Parfait 💪
On développe maintenant **le Chapitre 6 en détail**, comme si tu préparais l'examen.

✅ CHAPITRE 6 : Travailler avec les types de données composites

(Types composés en PL/SQL)

🔲ntroduction aux types composites

1. Définition

Un **type composite** est un type de donnée qui contient **plusieurs valeurs** regroupées sous une seule structure.

Contrairement aux types scalaires :

Type scalaire	Type composite
NUMBER	RECORD
VARCHAR2	TABLE INDEX BY
DATE	%ROWTYPE

---

2. Pourquoi utiliser les types composites ?

✓ Regrouper plusieurs informations liées
✓ Manipuler une ligne complète d'une table
✓ Stocker temporairement des données
✓ Éviter de déclarer plusieurs variables séparées

Exemple simple :

Au lieu de :

v_id NUMBER;

v_name VARCHAR2(30);

v_salary NUMBER;

On peut utiliser un RECORD.

🗂️ Les RECORD

---

1. RECORD personnalisé

- **Définition**

Un RECORD est une structure qui contient plusieurs champs.

Syntaxe :

```
TYPE nom_type IS RECORD (
  champ1 datatype,
  champ2 datatype,
  champ3 datatype
);

variable nom_type;
```

- **Exemple complet**

```
DECLARE
  TYPE emp_record IS RECORD (
    v_id NUMBER,
    v_name VARCHAR2(30),
    v_salary NUMBER
  );

  emp emp_record;
BEGIN
  emp.v_id := 100;
  emp.v_name := 'Bradley';
  emp.v_salary := 5000;
```

```
  DBMS_OUTPUT.PUT_LINE(emp.v_name);
END;
/
```

2️⃣ RECORD basé sur une table : %ROWTYPE

🔹 Définition

%ROWTYPE permet de créer un RECORD basé sur la structure complète d'une table.

Syntaxe :

variable table_name%ROWTYPE;

🔹 Exemple :

```
DECLARE
  v_emp EMPLOYEES%ROWTYPE;
BEGIN
  SELECT *
  INTO v_emp
  FROM EMPLOYEES
  WHERE EMPLOYEE_ID = 100;

  DBMS_OUTPUT.PUT_LINE(v_emp.FIRST_NAME);
END;
/
```

🎯 Avantage principal

Si la structure de la table change → ton code ne change pas.

Très important à l'examen !

3️⃣ %TYPE

🔹 **Définition**

%TYPE permet de déclarer une variable en utilisant le type d'une colonne.

Exemple :

```
DECLARE
  v_salary EMPLOYEES.SALARY%TYPE;
BEGIN
  SELECT SALARY
  INTO v_salary
  FROM EMPLOYEES
  WHERE EMPLOYEE_ID = 100;
END;
/
```

🎯 **Pourquoi utiliser %TYPE ?**

✓ Évite les erreurs de type
✓ Suit automatiquement les modifications de la table

II Les tables INDEX BY (Tableaux associatifs)

1️⃣ **Définition**

Une table INDEX BY est un tableau en mémoire (temporaire).

Elle stocke plusieurs valeurs du même type.

---

2 Syntaxe

```
TYPE table_type IS TABLE OF datatype
INDEX BY BINARY_INTEGER;

variable table_type;
```

3️⃣ Exemple simple

```
DECLARE
  TYPE salary_table IS TABLE OF NUMBER
  INDEX BY BINARY_INTEGER;

  v_salaries salary_table;
BEGIN
  v_salaries(1) := 1000;
  v_salaries(2) := 2000;
  v_salaries(3) := 3000;

  DBMS_OUTPUT.PUT_LINE(v_salaries(2));
END;
/
```

4️⃣ Parcourir une table INDEX BY

```
FOR i IN 1..3 LOOP
  DBMS_OUTPUT.PUT_LINE(v_salaries(i));
END LOOP;
```

IV️⃣ Table d’enregistrements INDEX BY

Très important pour examen.

On combine :

- RECORD
- INDEX BY

- **Exemple complet**

```
DECLARE
  TYPE dept_record IS RECORD (
    v_id NUMBER,
    v_name VARCHAR2(30),
    v_location NUMBER
  );

  TYPE dept_table IS TABLE OF dept_record
  INDEX BY BINARY_INTEGER;

  v_depts dept_table;

BEGIN
  v_depts(1).v_id := 10;
  v_depts(1).v_name := 'Administration';
  v_depts(1).v_location := 1700;

  DBMS_OUTPUT.PUT_LINE(v_depts(1).v_name);
END;
/
```

✅ Différences importantes (souvent demandées)

Élément	Rôle
%TYPE	Type d’une colonne
%ROWTYPE	Ligne complète d’une table
RECORD	Structure personnalisée
INDEX BY	Tableau temporaire

🎯 Ce que le prof peut demander à l'examen

1. Créer un RECORD personnalisé
2. Utiliser %ROWTYPE
3. Déclarer une table INDEX BY
4. Parcourir une table avec une boucle
5. Créer une table d'enregistrements

⚠️ Erreurs fréquentes

❌ Oublier SELECT INTO
❌ Mauvais index
❌ Accéder à un index non initialisé
❌ Confondre %TYPE et %ROWTYPE

📌 Résumé ultra important

Chapitre 6 =
👉 Manipuler des structures complexes
👉 Stocker plusieurs données
👉 Travailler comme avec des mini-objets en mémoire

Si tu veux 👇

Je peux maintenant te faire :

1️⃣ Exercices type examen corrigés du chapitre 6
2️⃣ Une fiche ultra synthétique en 2 pages
3️⃣ Un TP complet corrigé comme ton TP pratique
4️⃣ Les pièges possibles à l'examen

On continue 💪

Parfait 💪
On développe maintenant **le CHAPITRE 10 : Création de procédures** (très important pour l'examen 🔥).

✅ CHAPITRE 10 : Création de procédures

🔲ntroduction aux procédures

1. Définition

Une **procédure** est un programme PL/SQL stocké dans la base de données qui exécute une ou plusieurs instructions.

Elle peut :

- Insérer des données (INSERT)
- Modifier des données (UPDATE)
- Supprimer des données (DELETE)
- Interroger des données (SELECT INTO)

---

2. Pourquoi utiliser une procédure ?

✓ Réutilisation du code
✓ Sécurité (contrôle d'accès)
✓ Centralisation de la logique métier
✓ Meilleure organisation

🔲Syntaxe générale

```
CREATE OR REPLACE PROCEDURE nom_procedure
IS
BEGIN
  instructions;
END nom_procedure;
/
```

Exemple simple

```
CREATE OR REPLACE PROCEDURE hello
```

```
IS
BEGIN
  DBMS_OUTPUT.PUT_LINE('Bonjour');
END;
/
```

Pour exécuter :

```
EXEC hello;
```

II Les paramètres

Les procédures peuvent recevoir des paramètres.

1️⃣ Modes des paramètres

Mode	Rôle
IN	Entrée (valeur fournie)
OUT	Sortie (valeur retournée)
IN OUT	Entrée + sortie

🔹 Paramètre IN

```
CREATE OR REPLACE PROCEDURE afficher_nom(
  p_nom IN VARCHAR2
)
IS
BEGIN
  DBMS_OUTPUT.PUT_LINE(p_nom);
END;
/
```

- **Paramètre OUT**

```
CREATE OR REPLACE PROCEDURE get_salary(
  p_id IN NUMBER,
  p_salary OUT NUMBER
)
IS
BEGIN
  SELECT SALARY
  INTO p_salary
  FROM EMPLOYEES
  WHERE EMPLOYEE_ID = p_id;
END;
/
```

Appel :

```
VARIABLE g_salary NUMBER
EXEC get_salary(100, :g_salary)
PRINT g_salary
```

- **Paramètre IN OUT**

```
CREATE OR REPLACE PROCEDURE incrementer(
  p_nombre IN OUT NUMBER
)
IS
BEGIN
  p_nombre := p_nombre + 1;
END;
/
```

IV. Procédures avec DML

---

1. INSERT

```
CREATE OR REPLACE PROCEDURE add_job(
  p_id IN VARCHAR2,
  p_title IN VARCHAR2
)
IS
BEGIN
  INSERT INTO JOBS (JOB_ID, JOB_TITLE)
  VALUES (p_id, p_title);
END;
/
```

---

2. UPDATE

```
CREATE OR REPLACE PROCEDURE upd_job(
  p_id IN VARCHAR2,
  p_new_title IN VARCHAR2
)
IS
BEGIN
  UPDATE JOBS
  SET JOB_TITLE = p_new_title
  WHERE JOB_ID = p_id;
END;
/
```

---

3. DELETE

```
CREATE OR REPLACE PROCEDURE del_job(
  p_id IN VARCHAR2
)
IS
BEGIN
  DELETE FROM JOBS
  WHERE JOB_ID = p_id;
END;
/
```

4️⃣ SELECT INTO

⚠ Très important : SELECT INTO doit retourner UNE seule ligne.

Sinon :

- NO_DATA_FOUND
- TOO_MANY_ROWS

✅ Gestion des exceptions

Très important pour examen 🔥

1️⃣ Exceptions prédéfinies

```
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    DBMS_OUTPUT.PUT_LINE('Aucune donnée trouvée');
  WHEN TOO_MANY_ROWS THEN
    DBMS_OUTPUT.PUT_LINE('Plusieurs lignes trouvées');
```

2️⃣ Vérifier si UPDATE ou DELETE a réussi

Utiliser :

SQL%ROWCOUNT

Exemple :

```
IF SQL%ROWCOUNT = 0 THEN
  DBMS_OUTPUT.PUT_LINE('Aucune ligne modifiée');
END IF;
```

Exemple complet avec gestion d’erreur

```
CREATE OR REPLACE PROCEDURE upd_job(
  p_id IN VARCHAR2,
  p_new_title IN VARCHAR2
)
IS
BEGIN
  UPDATE JOBS
  SET JOB_TITLE = p_new_title
  WHERE JOB_ID = p_id;

  IF SQL%ROWCOUNT = 0 THEN
    RAISE_APPLICATION_ERROR(-20001, 'Job inexistant');
  END IF;

EXCEPTION
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('Erreur : ' || SQLERRM);
END;
/
```

VI. Compilation et erreurs

Après création :

SHOW ERRORS

Voir erreurs :

SELECT * FROM USER_ERRORS;

VI Appel d'une procédure

---

1 Avec EXEC

EXEC add_job('IT_DBA','Database Administrator');

---

2 Dans un bloc PL/SQL

```
BEGIN
  add_job('IT_DBA','Database Administrator');
END;
/
```

VII Différence Procédure vs Fonction

Procédure	Fonction
Peut ne rien retourner	Doit retourner une valeur
Utilise OUT	Utilise RETURN
Appelée avec EXEC	Utilisable dans SELECT

Ce que le prof peut demander

1. Créer une procédure avec paramètres
2. Ajouter gestion d'exception
3. Utiliser SQL%ROWCOUNT
4. Faire un SELECT INTO

5. Expliquer IN / OUT / IN OUT

⚠️ Pièges fréquents

❌ Oublier le "/"
❌ Oublier SELECT INTO
❌ Mauvais mode paramètre
❌ Oublier RETURN dans une fonction (chap 11)

📌 Résumé clé examen

Chapitre 10 =
👉 Créer des programmes stockés
👉 Manipuler INSERT/UPDATE/DELETE
👉 Gérer les erreurs
👉 Utiliser paramètres

Si tu veux maintenant 👇

On fait :

1️⃣ Chapitre 11 (fonctions)
2️⃣ Exercices type examen chapitre 10
3️⃣ TP complet corrigé
4️⃣ Simulation d'examen

On continue 💪

Parfait 💪
On termine avec le **CHAPITRE 11 : Création de fonctions**
(Pages 143 à 150 – très stratégique pour l'examen 🔥)

✅ CHAPITRE 11 : Création de fonctions

Introduction aux fonctions

1. Définition

Une **fonction** est un programme PL/SQL stocké dans la base de données qui :

✓ Effectue un traitement
✓ **Retourne obligatoirement une valeur**

👉 Contrairement à une procédure, une fonction doit toujours utiliser RETURN.

---

2. Pourquoi utiliser une fonction ?

✓ Calculer une valeur
✓ Retourner un résultat
✓ Être utilisée dans une requête SQL
✓ Réutiliser une logique métier

Différence entre procédure et fonction

Procédure	Fonction
Peut ne rien retourner	Doit retourner une valeur
Utilise OUT	Utilise RETURN
Appelée avec EXEC	Peut être utilisée dans SELECT
Action (ex : INSERT)	Calcul (ex : salaire annuel)

Syntaxe générale

```
CREATE OR REPLACE FUNCTION nom_fonction
RETURN datatype
IS
BEGIN
  RETURN valeur;
END nom_fonction;
/
```

Exemple simple

```
CREATE OR REPLACE FUNCTION dire_bonjour
RETURN VARCHAR2
IS
BEGIN
  RETURN 'Bonjour Bradley';
END;
/
```

Appel :

```
SELECT dire_bonjour FROM dual;
```

IV Fonction avec paramètre

Exemple : Calcul TVA (16%)

```
CREATE OR REPLACE FUNCTION calcul_tva(
  p_montant IN NUMBER
)
RETURN NUMBER
IS
  v_tva NUMBER;
BEGIN
  v_tva := p_montant * 0.16;
  RETURN v_tva;
END;
/
```

Appel :

```
SELECT calcul_tva(1000) FROM dual;
```

Résultat : 160

✅ Fonction avec SELECT INTO

Très important 🔥

Exemple : Retourner le salaire d'un employé

```
CREATE OR REPLACE FUNCTION get_salary(
  p_id IN NUMBER
)
RETURN NUMBER
IS
  v_salary NUMBER;
BEGIN
  SELECT SALARY
  INTO v_salary
  FROM EMPLOYEES
  WHERE EMPLOYEE_ID = p_id;

  RETURN v_salary;

EXCEPTION
  WHEN NO_DATA_FOUND THEN
    RETURN 0;
END;
/
```

✅ Gestion des exceptions

Une fonction doit toujours retourner quelque chose.

⚠ Même en cas d'erreur !

Exceptions importantes

- NO_DATA_FOUND
- TOO_MANY_ROWS
- ZERO_DIVIDE
- WHEN OTHERS

Exemple avec gestion complète

```
CREATE OR REPLACE FUNCTION division(
  a NUMBER,
  b NUMBER
)
RETURN NUMBER
IS
BEGIN
  RETURN a / b;

EXCEPTION
  WHEN ZERO_DIVIDE THEN
    RETURN 0;
END;
/
```

V Utilisation d'une fonction

---

1 Dans un bloc PL/SQL

```
DECLARE
  v_result NUMBER;
BEGIN
```

```
  v_result := calcul_tva(2000);
  DBMS_OUTPUT.PUT_LINE(v_result);
END;
/
```

2️⃣ Dans une requête SQL

Très puissant 🔥

```
SELECT EMPLOYEE_ID,
    get_salary(EMPLOYEE_ID)
FROM EMPLOYEES;
```

VII Règles importantes à l'examen

✓ Une fonction doit obligatoirement avoir RETURN
✓ Le type après RETURN doit correspondre
✓ SELECT INTO doit retourner une seule ligne
✓ Toujours gérer les exceptions

IX Erreurs fréquentes

❌ Oublier RETURN
❌ Mauvais type de retour
❌ Ne pas gérer NO_DATA_FOUND
❌ Utiliser OUT (interdit dans SQL pur)

🎯 Ce que le prof peut demander

1. Créer une fonction qui calcule la TVA
2. Créer une fonction qui retourne un salaire
3. Utiliser SELECT INTO
4. Ajouter gestion d'exception
5. Expliquer différence procédure / fonction

📌 **Résumé stratégique examen**

Chapitre 11 =

👉 Retourner une valeur
👉 Utiliser RETURN
👉 Peut être utilisée dans SELECT
👉 Toujours gérer les exceptions

🔥 **Maintenant tu maîtrises :**

- Chapitre 6 → Types composites
- Chapitre 10 → Procédures
- Chapitre 11 → Fonctions

Si tu veux maintenant 👇

1️⃣ Simulation complète d'examen type UPC corrigée
2️⃣ Fiche ultra synthétique des 3 chapitres
3️⃣ Série d'exercices pièges corrigés
4️⃣ Questions théoriques probables

On passe au niveau supérieur 💪