



PL/SQL

Première année

M. Nassar

**Ecole Nationale Supérieure d'Informatique
et d'Analyse des Systèmes
- Rabat -**

PL/SQL FOR ORACLE

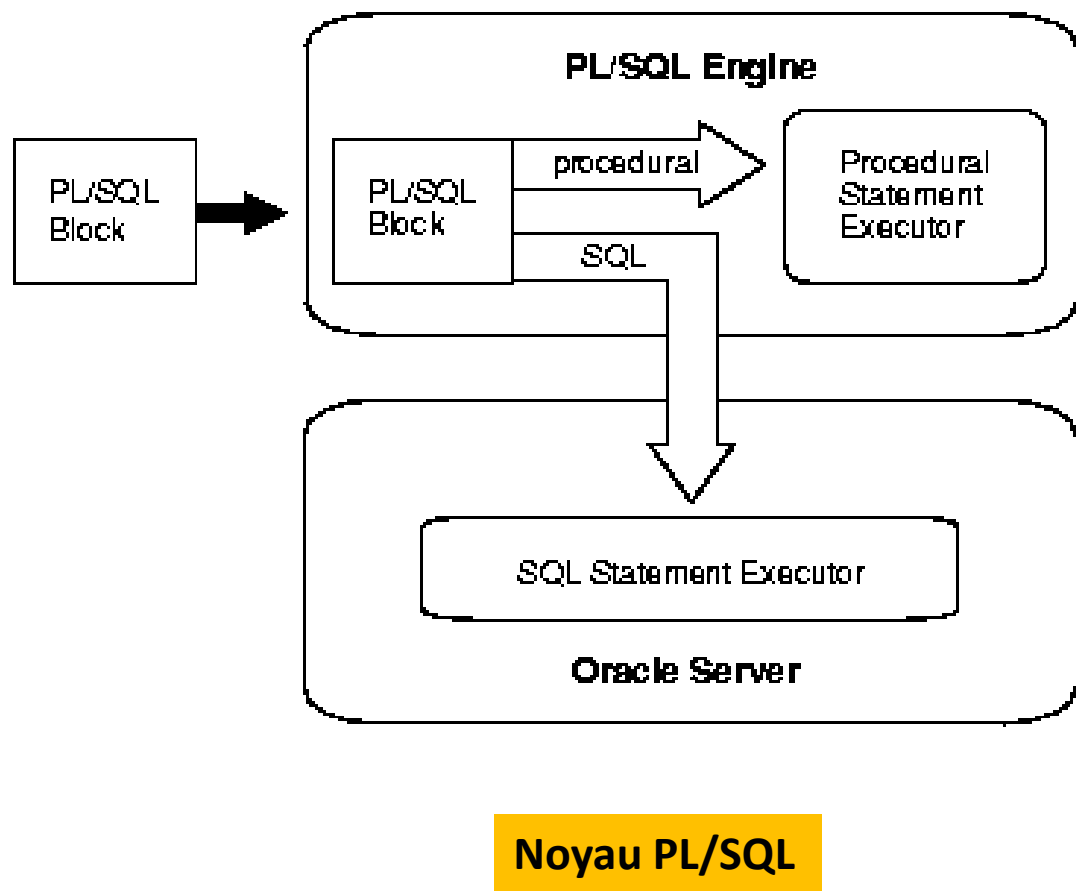
Concepts de base

M. NASSAR & R. OULAD HAJ THAMI
ENSIAS - Rabat

SOMMAIRE GENERAL

- 1. MOTIVATIONS**
- 2. STRUCTURE D' UN BLOC PL/SQL**
- 3. LES VARIABLES**
- 4. LES ENREGISTREMENTS**
- 5. ASSIGNATION DES VARIABLES ET AFFECTATION**
- 6. STRUCTURES DE CONTRÔLE**
- 7. LES COLLECTIONS**
- 8. LES TRANSACTIONS**
- 9. INSERT-UPDATE-DELETE DANS UN BLOC PL/SQL**
- 10. GESTION DES ERREURS ET DES EXCEPTIONS**
- 11. LES CURSEURS**
- 12. LES PROCEDURES ET LES FOCNTIONS STOCKEES**
- 13. LES PACKAGES**
- 14. LES TRIGGERS**

MOTIVATION



Avantages de PL / SQL

Prise en charge de SQL

Prise en charge de la programmation orientée objet

Meilleure performance

Une productivité accrue

La portabilité

L'intégration très forte avec Oracle

Haute sécurité

PARTIE 1:
LES FONDAMENTAUX

PARTIE 1: SOMMAIRE

STRUCTURE D' UN BLOC PL/SQL

LES VARIABLES

LES ENREGISTREMENTS

ASSIGNATION DES VARIABLES ET AFFECTATION

Dans le bloc PL/SQL

A partir du clavier

A partir d' une BD

STRUCTURES DE CONTRÔLE

IF - THEN - ELSE - END IF

LOOP - EXIT WHEN - END LOOP

WHILE - LOOP - END LOOP

FOR - IN – LOOP

CASE - WHEN - THEN - ELSE - END CASE

TRAVAUX PRATIQUES

STRUCTURE D' UN BLOC PL/SQL

DECLARE *--section optionnelle*
déclaration variables, constantes, types, curseurs,...

BEGIN *--section obligatoire*
contient le code PL/SQL

EXCEPTION *--section optionnelle*
traitement des erreurs

END; *--obligatoire*

DECLARE *--section optionnelle*

déclaration variables, constantes, types, curseurs,...

BEGIN *--section obligatoire*

contient le code PL/SQL

DECLARE *--section optionnelle*

déclaration variables, constantes, types, curseurs,...

BEGIN *--section obligatoire*

contient le code PL/SQL

EXCEPTION *--section optionnelle*

traitement des erreurs

END; *--obligatoire*

EXCEPTION *--section optionnelle*

traitement des erreurs

END; *--obligatoire*

REMARQUE

LA PORTEE DES VARIABLES EST LA MEME QUE DANS LES LANGAGES DE PROGRAMMATION

LES VARIABLES

nom variable [**CONSTANT**] **type** [[**NOT NULL**] [**:=** expression | **DEFAULT** expression];

nom variable représente le nom de la variable composé de lettres, chiffres, \$, _ ou #
Le nom de la variable ne peut pas excéder 30 caractères

CONSTANT indique que la valeur ne pourra pas être modifiée dans le code du bloc PL/SQL

NOT NULL indique que la variable ne peut pas être NULL, et dans ce cas **expression** doit être indiqué.

type représente de type de la variable correspondant à l'un des types suivants :

Remarque

Si une variable est déclarée avec l'option **CONSTANTE**, elle doit être initialisée
Si une variable est déclarée avec l'option **NOT NULL**, elle doit être initialisée

TYPE	SEMANTIQUE
NUMBER[(e,d)]	Nombre réel avec e chiffres significatifs stockés et d décimales
PLS_INTEGER	Nombre entier compris entre -2 147 483 647 et +2 147 483 647
CHAR [(n)]	Chaîne de caractères de longueur fixe avec n compris entre 1 et 32767 (par défaut 1)
VARCHAR2[(n)]	Chaîne de caractères de longueur variable avec n compris entre 1 et 32767
BOOLEAN	
DATE	
RAW[[[(n)]]	Chaîne de caractères ou données binaires de longueur variable avec n compris entre 1 et 32767. Le contenu d'une variable de ce type n'est pas interprété par PL/SQL
LONG RAW	Identique au type LONG qui peut contenir des données binaires
LONG	Chaîne de caractères de longueur variable avec au maximum 32760 octets
ROWID	Permet de stocker l'adresse absolue d'une ligne dans une table sous la forme d'une chaîne de caractères

Exemples de types de bases PL/SQL

SUBTYPE nom_sous_type **IS** type ;

Exemple:

SUBTYPE nom_employe IS VARCHAR2(20) NOT NULL;

nom nom_employe;

nom_variable nom_table.nom_colonne%TYPE ;
nom_variable nom_variable_ref%TYPE ;

Exemple:

Nom **E_EMPLOYE.NOM%TYPE;**
Dat_COM DATE;
Dat_LIV **Dat_COM%TYPE;**

nom_variable nom_table%ROWTYPE ;

Exemple:

EMPLOYEE **E_EMPLOYE%ROWTYPE;**

LES ENREGISTREMENTS


```
TYPE nom_type_rec IS RECORD (  
    nom_champ1      type_élément1 [[ NOT NULL ] := expression ],  
    nom_champ2      type_élément2 [[ NOT NULL ] := expression ],  
    ...  
    nom_champN      type_élémentN[[ NOT NULL ] := expression ]  
);  
Nom_variable      nom_type_rec ;
```

Exemple:

```
TYPE T_REC_EMP IS RECORD (  
    Num      E_EMPLOYE.NO%TYPE,  
    Nom      E_EMPLOYE.NOM%TYPE,  
    Pre      E_EMPLOYE.PRENOM%TYPE  
);  
  
EMP      T_REC_EMP ;
```

ACCES:

EMP.num	EMP.NOM	et	EMP.Pre
---------	---------	----	---------

**ASSIGNATION DES
VARIABLES
(AFFECTATION)**

VARIABLE := EXPRESSION

- Lors de la déclaration
- Dans le bloc PL/SQL

MON_NUM:= 10 ;

MA_CHAINE := 'Chaîne de caractères' ;

MA_TAXE :=PRIX*TAUX;

MON_BOOLEAN := FALSE; MON_BOOLEAN := (NOM='toto');

BONUS := SALAIRE * 0.10;

MA_LIMITE_BUDGET CONSTANT REAL := 5000.00;

MA_DATE:= '12/12/2012'

MON_DEP := DEPARTEMENT.NUMDEP;

BASES DE DONNEES RELATIONNELLES

Exemple 1:

--partie des messages pour inviter l'utilisateur à saisir des données

prompt CREATION D'UN NOUVEAU CLIENT

prompt ENTREZ MES DONNEES DU NOUVEAU CLIENT:

ACCEPT L_NO **prompt** 'Employee : '

prompt

ACCEPT L_NOM **prompt** 'NOM: '

prompt

ACCEPT L_SAL **prompt** 'SALAIRE : '

prompt

ACCEPT L_DATE **prompt** 'DATE (mm/dd/yyyy): '

prompt

DECLARE

NO NUMBER(4):=**&L_NO**;

NOM VARCHAR2(20):=**'&L_NOM'**;

SAL NUMBER(10,2):=**&L_SAL**;

DT_REC DATE:=**'&L_DATE'**;

BEGIN

DBMS_OUTPUT.PUT_LINE ('NUMERO: ' || NO);

DBMS_OUTPUT.PUT_LINE ('NOM: ' || NOM);

DBMS_OUTPUT.PUT_LINE ('SALAIRE: ' || SAL);

DBMS_OUTPUT.PUT_LINE ('DATE: ' || DT_REC);

END;

/

Exemple 1: exécution

SQL> --partie des messages pour inviter l'utilisateur à saisir des données

SQL> **prompt CREATION D'UN NOUVEAU CLIENT**

CREATION D'UN NOUVEAU CLIENT

SQL> **prompt ENTREZ MES DONNEES DU NOUVEAU CLIENT:**

ENTREZ MES DONNEES DU NOUVEAU CLIENT:

SQL>

SQL> **ACCEPT L_NO** **prompt 'Employee : '**

Employee : 99

SQL> **prompt**

SQL> **ACCEPT L_NOM** **prompt 'NOM: '**

NOM: toto

SQL> **prompt**

SQL> **ACCEPT L_SAL** **prompt 'SALAIRE : '**

SALAIRE : 5005.50

SQL> **prompt**

SQL> **ACCEPT L_DATE** **prompt 'DATE (mm/dd/yyyy): '**

DATE (mm/dd/yyyy): 10/12/2012

SQL> **prompt**

```
SQL> DECLARE
2      NO      NUMBER(4):=&L_NO;
3      NOM     VARCHAR2(20):='&L_NOM';
4      SAL     NUMBER(10,2):=&L_SAL;
5      DT_REC  DATE:='&L_DATE';
6 BEGIN
7      DBMS_OUTPUT.PUT_LINE ('NUMERO:          ' || NO);
8      DBMS_OUTPUT.PUT_LINE ('NOM:           ' || NOM);
9      DBMS_OUTPUT.PUT_LINE ('SALAIRE:       ' || SAL);
10     DBMS_OUTPUT.PUT_LINE ('DATE:          ' || DT_REC);
11 END;
12 /
```

ancien 2 :	NO	NUMBER(4):=&L_NO;
nouveau 2 :	NO	NUMBER(4):=99;
ancien 3 : NOM		VARCHAR2(20):='&L_NOM';
nouveau 3 :	NOM	VARCHAR2(20):='toto';
ancien 4 : SAL		NUMBER(10,2):=&L_SAL;
nouveau 4 :	SAL	NUMBER(10,2):=5005.50;
ancien 5 : DT_REC		DATE:='&L_DATE';
nouveau 5 :	DT_REC	DATE:='10/12/2012';
NUMERO:		99
NOM:		toto
SALAIRE:		5005,5
DATE:		10/12/12

Procédure PL/SQL terminée avec succès.

Exemple 2: exécution

SQL> **SET VERIFY OFF;** --pour ne pas afficher les anciennes valeurs &...

SQL> SET SERVEROUTPUT ON

SQL>

SQL> DECLARE

2 NO NUMBER(4):=&L_NO;

3 NOM VARCHAR2(20):='&L_NOM';

4 SAL NUMBER(10,2):=&L_SAL;

5 DT_REC DATE:='&L_DATE';

6 BEGIN

7 DBMS_OUTPUT.PUT_LINE ('NUMERO: ' || NO);

8 DBMS_OUTPUT.PUT_LINE ('NOM: ' || NOM);

9 DBMS_OUTPUT.PUT_LINE ('SALAIRE: ' || SAL);

10 DBMS_OUTPUT.PUT_LINE ('DATE: ' || DT_REC);

11 END;

12 /

NUMERO: **77**

NOM: **titi**

SALAIRE: **1000**

DATE: **12/12/12**

Procédure PL/SQL terminée avec succès.

**AFFECTATION DES VARIABLES
A PARTIR D' UNE BD**

SCHEMA DE LA BASE D' EXEMPLES

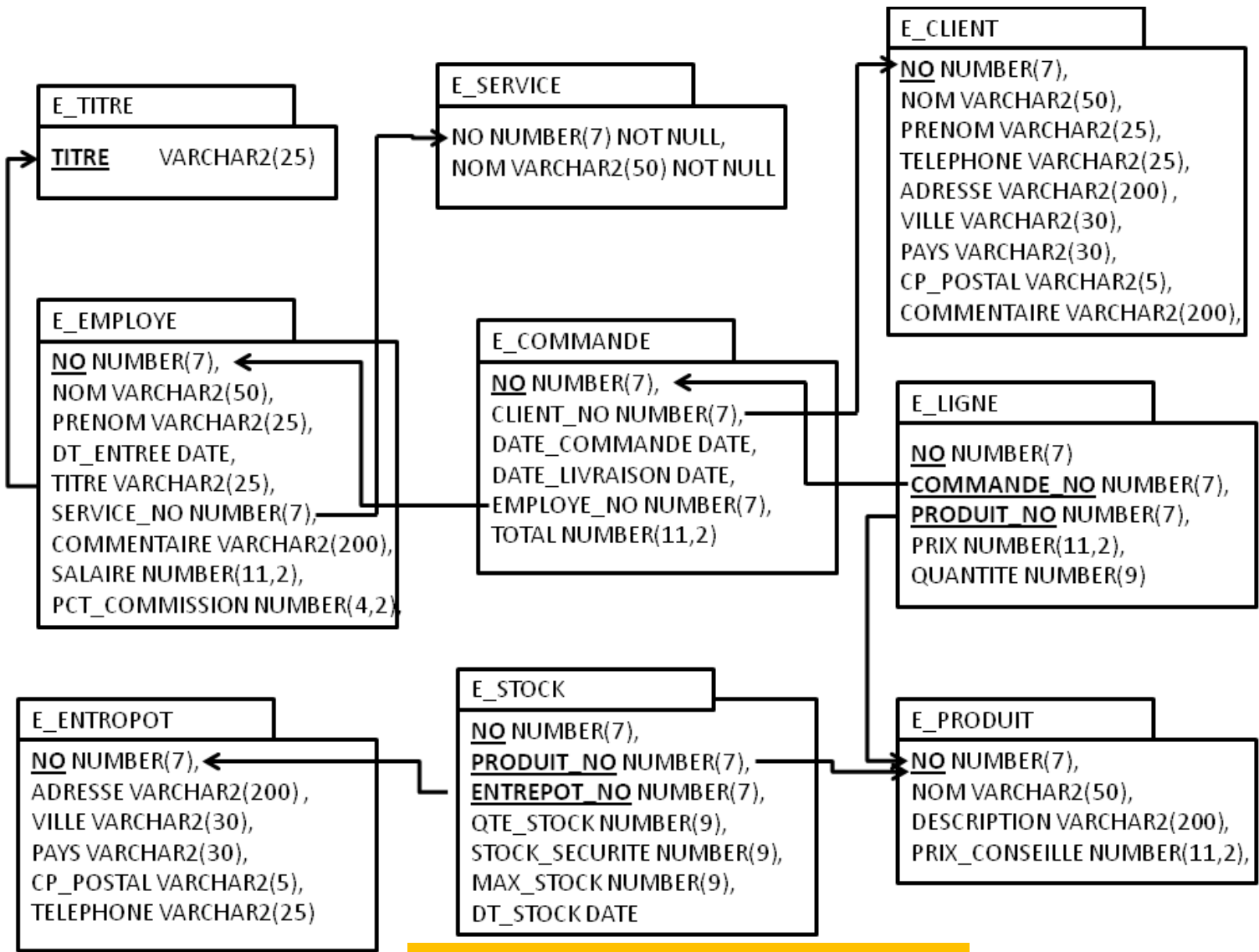


Schéma logique la BD exemples

```
SELECT <COLONNE_OU_TUPLE> INTO <VAR> FROM NOM_TABLE WHERE CONDITION;
```

Exemple 1:

```
SQL> SET SERVEROUTPUT ON
SQL> DECLARE
2      NOM_EMP  VARCHAR2(20);
3 BEGIN
4  SELECT NOM INTO NOM_EMP
5  FROM E_CLIENT
6  WHERE NO=1;
7  DBMS_OUTPUT.PUT_LINE ('Le nom du client NO 1 est ' || NOM_EMP);
8 END;
9 /
```

Le nom du client NO 1 est Idrissi

Procédure PL/SQL terminée avec succès.

Exemple 2:

```
SQL> DECLARE
2      NOM_EMP  E_CLIENT.NOM%TYPE;
3 BEGIN
4  SELECT NOM INTO NOM_EMP
5  FROM E_CLIENT
6  WHERE NO=1;
7  DBMS_OUTPUT.PUT_LINE ('Le nom du client NO 1 est ' || NOM_EMP);
8 END;
9 /
```

Le nom du client NO 1 est Idrissi

Procédure PL/SQL terminée avec succès.

Exemple 3:

```
SQL> DECLARE
2      NOM_EMP  VARCHAR2(20);
3      PRE_EMP  VARCHAR2(20);
4 BEGIN
5  SELECT NOM, PRENOM INTO NOM_EMP, PRE_EMP
6  FROM E_CLIENT
7  WHERE NO=1;
8  DBMS_OUTPUT.PUT_LINE ('Le nom du client NO 1 est ' || NOM_EMP);
9  DBMS_OUTPUT.PUT_LINE ('son prénom est ' || PRE_EMP);
10 END;
11 /
```

Le nom du client NO 1 est Idrissi
son prénom est Mohammed

Procédure PL/SQL terminée avec succès.

Exemple 4:

```
SQL> DECLARE
2      TYPE T_EMP IS RECORD (
3          NOM_EMP  VARCHAR2(20),
4          PRE_EMP  VARCHAR2(20)
5      );
6      EMP T_EMP;
7 BEGIN
8  SELECT NOM, PRENOM INTO EMP.NOM_EMP, EMP.PRE_EMP
9  FROM E_CLIENT
10 WHERE NO=1;
11 DBMS_OUTPUT.PUT_LINE ('Le nom du client NO 1 est ' || EMP.NOM_EMP);
12 DBMS_OUTPUT.PUT_LINE ('son prénom est ' || EMP.PRE_EMP);
13 END;
14 /
```

Le nom du client NO 1 est Idrissi
son prénom est Mohammed

Procédure PL/SQL terminée avec succès.

Exemple 5:

```
SQL> DECLARE
2      TYPE T_EMP IS RECORD (
3          NOM_EMP  VARCHAR2(20),
4          PRE_EMP  VARCHAR2(20)
5      );
6      EMP T_EMP;
7 BEGIN
8  SELECT NOM, PRENOM INTO EMP
9  FROM E_CLIENT
10 WHERE NO=1;
11 DBMS_OUTPUT.PUT_LINE ('Le nom du client NO 1 est ' || EMP.NOM_EMP);
12 DBMS_OUTPUT.PUT_LINE ('son prénom est ' || EMP.PRE_EMP);
13 END;
14 /
```

Le nom du client NO 1 est Idrissi
son prénom est Mohammed

Procédure PL/SQL terminée avec succès.

Exemple 6:

```
SQL> DECLARE
2      EMP E_CLIENT%ROWTYPE;
3 BEGIN
4  SELECT * INTO EMP
5  FROM E_CLIENT
6  WHERE NO=1;
7  DBMS_OUTPUT.PUT_LINE ('client NO 1 est :');
8  DBMS_OUTPUT.PUT_LINE ('NOM      || EMP.NOM);
9  DBMS_OUTPUT.PUT_LINE ('PRENOM   || EMP.PRENOM);
10 DBMS_OUTPUT.PUT_LINE ('TELEHPONE || EMP.TELEPHONE);
11 DBMS_OUTPUT.PUT_LINE ('ADRESSE   || EMP.ADRESSE);
12 DBMS_OUTPUT.PUT_LINE ('VILLE    || EMP.VILLE);
13 DBMS_OUTPUT.PUT_LINE ('PAYS      || EMP.PAYS);
14 DBMS_OUTPUT.PUT_LINE ('CP_POSTAL  || EMP.CP_POSTAL);
15 DBMS_OUTPUT.PUT_LINE ('COMMENTAIRE || EMP.COMMENTAIRE);
16 END;
17 /
```

client NO 1 est :	
NOM	Idrissi
PRENOM	Mohammed
TELEHPONE	O60000000
ADRESSE	Rue 1, N° 23
VILLE	Rabat
PAYS	Maroc
CP_POSTAL	5000
COMMENTAIRE	Pas de commentaire

Procédure PL/SQL terminée avec succès.

Exemple 7:

```
SQL> DECLARE
2      NOM_EMP  VARCHAR2(20);
3 BEGIN
4  SELECT NOM INTO NOM_EMP
5  FROM E_CLIENT
6  WHERE NO=99;
7 END;
8 /
```

```
DECLARE
*
```

ERREUR à la ligne 1 :
ORA-01403: aucune donnée trouvée
ORA-06512: à ligne 4

CURSEUR

SOLUTION

Exemple 8:

```
SQL> DECLARE
2      NOM_EMP  VARCHAR2(20);
3 BEGIN
4  SELECT NOM INTO NOM_EMP
5  FROM E_CLIENT
6  WHERE NO=1 OR NO=2;
7 END;
8 /
```

```
DECLARE
*
```

ERREUR à la ligne 1 :
ORA-01422: l'extraction exacte ramène plus que le
nombre de lignes demandé
ORA-06512: à ligne 4

STRUCTURES DE CONTRÔLE

Structures de contrôles PLSQL

IF - THEN - ELSE - END IF

IF condition THEN

```

    instruction1      ;
    instruction 2    ;
    .....
    instruction 2    ;

```

END IF;

PLSQL IF-THEN-END IF: Exemple 1

```

SQL> SET SERVEROUTPUT ON;
SQL> DECLARE
2  x integer := 10; y integer := 15;
3  BEGIN
4      IF x<y THEN
5          DBMS_OUTPUT.PUT_LINE (x || ' < ' || y);
6      END IF ;
7  END;
8  /
10 < 15

```

Procédure PL/SQL terminée avec succès.

```
IF condition1 THEN
    instruction1;
    instruction 2;
ELSE
    instruction3;
END IF;
```

PLSQL IF-THEN-ELSE-END IF: Exemple 2

```
SQL> SET SERVEROUTPUT ON;
SQL> DECLARE
2  x integer := 20; y integer := 15;
3  BEGIN
4      IF x<y THEN
5          DBMS_OUTPUT.PUT_LINE (x || ' < ' || y);
6      ELSE
7          DBMS_OUTPUT.PUT_LINE (x || ' >= ' || y);
8      END IF ;
9  END;
10 /
20 >= 15
```

Procédure PL/SQL terminée avec succès.

```
IF condition1 THEN
    instruction 1;
    instruction 2;
ELSIF condition2 THEN
    instruction 3;
    instruction 4;
ELSIF condition3 THEN
    instruction 5;
    instruction 6;
ELSE
    instruction 7;
END IF;
```

PLSQL IF-THEN-ELSIF-THEN.....END IF: Exemple 3

```
SQL> SET SERVEROUTPUT ON;
SQL> DECLARE
2  x integer := 20; y integer := 20;
3  BEGIN
4      IF x<Y THEN
5          DBMS_OUTPUT.PUT_LINE (x || ' < ' || y);
6      ELSIF x=Y THEN
7          DBMS_OUTPUT.PUT_LINE (x || ' = ' || y);
8      ELSIF x<Y THEN
9          DBMS_OUTPUT.PUT_LINE (x || ' < ' || y);
10     ELSE
11         DBMS_OUTPUT.PUT_LINE ('Bizzare!!');
12     END IF ;
13 END;
14 /
20 = 20
```

Procédure PL/SQL terminée avec succès.

Structures répétitives PLSQL

LOOP - EXIT WHEN - END LOOP

<<label>>

LOOP

instruction1;

instruction2;

EXIT [*label*][**WHEN** condition1]

END LOOP *label*;

- | | |
|--|--|
| • EXIT | force la sortie de la boucle sans conditions. |
| • EXIT WHEN | permet une sortie de boucle si la condition est vraie. |
| • EXIT <<label>> WHEN | permet une sortie d'une boucle nommée label si la condition est vraie. |
| • EXIT <<label>> | force une sortie de boucle nommée label. |

PLSQL LOOP - END LOOP : Exemple 1

```
SQL> SET SERVEROUTPUT ON;
SQL> DECLARE
  2  x integer := 0;
  3  BEGIN
  4  LOOP
  5      x := x + 1;
  6      DBMS_OUTPUT.PUT_LINE (x);
  7  EXIT WHEN x = 10;
  8  END LOOP;
  9  END;
10 /
```

1
2
3
4
5
6
7
8
9
10

Procédure PL/SQL terminée avec succès.

PLSQL LOOP - END LOOP : Exemple 2

```
SQL> SET SERVEROUTPUT ON;
SQL> DECLARE
  2   x integer := 0; y integer := 0;
  3 BEGIN
  4 LOOP
  5     x := x + 1; y := 0;
  6     <<label1>>
  7     LOOP y := y + 1;
  8     EXIT label1 WHEN y > 5;
  9     DBMS_OUTPUT.PUT_LINE (x || ' X ' || y || ' = ' || x*y);
 10     END LOOP label1;
 11 EXIT WHEN x = 2;
 12 END LOOP;
 13 END;
 14 /
1 X 1 = 1
1 X 2 = 2
1 X 3 = 3
1 X 4 = 4
1 X 5 = 5
2 X 1 = 2
2 X 2 = 4
2 X 3 = 6
2 X 4 = 8
2 X 5 = 10
```

Procédure PL/SQL terminée avec succès.

Structures répétitives PLSQL

WHILE - LOOP - END LOOP

```
WHILE conditions
LOOP
    instruction1;
    instruction2;
END LOOP;
```

PLSQL LOOP – WHILE LOOP-END LOOP: Exemple 1

```
SQL> SET SERVEROUTPUT ON;
SQL> DECLARE
2      cpt          INTEGER := 0;
3 BEGIN
4      WHILE cpt<10 LOOP
5          DBMS_OUTPUT.PUT_LINE ('Valeur suivante de X : ' || cpt);
6          cpt:=cpt+1;
7      END LOOP;
8 END;
9 /
```

Valeur suivante de X : 0

Valeur suivante de X : 1

Valeur suivante de X : 2

Valeur suivante de X : 3

Valeur suivante de X : 4

Valeur suivante de X : 5

Valeur suivante de X : 6

Valeur suivante de X : 7

Valeur suivante de X : 8

Valeur suivante de X : 9

Procédure PL/SQL terminée avec succès.

Structures répétitives PLSQL

FOR - IN - LOOP

FOR compteur **IN** [REVERSE] borne_inf..borne_sup **LOOP**

instruction1 ;

instruction2 ;

instruction3 ;

[EXIT WHEN condition];

END LOOP;

PLSQL FOR -IN-LOOP: Exemple 1

```
SQL> SET SERVEROUTPUT ON
SQL> BEGIN
 2  FOR i IN 1..5 LOOP
 3      DBMS_OUTPUT.PUT_LINE (i);
 4  END LOOP;
 5  END;
 6  /
```

1
2
3
4
5

Procédure PL/SQL terminée avec succès.

PLSQL FOR -IN-LOOP: Exemple 2

```
SQL> SET SERVEROUTPUT ON
SQL> BEGIN
  2  FOR i IN REVERSE 1..5 LOOP
  3      DBMS_OUTPUT.PUT_LINE (i);
  4  END LOOP;
  5  END;
  6  /
```

5
4
3
2
1

Procédure PL/SQL terminée avec succès.

SQL>

PLSQL FOR –IN-LOOP: Exemple 3

```
SQL>  
SQL> SET SERVEROUTPUT ON  
SQL> BEGIN  
  2  FOR i IN 1..5 LOOP  
  3      DBMS_OUTPUT.PUT_LINE (i);  
  4  EXIT WHEN i>3;  
  5  END LOOP;  
  6  END;  
  7  /
```

1
2
3
4

Procédure PL/SQL terminée avec succès.

SQL>

Structures de contrôles PLSQL

CASE - WHEN - THEN - ELSE - END CASE

CASE selecteur

WHEN expression1 **THEN** *instruction1*;

WHEN expression2 **THEN** *instruction2*;

...

WHEN expression3 **THEN** *instruction3*;

ELSE *instruction4*;

END CASE;

PLSQL CASE -WHEN -ELSE -END CASE: Exemple 1

```
SQL> SET SERVEROUTPUT ON;
```

```
SQL> DECLARE
```

```
2  x integer := 2;
```

```
3  BEGIN
```

```
4      CASE X
```

```
5      WHEN 1 THEN DBMS_OUTPUT.PUT_LINE ('Le premier');
```

```
6      WHEN 2 THEN DBMS_OUTPUT.PUT_LINE ('Le deuxième');
```

```
7      WHEN 3 THEN DBMS_OUTPUT.PUT_LINE ('Le troisième');
```

```
8      ELSE      DBMS_OUTPUT.PUT_LINE ('Le dernier');
```

```
9      END CASE;
```

```
10 END;
```

```
11 /
```

Le deuxième

Procédure PL/SQL terminée avec succès.

CASE selecteur

WHEN expression1 **THEN** *instruction1*;

WHEN expression2 **THEN** *instruction2*;

...

WHEN expression3 **THEN** *instruction3*;

ELSE *instruction4*;

END CASE;

PLSQL CASE -WHEN -ELSE -END CASE: Exemple 2

```
SQL> SET SERVEROUTPUT ON;
```

```
SQL> DECLARE
```

```
2  x integer := 2;
```

```
3  BEGIN
```

```
4      CASE
```

```
5          WHEN x=1 THEN DBMS_OUTPUT.PUT_LINE ('Le premier');
```

```
6          WHEN x=2 THEN DBMS_OUTPUT.PUT_LINE ('Le deuxième');
```

```
7          WHEN x=3 THEN DBMS_OUTPUT.PUT_LINE ('Le troisième');
```

```
8          ELSE      DBMS_OUTPUT.PUT_LINE ('Le dernier');
```

```
9      END CASE;
```

```
10 END;
```

```
11 /
```

Le deuxième

Procédure PL/SQL terminée avec succès.

TRAVAUX PRATIQUES

PL/SQL : séance 1

Objectifs:

manipuler les variables,
les bloc PL/SQL
et les structures de contrôle

On considère le schéma de la base de données précédente.

Sachant que les numéro des clients sont numérotés dans l'ordre (1,2,...):

Ecrire un bloc PL/SQL

1. Qui affiche le nom et la ville du client numéro 3;
2. qui affiche le numéro, le nom et la ville de chaque client;
3. qui affiche uniquement les client qui habitent rabat
4. qui vérifie si un client, dont le numéro est saisi au clavier, existe dans la liste des clients.
5. qui récupère le numéro d' un client du clavier et affiche ses informations

PL/SQL FOR ORACLE

Transactions & Curseurs

M. NASSAR & R. OULAD HAJ THAMI
ENSIAS - Rabat

SOMMAIRE GENERAL

MOTIVATIONS

STRUCTURE D'UN BLOC PL/SQL

LES VARIABLES

LES ENREGISTREMENTS

ASSIGNATION DES VARIABLES ET AFFECTATION

STRUCTURES DE CONTRÔLE

LES COLLECTIONS

LES TRANSACTIONS

INSERT-UPDATE-DELETE DANS UN BLOC PL/SQL

GESTION DES ERREURS ET DES EXCEPTIONS

LES CURSEURS

LES PROCEDURES ET LES FOCNTIONS STOCKEES

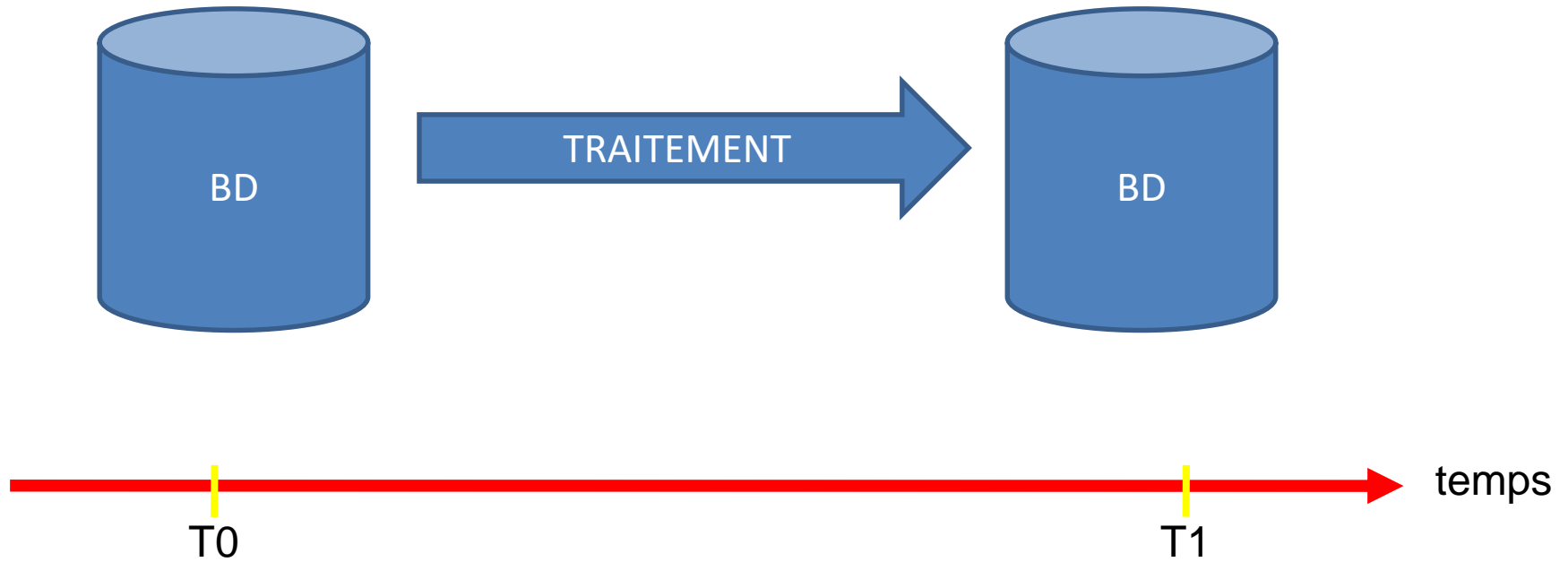
LES PACKAGES

LES TRIGGERS

**INSERT, UPDATE, DELETE
DANS UNE BD**

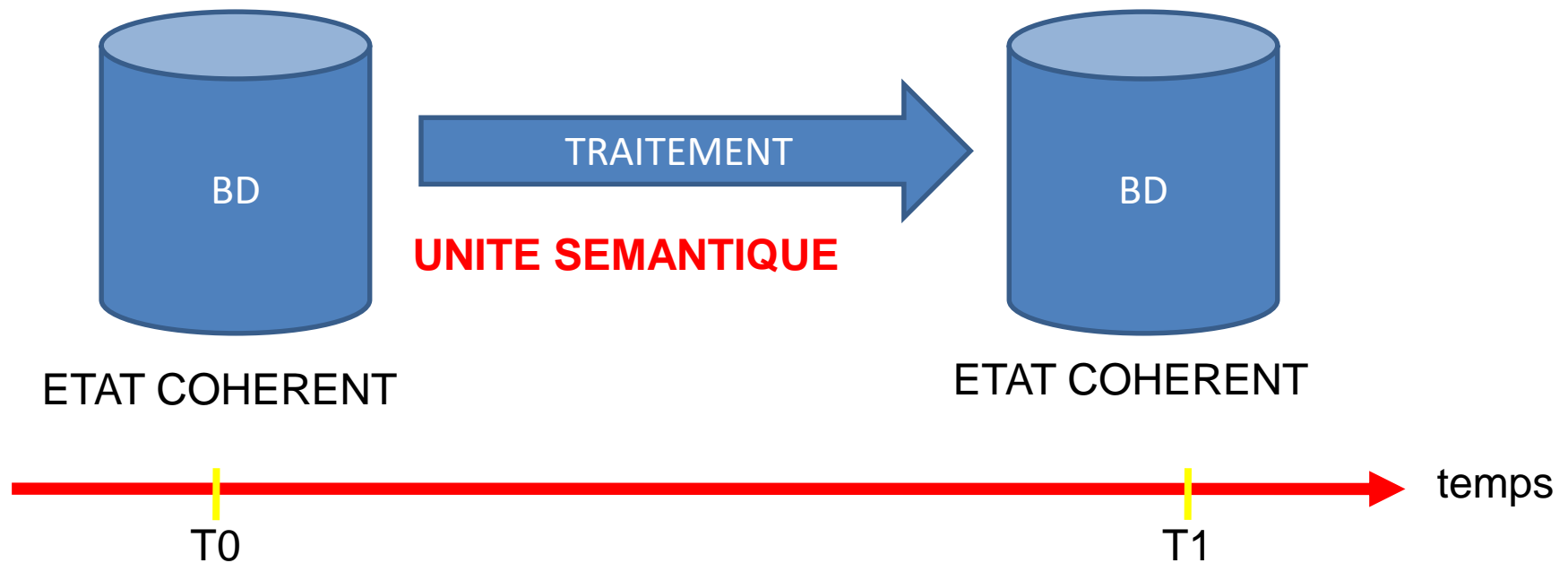
TRANSACTION

PROBLEME:



BASES DE DONNEES RELATIONNELLES

PROBLEME:



Exemple: Transfert d'argent entre 2 comptes:

```
UPDATE Compte  
SET Val = Val - 100  
Where NumCompte=1;
```

```
UPDATE Compte  
SET Val = Val + 100  
Where NumCompte=2;
```

Transaction: définition

Une transaction est une séquence d'actions <a11, a12, ..., a1ni>

Exemple:

DEBUT TRANSACTION

```
INSERT INTO compte_1 (...) VALUES (...);  
INSERT INTO compte_2 (...) VALUES (...);  
DELETE FROM comptabilité WHERE num=123;  
UPDATE .....
```

FIN TRANSACTION ;

Début de la transaction

Fin de la transaction

Transaction: Propriétés

Atomicité

Consistance

Isolation

Durabilité

Transaction: Propriétés

Atomicité

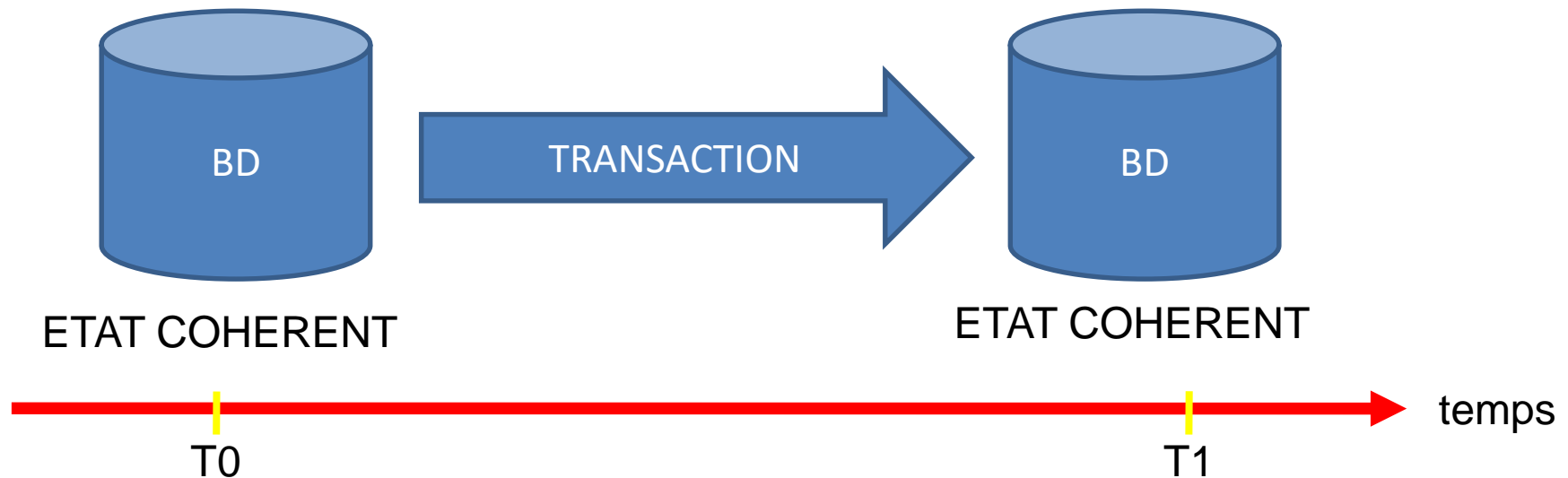
L'ensemble des opérations d'une transaction apparaît comme une seule opération atomique

Soit toutes les opérations sont validées ou toutes annulées (tout ou rien)

Transaction: Propriétés

Consistance

L'exécution de la transaction fait passer la base de données d'un état consistant à un autre état consistant



Transaction: Propriétés

Isolation

Chaque transaction est indépendante des autres transactions concurrentes.

Sérialisation des transactions.

Les résultats d'une transaction ne sont visibles aux autres transactions qu'une fois la transaction validée.

Les concurrences sont parfaitement contrôlées

Transaction: Propriétés

Durabilité

C'est la persistance des mises à jour d'une transaction validée.

Les effets d'une transaction validée sont durables et permanents, quelques soient les problèmes logiciels ou matériels, notamment après la fin de la transaction.

Transaction: primitives de gestion

BEGIN TRANSACTION
SET TRANSACTION
OUVERTURE DE
SESSION

DEBUT_DE_TRANSACTION

```
INSERT INTO compte_1 (...) VALUES (...);  
.....  
INSERT INTO compte_2 (...) VALUES (...);  
.....  
DELETE FROM comptabilité WHERE num=123;  
.....
```

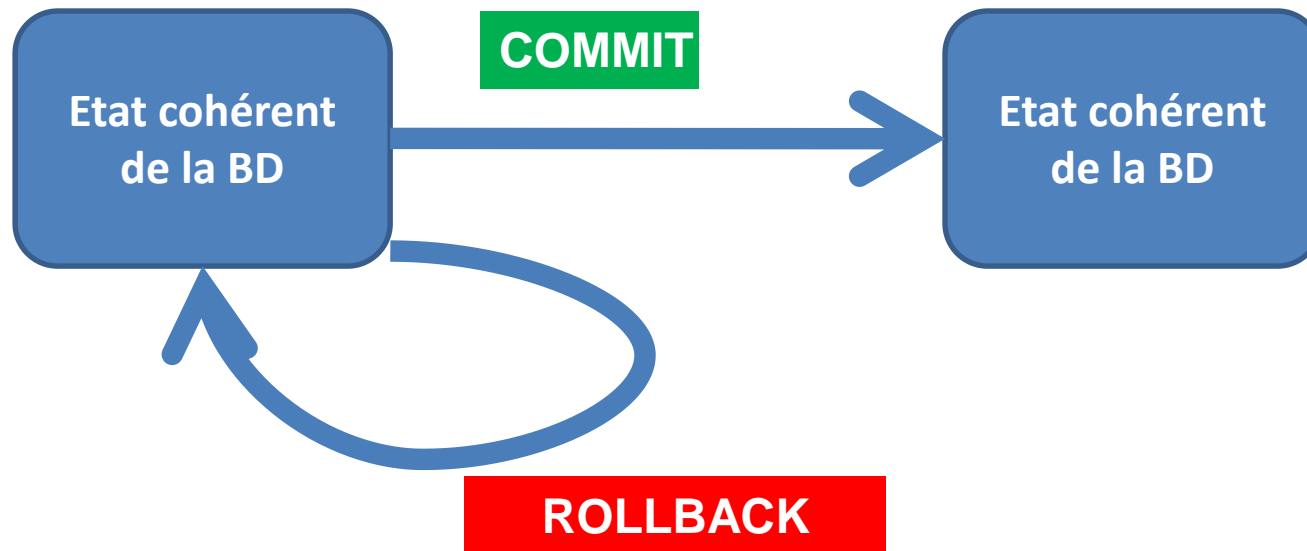
FIN_DE_TRANSACTION ;

COMMIT

- Validation de la transaction
- Rend effectives toutes les mises à jour de la transaction

ROLLBACK, ABORT

- Annulation de la transaction
- Défait toutes les mises à jour de la transaction



**TRANSACTION
SOUS
ORACLE**

BASES DE DONNEES RELATIONNELLES

SET TRANSACTION READ [ONLY | WRITE]

INSERT INTO compte_1 (...) VALUES (...);

INSERT INTO compte_2 (...) VALUES (...);

SAVEPOINT <NOM>

DELETE FROM comptabilité WHERE num=123;

UPDATE

SAVEPOINT <NOM>

DELETE FROM

[COMMIT | ROLLBACK] ;

Une transaction commence soit à la connexion ou en début de session, soit à la fin d'une transaction précédente annulée ou validée.

Le début d'une transaction dans Oracle peut être IMPLICITE (pas besoin de SET TRANSACTION)

COMMIT	VALIDE entièrement la transaction
--------	-----------------------------------

ROLLBACK	ANNULE entièrement la transaction
----------	-----------------------------------

Les savepoint sont des points de contrôle utilisés dans les transactions pour annuler partiellement l'une d'elles.

Il est possible de définir des Savepoint qui sont des points de contrôle utilisés dans une transaction ainsi on a la possibilité de faire des annulations partielles de transaction.

EXEMPLE 1 : COMMIT et ROOLLBACK

```
SQL> create table trans (  
2 id number(5) primary key,  
3 nom varchar2(20)  
4 );
```

Table créée.

```
SQL>  
SQL>  
SQL> desc trans;
```

Nom	NULL ?	Type

ID	NOT NULL	NUMBER(5)
NOM		VARCHAR2(20)


```
SQL> --transaction implicite dans oracle
SQL> insert into trans values (1, 'Rachid');
```

1 ligne créée.

```
SQL>
SQL>
SQL> select * from trans;
```

ID	NOM
1	Rachid

```
SQL>
SQL>
SQL> rollback;
```

Annulation (rollback) effectuée.

```
SQL>
SQL> select * from trans;
```

aucune ligne sélectionnée

```
SQL>  
SQL> insert into trans values (1, 'Rachid');
```

1 ligne créée.

```
SQL> commit;
```

Validation effectuée.

```
SQL>  
SQL>  
SQL> select * from trans;
```

ID	NOM
1	Rachid

```
SQL>
```

```
SQL> set transaction read only;
```

Transaction définie.

```
SQL>
```

```
SQL>
```

```
SQL> --test d'insertion
```

```
SQL>
```

```
SQL> insert into trans values (4, 'Said');  
insert into trans values (4, 'Said')
```

*

ERREUR à la ligne 1 :

ORA-01456: impossible d'exécuter l'opération insérer/supprimer/modifier dans une transaction READ ONLY

```
SQL>
```

```
SQL> update trans set nom='Filali' where id=1;  
update trans set nom='Filali' where id=1
```

*

ERREUR à la ligne 1 :

ORA-01456: impossible d'exécuter l'opération insérer/supprimer/modifier dans une transaction READ ONLY

```
SQL> set transaction read write;
```

Transaction définie.

```
SQL>
```

```
SQL> select * from trans;
```

ID	NOM
----	-----

1	Rachid
---	--------

```
SQL> update trans set nom='Mohammed' where id=1;
```

1 ligne mise à jour.

```
SQL> commit;
```

Validation effectuée.

```
SQL> select * from trans;
```

ID	NOM
----	-----

1	Mohammed
---	----------

EXEMPLE 2 : SAVEPOINT et ROLLBACK TO

```
SQL> --test savepoint
```

```
SQL>
```

```
SQL> select * from trans;
```

aucune ligne sélectionnée

```
SQL>
```

```
SQL> insert into trans values (1, 'Rachid');
```

1 ligne créée.

```
SQL>
```

```
SQL> select * from trans;
```

ID	NOM
1	Rachid

```
SQL>
```

```
SQL> savepoint P1;
```

Savepoint créé.

```
SQL>  
SQL> insert into trans values (2, 'Said');
```

1 ligne créée.

```
SQL>  
SQL> select * from trans;
```

ID	NOM
1	Rachid
2	Said

```
SQL>  
SQL> savepoint P2;
```

Savepoint créé.

```
SQL>
```

```
SQL>  
SQL> insert into trans values (3, 'Mohammed');
```

1 ligne créée.

```
SQL>  
SQL> select * from trans;
```

ID	NOM
----	-----

1	Rachid
---	--------

2	Said
---	------

3	Mohammed
---	----------

```
SQL>  
SQL> savepoint P3;
```

Savepoint créé.

BASES DE DONNEES RELATIONNELLES

SQL>

SQL> **rollback to P3;**

Annulation (rollback) effectuée.

SQL>

SQL> select * from trans;

ID NOM

1 Rachid

2 Said

3 Mohammed

SQL> **rollback to P2;**

Annulation (rollback) effectuée.

SQL>

SQL> select * from trans;

ID NOM

1 Rachid

2 Said


```
SQL> rollback to P1;
```

Annulation (rollback) effectuée.

```
SQL>
```

```
SQL> select * from trans;
```

ID	NOM
----	-----

1	Rachid
---	--------

```
SQL> rollback;
```

Annulation (rollback) effectuée.

```
SQL>
```

```
SQL>
```

```
SQL> select * from trans;
```

aucune ligne sélectionnée

EXEMPLE 3 : SAVEPOINT et ROLLBACK GLOBAL

SQL> --test savepoint et rollback et commit;
SQL> insert into trans values (1, 'Rachid');
1 ligne créée.

SQL> **savepoint P1;**
Savepoint créé.

SQL> insert into trans values (2, 'Said');
1 ligne créée.

SQL> **savepoint P2;**
Savepoint créé.

SQL> select * from trans;

ID	NOM
1	Rachid
2	Said

SQL> **rollback;**
Annulation (rollback) effectuée.

SQL> select * from trans;
aucune ligne sélectionnée

EXEMPLE 4 : SAVEPOINT et COMMIT GLOBAL

SQL> insert into trans values (1, 'Rachid');
1 ligne créée.

SQL> **savepoint P1;**
Savepoint créé.

SQL> insert into trans values (2, 'Said');
1 ligne créée.

SQL> **savepoint P2;**
Savepoint créé.

SQL> insert into trans values (3, 'Mohammed');
1 ligne créée.

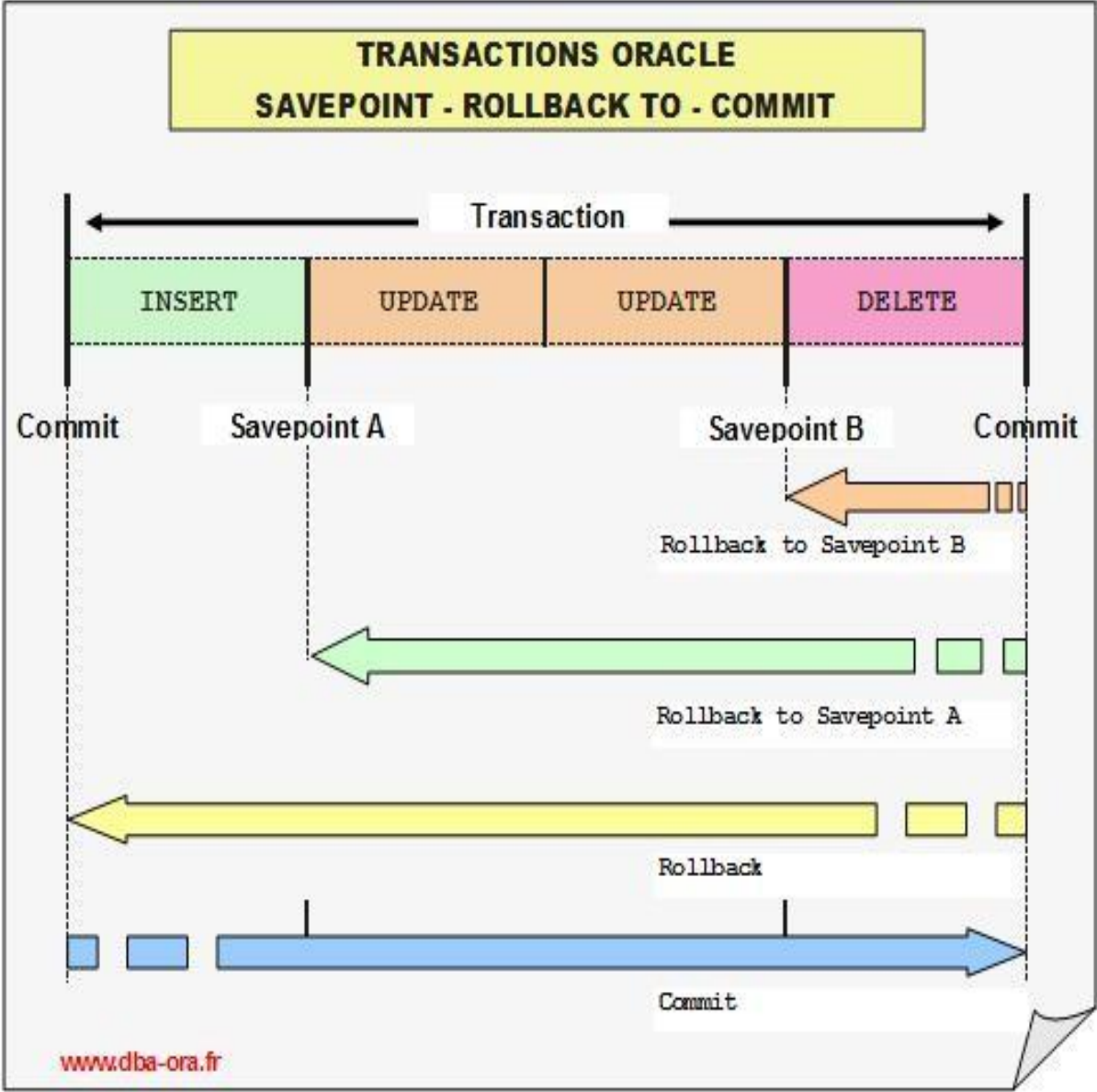
SQL> **commit;**
Validation effectuée.

SQL> select * from trans;

ID NOM

1 Rachid
2 Said
3 Mohammed

SYNTHESE



**TRAVAUX
PRATIQUES**

Partie 1: Transaction et ordre CREATE TABLE ET DROP TABLE

Pour réaliser cette première partie du TP vous devez avoir deux sessions différentes ouvertes sur la même base (connectez vous avec deux SQLPLUS avec le même compte)

1. créer la table "transa" comme présenté ci-après.

```
CREATE TABLE transa (  
    ID          NUMBER(5) PRIMARY KEY,  
    NOM         VARCHAR2(20)  
);
```

2. Considérons les ordres CREATE et DROP. La création et la suppression d'une table sont-elles transactionnelles ?

Pour vérifier cela, avec vos deux connexions, tentez de créer la table "transa" dans une transaction, et de vérifier dans l'autre session si vous la voyez .

3. Que constatez-vous ?

4. Exécutez la même tentative avec un DROP.

5. Conclusion ?

Partie 2 : atomicité d'une transaction courante

- 6. Insérez trois ou quatre lignes dans la table transa et les voir,;**
- 7. Modifiez une ligne, en supprimer une autre, enfin annuler les mises à jour venant d'être effectuées (en écrivant « ROLLBACK ; »).**
- 8. Vérifier le contenu de le contenu de la table et sa structure.**
- 9. Conclusion?**
- 10. Insérer à nouveau trois ou quatre lignes, les modifier et les détruire partiellement, puis valider (en écrivant « COMMIT ; ») ces mises à jour,**
- 11. Faites maintenant un ROLLBACK. Que s'est-il passé ?**
- 12. Maintenant détruire les données de votre table et valider.**
- 13. Insérer à nouveau dans votre table vide trois ou quatre lignes et clure la transaction par un EXIT .**
- 14. Reconnectez vous à SQLPLUS. Que s'est-il passé ? Expliquez**
- 15. Dans votre table, insérez à nouveau deux ou trois lignes dans la table et fermez brutalement votre session.**
- 16. Reconnectez vous à SQLPLUS. Les données saisies ont-elles été préservées ? Expliquez!**
- 17. Insérer à nouveau deux ou trois lignes dans la table, puis ajouter une nouvelle colonne à la table et essayer d'annuler les dernières insertions puis faites un DESC de la table. Conclusion.**
- 18. Videz votre table par un delete.**

BASES DE DONNEES RELATIONNELLES

Partie 2 : atomicité d'une transaction courante

19. Insérez trois ou quatre lignes dans la table transa et les voir,;
20. Insérez deux ou trois lignes puis faites une sauvegarde partielle de la transaction (SAVEPOINT)
21. Insérez deux ou trois lignes puis faites une sauvegarde partielle de la transaction (SAVEPOINT)
22. Faites un ROLLBACK puis vérifiez le contenu votre table. Conclusion?
23. Insérez deux ou trois lignes puis faites une sauvegarde partielle de la transaction (SAVEPOINT)
24. Insérez deux ou trois lignes puis faites une sauvegarde partielle de la transaction (SAVEPOINT)
25. Faites un COMMIT puis vérifiez le contenu votre table. Conclusion?
26. Videz votre table par un delete.
27. Insérez deux ou trois lignes puis faites une sauvegarde partielle de la transaction (SAVEPOINT)
28. Insérez deux ou trois lignes puis faites une sauvegarde partielle de la transaction (SAVEPOINT)
29. Faites une annulation partielle de la transaction (ROLLBACK TO ...)
30. Vérifier le contenu de la table par un SELECT.
31. Faites un COMMIT. Et vérifiez le contenu de la table. Conclusion?

Partie 3 : transaction et ordre DDL

32. Videz votre table par un delete.
33. Insérez deux ou trois lignes.
34. Faites un ALTER TABLE (par exemple, modification du schéma logique en général).
35. Faites un DESC de la table
36. Vérifiez le contenu de la table. Que constatez-vous? Conclusion??

**LA SUITE
MISE EN ŒUVRE
PL/SQL \leftrightarrow BD**

**SCHEMA DE LA BASE
D'EXEMPLES**

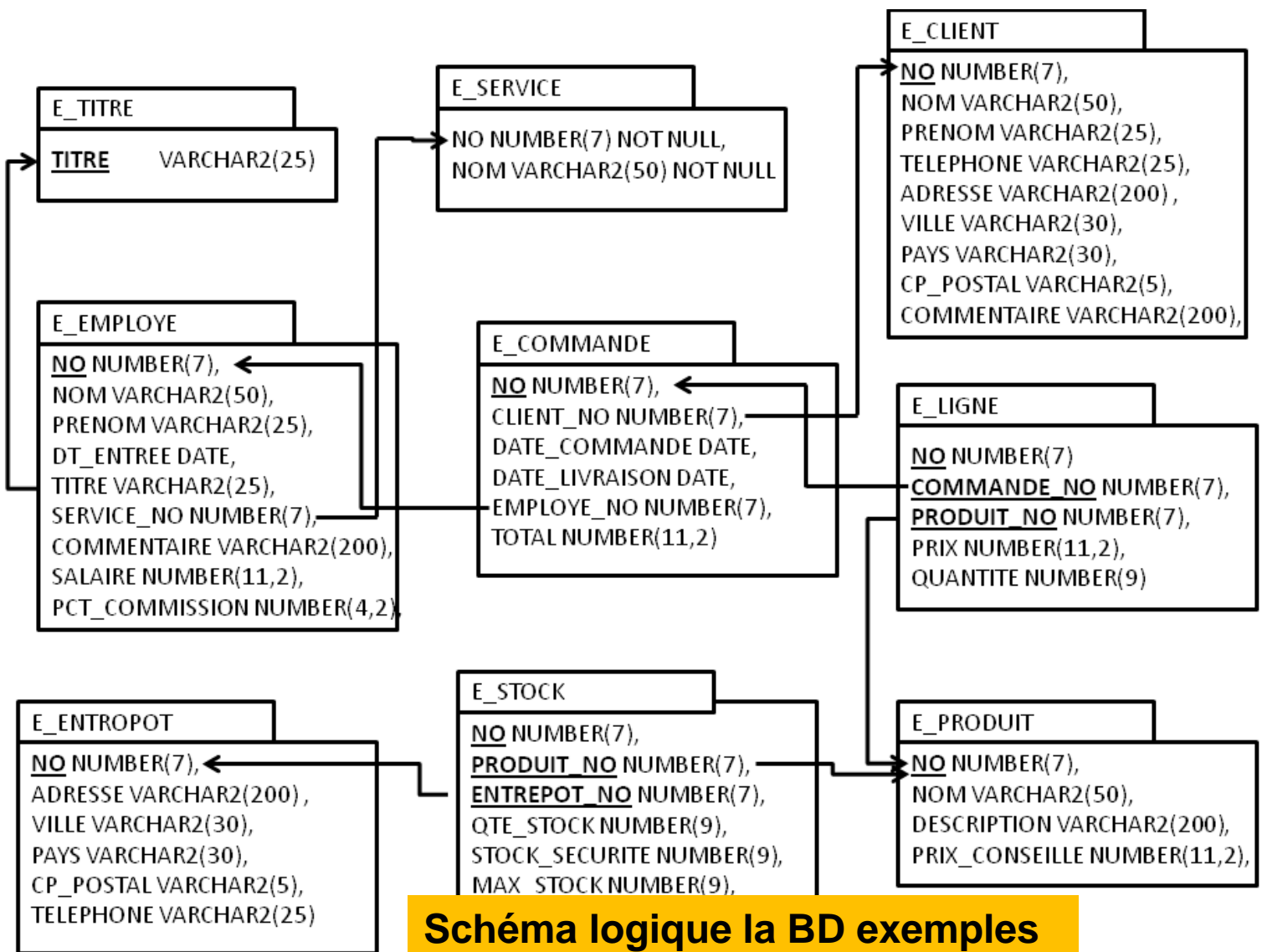


Schéma logique la BD exemples

INSERT

Même syntaxe que SQL

```
INSERT INTO NOM_TABLE [(col1, col2, ...,coln)] VALUES (val1, val2, ..., valn);
```

Exemple 1:

```
SQL> DECLARE
2
3 BEGIN
4     INSERT INTO E_CLIENT (NO, NOM, PRENOM) VALUES(99,'Filali','Said');
5     COMMIT;
6 END;
7 /
```

Procédure PL/SQL terminée avec succès.

Exemple 2:

```
DECLARE
    ID                E_CLIENT.NO%TYPE;
    NOM_EMP           E_CLIENT.NOM%TYPE;
    PRE_EMP           E_CLIENT.PRENOM%TYPE;
BEGIN
    ID:=99; NOM_EMP:='Filali'; PRE_EMP:='Said';
    INSERT INTO E_CLIENT (NO, NOM, PRENOM) VALUES(ID,NOM_EMP, PRE_EMP);
    COMMIT;
END;
```

```
CREATE SEQUENCE NOM_SEQ INCREMENT BY PAS_INC START WITH VAL_DEPART  
MAXVALUE VAL_MAX;
```

Exemple 3

```
SQL>  
SQL> CREATE SEQUENCE SEQ_NO_CL INCREMENT BY 1 START WITH 200 MAXVALUE  
99999;
```

Séquence créée.

```
SQL>  
SQL> DECLARE  
2  
3 BEGIN  
4     INSERT INTO E_CLIENT (NO, NOM, PRENOM)  
VALUES(SEQ_NO_CL.NEXTVAL,'Filali','Said');  
5     COMMIT;  
6 END;  
7 /
```

Procédure PL/SQL terminée avec succès.

UPDATE

BASES DE DONNEES RELATIONNELLES

Même syntaxe que SQL: UPDATE NOM_TABLE SET WHERE

```
SQL> DECLARE
2      VILLE_AVM          E_CLIENT.VILLE%TYPE;
3      VILLE_APM          E_CLIENT.VILLE%TYPE;
4 BEGIN
5      SELECT VILLE INTO VILLE_AVM FROM E_CLIENT WHERE NO=1;
6      DBMS_OUTPUT.PUT_LINE ('LA VILLE AVANT MODIFICATION EST :      '|| VILLE_AVM );
7
8      UPDATE E_CLIENT
9      SET VILLE='tantan'
10     WHERE NO=1;
11
12     COMMIT;
13
14     SELECT VILLE INTO VILLE_APM FROM E_CLIENT WHERE NO=1;
15     DBMS_OUTPUT.PUT_LINE ('LA VILLE APRES MODIFICATION EST :      '|| VILLE_APM );
16 END;
17 /
LA VILLE AVANT MODIFICATION EST :      Rabat
LA VILLE APRES MODIFICATION EST :      tantan
```

Procédure PL/SQL terminée avec succès.

DELETE

Même syntaxe que SQL: DELETE NOM_TABLE WHERE

```
SQL>  
SQL> DECLARE  
2 BEGIN  
3     DELETE E_CLIENT WHERE NO=9;  
4  
5     COMMIT;  
6 END;  
7 /
```

Procédure PL/SQL terminée avec succès.

```
SQL>
```

TRAITEMENT DE PLUSIEURS TUPELS

Exemple 7:

```
SQL> DECLARE
2     NOM_EMP    VARCHAR2(20);
3 BEGIN
4     SELECT NOM INTO NOM_EMP
5     FROM E_CLIENT
6     WHERE NO=99;
7 END;
8 /
```

DECLARE

*

ERREUR à la ligne 1 :
ORA-01403: aucune donnée trouvée
ORA-06512: à ligne 4

CURSEUR

SOLUTION

Exemple 8:

```
SQL> DECLARE
2     NOM_EMP    VARCHAR2(20);
3 BEGIN
4     SELECT NOM INTO NOM_EMP
5     FROM E_CLIENT
6     WHERE NO=1 OR NO=2;
7 END;
8 /
```

DECLARE

*

ERREUR à la ligne 1 :
ORA-01422: l'extraction exacte ramène plus que
le nombre de lignes demandé
ORA-06512: à ligne 4

LES CURSEURS

Un curseur est une zone mémoire de taille fixe, utilisée par le moteur SQL pour analyser et interpréter un ordre SQL

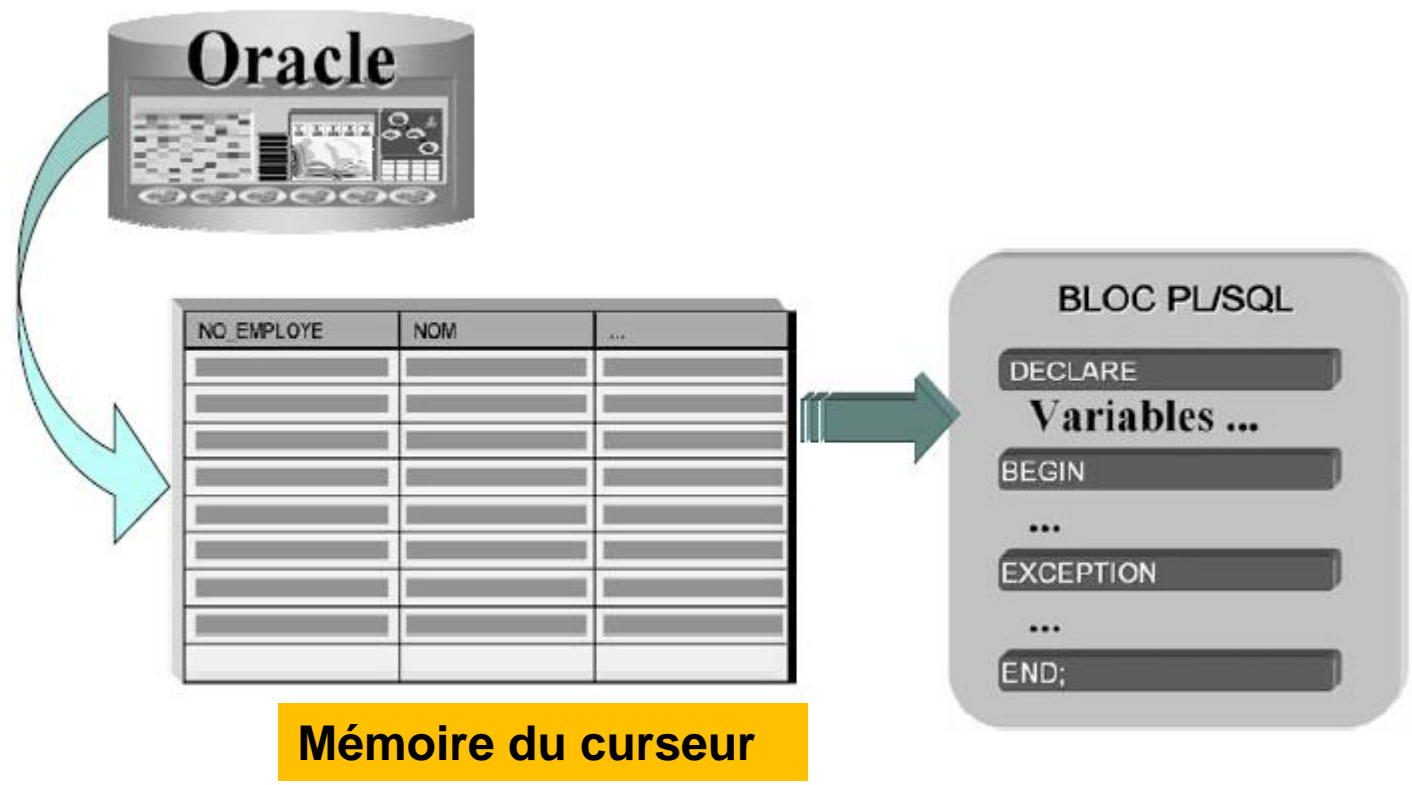
Curseur implicite: dans un ordre SQL et géré par le compilateur PL/SQL

Un curseur explicite est géré par l'utilisateur pour traiter un ordre Select qui ramène plusieurs tuples

Remarque

EN TERME D'EXECUTION, LES CURSEURS IMPLICITES SONT PLUS RAPIDES QUE LES CURSEURS EXPLICITES

**LES
CURSEURS IMPLICITES**




Curseur implicite: Exemple 1:

SQL> SELECT * FROM E_COMMANDE;

NO	CLIENT_NO	DATE_COM	DATE_LIV	EMPLOYE_NO	TOTAL
1	1	11/11/01	12/11/01	1	50000
2	1	11/12/01	12/02/02	2	60000
3	2	10/11/01	12/11/01	1	40500
4	3	11/11/02	12/11/02	1	50030
5	1	12/11/03	01/01/04	2	71000
6	4	15/11/03	19/11/03	3	80100
7	6	11/11/03	01/01/04	4	410000
8	6	02/01/04	12/01/04	1	55000
9	8	10/01/04	15/01/04	5	69000
10	13	11/01/04	16/01/04	1	150800

10 ligne(s) sélectionnée(s).



1	1	11/11/01	12/11/01	1	50000
2	1	11/12/01	12/02/02	2	60000
5	1	12/11/03	01/01/04	2	71000

CURSOR IMPLICITE

SQL> DECLARE

2 SOMME E_COMMANDE.TOTAL%TYPE;

3 BEGIN

4 SELECT SUM(TOTAL) INTO SOMME FROM E_COMMANDE WHERE CLIENT_NO=1;

5 DBMS_OUTPUT.PUT_LINE('la somme est : '||SOMME);

6 END;

7 /

la somme est : 181000

Procédure PL/SQL terminée avec succès.

Curseur implicite: Exemple 2:

```
SQL> DECLARE
2      NBR_LIGNE_DELETE      NUMBER(3);
3 BEGIN
4      DELETE E_COMMANDE WHERE CLIENT_NO=1;
5      NBR_LIGNE_DELETE:=SQL%ROWCOUNT;
6      DBMS_OUTPUT.PUT_LINE('Nombre de ligne détruites : '||NBR_LIGNE_DELETE);
7 END;
8 /
```

Nombre de ligne détruites : 3

Procédure PL/SQL terminée avec succès.

Curseur implicite: Exemple 3:

```
SQL> DECLARE
2          NBR_LIGNE          NUMBER(3):=0;
3          SOMME              E_COMMANDE.TOTAL%TYPE:=0;
4 BEGIN
5     FOR LIGNE IN (SELECT * FROM E_COMMANDE WHERE CLIENT_NO=1) LOOP
6         DBMS_OUTPUT.PUT_LINE('NUM COMMANDE: '|| LIGNE.NO||' NUM CLIENT: '||LIGNE.CLIENT_NO);
7         SOMME:=SOMME+LIGNE.TOTAL;
8         NBR_LIGNE := NBR_LIGNE +1;
9     END LOOP;
10        DBMS_OUTPUT.PUT_LINE('LA somme est : '||SOMME);
11        DBMS_OUTPUT.PUT_LINE('Nombre de lignes est : '||NBR_LIGNE);
12 END;
13 /
```

Exécution :

```
NUM COMMANDE: 1  NUM CLIENT: 1
NUM COMMANDE: 2  NUM CLIENT: 1
NUM COMMANDE: 5  NUM CLIENT: 1
LA somme est : 181000
Nombre de lignes est : 3
```

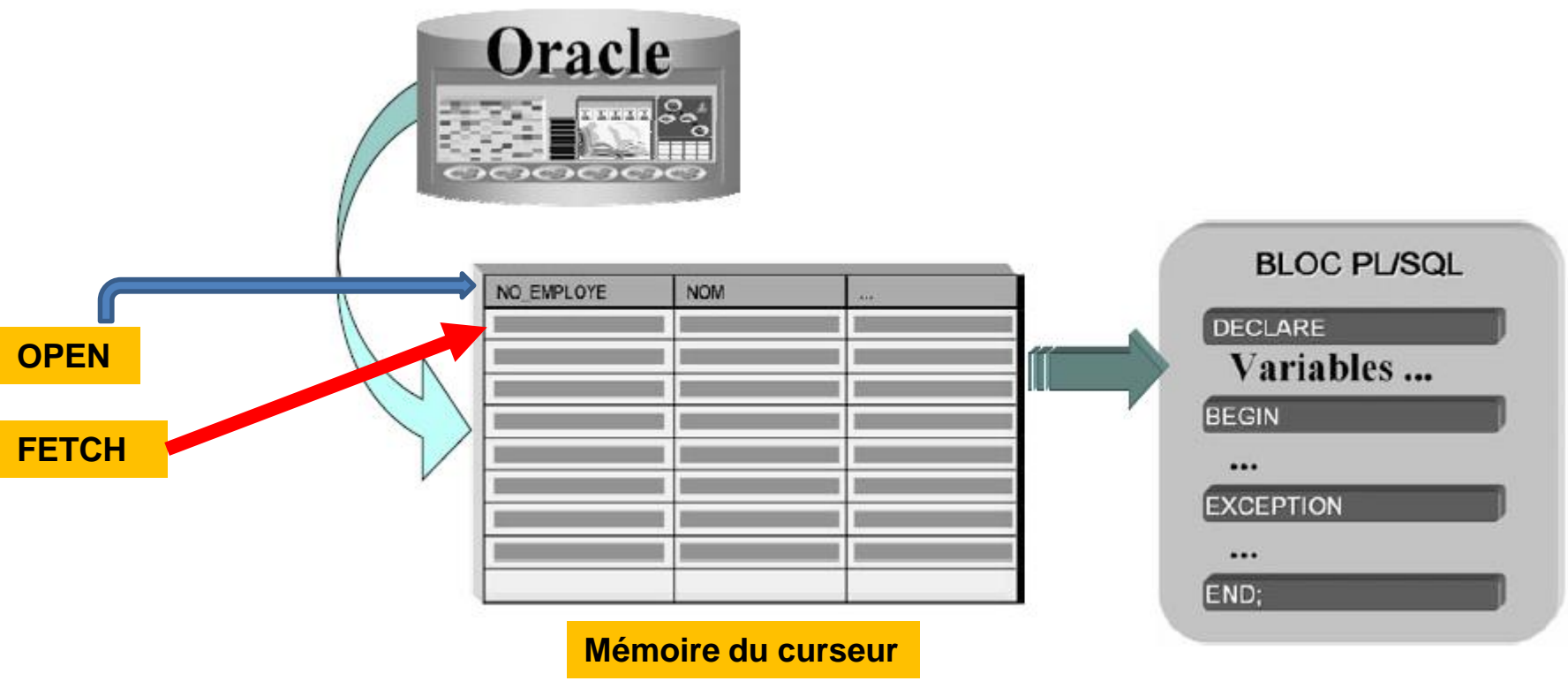
Procédure PL/SQL terminée avec succès.

**LES
CURSEURS EXPLICITES**

CURSOR nom_curseur [(parametres)] **IS** requête_select;

OPEN nom_curseur	OUVRE LE CURSEUR. Aucune EXCEPTION si la requête ne ramène aucune ligne.
FETCH nom_curseur INTO listes_variables nom_Record	Positionnement sur la ligne suivante et chargement des valeurs dans les variables INTO ou record
CLOSE nom_curseur	Fermeture du curseur
Nom_curseur% ISOPEN	Retourne TRUE si le curseur est ouvert sinon FALSE
Nom_curseur% NOTFOUND	Retourne FALSE si le dernier FETCH n'a pas renvoyé de ligne
Nom_curseur% FOUND	Retourne TRUE si le dernier FETCH a renvoyé une ligne
Nom_curseur% ROWCOUNT	Retourne le nombre totale de lignes traités jusqu'à maintenant

FONCTIONS ASSOCIEES AU CURSEUR



Curseur explicite: Exemple 4:

```
SQL> DECLARE
2  CURSOR CLIENT_RABAT IS
3  SELECT NO,NOM FROM E_CLIENT
4  WHERE VILLE='Rabat';
5
6  NOMCL E_CLIENT.NOM%TYPE;
7  NOCL E_CLIENT.NO%TYPE;
8
9  BEGIN
10 OPEN CLIENT_RABAT;
11
12 DBMS_OUTPUT.PUT_LINE('Nbr lignes traitées : '|| CLIENT_RABAT%ROWCOUNT);
13 FETCH CLIENT_RABAT INTO NOCL, NOMCL;
14 WHILE (CLIENT_RABAT%FOUND) LOOP
15     DBMS_OUTPUT.PUT_LINE('Client no : '|| NOCL ||' de nom : ' || NOMCL);
16     DBMS_OUTPUT.PUT_LINE('Nbr lignes traitées : '|| CLIENT_RABAT%ROWCOUNT);
17     FETCH  CLIENT_RABAT INTO NOCL, NOMCL;
18 END LOOP;
19
20 CLOSE CLIENT_RABAT;
21 END;
22 /
```

Exécution :

Nbr lignes traitées : 0

Client no : 1 de nom : Idrissi

Nbr lignes traitées : 1

Client no : 2 de nom : Soufiani

Nbr lignes traitées : 2

Client no : 6 de nom : Doukkali

Nbr lignes traitées : 3

Client no : 8 de nom : Zahraoui

Nbr lignes traitées : 4

Client no : 11 de nom : Meknassi

Nbr lignes traitées : 5

Procédure PL/SQL terminée avec succès.

Curseur explicite: Exemple 5:

```
SQL> DECLARE
2  CURSOR CLIENT_RABAT IS
3  SELECT NO,NOM FROM E_CLIENT
4  WHERE VILLE='Rabat';
5
6  CL_RB CLIENT_RABAT%ROWTYPE;
7
8  BEGIN
9  OPEN CLIENT_RABAT;
10
11  FETCH CLIENT_RABAT INTO CL_RB;
12  WHILE (CLIENT_RABAT%FOUND) LOOP
13      DBMS_OUTPUT.PUT_LINE('Client no : ' || CL_RB.NO || ' de nom : ' || CL_RB.NOM);
14      FETCH      CLIENT_RABAT INTO CL_RB;
15  END LOOP;
16
17  CLOSE CLIENT_RABAT;
18  END;
19 /
```

Exécution :

Client no : 1 de nom : Idrissi
Client no : 2 de nom : Soufiani
Client no : 6 de nom : Doukkali
Client no : 8 de nom : Zahraoui
Client no : 11 de nom : Meknassi

Procédure PL/SQL terminée avec succès.

Curseur explicite: Exemple 6: BOUCLE FOR

```
SQL>
SQL> DECLARE
2  CURSOR CLIENT_RABAT IS
3  SELECT NO,NOM FROM E_CLIENT
4  WHERE VILLE='Rabat';
5
6 BEGIN
7  FOR CL_RB IN CLIENT_RABAT LOOP
8      DBMS_OUTPUT.PUT_LINE('Client no : ' || CL_RB.NO || ' de nom : ' || CL_RB.NOM);
9  END LOOP;
10 END;
11 /
```

Exécution :

Client no : 1 de nom : Idrissi
Client no : 2 de nom : Soufiani
Client no : 6 de nom : Doukkali
Client no : 8 de nom : Zahraoui
Client no : 11 de nom : Meknassi

Procédure PL/SQL terminée avec succès.

•PAS DE OPEN: OUVERTURE IMPLICITE
•PAS DE CLOSE: FERMETURE IMPLICITE
•PAS DE FETCH: PARCOURS IMPLICITE
•PAS DE DECLARATION DE TYPE DE LA
VARIABLE DE RETOUR:
IMPLICITE CURSOR%ROWTYPE

Curseur explicite: Exemple 7: PARAMETRE DE CURSOR

```
SQL> DECLARE
 2  CURSOR CLIENT_PAR_VILLE ( NOM_VILLE IN E_CLIENT.VILLE%TYPE) IS
 3  SELECT NO,NOM FROM E_CLIENT
 4  WHERE VILLE=NOM_VILLE;
 5
 6  NV E_CLIENT.VILLE%TYPE;
 7
 8  BEGIN
 9  NV:='Rabat';
10  DBMS_OUTPUT.PUT_LINE('Les clients de '|| NV ||' sont : ');
11  FOR CL_RV IN CLIENT_PAR_VILLE(NV) LOOP
12      DBMS_OUTPUT.PUT_LINE('Client no : '|| CL_RV.NO ||' de nom : ' || CL_RV.NOM);
13  END LOOP;
14  NV:='Casa';
15  DBMS_OUTPUT.PUT_LINE('Les clients de '|| NV ||' sont : ');
16  FOR CL_RV IN CLIENT_PAR_VILLE(NV) LOOP
17      DBMS_OUTPUT.PUT_LINE('Client no : '|| CL_RV.NO ||' de nom : ' || CL_RV.NOM);
18  END LOOP;
19  END;
20 /
```

Exécution :

Les clients de Rabat sont :

Client no : 1 de nom : Idrissi

Client no : 2 de nom : Soufiani

Client no : 6 de nom : Doukkali

Client no : 8 de nom : Zahraoui

Client no : 11 de nom : Meknassi

Les clients de Casa sont :

Client no : 3 de nom : Miliani

Client no : 4 de nom : Zamouri

Client no : 7 de nom : Idrissi

Procédure PL/SQL terminée avec succès.

Curseur explicite: Exemple 7_1: PARAMETRE DE CURSOR

```
SQL>
SQL> DECLARE
 2  CURSOR COMMANDE_DATE_LIV ( DTD IN DATE, DTF IN DATE) IS
 3  SELECT NO,DATE_COMMANDE, DATE_LIVRAISON FROM E_COMMANDE
 4  WHERE DATE_LIVRAISON BETWEEN DTD AND DTF;
 5
 6 BEGIN
 7  DBMS_OUTPUT.PUT_LINE('Les commandes à livrer entre 01/01/04 et le 15/01/04 sont :');
 8  FOR COM IN COMMANDE_DATE_LIV('01/01/04','15/01/04') LOOP
 9      DBMS_OUTPUT.PUT_LINE('Commande no : '||COM.NO|| ' à livrer entre
      '||COM.DATE_COMMANDE ||' et ' || COM.DATE_LIVRAISON);
10  END LOOP;
11 END;
12 /
```

Exécution :

Les commandes à livrer entre 01/01/04 et le 15/01/04 sont :

Commande no : 5 à livrer entre 12/11/03 et 01/01/04

Commande no : 7 à livrer entre 11/11/03 et 01/01/04

Commande no : 8 à livrer entre 02/01/04 et 12/01/04

Commande no : 9 à livrer entre 10/01/04 et 15/01/04

Procédure PL/SQL terminée avec succès.

**LES
CURSEURS EXPLICITES
Et
LA MISE A JOUR DE LA BASE
(UPDATE, DELETE)**

CURSOR nom_curseur [(parametres)] [RETURN ROWTYPE] IS requête_select
FOR UPDATE;

OBJECTIF:

Verrouiller les tuples du curseur à MODIFIER ou à SUPPRIMER dans la table

CONDITION:

- PAS DE DISTINCT DANS LA REQUETE DU CURSEUR
- PAS DE GROUP BY DANS LA REQUETE DU CURSEUR
- PAS DE UNION, INTERSECT ou MINUS DANS LA REQUETE DU CURSEUR
- PAS DE FONCTION D'AGREGATION DANS LA REQUETE DU CURSEUR

PRECAUTION:

UN COMMIT A LA FIN, SINON LES MODIFICATION ET LES SUPPRESSION
SERONT ANNULEES

BASES DE DONNEES RELATIONNELLES

Curseur explicite: Exemple 8: CURSOR...FOR UPDATE

```
SQL> DECLARE
2  CURSOR CLIENT_RABAT IS
3  SELECT NO,NOM, VILLE FROM E_CLIENT
4  WHERE VILLE='Rabat'
5  FOR UPDATE;
6
7  NC_TANTAN    NUMBER(3);
8 BEGIN
9  DBMS_OUTPUT.PUT_LINE('Les clients avant modification: ');
10 FOR CLR IN CLIENT_RABAT LOOP
11    DBMS_OUTPUT.PUT_LINE('Client no : '|| CLR.NO ||' de nom : ' || CLR.NOM || ' et de ville : '|| CLR.VILLE);
12 END LOOP;
13
14 SELECT COUNT(*) INTO NC_TANTAN FROM E_CLIENT WHERE VILLE='TANTAN';
15 DBMS_OUTPUT.PUT_LINE('Les clients habitant TANTAN sont en nombre : '|| NC_TANTAN);
16 --modification de la ville Rabat à TANTAN
17
18 FOR CLR IN CLIENT_RABAT LOOP
19    UPDATE E_CLIENT SET VILLE='TANTAN' WHERE CURRENT OF CLIENT_RABAT;
20 END LOOP;
21
22 COMMIT;
23
24 DBMS_OUTPUT.PUT_LINE('Les clients APRES modification : ');
25 FOR CLR IN (SELECT NO, NOM, VILLE FROM E_CLIENT WHERE VILLE='TANTAN') LOOP
26    DBMS_OUTPUT.PUT_LINE('Client no : '|| CLR.NO ||' de nom : ' || CLR.NOM || ' et de ville : '|| CLR.VILLE);
27 END LOOP;
28 END;
29 /
```

Exécution :

Les clients avant modification :

Client no : 1 de nom : Idrissi et de ville : Rabat
Client no : 2 de nom : Soufiani et de ville : Rabat
Client no : 6 de nom : Doukkali et de ville : Rabat
Client no : 8 de nom : Zahraoui et de ville : Rabat
Client no : 11 de nom : Meknassi et de ville : Rabat

Les clients habitant TANTAN sont en nombre : 0

Les clients APRES modification :

Client no : 1 de nom : Idrissi et de ville : TANTAN
Client no : 2 de nom : Soufiani et de ville : TANTAN
Client no : 6 de nom : Doukkali et de ville : TANTAN
Client no : 8 de nom : Zahraoui et de ville : TANTAN
Client no : 11 de nom : Meknassi et de ville : TANTAN

Procédure PL/SQL terminée avec succès.

Travaux pratiques

Question 1:

Créer une table E_Augmentation comprenant les champs suivants :

No Number(7), Augmentation Number(11,2), Date_Augmentation Date, Emp_No Number(7)

Question 2: (Curseur, Instructions OPEN, FETCH, CLOSE, %FOUND)

Ecrire un programme permettant :

- **la mise à jour du salaire de tous les employés de la table E_Employe selon les conditions suivantes:**

- si son année d'entrée dans la société est 1995, augmenter le salaire de 50%.**

- si son année d'entrée dans la société est 1996, augmenter le salaire de 25%.**

- si son année d'entrée dans la société est 1997, augmenter le salaire de 10%.**

- **l'insertion des modifications dans la table E_Augmentation des informations suivantes:**

- le montant d'augmentation, la date d'augmentation, le numéro de l'employé, ainsi que le champ No qui sert d'identifiant de ligne pour la table.**

Question 3: (Utilisation d'un curseur avec la boucle FOR...LOOP)

Créer une table E_Resultat :

E_Resultat (No Number(2), LB_Resultat Varchar2(60), VL_Resultat Number(11,2))

Ecrire un programme permettant de faire le total des commandes gérées par chaque employé.

Pour les employés qui ont géré des commandes, faire les insertions dans la table E_Resultat selon le schéma suivant :

-No --> <<No-programme>>

-LB_Resultat --> <<Nom_Employé>> totalise

-VL_Resultat --> <<Total des commandes gérées>>

Question 4: (Utilisation d'un curseur paramétré)

Ecrire un programme qui insère dans la table E_Resultat le nom de l'employé et son salaire pour les employés dont le salaire vérifie les conditions suivantes :

• Si le salaire >3500, insérer dans la table E_Resultat les données suivantes :

-No --> <<No-programme>>

-LB_Resultat --> <<Variable_Nom_Employé>> a un salaire > 3500

-VL_Resultat --> <<Variable_Salaire_Employé>>

• Si le salaire >4500, insérer dans la table E_Resultat les données suivantes :

-No --> <<No-programme>>

-LB_Resultat --> <<Variable_Nom_Employé>> a un salaire > 4500

-VL_Resultat --> <<Variable_Salaire_Employé>>

Question 5 (Curseur et clause CURRENT OF)

Ecrire un programme permettant :

- d'abaisser de 30% le prix conseillé des produits qui ne figurent sur aucune des commandes.
- Insérer dans la table E_Resultat les produits concernés par la réduction :
 - No --> <<No-programme>>
 - LB_Resultat --> <<Variable_Numéro_Produit>> - <<Nom_Produit>> baisse de 30%
 - VL_Resultat --> <<Variable_Prix_Conseillé_avant_de_30%>>

PL/SQL FOR ORACLE

**La gestion des exceptions,
les fonctions et les procédures,
les packages**

M. NASSAR & R. OULAD HAJ THAMI

SOMMAIRE GENERAL

MOTIVATIONS

STRUCTURE D'UN BLOC PL/SQL

LES VARIABLES

LES ENREGISTREMENTS

ASSIGNATION DES VARIABLES ET AFFECTATION

STRUCTURES DE CONTRÔLE

LES COLLECTIONS

LES TRANSACTIONS

INSERT-UPDATE-DELETE DANS UN BLOC PL/SQL

GESTION DES ERREURS ET DES EXCEPTIONS

LES CURSEURS

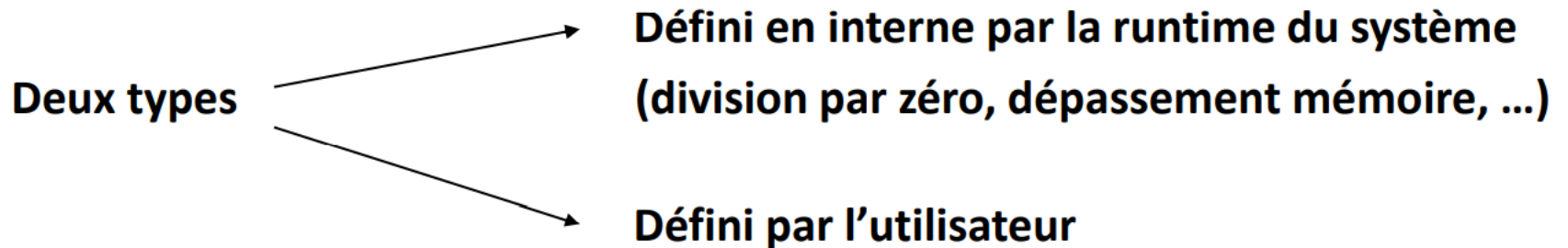
LES PROCEDURES ET LES FOCNTIONS STOCKEES

LES PACKAGES

LES TRIGGERS

Gestion des exceptions

Le gestion des exceptions permet d'affecter un traitement approprié aux erreurs qui apparaissent lors de l'exécution d'un bloc PL/SQL



■ Codes exceptions internes

Code d'erreur SQLCODE	Erreur
+1403	NO_DATA_FOUND
-1	DUP_VAL_ON_INDEX
-6502	VALUE_ERROR
-1001	INVALID CURSOR
-1722	INVALID NUMBER
-6501	PROGRAM ERROR
-1017	LOGIN DENIED
-1422	TOO_MANY_ROWS
-1476	ZERO_DIVIDE

■ Fonctions PL/SQL pour la gestion d'erreurs

- **SQLCODE** : Code de la dernière exception levée
- **SQLERRM** : message d'erreur correspondant

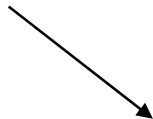
■ Exceptions externes

- **Déclaration** :

DECLARE Nom_exception **EXCEPTION**

- **Lever une exception** :

RAISE Nom_exception ;



Arrête l'exécution normale du bloc PL/SQL et transfère le contrôle au gestionnaire de l'exception

➤ Traitement des exceptions :

DECLARE

...

BEGIN

...

EXCEPTION

WHEN Nom_exception1 THEN

-Instructions1

WHEN Nom_exception2 OR Nom_exception3 THEN

Instructions2

WHEN ...

WHEN OTHERS THEN

Gérer les autres erreurs

END;

TRAVAUX PRATIQUES

GESTION DES EXCEPTIONS

Exercice 1 :

Ecrire un programme PL/SQL permettant de :

- lire un numéro de client
- rechercher le nom du client
- calculer le rapport ($\text{rapport} = \text{Mt_Commandes} / \text{Mt_Client}$) où Mt_Commandes est le montant total des commandes et Mt_Client est le montant total des commandes du client.

Insérer dans la table E_Resultat :

NO ----> <<no_programme>>

LB_Resultat ----> Rapport des commandes du client <<variable_nom_client>>

VL_Resultat -----> <<variable_rapport>>

gérer les possibilités suivantes (exceptions) :

- Numéro de client erroné
- Montant total des commandes du client (Mt_Client) = 0

**PROCEDURES ET
FONCTIONS STOCKEES
ET
PACKAGE**

Définitions:

Une procédure est un ensemble de code PL/SQL nommé, défini par l'utilisateur et stocké dans la BDD

Une fonction est identique à une procédure à la différence qu'elle retourne une valeur

Un paquetage est le regroupement de plusieurs procédures et fonctions dans un objet distinct

Avantages :

Le code relatif aux règles de gestion est centralisé.

Partage du code entre plusieurs applications

Amélioration des performances, car le code stocké est pré-compilé

Accessibilité des procédures stockées par Sql*Plus, Forms, Reports, Pro*C, Pro*Cobol, etc.

LES PROCEDURES

Privilèges système nécessaires:

Créer un objet procédural dans votre propre schéma :

GRANT CREATE PROCEDURE TO USER;

Créer un objet procédural dans n'importe quel schéma

GRANT CREATE ANY PROCEDURE TO USER;

Autoriser un autre schéma à exécuter une procédure de votre schéma, vous devez lui octroyer le privilège EXECUTE

GRANT EXECUTE ON MA_PROCEDURE TO AUTRE_SCHEMA;

BASES DE DONNEES RELATIONNELLES

Syntaxe :

```
CREATE [OR REPLACE] PROCEDURE NOM_PROC [(PARAMETRES)]  
      [AUTHID [CURRENT USER | DEFINIR]]  
[IS | AS]  
BLOC/PLSQL
```

CREATE indique que l'on veut créer une procédure stockée dans la base

OR REPLACE facultative. Permet d'écraser une procédure existante portant le même nom

nom procédure le nom donné par l'utilisateur à la procédure

AUTHID indique sur quel schéma la procédure s'applique

CURRENT_USER Indique que la procédure utilise les objets du schéma de l'utilisateur qui appelle la procédure

DEFINER(défaut) Indique que la procédure utilise les objets du schéma de création de la procédure

PAR DEFAULT: le schéma et les objets utilisés par la procédure sont ceux de son schéma de création

Paramètres d'une procédure

NOM_PARAM [**IN**|**OUT NOCOPY** |**IN OUT NOCOPY**] **TYPE_PARAM** [**:=**|**DEFAULT**] **EXPRESSION**]

nom paramètre	est le nom donné par l'utilisateur au paramètre transmis
IN	passage de paramètre en mode VALEUR. PAS D'AFFECTATION DANS LA PROCEDURE
OUT	passage de paramètre en mode ADRESSE
IN OUT	indique que le paramètre est transmis par le programme appelant et renseigné par la procédure
NOCOPY	comme IN OUT sans copie de l'objet dans la mémoire locale de la procédure appelée. A utiliser pour les grand objets (LOB, TABLE, GRAND RECORD, etc.)
TYPE_PARAM	Le type SQL ou PL/SQL du paramètre
:=	Symbole d'assignation d'une valeur par défaut
DEFAULT	identique à := la valeur par défaut de l'argument
EXPRESSION	La valeur par défaut du paramètre

PROCEDURE: Exemple 1: MODE DE PASSAGE DE PARAMETRES

```
SQL> CREATE OR REPLACE PROCEDURE TESTS_PARAMS
2      (V1 IN NUMBER, V2 OUT NUMBER, V3 IN OUT NUMBER, V4 IN OUT NOCOPY NUMBER)
3  IS
4  BEGIN
5      DBMS_OUTPUT.PUT_LINE('DEBUT TESTS_PARAMS');
6      --V1:=10; RETOURNE UNE ERREUR PAS D'AFFECTATION D'UN ARGUMENT IN
7      DBMS_OUTPUT.PUT_LINE('TESTS_PARAMS, valeur de V1 : ' || V1);
8      V2:=20;
9      DBMS_OUTPUT.PUT_LINE('TESTS_PARAMS, valeur de V2 : ' || V2);
10     V3:=30;
11     DBMS_OUTPUT.PUT_LINE('TESTS_PARAMS, valeur de V3 : ' || V3);
12     V4:=40;
13     DBMS_OUTPUT.PUT_LINE('TESTS_PARAMS, valeur de V4 : ' || V4);
14     DBMS_OUTPUT.PUT_LINE('FIN TESTS_PARAMS');
15 END;
16 /
```

Procédure créée.

PROCEDURE: Exemple 1: MODE DE PASSAGE DE PARAMETRES

```
SQL>
SQL> DECLARE
2      T1      NUMBER:=100;
3      T2      NUMBER:=100;
4      T3      NUMBER:=100;
5      T4      NUMBER:=100;
6 BEGIN
7      DBMS_OUTPUT.PUT_LINE('DEBUT BLOC PL/SQL');
8
9      TESTS_PARAMS(T1,T2,T3,T4);
10
11     DBMS_OUTPUT.PUT_LINE('DEBUT BLOC PL/SQL, valeur de T1 : ' || T1);
12     DBMS_OUTPUT.PUT_LINE('DEBUT BLOC PL/SQL, valeur de T2 : ' || T2);
13     DBMS_OUTPUT.PUT_LINE('DEBUT BLOC PL/SQL, valeur de T3 : ' || T3);
14     DBMS_OUTPUT.PUT_LINE('DEBUT BLOC PL/SQL, valeur de T4 : ' || T4);
15     DBMS_OUTPUT.PUT_LINE('FIN BLOC PL/SQL');
16 END;
17 /
```

```
DEBUT BLOC PL/SQL
DEBUT TESTS_PARAMS
TESTS_PARAMS, valeur de V1 : 100
TESTS_PARAMS, valeur de V2 : 20
TESTS_PARAMS, valeur de V3 : 30
TESTS_PARAMS, valeur de V4 : 40
FIN TESTS_PARAMS
DEBUT BLOC PL/SQL, valeur de T1 : 100
DEBUT BLOC PL/SQL, valeur de T2 : 20
DEBUT BLOC PL/SQL, valeur de T3 : 30
DEBUT BLOC PL/SQL, valeur de T4 : 40
FIN BLOC PL/SQL
```

BASES DE DONNEES RELATIONNELLES

PROCEDURE: Exemple 3: APPEL DANS UN BLOC PL/SQL OU UNE AUTRE PROCEDURE

```
SQL> CREATE OR REPLACE PROCEDURE TOTAL_CLIENT
2      (NCL IN NUMBER, NOMBCOM OUT NUMBER, TOTALCL OUT NUMBER)
3  IS
4  BEGIN
5      SELECT COUNT(*), SUM(TOTAL) INTO NOMBCOM, TOTALCL
6      FROM E_COMMANDE
7      WHERE CLIENT_NO=NCL;
8  END TOTAL_CLIENT;
9  /
```

Procédure créée.

```
SQL> DECLARE
2      NCL    NUMBER:=1;
3      NCOM   NUMBER;
4      TOTAL  NUMBER;
5  BEGIN
6      TOTAL_CLIENT(1,NCOM,TOTAL);
7      DBMS_OUTPUT.PUT_LINE('CLIENT NUMERO : ' || NCL);
8      DBMS_OUTPUT.PUT_LINE('NOMBRE DE COMMANDE: ' || NCOM);
9      DBMS_OUTPUT.PUT_LINE('TOTAL DES COMMANDES: ' || TOTAL);
10 END;
11 /
```

CLIENT NUMERO : 1
NOMBRE DE COMMANDE: 2
TOTAL DES COMMANDES: 2300

Procédure PL/SQL terminée avec succès.

SQL>

PROCEDURE: Exemple 2: APPEL DANS SQL*PLUS POUR LES TESTS

```
SQL> CREATE OR REPLACE PROCEDURE TOTAL_CLIENT
  2      (NCL IN NUMBER, NOMBCOM OUT NUMBER, TOTALCL OUT NUMBER)
  3 IS
  4 BEGIN
  5      SELECT COUNT(*), SUM(TOTAL) INTO NOMBCOM, TOTALCL
  6      FROM E_COMMANDE
  7      WHERE CLIENT_NO=NCL;
  8 END TOTAL_CLIENT;
  9 /
```

Procédure créée.

PROCEDURE: Exemple 3: APPEL DANS SQL*PLUS POUR LES TESTS

```
SQL> --déclaration des arguments dans SQL*PLUS
```

```
SQL> VARIABLE      NCOM  NUMBER;
```

```
SQL> VARIABLE      TOTAL  NUMBER;
```

```
SQL>
```

```
SQL> --exécution de la procédure
```

```
SQL> EXECUTE TOTAL_CLIENT(1,:NCOM,:TOTAL);
```

Procédure PL/SQL terminée avec succès.

```
SQL>
```

```
SQL> --AFFICHAGE DES VALEURS DES ARGUMENTS
```

```
SQL> PRINT  NCOM;
```

NCOM

2

```
SQL> PRINT  TOTAL;
```

TOTAL

2300

```
SQL>
```

PROCEDURE: Exemple 4: APPEL DANS SQL*PLUS POUR LES TESTS

```
SQL> --declarartion des arguments dans SQL*PLUS
SQL> VARIABLE          NCOM  NUMBER;
SQL> VARIABLE          TOTAL  NUMBER;
SQL>
SQL> --exécution de la procédure
SQL> EXECUTE TOTAL_CLIENT(1,TOTALCL=>:TOTAL, NOMBCOM=>:NCOM);
```

Procédure PL/SQL terminée avec succès.

```
SQL> --AFFICHAGE DES VALEURS DES ARGUMENTS
SQL> PRINT  NCOM;
```

```
      NCOM
-----
        2
```

```
SQL> PRINT  TOTAL;
```

```
     TOTAL
-----
    2300
```

```
CREATE OR REPLACE PROCEDURE TOTAL_CLIENT
(NCL IN NUMBER, NOMBCOM OUT NUMBER,
TOTALCL OUT NUMBER)
IS ...
```

⇒ Pour un appel avec les arguments nommés
ARG_FORMEL=>ARG_EFFECTIF
L'ordre des arguments dans ce cas n'est pas important

MEME CHOSE POUR UN APPEL DANS BLOC PL/SQL OU UNE
AUTRE POCEDURE OU FONCTION

LES FONCTIONS

Syntaxe :

```
CREATE [OR REPLACE] FUNCTION NOM_FCT [(PARAMETRES)] RETURN TYPE_RETOUR_FCT
    [AUTHID [CURRENT USER | DEFINIR]]
[IS | AS]
BEGIN

RETURN RESULTAT;
END [NOM_FCT];
/
```

RETURN TYPE_RETOUR_FCT indique le type de résultat d'une fonction

Le reste est identique que pour une procédure

PAR DEFAUT: le schéma et les objets utilisés par la fonction sont ceux de son schéma de création

FONCTION: Exemple 5:

```
SQL>
SQL> CREATE OR REPLACE FUNCTION TOTALCLIENT
2      (NCL IN NUMBER)
3
4  RETURN NUMBER
5
6  IS
7      TOT    NUMBER:=0;
8  BEGIN
9      SELECT SUM(TOTAL) INTO TOT
10     FROM E_COMMANDE
11     WHERE CLIENT_NO=NCL;
12
13     RETURN TOT;
14
15 END TOTALCLIENT;
16 /
```

Fonction créée.

FONCTION: Exemple 6: L'APPEL D'UNE FONCTION

```
SQL> CREATE OR REPLACE FUNCTION TOTALCLIENT
2      (NCL IN NUMBER) RETURN NUMBER
3  IS
4      TOT    NUMBER:=0;
5  BEGIN
6      SELECT SUM(TOTAL) INTO TOT
7      FROM E_COMMANDE
8      WHERE CLIENT_NO=NCL;
9      RETURN TOT;
10 END TOTALCLIENT;
11 /
```

Fonction créée.

```
SQL> DECLARE
2      SOM      NUMBER;
3  BEGIN
4      SOM:=TOTALCLIENT (1);
5      DBMS_OUTPUT.PUT_LINE('LA SOMME EST : '|| SOM);
6  END;
7  /
```

LA SOMME EST : 2300

Procédure PL/SQL terminée avec succès.

FONCTION: Exemple 7: L'APPEL D'UNE FONCTION DANS SQL*PLUS

```
SQL>
SQL> --déclarartion de variable
SQL> VARIABLETOTAL      NUMBER;
SQL>
SQL> --Exécution de la fonction
SQL> EXECUTE :TOTAL:=TOTALCLIENT(1);
```

Procédure PL/SQL terminée avec succès.

```
SQL>
SQL> --Affichage du résultat retourné
SQL> PRINT TOTAL
```

```
TOTAL
-----
2300
```

```
SQL>
```

FONCTION: Exemple 7: L'APPEL D'UNE FONCTION DANS UNE REQUETE

```
SQL> SELECT DISTINCT(CLIENT_NO), TOTALCLIENT(CLIENT_NO)
2 FROM E_COMMANDE;
```

CLIENT_NO TOTALCLIENT(CLIENT_NO)	

4	80100
1	2300
2	0
3	50030
6	0
13	150800
8	0

7 ligne(s) sélectionnée(s).

```
SQL>
SQL>
SQL> INSERT INTO E_COMMANDE VALUES (
2 1111,
3 1,
4 '12/12/2012',
5 '23/12/2012',
6 1,
7 TOTALCLIENT(4)
8 );

1 ligne créée.
```

UTILISATION: PROJECTION, WHERE, HAVING, ORDER BY, GROUP BY, VALUES, UPDATE SET

RESTRICTION: CHECK, TYPE PL/SQL, UPDATE, INSERT, DELETE, ARGUMENT OUT ou IN OUT

Recompilation d'une procédure ou d'une fonction

```
ALTER PROCEDURE nom_procedure COMPILE;  
ALTER FUNCTION nom_fonction COMPILE;
```

Suppression d'une procédure ou d'une fonction

```
DROP PROCEDURE nom_procedure ;  
DROP FUNCTION nom_fonction ;
```

LES PACKAGES

Définitions:

**Un PACKAGE est une ensemble de:
VARIABLES,
PROCEDURES,
Et FONCTIONS.**

Avantages:

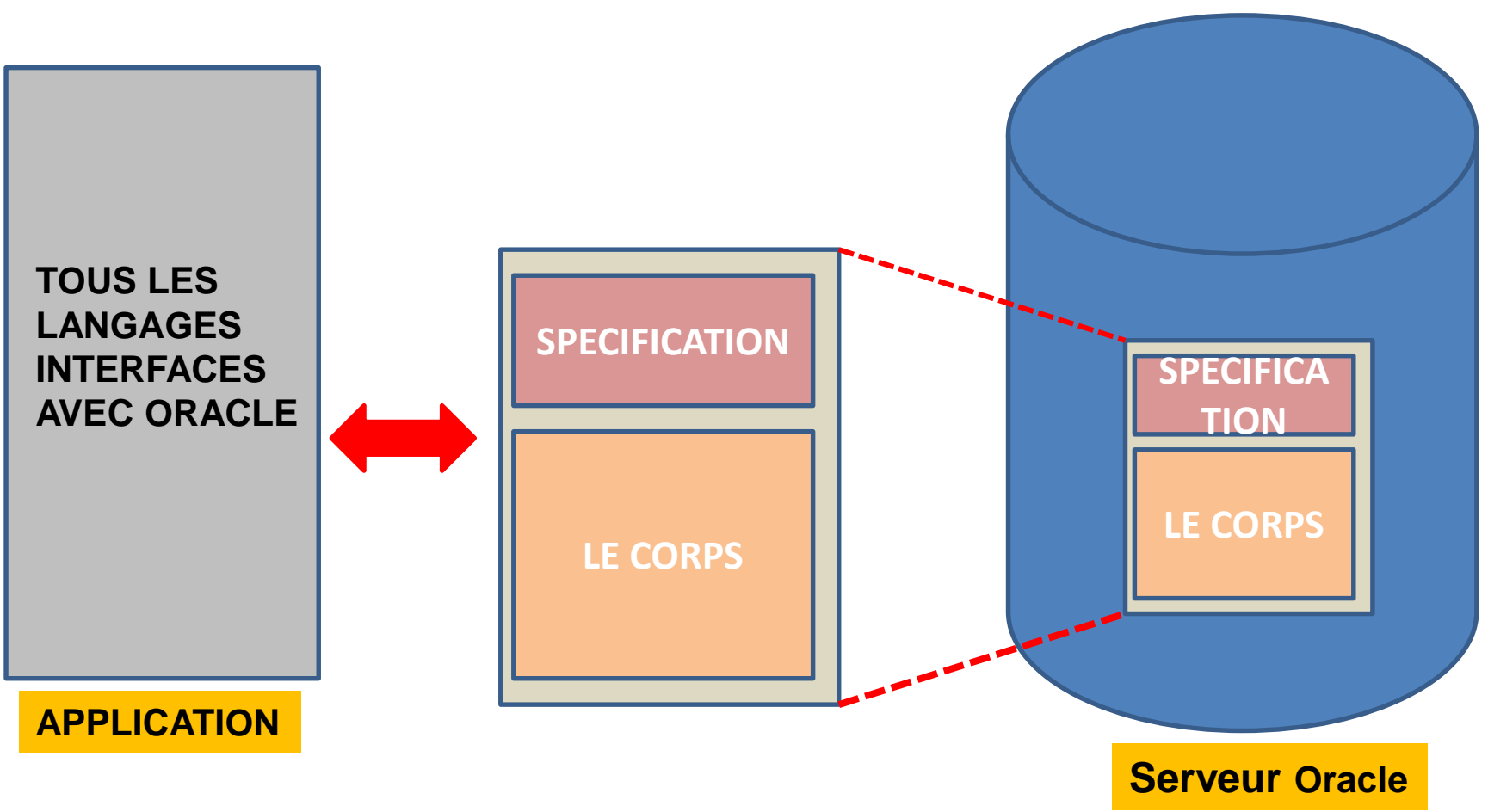
Maintenance

Partage

Modularité

Extensibilité

Réutilisabilité



```
CREATE PACKAGE nom_package
AS
```

```
    PROCEDURE P1 (....);
    FUNCTION F1(...) RETURN .....;
    VARIABLES
    EXCEPTIONS
```

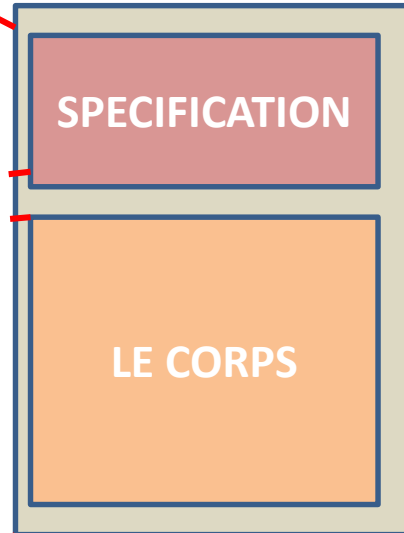
```
END [nom_package];
```

```
CREATE PACKAGE BODY
nom_package
```

```
    PROCEDURE P1 (....) IS
    BEGIN
        .....
    END P1;
```

```
    FUNCTION F1(...) RETURN .....IS
    BEGIN
        .....
    END F1;
```

```
.....
END [nom_package];
```



COMPILATION:

SQL> -- comme pour un bloc PL/SQL

SQL> **START** nom_fichier_contenant_le_package

SQL> **@** nom_fichier_contenant_le_package

RECOMPILATION:

SQL> **ALTER PACKAGE** nom_package **COMPILE BODY;**

SQL> **ALTER PACKAGE** nom_package **COMPILE PACKAGE;**
OU

CREATE OR REPLACE PACKAGE
CREATE OR REPLACE PACKAGE BODY

DESTRUCTION:

SQL> **DROP PACKAGE BODY** nom_package;

SQL> **DROP PACKAGE** nom_package;

PACKAGE: Exemple 1:

```
SQL> CREATE OR REPLACE PACKAGE GEST_EMP AS
2      PROCEDURE EMP_LE_PLUS_PAYE(N OUT E_EMPLOYE.NOM%TYPE, S OUT
                                     E_EMPLOYE.SALAIRE%TYPE);
3      FUNCTION MASSE_SALAIRE RETURN NUMBER;
4  END GEST_EMP;
5  /
```

Package créé.

SQL>

PACKAGE: Exemple 1 suite:

```
SQL> CREATE OR REPLACE PACKAGE BODY GEST_EMP AS
2
3     PROCEDURE EMP_LE_PLUS_PAYE(N OUT E_EMPLOYEE.NOM%TYPE, S OUT
E_EMPLOYEE.SALAIRE%TYPE)
4     IS
5     BEGIN
6         SELECT E.NOM, E.SALAIRE INTO N, S
7         FROM E_EMPLOYEE E
8         WHERE E.SALAIRE >= ALL (SELECT SALAIRE FROM E_EMPLOYEE);
9     END EMP_LE_PLUS_PAYE;
10
11    FUNCTION MASSE_SALAIRE RETURN NUMBER
12    IS
13        MAS    NUMBER:=0;
14    BEGIN
15        SELECT SUM(SALAIRE) INTO MAS
16        FROM E_EMPLOYEE;
17
18        RETURN MAS;
19    END MASSE_SALAIRE;
20 END GEST_EMP;
21 /
```

Corps de package créé.

SQL> DECLARE

5 BEGIN

```
7 DBMS_OUTPUT.PUT_LINE('EMPLOYE LE MIEUX PAYE :'|| NOM ||' SON  
SALAIRE EST: '||SAL);
```

```
9 DBMS_OUTPUT.PUT_LINE('LA MASSE SALARIALE EST :'|| MAS);
```

11 /

LA MASSE SALARIALE EST :43555,8

ORACLE

PACKAGE: Exemple 2 : VARIABLES

```
SQL> CREATE OR REPLACE PACKAGE AUG_EMP AS
2
3     NBR_AUG                                NUMBER:=0;
4
5     TAUX_ZERO                                EXCEPTION;
6     ERR_TRANCHE                                EXCEPTION;
7     PRAGMA EXCEPTION_INIT( TAUX_ZERO,    -22100 ) ;
8     PRAGMA EXCEPTION_INIT( ERR_TRANCHE,    -6502 ) ;
9
10    PROCEDURE AUGMENTER(TAUX IN NUMBER,INF IN
        E_EMPLOYE.SALAIRE%TYPE,SUP IN E_EMPLOYE.SALAIRE%TYPE);
11 END AUG_EMP;
12 /
```

Package créé.

PACKAGE: Exemple 2 : VARIABLES

```
SQL> CREATE OR REPLACE PACKAGE BODY AUG_EMP AS
2
3     CURSOR LISTE_EMPS IS
4     SELECT NO, SALAIRE
5     FROM E_EMPLOYE;
6
7
.
.
.
.
29
30 END AUG_EMP;
31 /
```

BASES DE DONNEES RELATIONNELLES

```
7      PROCEDURE AUGMENTER(TAUX IN NUMBER,INF IN E_EMPLOYE.SALAIRE%TYPE,SUP IN
      E_EMPLOYE.SALAIRE%TYPE)
8      IS
9      AUG    NUMBER(5,2):=(TAUX/100)+1;
10     BEGIN
11         IF TAUX=0 THEN RAISE TAUX_ZERO; END IF;
12         IF (INF>=SUP) THEN RAISE ERR_TRANCHE; END IF;
13         FOR EMP IN LISTE_EMPS LOOP
14             IF (INF<=EMP.SALAIRE) AND (EMP.SALAIRE<=SUP)
15                 THEN
16                     UPDATE E_EMPLOYE E
17                     SET E.SALAIRE=E.SALAIRE*AUG
18                     WHERE EMP.NO=E.NO;
19                     NBR_AUG:=NBR_AUG+1;
20                 END IF;
21         END LOOP;
22         COMMIT;
23         EXCEPTION
24         WHEN TAUX_ZERO THEN DBMS_OUTPUT.PUT_LINE ('ERR RADIN: AUGMENTATION NULLE!!');
25         WHEN ERR_TRANCHE THEN DBMS_OUTPUT.PUT_LINE ('ERR: TRANCHE ERRONEE!!');
26         WHEN OTHERS THEN DBMS_OUTPUT.PUT_LINE ('ERR: ERREUR INCONNUE!!');
27         COMMIT;
28     END AUGMENTER;
29
30 END AUG_EMP;
31 /
Corps de package créé.
```

BASES DE DONNEES RELATIONNELLES

PACKAGE: Exemple 2 : VARIABLES: APPEL

```
SQL> SELECT NO, SALAIRE FROM
E_EMPLOYE;
      NO  SALAIRE
-----
1      1  4500,5
2      2  3794,22
3      3  4500,5
4      4  4079,2
5      5  4126,17
6      6  3320,46
7      7  5400,5
8      8  2845,78
9      9  3794,22
10     10 6000,34
```

10 ligne(s) sélectionnée(s).

```
SQL> SELECT NO, SALAIRE FROM
E_EMPLOYE;
      NO  SALAIRE
-----
1      1  4500,5
2      2  4021,87
3      3  4500,5
4      4  4323,95
5      5  4373,74
6      6  3519,69
7      7  5400,5
8      8  2845,78
9      9  4021,87
10     10 6000,34
```

10 ligne(s) sélectionnée(s).

```
SQL> BEGIN
2      AUG_EMP.AUGMENTER(5.5, 3000, 4500);
3      DBMS_OUTPUT.PUT_LINE ('LE NOMBRE D EMPLOYES AUGMENTES EST: '||
      AUG_EMP.NBR_AUG );
4 END;
5 /
LE NOMBRE D EMPLOYES AUGMENTES EST: 5
Procédure PL/SQL terminée avec succès.
```

Remarques

- **les objets déclarés au niveau des spécifications d'un package ou de son corps persistent le long de la session et après le premier appel à ce package.**
- **On peut initialiser les variables explicitement en introduisant dans le corps d'un package un bloc séparé par BEGIN et END**
- **Possibilité de nommer plusieurs procédures (fonctions) de la même façon (SURCHARGES).**

PACKAGE: Exemple 3: SURCHARGES DES PROCEDURES ET FONCTIONS

Create package Ges_employees IS

function CA(NO EMP Number) **Return** Number;

function CA(NO EMP Number, No CLi Number) **Return** Number;

procedure Augmentation_ann;

END Ges_employees;

créer un package composé de 2 fonctions qui calculent le chiffre d'affaires global et le chiffre d'affaires par client pour un employé donné, et une procédure d'augmentation annuelle (10%) des salaires des employés.

PACKAGE: Exemple 3: SURCHARGES DES PROCEDURES ET FONCTIONS

Create package Body Ges_employees IS

Function CA(**NO_EMP Number**) Return Number IS

CA1 Number :=0;

begin

select sum(montant) into CA1

from Commande

where Employe_NO= No_Emp;

return (CA1);

end CA;

Function CA(**NO_EMP Number, No_CLI Number**) Return Number IS

CA2 Number :=0;

begin

select sum(montant) into CA2

from Commande

where Employe_NO = No_Emp and client_No=No_Cli;

return (CA2);

end CA;

PACKAGE: Exemple 3: SURCHARGES DES PROCEDURES ET FONCTIONS

```
Procedure Augmentation_ann IS
begin
  if (To_char(SYSDATE,'DDMM')='0501' Then
    update E_employe
      set salaire = salaire * 1.1;
    commit;
  end if;
end Augmentation_ann;

End Ges_employes;
```

**TRAVAUX
PRATIQUES**

PROCEDURES, FONCTIONS ET PACKAGES

Exercice 1

1.1. Créer une procédure *Sup_emp* qui permet de:

supprimer un enregistrement dans la table *E_employe*, avec en paramètre d'entrée le numéro de l'enregistrement à supprimer.

Si ce dernier comporte un enregistrement fils cela génère l'erreur **ORA-02292** qu'il faudra traiter.

1.2 Créer un programme qui fait appel à la procédure *Sup_emp*.

1.3. Tester et valider

Exercice 2

1.1. Créer une procédure ***augmente_salaire*** qui prend comme argument *le taux d'augmentation* et qui augmente les salaires des employés et affiche pour chaque employé le numéro, le nom, l'ancien salaire et le nouveau salaire.

Le taux de l'augmentation doit être supérieur strictement à 0 (à gérer sous forme d'exception externe)

1.2 Créer un programme qui fait appel à la procédure ***augmente_salaire***.

1.3. Tester et valider

Exercice 3

1.1. Créer une fonction **Recette_annuelle** qui prend comme argument *une année* et retourne la somme des totaux de toutes les commandes réalisées en cette année.

1.2 Créer un programme qui fait appel à la fonction Recette_annuelle.

1.3. Tester et valider

PL/SQL FOR ORACLE

Les déclencheurs (Triggers)

M. NASSAR & R. OULAD HAJ THAMI

SOMMAIRE GENERAL

MOTIVATIONS

STRUCTURE D'UN BLOC PL/SQL

LES VARIABLES

LES ENREGISTREMENTS

ASSIGNATION DES VARIABLES ET AFFECTATION

STRUCTURES DE CONTRÔLE

LES COLLECTIONS

LES TRANSACTIONS

INSERT-UPDATE-DELETE DANS UN BLOC PL/SQL

GESTION DES ERREURS ET DES EXCEPTIONS

LES CURSEURS

LES PROCEDURES ET LES FOCNTIONS STOCKEES

LES PACKAGES

LES TRIGGERS

**LES
TRIGGERS
(DECLENCHEURS)**

Définition :

Un déclencheur est un programme qui se déclenche automatiquement suite à un évènement sur une table, une base, une application ou un évènement système

Utilisation : les déclencheurs peuvent être utilisés pour :

la sécurité

l'audit

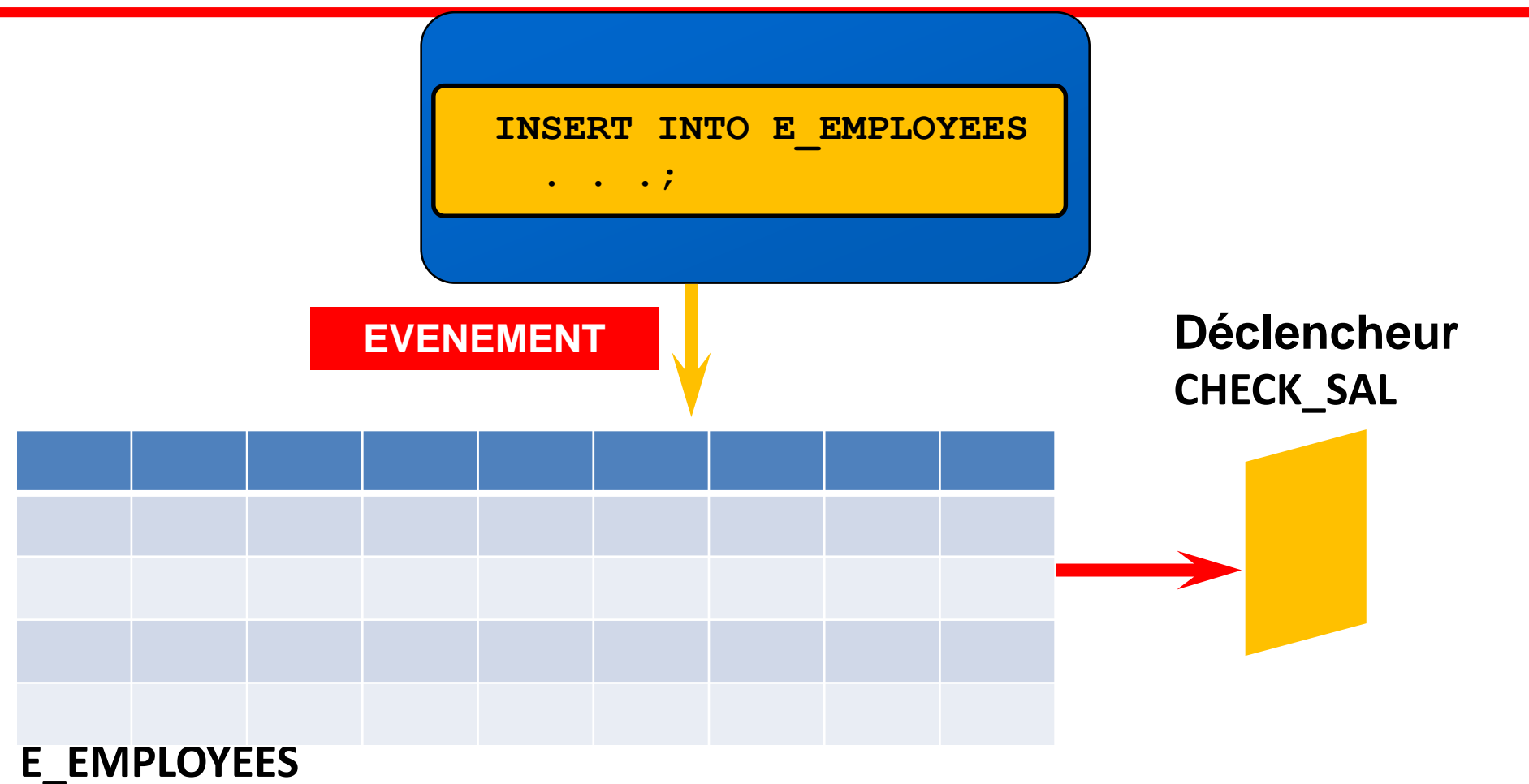
l'intégrité des données

l'intégrité référentielle

la réplication de table

le calcul automatique des données dérivées

**LES
TRIGGERS LMD
(DECLENCHEURS)**



QUESTIONS

EVENEMENT DECLENCHEUR??
TEMPS DE L'EXECUTION? APRES L'EVENEMENT DECLENCHEUR? AVANT?
SUR TOUTE LA TABLE? SUR CHAQUE LIGNE???

Syntaxe: TRIGGER SUR TABLE

```
CREATE [OR REPLACE] TRIGGER nom_trigger
  timing
  event1 [OR event2 OR event3]
ON nom_table
Corps_de_trigger
```

timing	::= BEFORE AFTER
Event	::= INSERT DELETE UPDATE

INSERT	IF	INSERTING	THEN
DELETE	IF	DELETING	THEN ...
UPDATE	IF	UPDATING('ATTRIBUT')	THEN ... (modification de l'attribut)
	IF	UPDATING	THEN ... (n'importe quelle modification sur la table)

LE TRIGGER SUR LA TABLE EST DECLENCHE UNE SEULE FOIS SUITE A UN EVENEMENT SUR LA TABLE

TRIGGER SUR TABLE: Exemple 1:

```
SQL> CREATE OR REPLACE TRIGGER ACCES_EMP
  2 BEFORE INSERT ON E_EMPLOYE
  3 BEGIN
  4   IF (TO_CHAR (sysdate, 'DY') IN ('SAT', 'SUN'))
  5       OR (TO_CHAR(sysdate, 'HH24') NOT BETWEEN
  6         '08' AND '18'
  7       THEN RAISE_APPLICATION_ERROR (-20500,
  8         'Vous ne pouvez pas utiliser la table E_EMPLOYE
           que pendant les heures normales. ');
 10   END IF;
 11 END;
 12 /
```

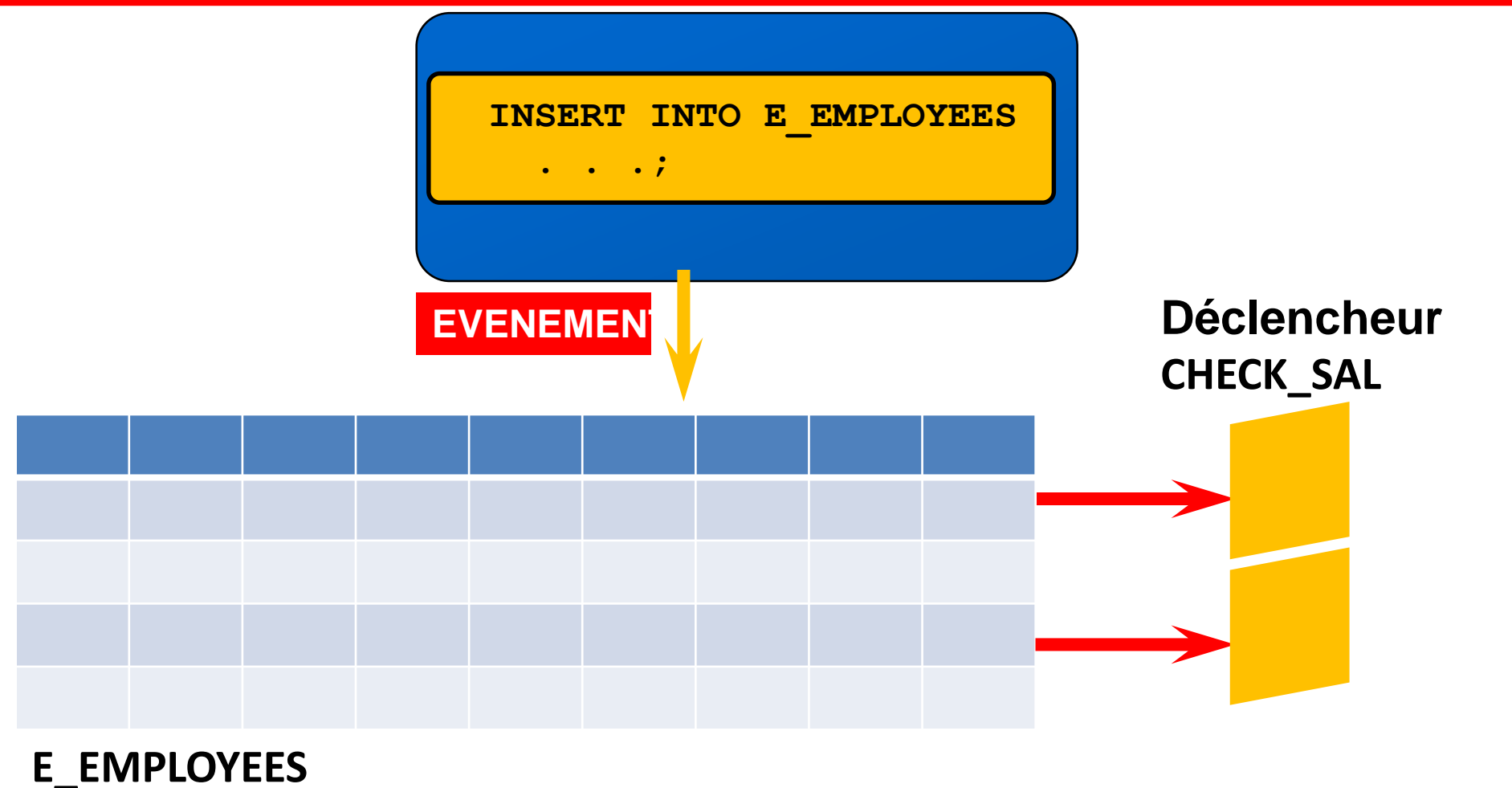
TRIGGER SUR TABLE: Exemple 2:

```
CREATE OR REPLACE TRIGGER ACCES_EMP
BEFORE INSERT OR UPDATE OR DELETE ON E_EMPLOYES
BEGIN
  IF (TO_CHAR (SYSDATE, 'DY') IN ('SAT', 'SUN')) OR
     (TO_CHAR (SYSDATE, 'HH24') NOT BETWEEN '08' AND '18')
  THEN
    IF      DELETING THEN
      RAISE_APPLICATION_ERROR (-20502, 'Vous ne pouvez pas supprimer dans la table
E_EMPLOYE que pendant les heures normales. ');
    ELSIF  INSERTING THEN
      RAISE_APPLICATION_ERROR (-20500, 'Vous ne pouvez pas ajouter dans la table
E_EMPLOYE que pendant les heures normales. ');
    ELSIF  UPDATING ('SALAIRE') THEN
      RAISE_APPLICATION_ERROR (-20503, 'Vous ne pouvez pas modifier le SALAIRE dans la
table E_EMPLOYE que pendant les heures normales. ');
    ELSE
      RAISE_APPLICATION_ERROR (-20504, 'Vous ne pouvez pas
      modifier la table E_EMPLOYE que pendant les heures normales. ');
    END IF;
  END IF;
END;
```

Syntaxe : TRIGGER SUR LIGNE

```
CREATE [OR REPLACE] TRIGGER nom_trigger  
timing  
    event1 [OR event2 OR event3]  
ON nom_table  
FOR EACH ROW  
Corps_de_trigger
```

**LE TRIGGER SERA EXECUTE POUR CHAQUE LIGNE DE LA TABLE
CONCERNEE PAR L 'EVENEMENT DECLENCHEUR**



LES ANCIENS VALEURS DE LA LIGNE SONT REFERENCEES PAR :OLD
LES NOUVELLES VALEURS DE LA LIGNE SONT REFERENCEES PAR :NEW

TRIGGER SUR LIGNE: Exemple 1:

```
SQL>
SQL> CREATE OR REPLACE TRIGGER CHECK_SAL
 2 BEFORE UPDATE ON E_EMPLOYE
 3 FOR EACH ROW
 4 BEGIN
 5     IF :NEW.SALAIRE<:OLD.SALAIRE
 6     THEN RAISE_APPLICATION_ERROR(-20200,'ERR: LE SALAIRE DOIT
SUPERIEUR A L      ANCIEN SALAIRE!!');
 7     END IF;
 8 END;
 9 /
```

Déclencheur créé.

```
SQL>
```

REGLE DE GESTION : UN SALAIRE NE DOIT JAMAIS BAISSER

```
SQL> select NO, SALAIRE FROM
E_EMPLOYE;
```

NO	SALAIRE
1	15189,2
2	4000,5
3	4500,5
4	4300,98
5	4350,5
6	3500,98
7	5400,5
8	3000,5
9	4000,5
10	6000,34

10 ligne(s) sélectionnée(s).

```
SQL>
SQL> --test
SQL> UPDATE E_EMPLOYE SET
SALAIRE=1000 WHERE NO=1;
UPDATE E_EMPLOYE SET SALAIRE=1000
WHERE NO=1
*
```

ERREUR à la ligne 1 :

ORA-20200: ERR: LE SALAIRE DOIT SUPERIEUR A L ANCIEN SALAIRE!!

ORA-06512: à "RACHID.CHECK_SAL",
ligne 3

ORA-04088: erreur lors d'exécution du déclencheur 'RACHID.CHECK_SAL'

BASES DE DONNEES RELATIONNELLES

TRIGGER SUR LIGNE: Exemple 2: TRACABILITE DES SUPPRESSIONS SUR UNE TABLE

```
SQL> SELECT NO, NOM, PRENOM, DT_ENTREE, SALAIRE FROM E_EMPLOYE;
```

NO	NOM	PRENOM	DT_ENTRE	SALAIRE
---	-----	-----	-----	-----
1	Alaoui	Said	01/01/95	4500.50
2	Filali	Mohammed	11/01/95	4021.87
3	Hayani	Mourad	13/02/97	4500.50
4	Ansari	Zouhair	25/01/96	4323.95
5	Naciri	Abdallah	01/11/96	4373.74
6	Rabii	Khalid	09/01/95	3519.69
7	Touzani	Said	06/01/96	5400.50
8	ElBasri	Samir	11/01/98	3000.50
9	Bahia	Salah	01/01/03	4000.50
10	Bahja	Brahim	19/01/95	6000.34

10 ligne(s) selectionnee(s).

REGLE DE GESTION: CHAQUE EMPLOYE SUPPRIME DOIT ETRE ARCHIVE

TRIGGER SUR LIGNE: Exemple 2: TRACABILITE DES SUPPRESSIONS SUR UNE TABLE

```
SQL> CREATE SEQUENCE SEQ_TR_EMP START WITH 1;  
Séquence créée.
```

```
SQL>  
SQL> CREATE TABLE TRACE_EMPS(  
 2  NUMOP          NUMBER(7),  
 3  DT_OP          DATE,  
 4  OPERAT         VARCHAR2(20),  
 5  NOEMP          NUMBER(7),  
 6  NOMEMP         VARCHAR2(50),  
 7  PREEMP         VARCHAR2(25),  
 8  DTENTREMP      DATE,  
 9  TITREEMP       VARCHAR2(25),  
10  SER_NOEMP      NUMBER(7),  
11  COMMENTAIRE    VARCHAR2(200),  
12  SALEMP        NUMBER(11,2),  
13  OLD_PCT_COMM   NUMBER(4,2)  
14 );
```

Table créée.

TRIGGER SUR LIGNE: Exemple 2: TRACABILITE DES SUPPRESSIONS SUR UNE TABLE

SQL>

SQL> **CREATE OR REPLACE TRIGGER** TRACE_EMPS

2 **AFTER DELETE** ON E_EMPLOYE

3 **FOR EACH ROW**

4 **BEGIN**

5 **IF DELETING** THEN

6 **INSERT INTO** TRACE_EMPS **VALUES**(

7 **SEQ_TR_EMP.NEXTVAL**,

8 **SYSDATE**,

9 **USER**,

10 **:OLD.NO**,

11 **:OLD.NOM**,

12 **:OLD.PRENOM**,

13 **:OLD.DT_ENTREE**,

14 **:OLD.TITRE**,

15 **:OLD.SERVICE_NO**,

16 **:OLD.COMMENTAIRE**,

17 **:OLD.SALAIRE**,

18 **:OLD.PCT_COMMISSION**

19 **);**

20 **END IF;**

21 **END;**

22 /

Déclencheur créé.

TRIGGER SUR LIGNE: Exemple 2: TRACABILITE DES SUPPRESSIONS SUR UNE TABLE

```
SQL> --test
SQL> DELETE E_EMPLOYE WHERE NO IN (7, 8, 9,10);
```

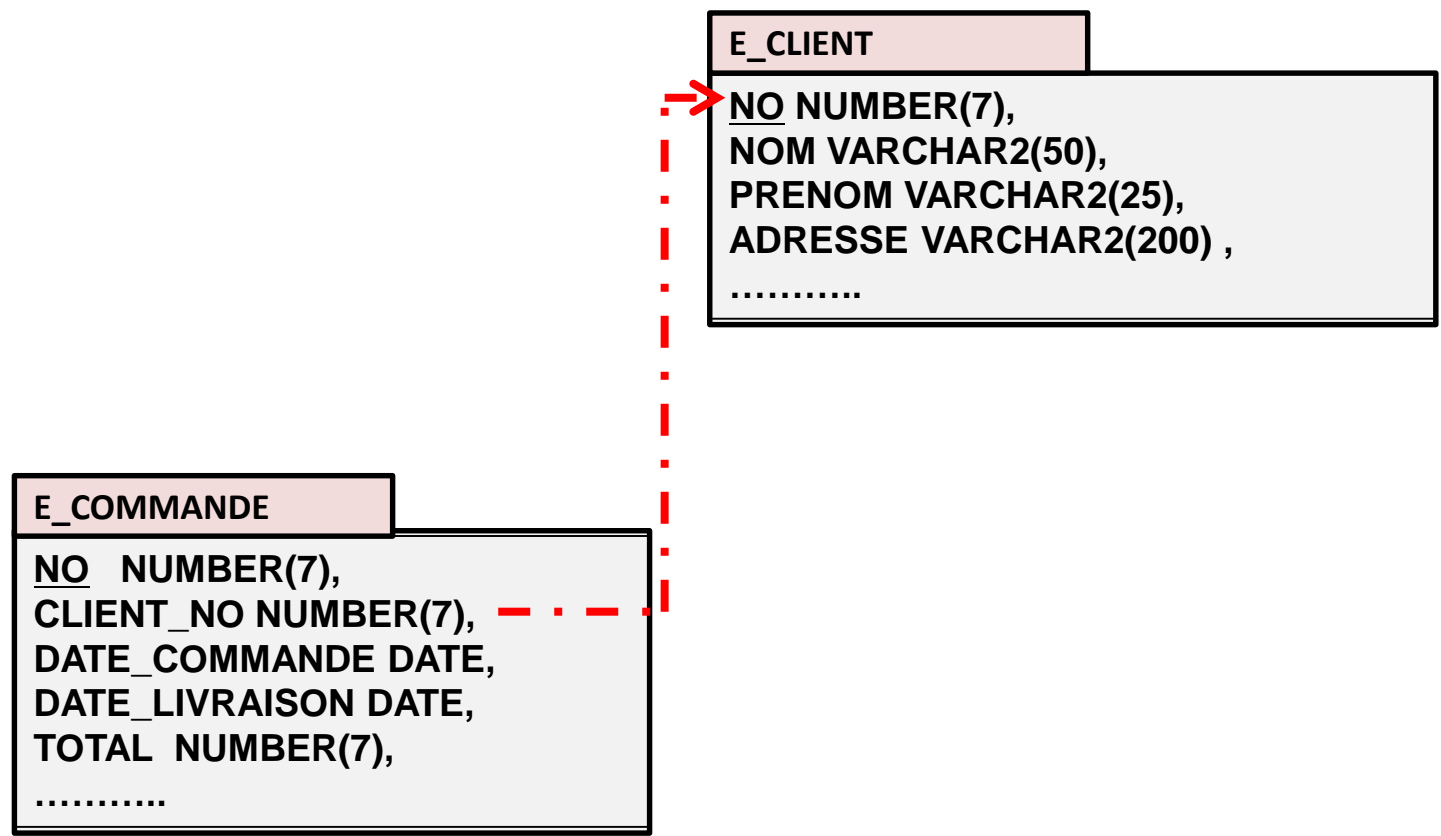
4 ligne(s) supprimée(s).

```
SQL> COMMIT; --si on commite pas, les tuples ajoutés dans TRACE_EMPS seront vidés
2
```

```
SQL> SELECT NUMOP,DT_OP, OPERAT, NOEMP, NOMEMP, PREEMP FROM TRACE_EMPS;
```

NUMOP	DT_OP	OPERAT	NOEMP	NOMEMP	PREEMP
1	17/01/13	RACHID	7	Touzani	Said
2	17/01/13	RACHID	8	ElBasri	Samir
3	17/01/13	RACHID	9	Bahja	Salah
4	17/01/13	RACHID	10	Bahja	Brahim

TRIGGER SUR LIGNE: Exemple 2: CONTRAINTE D'INTEGRITE REFRENTIELLE



POUR L'EXEMPLE, LA CONTRAINTE D'INTEGRITE REFRENTIELLE N'EST PAS DECLAREE LORS DE LA CREATION DES TABLES

TRIGGER SUR LIGNE: Exemple 2: CONTRAINTES D'INTEGRITE REFRENTIELLE

```
create or replace trigger CIF_CL
AFTER DELETE OR UPDATE ON E_CLIENT
FOR EACH ROW
DECLARE
    num_com          NUMBER(7);
BEGIN
    IF DELETING THEN
        insert into arch_client values
            (:old.NO,:old.NOM,:old.PRENOM,:old.ADRESSE);
        --suppression des commandes du client supprimé
        delete e_commande where client_no=:old.no;
    ELSIF UPDATING('NO') THEN
        update e_commande
        set client_no=:new.no
        where client_no=:old.no;
    END IF;
END;
/
```

TRIGGER SUR LIGNE: Exemple 2: CONTRAINTE D'INTEGRITE REFRENTIELLE

```
--trigger au niveau de la table commande
create or replace trigger CIF_COM
AFTER DELETE OR UPDATE OR INSERT ON E_COMMANDE
FOR EACH ROW
DECLARE
    NBR_CL NUMBER(3);
BEGIN
    IF DELETING THEN
        insert into arch_commande values
            (:old.NO,:old.CLIENT_NO,:old.DATE_COMMANDE,:old.DATE_LIVRAISON,:old.TOTAL);
    ELSIF INSERTING THEN
        select count(*) into NBR_CL
        from e_client
        where no=:new.client_no;
        IF NBR_CL=0 then RAISE_APPLICATION_ERROR (-20500,'ERR: VIOLATION DE CONTRAINTE!');
        END IF;
    ELSIF UPDATING('CLIENT_NO') THEN
        RAISE_APPLICATION_ERROR (-20500,'ERR: VOUS NE POUVEZ CHANGER LE
        CLIENT!');
    END IF;
END;
/
```

TRIGGER SUR VUE: Exemple 1:

SQL> --Exemple de trigger pour gérer les interdictions sur une VUE

SQL>

SQL> create OR REPLACE view COM_CLIENT

2 (NUM_COM, NOM_CL, ADR_LIV, MONTANT_COM, DT_LIV_COM)

3 AS

4 SELECT

5 CO.NO AS NUM_COM,

6 CL.NOM AS NOM_CL,

7 CL.ADRESSE AS ADR_LIV,

8 CO.TOTAL AS MONTANT_COM,

9 CO.DATE_LIVRAISON AS DT_LIV_COM

10 FROM E_CLIENT CL, E_COMMANDE CO

11 WHERE CO.CLIENT_NO = CL.NO;

Vue créée.

SQL> CREATE SEQUENCE SEQ_ID_CLIENT START WITH 2000 ORDER;

Séquence créée.

TRIGGER SUR VUE: Exemple 1:

```
SQL> --CREATE SEQUENCE SEQ_ID_COM START WITH 2000 ORDER;
SQL>
SQL> CREATE OR REPLACE TRIGGER TRIG_INS_VUE_COM_CLIENT
 2  INSTEAD OF INSERT OR UPDATE OR DELETE ON COM_CLIENT
 3  FOR EACH ROW
 4  DECLARE
 5      NEW_ID_CL          NUMBER(5);
 6  BEGIN
 7      IF INSERTING THEN
 8          SELECT last_number INTO NEW_ID_CL
 9          FROM user_sequences WHERE sequence_name = 'SEQ_ID_CLIENT';
10
11          INSERT INTO E_CLIENT (NO, NOM, ADRESSE) VALUES (NEW_ID_CL,
                  :NEW.NOM_CL,:NEW.ADR_LIV );
12
13          INSERT INTO E_COMMANDE (NO, CLIENT_NO, TOTAL, DATE_LIVRAISON) VALUES
                  (:NEW.NUM_COM, NEW_ID_CL,:NEW.MONTANT_COM,:NEW.DT_LIV_COM);
14      END IF;
15  END;
16 /
```

Déclencheur créé.

```
SQL> --test
SQL> INSERT INTO COM_CLIENT VALUES (
2      111,
3      'ouldbah',
4      'chez lui',
5      15000,
6      '25/01/2013'
7 );
```

1 ligne créée.

```
SQL> commit;
```

Validation effectuée.

```
SQL>
```

```
SQL> SELECT * FROM E_COMMANDE WHERE NO=111;
```

NO	CLIENT_NO	DATE_COM	DATE_LIV	EMPLOYE_NO	TOTAL
111	2000	25/01/13		15000	


```
SQL> SELECT * FROM E_CLIENT WHERE NOM='ouldbah';
```

NO	NOM	PRENOM	TELEPHONE
2000	ouldbah	chez lui	

TRIGGER CLAUSE WHEN: Exemple 1:

```
SQL> CREATE OR REPLACE TRIGGER INS_DT_EMPS
2 BEFORE INSERT ON E_EMPLOYE
3 FOR EACH ROW WHEN (NEW.DT_ENTREE IS NULL)
4 BEGIN
5     :NEW.DT_ENTREE:=SYSDATE;
6 END;
7 /
```

Déclencheur créé.

SQL>

PAS DE ':' POUR LE OLD ET LE NEW DANS LA CLAUSE WHEN

REGLE DE GESTION: SI LA DATE N'EST DONNEE, ON PREND LA DATE DU JOUR

TRIGGER CLAUSE WHEN: Exemple 1:

```
SQL> --test
SQL> INSERT INTO E_EMPLOYE (NO, NOM, PRENOM, SALAIRE) VALUES
2 (      555,
3      'toto',
4      'titi',
5      6000
6 );
```

1 ligne créée.

```
SQL> commit;
```

Validation effectuée.

```
SQL>
SQL> SELECT NO, NOM, PRENOM, SALAIRE, DT_ENTREE FROM E_EMPLOYE WHERE NO=555;
```

NO	NOM	PRENOM	SALAIRE	DT_ENTRE
555	toto	titi	6000.00	18/01/13

SQL>

**TRAVAUX
PRATIQUES**

EXERCICE 1:

- Créer la table Trace_Salaire suivante :

Trace_Salaire(No_emp	Number,
	AncienSal	Number(7,2),
	NouveauSal	Number(7,2),
	Date_Modif	Date,
	Commentaire	Varchar(30)

);

Le commentaire concerne le type d'opération (UPDATE, INSERT, DELETE)

1. Créer un trigger qui permet de constituer l'historique de toutes les modifications apportées aux salaires des employés.
2. Créer un trigger qui met la date du jour de la création d'une ligne dans la colonne Dt_Entree de la table E_employe, pour chaque ligne, si Dt_Entree est à NULL lors de l'insertion.
3. Créer un trigger qui se déclenche sur la mise à jour de chaque ligne de la table E_employe et qui renseigne la table E_augmentation.

EXERCICE 2:

On considère le schéma suivant :

```
create table CAT_PROD(  
  NumCat          NUMBER(3) PRIMARY KEY,          --numéro de la catégorie du produit  
  PRIXACHATMAX    NUMBER(5,2),                    --Prix maximum du produit à l'achat  
  MARGEMAX        NUMBER(2)                      --Marge maximale à la vente  
);
```

```
create table PRODS(  
  NumPROD          Number(3) PRIMARY KEY,          --numéro du produit  
  CatPROD          Number(3) REFERENCES CAT_PROD(NumCat), --catégorie du produit  
  PrixACHAT        Number(5,2),                    --Prix d'achat effectif du produit  
  PrixVENTE        Number(5,2),                    --Prix de vente effective du produit  
);
```

2.1. Ecrire un trigger qui vérifie lors de la mise à jour du prix de vente d'un produit que le prix de vente ne dépasse pas la marge correspondant à la catégorie du produit, sinon, un message d'erreur doit être retourné et la mise à jour avortée. De même, dans ce cas, on doit archiver dans la table ARCH_PRODS les anciennes valeurs du produit et les nouvelles valeurs. Le trigger doit aussi vérifier lors d'une promotion, que le prix de vente n'est pas inférieur au prix d'achat et lors de l'achat du produit que le prix d'achat ne dépasse pas le prix maximum autorisé (PRIXACHATMAX)

La table ARCH_PRODS est comme suit :

```
create table arch_emps(  
  NumPROD      Number(3),          -- numéro du produit concerné par la mise à jour  
  CatPROD      Number(3),          -- la catégorie du produit  
  PrixAchat    Number(5,2),        --le prix effectif d'achat du produit  
  OldPrixVente NUMBER(3),          --l'ancien prix de vente  
  NewPrixVente NUMBER(3),          --Nouveau prix de vente  
);
```


2.2. Ecrire, un trigger qui, si le prix de vente n'est pas donné lors de l'insertion, calcule automatiquement ce prix selon la catégorie du produit et la marge maximale et l'insère dans le tuple à insérer.

2.3. Créer une vue, contenant : NumCAT, PRIXACHATMAX, NumPROD, PrixAchat, PrixVente.

2.4.Ecrire un trigger qui autorise l'insertion et la suppression à travers la vue selon la sémantique suivante :

- a. Si NumCat=NULL→ERREUR OU PrixAchat=NULL→ERRUR SINON le prix de vente peut être calculé comme précédemment et le numéro de produit par une séquence.**
- b. Lors d'une suppression, seul le produit sera supprimé et archivé.**

FIN