Les blocs de code exécutables

I. Procédures :

1. Qu'est-ce qu'une procédure en PL/SQL ?

Une procédure en PL/SQL est un bloc de code qui effectue une tâche spécifique. Elle peut accepter des paramètres, exécuter des instructions SQL et retourner des valeurs.

2. Création d'une procédure

La syntaxe de base pour créer une procédure est la suivante :

```
CREATE OR REPLACE PROCEDURE nom_procedure (param1 TYPE, param2
TYPE) AS
BEGIN
    -- Instructions SQL
END nom_procedure;
```

3. Définition des paramètres

- IN : Paramètre d'entrée (par défaut).
- OUT : Paramètre de sortie.
- IN OUT : Paramètre d'entrée et de sortie.

4. Exemple de création d'une procédure

Voici un exemple simple qui calcule le carré d'un nombre :

```
CREATE OR REPLACE PROCEDURE calculer_carre (nombre IN NUMBER,
resultat OUT NUMBER) AS
BEGIN
    resultat := nombre * nombre;
END calculer_carre;
```

5. Appel d'une procédure

Pour appeler une procédure, utilisez la syntaxe suivante :

```
DECLARE
    v_resultat NUMBER;
BEGIN
    calculer_carre(5, v_resultat);
    DBMS_OUTPUT.PUT_LINE('Le carré est : ' || v_resultat);
END;
```

6. Exemples supplémentaires

- Procédure avec plusieurs paramètres :

```
CREATE OR REPLACE PROCEDURE ajouter_nombres (a IN NUMBER, b IN
NUMBER, somme OUT NUMBER) AS
BEGIN
    somme := a + b;
END ajouter_nombres;
```

- Appel de la procédure :

```
DECLARE
    v_somme NUMBER;
BEGIN
    ajouter_nombres(10, 20, v_somme);
    DBMS_OUTPUT.PUT_LINE('La somme est : ' || v_somme);
END;
```

II. Les fonctions :

1. Qu'est-ce qu'une fonction en PL/SQL ?

Une fonction en PL/SQL est un bloc de code qui effectue une tâche spécifique et retourne une valeur. Contrairement aux procédures, les fonctions doivent toujours retourner une valeur.

2. Création d'une fonction

La syntaxe de base pour créer une fonction est la suivante :

```
CREATE OR REPLACE FUNCTION nom_fonction (param1 TYPE, param2 TYPE)
RETURN TYPE AS
BEGIN
    -- Instructions SQL
    RETURN valeur;
END nom_fonction;
```

3. Définition des paramètres

Les paramètres peuvent être définis comme suit :

- IN : Paramètre d'entrée (par défaut).
- OUT : Non utilisé dans les fonctions, car elles retournent une valeur.
- IN OUT : Non utilisé dans les fonctions, car elles retournent une valeur.

4. Exemple de création d'une fonction

Voici un exemple simple qui calcule le carré d'un nombre :

```
CREATE OR REPLACE FUNCTION calculer_carre (nombre IN NUMBER) RETURN
NUMBER AS
BEGIN
    RETURN nombre * nombre;
END calculer_carre;
```

5. Appel d'une fonction

Pour appeler une fonction, utilisez la syntaxe suivante :

```
DECLARE
    v_resultat NUMBER;
BEGIN
    v_resultat := calculer_carre(5);
    DBMS_OUTPUT.PUT_LINE('Le carré est : ' || v_resultat);
END;
```

6. Exemples supplémentaires

- Fonction avec plusieurs paramètres :

```
CREATE OR REPLACE FUNCTION ajouter_nombres (a IN NUMBER, b IN NUMBER)
RETURN NUMBER AS
BEGIN
    RETURN a + b;
END ajouter_nombres;
```

- Appel de la fonction :

```
DECLARE
    v_somme NUMBER;
BEGIN
    v_somme := ajouter_nombres(10, 20);
    DBMS_OUTPUT.PUT_LINE('La somme est : ' || v_somme);
END;
```

III. Les Trigger

1. Qu'est-ce qu'un trigger ?

Un trigger est un bloc de code PL/SQL qui s'exécute automatiquement en réponse à certains événements sur une table ou une vue. Les triggers sont utilisés pour maintenir l'intégrité des données, effectuer des audits, ou automatiser des actions.

2. Types de triggers

- **DML** Triggers : S'exécutent lors des opérations **DML** (**INSERT, UPDATE, DELETE**).
- **DDL** Triggers : S'exécutent lors des opérations **DDL** (**CREATE, ALTER, DROP**).
- **LOGON/LOGOFF** Triggers : S'exécutent lors des connexions ou déconnexions d'un utilisateur.

3. Création d'un trigger

La syntaxe de base pour créer un trigger est la suivante :

```
CREATE OR REPLACE TRIGGER nom_trigger
{BEFORE | AFTER} {INSERT | UPDATE | DELETE} ON nom_table
FOR EACH ROW
BEGIN
    -- Instructions PL/SQL
END nom_trigger;
```

4. Exemple de création d'un trigger DML

Voici un exemple de trigger qui enregistre les modifications sur une table `employes` :

```
CREATE OR REPLACE TRIGGER trg_audit_employes
AFTER INSERT OR UPDATE OR DELETE ON employes
FOR EACH ROW
BEGIN
    INSERT INTO audit_employes (action, employe_id, date_action)
    VALUES (CASE
                WHEN INSERTING THEN 'INSERT'
                WHEN UPDATING THEN 'UPDATE'
                WHEN DELETING THEN 'DELETE'
            END,
            :NEW.id,
            SYSDATE);
END trg_audit_employes;
```

5. Appel d'un trigger

Les triggers s'exécutent automatiquement en réponse à l'événement spécifié. Par exemple, lorsque tu insères, mets à jour ou supprime un enregistrement dans la table `employes`, le trigger `trg_audit_employes` s'exécute automatiquement.

6. Exemples supplémentaires

- Trigger pour valider des données :

```
CREATE OR REPLACE TRIGGER trg_valider_salaire
BEFORE INSERT OR UPDATE ON employes
FOR EACH ROW
BEGIN
    IF :NEW.salaire < 0 THEN
        RAISE_APPLICATION_ERROR(-20001, 'Le salaire ne peut pas être
négatif.');
    END IF;
END trg_valider_salaire;
```

- Trigger pour mettre à jour une colonne :

```
CREATE OR REPLACE TRIGGER trg_mise_a_jour_date
BEFORE UPDATE ON employes
FOR EACH ROW
BEGIN
    :NEW.date_modification := SYSDATE;
END trg_mise_a_jour_date;
```

IV. Les packages

1. Qu'est-ce qu'un package ?

Un package en PL/SQL est un conteneur qui regroupe des objets liés, tels que des procédures, des fonctions, des variables, des types, et des curseurs. Les packages permettent d'organiser le code, de le rendre modulaire et de faciliter la réutilisation.

2. Composants d'un package

Un package se compose de deux parties :

- Spécification du package : Déclare les objets publics (procédures, fonctions, variables) accessibles à l'extérieur.
- Corps du package : Contient l'implémentation des objets déclarés dans la spécification.

3. Création d'un package

La syntaxe de base pour créer un package est la suivante :

Spécification :

```
CREATE OR REPLACE PACKAGE nom_package AS
    -- Déclarations des objets publics
END nom_package;
```

Corps :

```
CREATE OR REPLACE PACKAGE BODY nom_package AS
    -- Implémentations des objets
END nom_package;
```

4. Exemple de création d'un package

Voici un exemple simple d'un package qui gère des opérations sur des employés :

Spécification :

```
CREATE OR REPLACE PACKAGE pkg_employes AS
    PROCEDURE ajouter_employe(nom IN VARCHAR2, salaire IN NUMBER);
    FUNCTION obtenir_salaire(id IN NUMBER) RETURN NUMBER;
END pkg_employes;
```

Corps :

```
CREATE OR REPLACE PACKAGE BODY pkg_employes AS
    PROCEDURE ajouter_employe(nom IN VARCHAR2, salaire IN NUMBER) IS
    BEGIN
        INSERT INTO employes (nom, salaire) VALUES (nom, salaire);
    END ajouter_employe;

    FUNCTION obtenir_salaire(id IN NUMBER) RETURN NUMBER IS
        v_salaire NUMBER;
    BEGIN
        SELECT salaire INTO v_salaire FROM employes WHERE id = id;
        RETURN v_salaire;
    END obtenir_salaire;
END pkg_employes;
```

5. Appel d'un package

Pour appeler les procédures et fonctions d'un package, utilise la syntaxe suivante :

```
BEGIN
    pkg_employes.ajouter_employe('Alice', 50000);
    DBMS_OUTPUT.PUT_LINE('Salaire d\'Alice : ' ||
pkg_employes.obtenir_salaire(1));
END;
```

6. Avantages des packages

- Modularité : Regroupe des objets liés, facilitant la gestion du code.
- Encapsulation : Cache les détails d'implémentation, exposant uniquement les objets nécessaires.
- Performance : Charge le package en mémoire une seule fois, améliorant les performances lors des appels.

V. SQL Dynamique

1. Introduction

Le SQL statique est précompilé, ce qui signifie que sa structure est connue et validée à la compilation. Cependant, le SQL dynamique permet d'écrire des instructions SQL qui ne peuvent pas être déterminées à l'avance.

Pourquoi utiliser le SQL dynamique ?

- Gérer des tables ou colonnes dynamiques.
- Exécuter des requêtes personnalisées en fonction des besoins utilisateurs.
- Construire des utilitaires génériques (comme des scripts de migration).

2. Méthodes d'exécution

En PL/SQL, le SQL dynamique peut être implémenté via deux mécanismes principaux :

2.1. `EXECUTE IMMEDIATE`

La méthode la plus simple et recommandée pour exécuter des instructions SQL dynamiques. Elle prend en charge toutes les opérations SQL (à l'exception des commandes `SELECT INTO` avec plusieurs lignes).

Syntaxe de base :

```
EXECUTE IMMEDIATE '<instruction SQL>';
```

Avec liaison de variables :

```
EXECUTE IMMEDIATE '<instruction SQL>'
```

```
USING <valeur1>, <valeur2>, ...;
```

Exemple :

```
DECLARE
    v_table_name VARCHAR2(50) := 'EMPLOYEES';
    v_sql        VARCHAR2(200);
BEGIN
    v_sql := 'CREATE TABLE ' || v_table_name || ' (ID NUMBER, NAME
VARCHAR2(100))';
    EXECUTE IMMEDIATE v_sql;
END;
```

2.2. DBMS_SQL

Une alternative plus complexe mais plus puissante. DBMS_SQL est utile pour exécuter des instructions dynamiques qui nécessitent un traitement plus poussé, comme des curseurs dynamiques ou des instructions complexes.

Exemple :

```
DECLARE
    v_cursor  INTEGER;
    v_status  INTEGER;
    v_sql     VARCHAR2(200);
BEGIN
    v_sql := 'CREATE TABLE DYNAMIC_TABLE (ID NUMBER, NAME
VARCHAR2(100))';
    v_cursor := DBMS_SQL.OPEN_CURSOR;
    DBMS_SQL.PARSE(v_cursor, v_sql, DBMS_SQL.NATIVE);
    v_status := DBMS_SQL.EXECUTE(v_cursor);
    DBMS_SQL.CLOSE_CURSOR(v_cursor);
END;
```

3. Manipulation dynamique des données

3.1. Exécution de requêtes DML

Les commandes comme INSERT, UPDATE ou DELETE peuvent être exécutées dynamiquement.

Exemple :

```
DECLARE
    v_table_name VARCHAR2(50) := 'EMPLOYEES';
    v_sql        VARCHAR2(200);
BEGIN
    v_sql := 'INSERT INTO ' || v_table_name || ' (ID, NAME) VALUES (1,
''John Doe'')';
    EXECUTE IMMEDIATE v_sql;
END;
```

3.2. Exécution de requêtes SELECT

Pour exécuter des requêtes SELECT dynamiques, `EXECUTE IMMEDIATE` est utilisé avec `INTO` pour stocker les résultats.

Exemple :

```
DECLARE
    v_table_name VARCHAR2(50) := 'EMPLOYEES';
    v_sql        VARCHAR2(200);
    v_name       VARCHAR2(100);
BEGIN
    v_sql := 'SELECT NAME FROM ' || v_table_name || ' WHERE ID = 1';
    EXECUTE IMMEDIATE v_sql INTO v_name;
    DBMS_OUTPUT.PUT_LINE('Name: ' || v_name);
END;
```

4. Bonnes pratiques

1. **Validation des entrées utilisateur** : Protégez votre code contre les injections SQL.
    - Utilisez des paramètres liés dans `EXECUTE IMMEDIATE`.
    - Exemple :

```
EXECUTE IMMEDIATE 'SELECT NAME FROM EMPLOYEES WHERE ID = :1' INTO v_name
USING v_id;
```

2. **Gestion des exceptions** : Ajoutez des blocs `EXCEPTION` pour gérer les erreurs.

    Exemple :

```
BEGIN
    EXECUTE IMMEDIATE 'DROP TABLE NON_EXISTENT_TABLE';
EXCEPTION
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('Erreur : ' || SQLERRM);
END;
```

3. **Privilèges minimaux** : Assurez-vous que l'utilisateur dispose des privilèges nécessaires pour exécuter le SQL dynamique.
4. **Debugging** : Ajoutez des journaux pour suivre l'exécution.
    - Utilisez `DBMS_OUTPUT.PUT_LINE` ou des tables de log pour enregistrer les commandes SQL exécutées.

5. Cas d'utilisation avancés

5.1. Gestion de structures dynamiques

Création ou modification de tables à la volée en fonction des besoins.

Exemple :

```
DECLARE
    v_sql VARCHAR2(200);
BEGIN
    v_sql := 'ALTER TABLE EMPLOYEES ADD (AGE NUMBER)';
    EXECUTE IMMEDIATE v_sql;
END;
```

5.2. Curseurs dynamiques avec DBMS_SQL

Pour exécuter des instructions avec un nombre inconnu de colonnes ou de paramètres.

Exemple :

```
DECLARE
    v_cursor  INTEGER;
    v_sql     VARCHAR2(200);
    v_columns VARCHAR2(200);
    v_desc    DBMS_SQL.DESC_TAB;
    v_col_cnt INTEGER;
BEGIN
    v_sql := 'SELECT * FROM EMPLOYEES';
    v_cursor := DBMS_SQL.OPEN_CURSOR;
    DBMS_SQL.PARSE(v_cursor, v_sql, DBMS_SQL.NATIVE);

    -- Décrire les colonnes
    DBMS_SQL.DESCRIBE_COLUMNS(v_cursor, v_col_cnt, v_desc);
    DBMS_OUTPUT.PUT_LINE('Nombre de colonnes : ' || v_col_cnt);

    DBMS_SQL.CLOSE_CURSOR(v_cursor);
END;
```

6. Limitations

- Performance : Les instructions dynamiques sont compilées à l'exécution, ce qui peut être plus lent.

- Lisibilité : Le code peut devenir difficile à lire et à maintenir.
- Gestion des droits : L'exécution dynamique peut être bloquée si les privilèges ne sont pas correctement configurés.