 DECLARE, OPEN, FETCH, CLOSE

Il faut d'abord déclarer le curseur dans le bloc DECLARE, ensuite ouvrir le curseur avec la commande OPEN et penser à le fermé une fois celui-ci utilisé. Pour accéter aux données les unes après les autres il faut utiliser la commande FETCH.

Exemple 8.1. Traitement des rayons d'un étage donné.

```
DECLARE
detage trayon.etage%TYPE;
dcode_rayon trayon.code_rayon%TYPE;
dnom_rayon trayon.nom_rayon%TYPE;

CURSOR r IS select code_rayon, nom_rayon from trayon where etage = detage;

BEGIN
detage := 2;
 OPEN r;
FETCH  r  INTO dcode_rayon, dnom_rayon;

WHILE r%FOUND
LOOP
 ... traitement de la ligne lue ...
FETCH r INTO dcode_rayon, dnom_rayon;
END LOOP;
CLOSE r;
END;
```

Exercice 8. Caculer le pourcentage de produits ayant un stock nul. Proposer une solution avec curseur et une sans.

Exercice 9. Créer une table des produits classés par valeur du stock : No de produit, désignation, valeur du stock, classement.

Exercice 10. Créer une table des produits qui n'ont pas de fournisseur et une table des produits qui en ont avec les noms de leurs fournisseurs.

8.2 UPDATE, DELETE ... WHERE CURRENT OF curseur

Lors du parcours d'une table (ou une vue modifiable) par FETCH à l'aide d'un curseur, il est possible de demander à mettre à jour ou supprimer la ligne courante par la condition WHERE CURRENT OF nom du curseur. Le curseur doit avoir été déclaré FOR UPDATE

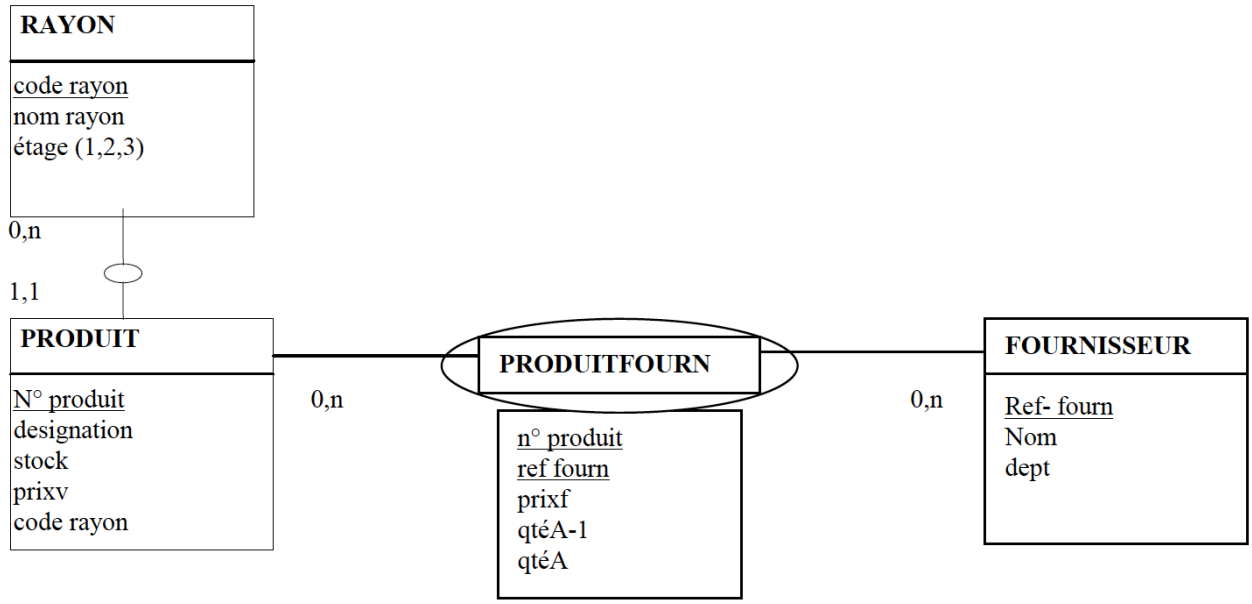
```
CURSOR  r  IS
select code_rayon, nom_rayon from trayon where etage = detage FOR UPDATE;
```

La mise à jour se fera par :

```
UPDATE trayon SET nom_rayon = dnom_rayon WHERE CURRENT OF r;
COMMIT;
```

⚠ Attention, terminer par `COMMIT`, sinon risque d'interblocage entre deux exécutions qui voudraient écrire dans la même table.

Exercice 11. Écrire un code PL/SQL qui augmenter de 10% les produits de plus de 1000€, et de 5% les autres.



prixv : prix de vente
prixf : prix du produit chez le fournisseur (dernier prix connu) ; doit être renseigné si qtéA ou qtéA-1 sont renseignées
qtéA-1 : quantité livrée l'année précédente par ce fournisseur
qtéA : idem pour l'année en cours

FIGURE 4 – Base de données pour les exercices.

Commandes utiles pour les TP

Première séance : Connexion : `sqlplus <user>/<password>@KIROV`
`source /etc/profile`
Lancer `sqlplus` avec cette commande `rlwrap sqlplus` vous permet d'avoir l'historique.
Lors de la première connexion modifier le mot de passe avec la commande SQL : `PASSWORD;`
Ce qui est équivalent à : `ALTER USER dupond IDENTIFIED BY password;`
Si le mot de passe est égaré quelque part, il faut se connecter en `ssh` sur `londres` et faire `oracle_passwd`
Pour lancer un fichier `.sql` en sqlplus, il suffit de taper : `@toto.sql;`
Pour quitter sqlplus, il suffit de taper : `quit;`

Mise en forme : Sous SQL :

```
Set linesize 150 -- positionne la taille d'une ligne
Set pagesize 300 --  positionne le nombre de lignes avant de réafficher les entêtes
Set pages 0 --  n'affiche pas les entêtes
Col <nom_colonne> for A10 --  défini que la colonne nom_colonne va être affiché sur 10 caractère
alphanumériques, 999.99 pour les valeurs numériques.
```

Corbeille : Vider les tables `BIN$$xxxx` : avec la commande `PURGE RECYCLEBIN;`

Débug : Afficher la strucutre de la table `nba` : `describle nba;`
Connaître l'utilisateur connecté : `show user;`
Liste des tables d'un `user` : `SELECT table_name FROM user_tables;`
Lister les tables accessibles par l'utilisateur : `SELECT table_name FROM all_tables;`
Lister toutes les tables possédées par un utilisateur `SELECT * FROM all_tables WHERE owner='PALAFOUR';`
Liste des vues d'un `user` : `SELECT view_name FROM user_views;`
Liste des contraintes : `SELECT * FROM user_constraints WHERE table_name='<table>';`
`SELECT * FROM user_cons_columns WHERE table_name='<table>";`
`show errros` affiche les erreurs des fonctions.

Contraintes : Description d'une table : `describe <table>` ou `desc <table>;`
Liste des colonnes concernées par les contraintes : `SELECT * FROM user_cons_columns;`
Lire une table d'un autre schéma :
`SELECT * FROM <schema>.<table_name>; -- ou schma = login de connexion de l'utilisateur`

Affichage : `SET HEADING OFF`
`SET FEEDBACK OFF`

Travailler à la maison Il est possible d'accèder par SSH à la machine `londres`. L'adresse de la passerelle est `ssh.iut-clermont.uca.fr` qui n'est accessible que par une authentification avec des clefs SSH.
Pour déposer vos clefs vous pouvez utiliser `https://homeweb.iut-clermont.uca.fr`

En cas de blocage : Dans un terminal executer la commande suivante pour voir les `pid` des processus zombies : `ps -aux | grep sqlplus`
Ensuite tuer ces zombies grâce à la commande `kill -9 numerodepid`