Cours PL/SQL

Programmation Oracle

Pr. Lamia ZIAD
Ecole Superieure de Technologie d'Essaouira

Contents

Chapter 1

Introduction a PL/SQL

Definition

PL/SQL (Procedural Language/Structured Query Language) est le langage de programmation procedurale d'Oracle. Il etend SQL avec des structures de controle, des boucles, des variables et bien plus.

1.1 Architecture PL/SQL

- **Bloc anonyme** : Execute immediatement, non stocke
- **Procedures** : Sous-programmes stockes dans la base
- **Fonctions** : Retournent une valeur
- **Packages** : Groupement de procedures et fonctions
- **Triggers** : Declenches automatiquement par evenements

1.2 Structure d'un Bloc PL/SQL

Important

Un bloc PL/SQL se compose de trois parties : DECLARE, BEGIN, EXCEPTION.

Exemple

```
DECLARE
    -- Section declaration (optionnelle)
    v_nombre NUMBER := 10;
    v_nom VARCHAR2(50);
BEGIN
    -- Section executable (obligatoire)
    v_nom := 'Jean Dupont';
    DBMS_OUTPUT.PUT_LINE('Bonjour ' || v_nom);
    DBMS_OUTPUT.PUT_LINE('Nombre: ' || v_nombre);
EXCEPTION
    -- Section exception (optionnelle)
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('Erreur: ' || SQLERRM);
END;
/
```

1.3 Avantages de PL/SQL

- **Performance** : Execution groupee reduit les appels reseau
- **Productivite** : Integration directe avec SQL
- **Portabilite** : Fonctionne sur toutes les plateformes Oracle
- **Securite** : Gestion des privileges via la base

Chapter 2

Variables et Types de Donnees

2.1 Declaration de Variables

Important

Syntaxe : nom_variable [CONSTANT] type [NOT NULL] [:= valeur];

Exemple

```
DECLARE
    v_salaire NUMBER(8,2) := 2500.50;
    v_nom VARCHAR2(50) NOT NULL := 'Martin';
    c_taux CONSTANT NUMBER := 1.20;
    v_date DATE DEFAULT SYSDATE;
    v_actif BOOLEAN := TRUE;
BEGIN
    DBMS_OUTPUT.PUT_LINE('Salaire: ' || v_salaire);
    DBMS_OUTPUT.PUT_LINE('Nom: ' || v_nom);
END;
/
```

2.2 Types de Donnees Scalarres

Type	Description
NUMBER	Numerique (precision, echelle)
VARCHAR2	Chaine variable (max 32767 en PL/SQL)
CHAR	Chaine fixe
DATE	Date et heure
BOOLEAN	Vrai/Faux/NULL
BINARY_INTEGER	Entier signe

2.3 Anchored Datatypes

Important

Utilisez %TYPE pour lier le type d'une variable a une colonne de table.

Exemple

```
DECLARE
    v_emp_name employees.last_name%TYPE;
    v_emp_salary employees.salary%TYPE;
    v_dept_name departments.department_name%TYPE;
BEGIN
    SELECT last_name, salary INTO v_emp_name, v_emp_salary
    FROM employees WHERE employee_id = 100;

    DBMS_OUTPUT.PUT_LINE(v_emp_name || ' gagne ' || v_emp_salary);
END;
/
```

2.4 Types Composites

Exemple

```
-- Record Type
DECLARE
    TYPE emp_record IS RECORD (
        id employees.employee_id%TYPE,
        nom employees.last_name%TYPE,
        salaire employees.salary%TYPE
    );
    v_emp emp_record;

-- Table Type
    TYPE nom_table IS TABLE OF VARCHAR2(50) INDEX BY BINARY_INTEGER;
    v_noms nom_table;
BEGIN
    v_emp.id := 100;
    v_emp.nom := 'Dupont';
    v_emp.salaire := 5000;

    v_noms(1) := 'Alice';
    v_noms(2) := 'Bob';
END;
/
```

Chapter 3

Structures de Controle

3.1 Structure IF-THEN-ELSE

Important

Permet l'execution conditionnelle de blocs de code.

Exemple

```
DECLARE
    v_salaire NUMBER := 3000;
    v_grade VARCHAR2(20);
BEGIN
    IF v_salaire > 5000 THEN
        v_grade := 'A';
    ELSIF v_salaire BETWEEN 3000 AND 5000 THEN
        v_grade := 'B';
    ELSE
        v_grade := 'C';
    END IF;

    DBMS_OUTPUT.PUT_LINE('Grade: ' || v_grade);
END;
/
```

3.2 Boucle LOOP Basique

Exemple

```
DECLARE
    v_compteur NUMBER := 1;
BEGIN
    LOOP
        DBMS_OUTPUT.PUT_LINE('Iteration: ' || v_compteur);
        v_compteur := v_compteur + 1;
        EXIT WHEN v_compteur > 5;
    END LOOP;
END;
/
```

3.3 Boucle WHILE

Exemple

```
DECLARE
    v_compteur NUMBER := 1;
BEGIN
    WHILE v_compteur <= 5 LOOP
        DBMS_OUTPUT.PUT_LINE('Compteur: ' || v_compteur);
        v_compteur := v_compteur + 1;
    END LOOP;
END;
/
```

3.4 Boucle FOR

Important

La boucle FOR est ideal quand le nombre d'iterations est connu.

Exemple

```
BEGIN
    -- FOR avec compteur
    FOR i IN 1..5 LOOP
        DBMS_OUTPUT.PUT_LINE('Iteration: ' || i);
    END LOOP;

    -- FOR reverse
    FOR i IN REVERSE 1..5 LOOP
        DBMS_OUTPUT.PUT_LINE('Reverse: ' || i);
    END LOOP;
END;
/
```

3.5 Boucle FOR avec Cursor

Exemple

```
BEGIN
    FOR emp IN (SELECT employee_id, last_name, salary
                FROM employees
                WHERE department_id = 50)
    LOOP
        DBMS_OUTPUT.PUT_LINE(emp.last_name || ' : ' || emp.salary);
    END LOOP;
END;
/
```

Chapter 4

Les Cursors

Definition

Un cursor est un pointeur vers une zone memoire contenant le resultat d'une requete SQL. Il permet de traiter les lignes une par une.

4.1 Cursor Implicite

Important

Oracle cree automatiquement un cursor implicite pour les instructions SQL.

Exemple

```
BEGIN
    UPDATE employees SET salary = salary * 1.1
    WHERE department_id = 50;

    DBMS_OUTPUT.PUT_LINE(SQL%ROWCOUNT || ' lignes mises a jour');

    IF SQL%FOUND THEN
        DBMS_OUTPUT.PUT_LINE('Mise a jour reussie');
    END IF;
END;
/
```

4.2 Cursor Explicite

Important

Les cursors explicites donnent plus de controle sur le traitement des resultats.

Exemple

```
DECLARE
    CURSOR c_emp IS
        SELECT employee_id, last_name, salary
        FROM employees
        WHERE department_id = 50;

    v_emp_id employees.employee_id%TYPE;
    v_nom employees.last_name%TYPE;
    v_salaire employees.salary%TYPE;
BEGIN
    OPEN c_emp;

    LOOP
        FETCH c_emp INTO v_emp_id, v_nom, v_salaire;
        EXIT WHEN c_emp%NOTFOUND;

        DBMS_OUTPUT.PUT_LINE(v_emp_id || ' - ' || v_nom || ' - ' ||
    v_salaire);
    END LOOP;

    CLOSE c_emp;
END;
/
```

4.3 Cursor avec Parametres

Exemple

```
DECLARE
    CURSOR c_emp(p_dept_id NUMBER) IS
        SELECT last_name, salary
        FROM employees
        WHERE department_id = p_dept_id;

    v_nom employees.last_name%TYPE;
    v_salaire employees.salary%TYPE;
BEGIN
    OPEN c_emp(50);  -- Departement 50

    LOOP
        FETCH c_emp INTO v_nom, v_salaire;
        EXIT WHEN c_emp%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE(v_nom || ' : ' || v_salaire);
    END LOOP;

    CLOSE c_emp;
END;
/
```

4.4 Cursor FOR Loop

Important

Simplifie l'utilisation des cursors avec gestion automatique.

Exemple

```
DECLARE
    CURSOR c_emp IS
        SELECT last_name, salary, department_id
        FROM employees
        WHERE salary > 5000;
BEGIN
    FOR emp_rec IN c_emp LOOP
        DBMS_OUTPUT.PUT_LINE(
            emp_rec.last_name || ' dans departement ' ||
            emp_rec.department_id || ' gagne ' || emp_rec.salary
        );
    END LOOP;
END;
/
```

Chapter 5

Gestion des Exceptions

Definition

Une exception est une erreur qui survient pendant l'execution du programme. PL/SQL permet de capturer et traiter ces erreurs gracieusement.

5.1 Exceptions Predefinies

Exception	**Description**
NO_DATA_FOUND	SELECT INTO ne retourne aucune ligne
TOO_MANY_ROWS	SELECT INTO retourne plusieurs lignes
DUP_VAL_ON_INDEX	Violation de contrainte d'unicite
INVALID_NUMBER	Conversion numerique invalide
ZERO_DIVIDE	Division par zero

Exemple

```
DECLARE
    v_emp_name employees.last_name%TYPE;
    v_salary employees.salary%TYPE;
BEGIN
    SELECT last_name, salary INTO v_emp_name, v_salary
    FROM employees WHERE employee_id = 9999; -- ID inexistant

EXCEPTION
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE('Aucun employe trouve avec cet ID');
    WHEN TOO_MANY_ROWS THEN
        DBMS_OUTPUT.PUT_LINE('Plusieurs employes trouves');
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('Erreur: ' || SQLCODE || ' - ' ||
   SQLERRM);
END;
/
```

5.2 Exceptions Utilisateur

Important

Vous pouvez definir vos propres exceptions pour gerer des cas specifiques.

Exemple

```
DECLARE
    e_salaire_trop_bas EXCEPTION;
    PRAGMA EXCEPTION_INIT(e_salaire_trop_bas, -20001);
    v_salaire employees.salary%TYPE := 800;
BEGIN
    IF v_salaire < 1000 THEN
        RAISE e_salaire_trop_bas;
    END IF;

EXCEPTION
    WHEN e_salaire_trop_bas THEN
        DBMS_OUTPUT.PUT_LINE('Erreur: Le salaire est trop bas');
        DBMS_OUTPUT.PUT_LINE('Le salaire minimum est 1000');
END;
/
```

5.3 RAISE_APPLICATION_ERROR

Exemple

```
DECLARE
    v_count NUMBER;
BEGIN
    SELECT COUNT(*) INTO v_count
    FROM employees WHERE department_id = 99;

    IF v_count = 0 THEN
        RAISE_APPLICATION_ERROR(-20002,
            'Le departement 99 n existe pas');
    END IF;

EXCEPTION
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('Code erreur: ' || SQLCODE);
        DBMS_OUTPUT.PUT_LINE('Message: ' || SQLERRM);
END;
/
```

Chapter 6

Procedures et Fonctions

6.1 Creation de Procedure

Important

Les procedures sont des sous-programmes qui executent une action sans retourner de valeur.

Exemple

```
CREATE OR REPLACE PROCEDURE augmenter_salaire (
    p_emp_id IN employees.employee_id%TYPE,
    p_pourcentage IN NUMBER
) IS
    v_ancien_salaire employees.salary%TYPE;
    v_nouveau_salaire employees.salary%TYPE;
BEGIN
    -- Recuperer l'ancien salaire
    SELECT salary INTO v_ancien_salaire
    FROM employees WHERE employee_id = p_emp_id;

    -- Calculer le nouveau salaire
    v_nouveau_salaire := v_ancien_salaire * (1 + p_pourcentage/100);

    -- Mettre a jour
    UPDATE employees SET salary = v_nouveau_salaire
    WHERE employee_id = p_emp_id;

    COMMIT;

    DBMS_OUTPUT.PUT_LINE('Salaire de ' || p_emp_id ||
        ' augmente de ' || p_pourcentage || '%');

EXCEPTION
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE('Employe non trouve');
    WHEN OTHERS THEN
        ROLLBACK;
        RAISE;
END augmenter_salaire;
/
```

6.2 Appel d'une Procedure

Exemple

```
-- Appel de la procedure
BEGIN
    augmenter_salaire(100, 10); -- Augmentation de 10% pour l'employe
    100
END;
/

-- Ou directement
EXECUTE augmenter_salaire(101, 15);
```

6.3 Creation de Fonction

Important

Les fonctions retournent une valeur et peuvent etre utilisees dans des expressions SQL.

Exemple

```
CREATE OR REPLACE FUNCTION calculer_salaire_annuel (
    p_emp_id IN employees.employee_id%TYPE
) RETURN NUMBER IS
    v_salaire_mensuel employees.salary%TYPE;
    v_commission employees.commission_pct%TYPE;
    v_salaire_annuel NUMBER;
BEGIN
    SELECT salary, NVL(commission_pct, 0)
    INTO v_salaire_mensuel, v_commission
    FROM employees WHERE employee_id = p_emp_id;

    v_salaire_annuel := (v_salaire_mensuel * 12) * (1 + v_commission)
    ;

    RETURN v_salaire_annuel;

EXCEPTION
    WHEN NO_DATA_FOUND THEN
        RETURN NULL;
    WHEN OTHERS THEN
        RAISE;
END calculer_salaire_annuel;
/
```

6.4 Utilisation d'une Fonction

Exemple

```
-- Appel dans un bloc PL/SQL
DECLARE
    v_salaire_annuel NUMBER;
BEGIN
    v_salaire_annuel := calculer_salaire_annuel(100);
    DBMS_OUTPUT.PUT_LINE('Salaire annuel: ' || v_salaire_annuel);
END;
/

-- Utilisation dans une requete SQL
SELECT employee_id, last_name,
       calculer_salaire_annuel(employee_id) as salaire_annuel
FROM employees
WHERE department_id = 50;
```

Chapter 7

Les Packages

Definition

Un package est un regroupement logique de procedures, fonctions, variables et cursors. Il se compose d'une specification (interface) et d'un body (implementation).

7.1 Specification du Package

Exemple

```
CREATE OR REPLACE PACKAGE gestion_employes AS
    -- Variables publiques
    g_taux_augmentation CONSTANT NUMBER := 1.1;

    -- Cursor public
    CURSOR c_emp_par_dept(p_dept_id NUMBER) RETURN employees%ROWTYPE;

    -- Procedures
    PROCEDURE afficher_employes_dept(p_dept_id NUMBER);
    PROCEDURE augmenter_salaire_dept(p_dept_id NUMBER, p_pourcentage
   NUMBER);

    -- Fonctions
    FUNCTION compter_employes_dept(p_dept_id NUMBER) RETURN NUMBER;
    FUNCTION salaire_moyen_dept(p_dept_id NUMBER) RETURN NUMBER;

END gestion_employes;
/
```

7.2 Body du Package

Exemple

```
CREATE OR REPLACE PACKAGE BODY gestion_employes AS

    -- Implementation du cursor
    CURSOR c_emp_par_dept(p_dept_id NUMBER) RETURN employees%ROWTYPE
    IS
        SELECT * FROM employees
        WHERE department_id = p_dept_id;

    -- Procedure pour afficher les employes
    PROCEDURE afficher_employes_dept(p_dept_id NUMBER) IS
    BEGIN
        FOR emp IN c_emp_par_dept(p_dept_id) LOOP
            DBMS_OUTPUT.PUT_LINE(
                emp.employee_id || ' - ' ||
                emp.last_name || ' - ' ||
                emp.salary
            );
        END LOOP;
    END afficher_employes_dept;

    -- Procedure pour augmenter les salaires
    PROCEDURE augmenter_salaire_dept(p_dept_id NUMBER, p_pourcentage
    NUMBER) IS
    BEGIN
        UPDATE employees
        SET salary = salary * (1 + p_pourcentage/100)
        WHERE department_id = p_dept_id;
        COMMIT;
    END augmenter_salaire_dept;

    -- Fonction pour compter les employes
    FUNCTION compter_employes_dept(p_dept_id NUMBER) RETURN NUMBER IS
        v_count NUMBER;
    BEGIN
        SELECT COUNT(*) INTO v_count
        FROM employees WHERE department_id = p_dept_id;
        RETURN v_count;
    END compter_employes_dept;

    -- Fonction pour le salaire moyen
    FUNCTION salaire_moyen_dept(p_dept_id NUMBER) RETURN NUMBER IS
        v_moyenne NUMBER;
    BEGIN
        SELECT AVG(salary) INTO v_moyenne
        FROM employees WHERE department_id = p_dept_id;
```

7.3 Utilisation du Package

Exemple

```
BEGIN
    -- Utiliser les fonctions et procedures du package
    DBMS_OUTPUT.PUT_LINE('Nombre employes dept 50: ' ||
        gestion_employes.compter_employes_dept(50));

    DBMS_OUTPUT.PUT_LINE('Salaire moyen dept 50: ' ||
        gestion_employes.salaire_moyen_dept(50));

    -- Afficher les employes
    gestion_employes.afficher_employes_dept(50);

    -- Augmenter les salaires
    gestion_employes.augmenter_salaire_dept(50, 10);
END;
/
```

Chapter 8

Les Triggers

Definition

Un trigger (declencheur) est un bloc PL/SQL qui s'execute automatiquement lorsqu'un evenement specifique se produit sur une table ou une vue.

8.1 Trigger sur INSERT

Exemple

```
CREATE OR REPLACE TRIGGER tr_audit_insert_emp
    BEFORE INSERT ON employees
    FOR EACH ROW
DECLARE
    v_username VARCHAR2(30);
BEGIN
    -- Recuperer l'utilisateur courant
    SELECT USER INTO v_username FROM DUAL;

    -- Inserer dans la table d'audit
    INSERT INTO audit_employees (
        audit_id, operation, user_name,
        employee_id, timestamp
    ) VALUES (
        audit_seq.NEXTVAL, 'INSERT', v_username,
        :NEW.employee_id, SYSDATE
    );
END;
/
```

8.2 Trigger sur UPDATE

Exemple

```
CREATE OR REPLACE TRIGGER tr_verif_salaire
    BEFORE UPDATE OF salary ON employees
    FOR EACH ROW
BEGIN
    -- Verifier que le nouveau salaire est superieur a l'ancien
    IF :NEW.salary < :OLD.salary THEN
        RAISE_APPLICATION_ERROR(-20003,
            'Le nouveau salaire ne peut pas etre inferieur a l ancien
   ');
    END IF;

    -- Verifier le salaire minimum
    IF :NEW.salary < 1000 THEN
        RAISE_APPLICATION_ERROR(-20004,
            'Le salaire minimum est 1000');
    END IF;
END;
/
```

8.3 Trigger sur DELETE

Exemple

```
CREATE OR REPLACE TRIGGER tr_archiver_emp
    BEFORE DELETE ON employees
    FOR EACH ROW
BEGIN
    -- Archiver l'employe supprime
    INSERT INTO employees_archive (
        employee_id, last_name, email,
        hire_date, salary, delete_date
    ) VALUES (
        :OLD.employee_id, :OLD.last_name, :OLD.email,
        :OLD.hire_date, :OLD.salary, SYSDATE
    );
END;
/
```

8.4 Trigger d'Instruction

Important

Les triggers d'instruction s'executent une fois par instruction, pas par ligne.

Exemple

```
CREATE OR REPLACE TRIGGER tr_verif_operations
    BEFORE INSERT OR UPDATE OR DELETE ON employees
DECLARE
    v_jour VARCHAR2(10);
BEGIN
    -- Recuperer le jour de la semaine
    SELECT TO_CHAR(SYSDATE, 'DY') INTO v_jour FROM DUAL;

    -- Interdire les operations le weekend
    IF v_jour IN ('SAT', 'SUN') THEN
        RAISE_APPLICATION_ERROR(-20005,
            'Operations interdites le weekend');
    END IF;

    -- Interdire les operations apres 18h
    IF TO_NUMBER(TO_CHAR(SYSDATE, 'HH24')) > 18 THEN
        RAISE_APPLICATION_ERROR(-20006,
            'Operations interdites apres 18h');
    END IF;
END;
/
```

Chapter 9

Gestion des Transactions

9.1 COMMIT et ROLLBACK

Important

PL/SQL herite des commandes de transaction SQL : COMMIT, ROLLBACK, SAVEPOINT.

Exemple

```
DECLARE
    v_success BOOLEAN := TRUE;
BEGIN
    -- Point de sauvegarde
    SAVEPOINT avant_transaction;

    -- Premiere operation
    UPDATE comptes SET solde = solde - 1000
    WHERE numero_compte = 123;

    -- Deuxieme operation
    UPDATE comptes SET solde = solde + 1000
    WHERE numero_compte = 456;

    -- Simulation d'une verification
    IF v_success THEN
        COMMIT;
        DBMS_OUTPUT.PUT_LINE('Transaction validee');
    ELSE
        ROLLBACK TO avant_transaction;
        DBMS_OUTPUT.PUT_LINE('Transaction annulee');
    END IF;

EXCEPTION
    WHEN OTHERS THEN
        ROLLBACK TO avant_transaction;
        DBMS_OUTPUT.PUT_LINE('Erreur - Transaction annulee: ' ||
   SQLERRM);
END;
/
```

9.2 AUTONOMOUS_TRANSACTION

Important

Permet d'executer une transaction independante dans une procedure ou fonction.

Exemple

```
CREATE OR REPLACE PROCEDURE journaliser_operation (
    p_message IN VARCHAR2
) IS
    PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
    INSERT INTO journal_operations (
        id, message, date_operation, utilisateur
    ) VALUES (
        journal_seq.NEXTVAL, p_message, SYSDATE, USER
    );

    COMMIT; -- Commit independant de la transaction principale
END journaliser_operation;
/

-- Utilisation
BEGIN
    UPDATE employes SET salaire = salaire * 1.1;

    -- Journalisation qui commit independamment
    journaliser_operation('Augmentation generale des salaires de 10%'
   );

    -- Le commit principal n'affecte pas la journalisation
    -- et vice versa
    ROLLBACK; -- N'annule pas la journalisation
END;
/
```

Chapter 10

SQL Dynamique

Definition

Le SQL dynamique permet de construire et executer des instructions SQL a l'execution.

10.1 EXECUTE IMMEDIATE

Important

EXECUTE IMMEDIATE est utilise pour executer des instructions SQL dynamiques.

Exemple

```
DECLARE
    v_table_name VARCHAR2(30) := 'employees';
    v_column_name VARCHAR2(30) := 'salary';
    v_emp_id NUMBER := 100;
    v_value NUMBER;
    v_sql VARCHAR2(1000);
BEGIN
    -- Construction dynamique de la requete
    v_sql := 'SELECT ' || v_column_name ||
             ' FROM ' || v_table_name ||
             ' WHERE employee_id = :1';

    -- Execution avec parametre
    EXECUTE IMMEDIATE v_sql INTO v_value USING v_emp_id;

    DBMS_OUTPUT.PUT_LINE('Valeur: ' || v_value);
END;
/
```

10.2 Dynamic SQL avec DDL

Exemple

```
CREATE OR REPLACE PROCEDURE creer_table_backup (
    p_table_name IN VARCHAR2
) IS
    v_sql VARCHAR2(1000);
BEGIN
    -- Construire la commande DDL dynamique
    v_sql := 'CREATE TABLE ' || p_table_name || '_backup AS ' ||
             'SELECT * FROM ' || p_table_name || ' WHERE 1=0';

    -- Executer la DDL
    EXECUTE IMMEDIATE v_sql;

    DBMS_OUTPUT.PUT_LINE('Table backup creee: ' || p_table_name || '
   _backup');
END creer_table_backup;
/

-- Utilisation
BEGIN
    creer_table_backup('employees');
END;
/
```

10.3 DBMS_SQL Package

Important

DBMS_SQL offre plus de flexibilite pour le SQL dynamique complexe.

Exemple

```
DECLARE
    v_cursor INTEGER;
    v_rows_processed INTEGER;
    v_emp_id employees.employee_id%TYPE;
    v_last_name employees.last_name%TYPE;
BEGIN
    -- Ouvrir un cursor
    v_cursor := DBMS_SQL.OPEN_CURSOR;

    -- Parse la requete
    DBMS_SQL.PARSE(v_cursor,
        'SELECT employee_id, last_name FROM employees WHERE
   department_id = 50',
        DBMS_SQL.NATIVE);

    -- Definir les colonnes de retour
    DBMS_SQL.DEFINE_COLUMN(v_cursor, 1, v_emp_id);
    DBMS_SQL.DEFINE_COLUMN(v_cursor, 2, v_last_name, 50);

    -- Executer
    v_rows_processed := DBMS_SQL.EXECUTE(v_cursor);

    -- Parcourir les resultats
    LOOP
        EXIT WHEN DBMS_SQL.FETCH_ROWS(v_cursor) = 0;
        DBMS_SQL.COLUMN_VALUE(v_cursor, 1, v_emp_id);
        DBMS_SQL.COLUMN_VALUE(v_cursor, 2, v_last_name);
        DBMS_OUTPUT.PUT_LINE(v_emp_id || ' - ' || v_last_name);
    END LOOP;

    -- Fermer le cursor
    DBMS_SQL.CLOSE_CURSOR(v_cursor);
END;
/
```