5. Sous programmes : Procédures et Fonctions

a. Les procédures : En PL/SQL, on définit la syntaxe d'une procédure de la sorte :

```
CREATE OR REPLACE PROCEDURE /* nom */ (/* parametres */) IS
        /* déclaration  des variables  locales */
BEGIN
        /* instructions */
END ;
```

Les paramètres sont une simple liste des couples : **NOM TYPE. Une procédure commence par un en-tête qui spécifie son nom et une liste de paramètres facultatifs. Chaque paramètre peut être en mode IN, OUT ou INOUT. Le mode du paramètre spécifie si un paramètre peut être lu ou écrit.**

- **IN : Un paramètre IN est en lecture seule. Vous pouvez faire référence à un paramètre IN dans une procédure, mais vous ne pouvez pas modifier sa valeur. Oracle utilise IN comme mode par défaut. Cela signifie que si vous ne spécifiez pas explicitement le mode pour un paramètre, Oracle utilisera le mode IN.**
- **OUT : Un paramètre OUT est accessible en écriture. En général, vous définissez une valeur de retour pour le paramètre OUT et la renvoyez au programme appelant. Notez qu'une procédure ignore la valeur que vous fournissez pour un paramètre OUT.**
- **INOUT : Un paramètre INOUT est à la fois accessible en lecture et en écriture.**

- **Récursivité** : Une **procédure ou fonction** est dite récursive si le corps du sous-programme (procédure ou fonction) contient un ou plusieurs appels à lui-même.
  Exemple d'une procédure récursive (compte à rebours) :

```
CREATE OR REPLACE PROCEDURE compteARebours ( n NUMBER) IS
BEGIN
        IF n >= 0 THEN
        DBMS_OUTPUT.PUT_LINE (n);
        compteARebours (n – 1);
        END IF;
END;
```

- **Invocation** : En PL/SQL, une procédure s'invoque avec son nom. Mais sous SQL+, on doit utiliser le mot-clé **CALL**. **Exemple :** Sous SQL+ avec la commande : **CALL compteARebours (20).**

b. Les fonctions : En PL/SQL, on définit la syntaxe d'une fonction de la sorte :

```
CREATE OR REPLACE FUNCTION /* nom */ (/* parametres */) RETURN /*type*/ IS
        /* déclaration  des variables  locales */
BEGIN
        /* instructions */
END ;
```

Les paramètres sont une simple liste des couples : **NOM TYPE** et l'instruction **RETURN** sert à retourner une valeur.

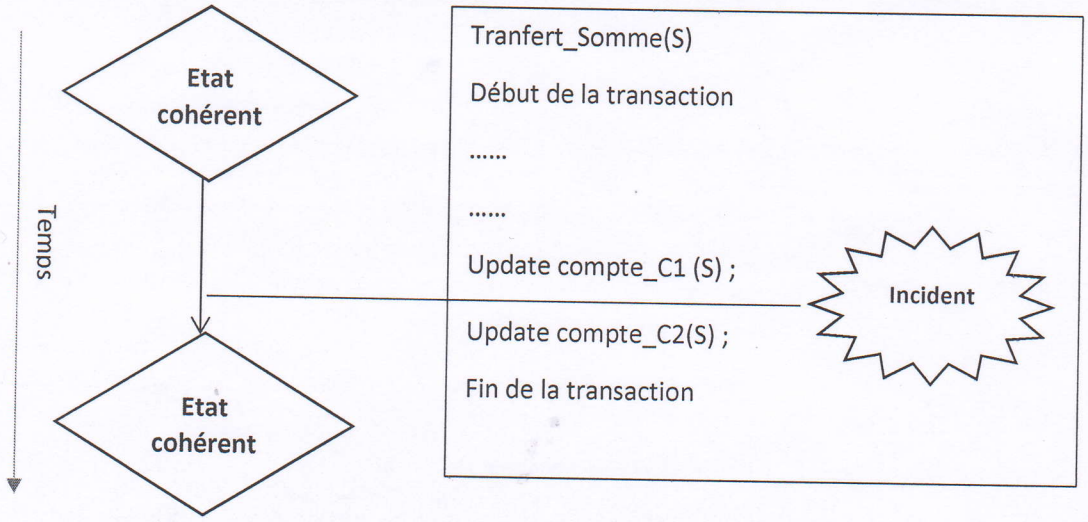
```
CREATE OR REPLACE FUNCTION module (a NUMBER, b NUMBER) RETURN NUMBER IS
BEGIN
        IF a < b THEN
            RETURN a;
        ELSE
            RETURN module (a – b, b);
        END IF;
END;
```

- **Invocation** : En PL/SQL, une fonction s'invoque aussi avec son nom comme le cas d'une procédure. Mais sous SQL+, on doit **passer par une pseudo-table nommée DUAL.**
  Exemple: SELECT module (21,12) FROM DUAL;

6. Transactions et validation des données

a. Définition

Une transaction est un ensemble d'opérations "atomiques" indivisible. Nous considèrerons qu'un ensemble d'opérations est indivisible si une exécution partielle de ces instructions poserait des problèmes d'intégrité dans la base de données. Une transaction est donc un bloc d'instructions faisant passer la BD d'un état cohérent à un autre état cohérent.
Par exemple, un virement d'un compte à compte se fait en deux phases : Créditer un compte C1 d'une somme **S** + Débiter un autre C2 de la même somme **S**. Si un problème logiciel ou matériel survient au cours de la transaction et que la transaction est interrompue, le virement est incomplet. Dans ce cas, aucune des instructions de la transaction ne devrait être effectuée.



Il convient donc de disposer d'un mécanisme permettant de se protéger de ce genre de désagrément. La grande majorité des transactions sous Oracle sont programmées en PL/SQL. Les langages plus évolués permettent de développer des transactions à travers des fonctions de leur API

b. Propriétés d'une transaction

En principe, une transaction assure les propriétés **ACID** :

- **Atomicité** des instructions qui sont considérées comme une seule opération (Le tout ou rien) ;
- **Cohérence** (passage d'un état cohérent de la base à un autre état cohérent) ;
- **Isolation** des transactions entre elles (lecture consistante) ;
- **Durabilité** des opérations (les MAJ perdurent même si une panne se produit après la transaction).

NB. On parle aussi des propriétés **BASE** dans le monde des bases de données NOSQL.

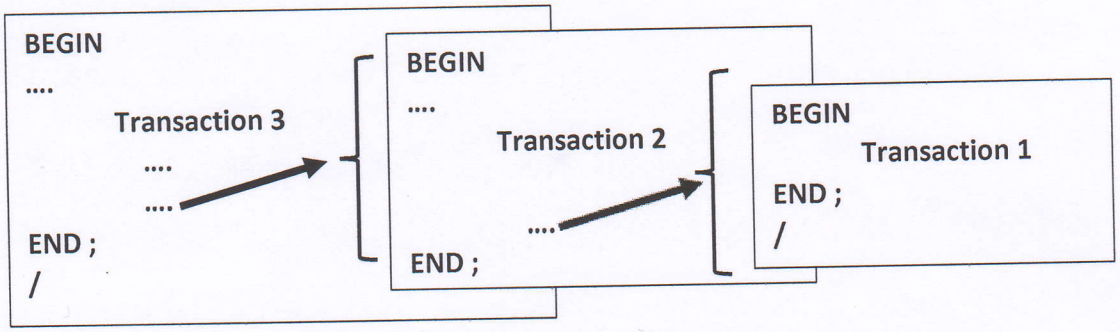
c. Validation des transactions

Afin de valider ou annuler une transaction, nous utilisons les instructions **COMMIT** et **ROLLBACK** : Le ROLLBACK annule toutes les modifications faites depuis le début de la transaction et COMMIT les enregistre définitivement dans la BD. Il faut signaler également que la variable d'environnement

AUTOCOMMIT permet d'activer (ON ou OFF) la gestion des transactions. Si elle est positionnée à ON, chaque instruction a des répercussions immédiates dans la base, sinon, les modifications ne sont effectives qu'une fois qu'un COMMIT a été exécuté.

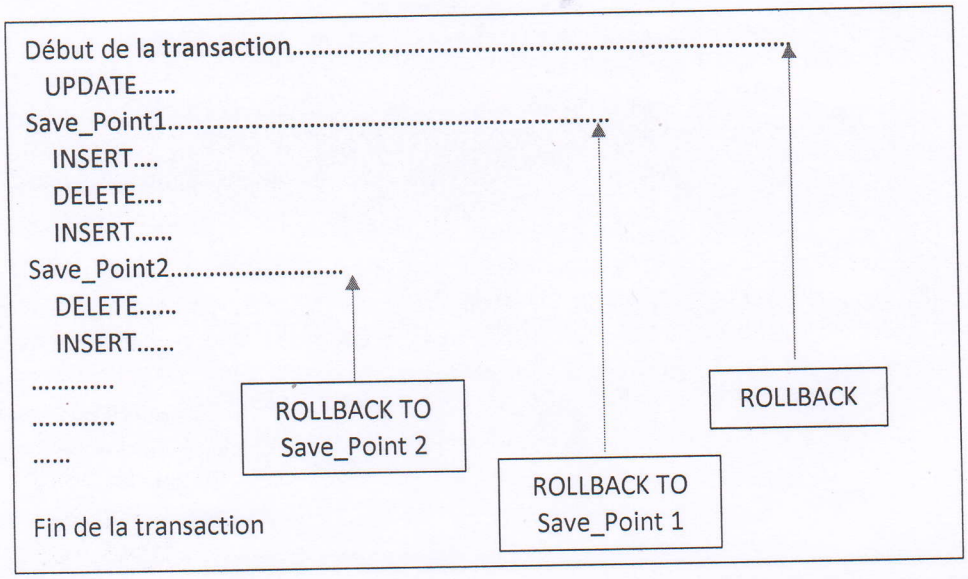
```
    /*instructions*/
IF  /*erreur*/  THEN
        ROLLBACK;
ELSE
        COMMIT;
END;
```

Il est possible de programmer également plusieurs transactions se déroulant dans des blocs imbriqués tout en respectant Les mécanismes d'atomicité, de cohérence, d'isolation et de durabilité (ACID) comme l'illustre la figure suivante :



d. Points de validation des transactions (Save points)

Il est opportun de pouvoir découper une transaction en insérant des points de validation (**Savepoints**) qui rendent possible l'annulation de tout ou bien une partie des opérations d'une transaction. Le programmeur aura le choix donc entre les instructions **ROLLBACK TO** pour valider tout ou une partie de la transaction. Il faudra finalement se décider entre **COMMIT** et **ROLLBACK**. La figure suivante illustre une transaction découpée en trois parties.



L'instruction ROLLBACK peut s'écrire sous différentes formes. Ainsi : **ROLLBACK TO Save_Point1** : invalidera les INSERT et le DELETE tout en laissant la possibilité de valider l'instruction UPDATE.
L'instruction « **SAVEPOINT** » permet de fixer des points de sauvegarde selon la Syntaxe suivante : SAVEPOINT <nom_savepoint>.