1. **Quels sont les avantages des sous-programmes (procédures ou fonctions stockées) par rapport aux autres techniques permettant de créer des programmes accédant aux BD?**
2. **Réutilisabilité : les sous-programmes sont des blocs de code réutilisables qui peuvent être appelés à partir de différents endroits dans le code. Cela permet de réduire la duplication de code et de faciliter la maintenance.**
3. **Sécurité : les sous-programmes peuvent être configurés pour n’autoriser que certains utilisateurs ou groupes d’utilisateurs à y accéder. Cela permet de limiter l'accès non autorisé aux données et d'améliorer la sécurité globale de l'application.**
4. **Performance : les sous-programmes sont souvent exécutés côté serveur, ce qui réduit la quantité de données échangées entre le serveur et le client. Cela peut améliorer considérablement les performances de l'application, surtout si les données sont volumineuses.**
5. **Modularité : les sous-programmes sont des blocs de code indépendants qui peuvent être modifiés et testés séparément du reste de l'application. Cela facilite la maintenance et le développement d'applications évolutives**
6. **Abstraction : les sous-programmes permettent d'abstraire les détails de mise en œuvre de la base de données. Cela facilite le changement de la base de données sous-jacente sans avoir à modifier l'ensemble de l'application**
7. **Supposons la table client (no number(5), nom varchar(20)) qui contient déjà certaines données. comment peut-on ajouter la contrainte clé primaire sur l'attribut 'no' en récupérant les données qui violent la contrainte à ajouter. Donner et expliquer les commandes nécessaires pour chaque étape.**
8. **Créer une nouvelle table temporaire avec la même structure que la table client mais sans la contrainte de clé primaire :**

**CREATE TABLE client\_temp AS**

**SELECT \***

**FROM client;**

1. **Supprimer la table client :**

**DROP TABLE client;**

1. **Recréer la table client avec la contrainte de clé primaire sur l’attribut ‘no’ :**

**CREATE TABLE client (**

**no NUMBER(5) PRIMARY KEY,**

**nom VARCHAR(20)**

**);**

1. **Insérer les données de la table temporaire dans la nouvelle table client :**

**INSERT INTO client (no, nom)**

**SELECT no, nom**

**FROM client\_temp;**

1. **Si certaines données violent la contrainte de clé primaire, elles ne seront pas insérées et un message d'erreur sera affiché. Pour récupérer ces données, il suffit d'exécuter la requête suivante**

**SELECT no, nom**

**FROM client\_temp**

**WHERE no NOT IN (SELECT no FROM client);**

**Cette requête renvoie toutes les données de la table temporaire qui ne sont pas présentes dans la nouvelle table client, c'est-à-dire les données qui violent la contrainte de clé primaire**

1. **Supprimer la table temporaire**

**DROP TABLE client\_temp;**

1. **Il existe plusieurs façons d'implémenter la contrainte "la valeur de l'attribut no\_dept doit être comprise entre 10 et 80" dans une table de base de données. Voici trois méthodes possibles avec les commandes et codes associés.**
2. **Utilisation de la clause CHECK :**

**CREATE TABLE emp (**

**id INT PRIMARY KEY,**

**nom VARCHAR(50),**

**no\_dept INT CHECK (no\_dept BETWEEN 10 AND 80)**

**);**

1. **Utilisation de la clause CONSTRAINT :**

**CREATE TABLE emp (**

**id INT PRIMARY KEY,**

**nom VARCHAR(50),**

**no\_dept INT,**

**CONSTRAINT chk\_dept CHECK (no\_dept BETWEEN 10 AND 80)**

**);**

1. **Utilisation de la clause ALTER TABLE :**

**ALTER TABLE emp**

**ADD CONSTRAINT chk\_dept CHECK (no\_dept BETWEEN 10 AND 80);**

1. **La table Client (numClient, nomClient, numRepresentant, NomRepresentant) n'est pas prête à être implémentée. Expliquer le pourquoi ? Quelles sont les solutions.**
2. **Manque de clés étrangères : Créer la table manquante.**
3. **Incohérence de données : Normaliser les données.**
4. **Mauvaise conception du schéma : Repenser le schéma.**
5. **Implémenter une procédure stockée qui nettoies l'espace occupé par tous les utilisateurs dont le login commence par 'Mar'. Pour nettoyer l'espace d'un utilisateur, on supprime tous ses objectes. Pour simplifier, on va utiliser la commande 'drop user user\_name cascade'. Par exemple la commande 'drop user toto cascade' supprime l'utilisateur "toto" en supprimant tous ses objects. En suite, on va recréer cet utilisateur 'create user toto indentified by password'.(password sera le même que login). La table all\_user (USERNAME, USER\_ID, CREATED) donne tous les utilisateurs existants. L'attribut username correspond au login.**

**DELIMITER $$**

**CREATE PROCEDURE clean\_users()**

**BEGIN**

**DECLARE user\_name VARCHAR(50);**

**DECLARE done INT DEFAULT FALSE;**

**DECLARE cur CURSOR FOR SELECT USERNAME FROM all\_user WHERE USERNAME LIKE 'Mar%';**

**DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;**

**OPEN cur;**

**read\_loop: LOOP**

**FETCH cur INTO user\_name;**

**IF done THEN**

**LEAVE read\_loop;**

**END IF;**

**SET @drop\_query = CONCAT('DROP USER ', user\_name, ' CASCADE');**

**PREPARE drop\_stmt FROM @drop\_query;**

**EXECUTE drop\_stmt;**

**DEALLOCATE PREPARE drop\_stmt;**

**SET @create\_query = CONCAT('CREATE USER ', user\_name, ' IDENTIFIED BY ', user\_name);**

**PREPARE create\_stmt FROM @create\_query;**

**EXECUTE create\_stmt;**

**DEALLOCATE PREPARE create\_stmt;**

**END LOOP;**

**CLOSE cur;**

**END$$**

**DELIMITER ;**

1. **Supposons le triggeur et le contenu de la table client (no, nom, salaire) suivants :**

**Create Trigger traitement\_client**

**Before update on client**

**Declare**

**err1 exception : a number ; b number ;**

**Begin**

**if(to\_char(sysdate, ‘d’) = 1) then raise err1;**

**endif;**

**Dbms\_output.put\_line(‘client traité’)**

**Exception when err1 when**

**Dbms\_output.put\_line(‘exception’)**

**End;**

**/**

**Table client**

|  |  |  |
| --- | --- | --- |
| **No** | **Nom** | **Salaire** |
| **10** | **Toto** | **1000** |
| **11** | **Titi** | **2000** |
| **12** | **Tata** | **1500** |
| **13** | **Matin** | **1000** |
| **14** | **Slyvie** | **2000** |

**Supposons que l’on execute la commande « update client set salaire = salaire + 100 when no > 12 ».**

**Si nous sommes lundi, combien de fois s’affiche le message ‘client traité’ ? Expliquer le pourquoi ?**

**Si nous sommes mardi, combien de fois s’affiche le message ‘client traité’ ? Expliquer le pourquoi ?**

**Si nous sommes lundi, le message ‘client traité’ ne s’affichera pas du tout car le trigger lève une exception lorsque la date du système est un lundi (c'est-à-dire lorsque to\_char(sysdate, ‘d’) = 1). Ainsi, la mise à jour de la table ne sera pas effectuée du tout et le message d'erreur "exception" sera affiché à la place.**

**Si nous sommes mardi, le message ‘client traité’ s'affichera pour chaque ligne de la table qui satisfait la condition no > 12, soit pour les lignes correspondant aux numéros de clients 13 et 14. La mise à jour sera effectuée normalement, car le trigger ne lève pas d'exception les jours autres que les lundis. Ainsi, le message "client traité" sera affiché pour chaque ligne qui a été mise à jour avec succès.**

**7.Supposons Ia table emp(id emp, nom, nobal, salaire) et la table batiment(id bal, nom, adresse ), 7.1.Ilmplementer le triggeur: Si un employe est affeché a un baltiment qui n'existe pas dans la table batiment, ce baltiment doit etre rajouté automatiquement avec pour valeur null pour les atributs nom et adresse. Si le nouveau salaire est inferieur à l'ancien salaire, il faudrait lui atribuer 《1000 》 comme salaire.L 'utilisateur "DUPONT' n'a pas le droit d'insérer des donnees dans la table emp.**

CREATE OR REPLACE TRIGGER emp\_trg

BEFORE INSERT ON emp

FOR EACH ROW

DECLARE

v\_nom batiment.nom%TYPE;

v\_adresse batiment.adresse%TYPE;

BEGIN

-- Vérifier si l'employé est affecté à un bâtiment qui n'existe pas dans la table batiment

IF :NEW.nobal NOT IN (SELECT id\_bal FROM batiment) THEN

-- Insérer le nouveau bâtiment avec des valeurs NULL pour nom et adresse

INSERT INTO batiment (id\_bal, nom, adresse) VALUES (:NEW.nobal, NULL, NULL);

END IF;

-- Vérifier si le nouveau salaire est inférieur à l'ancien salaire

IF :NEW.salaire < :OLD.salaire THEN

-- Affecter 1000 comme salaire

:NEW.salaire := 1000;

END IF;

-- Vérifier si l'utilisateur est DUPONT

IF USER = 'DUPONT' THEN

-- Annuler l'insertion en levant une erreur

raise\_application\_error(-20001, 'Vous n''êtes pas autorisé à insérer des données dans la table emp');

END IF;

END;

**7.2 Implementer le triggeur : Les inscriptions de nom sont irréversibles (impossible de modifier un nom introduit,un nom peut être NULL si on n'introduit pas de nom).**

CREATE OR REPLACE TRIGGER emp\_nom\_trg

BEFORE UPDATE ON emp

FOR EACH ROW

BEGIN

-- Vérifier si le nom a été modifié

IF :OLD.nom <> :NEW.nom THEN

-- Vérifier si le nouveau nom est NULL

IF :NEW.nom IS NULL THEN

-- Annuler la mise à jour en levant une erreur

raise\_application\_error(-20001, 'Le nom ne peut pas être NULL');

ELSE

-- Annuler la mise à jour en rétablissant l'ancien nom

:NEW.nom := :OLD.nom;

END IF;

END IF;

END;

**7.3 Supposons que la table TotalSal (montant) qui ne comporte au'une seul ligne avec la valeur 0 au depart existe pour stocker asomme des salaires des employees a tout moment.Implementer un triger qui actualise automatiouement la ligne de cette table.**

**Note) Attention au probleme de la table en mutation ! On ne peut pas uitiliser la commande 'select sur la table associee (meme une vue lié à cette table ) dans un triager. donc on ne peut pas faire 《 select ...From emp 》dans un trigger associé à la table emp .)**

CREATE OR REPLACE TRIGGER total\_sal\_trg

AFTER INSERT OR UPDATE OR DELETE ON emp

DECLARE

v\_total\_sal NUMBER;

BEGIN

SELECT SUM(salaire) INTO v\_total\_sal FROM emp;

UPDATE TotalSal SET montant = v\_total\_sal;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

INSERT INTO TotalSal (montant) VALUES (v\_total\_sal);

END;

**8.Supposons la table temp(id number10） avec les valeurs (1.4. 5). Donner le resultat affiche a lecran apres l’exécution de ce programme.**

**DECLARE**

**Outerid temp.id%type;**

**Innerid temp.id%type;**

**BEGIN**

**FOR i IN 1..4 LOOP**

**BEGIN**

**SELECT id INTO Outerid FROM temp WHERE id = i;**

**DBMS\_OUTPUT.PUT\_LINE('Outer i: ' || TO\_CHAR(i));**

**FOR j IN 4..5 LOOP**

**BEGIN**

**SELECT id INTO Innerid FROM temp WHERE id = j;**

**DBMS\_OUTPUT.PUT\_LINE('Inner j: ' || TO\_CHAR(j));**

**EXCEPTION**

**WHEN NO\_DATA\_FOUND THEN**

**DBMS\_OUTPUT.PUT\_LINE('Error inner j: ' || TO\_CHAR(j));**

**END;**

**END LOOP;**

**EXCEPTION**

**WHEN NO\_DATA\_FOUND THEN**

**DBMS\_OUTPUT.PUT\_LINE('Error outer i: ' || TO\_CHAR(i));**

**END;**

**END LOOP;**

**EXCEPTION**

**WHEN NO\_DATA\_FOUND THEN**

**DBMS\_OUTPUT.PUT\_LINE('No data found in temp table');**

**END;**

Ce code exécute deux boucles imbriquées. La boucle externe itère à travers les valeurs 1 à 4, et la boucle interne itère à travers les valeurs 4 et 5. Pour chaque valeur de la boucle externe, une requête SELECT est exécutée pour récupérer la valeur correspondante de la table "temp". Si la requête SELECT ne renvoie pas de résultats, une exception est levée et le message d'erreur correspondant est affiché à l'écran à l'aide de la fonction DBMS\_OUTPUT.PUT\_LINE. Si la requête SELECT réussit, la valeur de la boucle externe est affichée à l'écran.

Pour chaque valeur de la boucle interne, une requête SELECT est également exécutée pour récupérer la valeur correspondante de la table "temp". Si la requête SELECT ne renvoie pas de résultats, une exception est levée et le message d'erreur correspondant est affiché à l'écran. Si la requête SELECT réussit, la valeur de la boucle interne est affichée à l'écran.

Si aucune ligne n'est trouvée dans la table "temp", un message "No data found in temp table" sera affiché à l'écran.

**9.Supposons la table candidat (id, nom, age). Le programme pl/sql ci-dessous calcule la moyenne de l'age de tous les candidats dans la table candidat (age ne peut pas avoir la valeur null) . lndiquer les erreurs dans ce programme et corriger les s'il y en a.**

**Declare**

**Cursor age\_cursor is select age from candidat;**

**age\_total candidat.age%type:age\_moy candidat.age%type: temp candidat.age%type;num-number5);**

**Begin**

**age totalg= 0;numi= 0: age\_moyt= 0;**

**open age\_cursor;**

**Loop**

**Fetch age\_cursor into temp:age\_total := age\_total + temp,num:=num + 1;**

**Exit when (age\_cursor%found);**

**End Loop;**

**age\_moy:= age\_total / num;**

**Dbms\_output.put\_line ( age en moyenne: ' ll to\_char(age\_moy));**

**End;/**

DECLARE

CURSOR age\_cursor IS SELECT age FROM candidat;

age\_total candidat.age%TYPE := 0;

age\_moy candidat.age%TYPE := 0;

num NUMBER(5) := 0;

temp candidat.age%TYPE;

BEGIN

OPEN age\_cursor;

LOOP

FETCH age\_cursor INTO temp;

EXIT WHEN age\_cursor%NOTFOUND;

age\_total := age\_total + temp;

num := num + 1;

END LOOP;

CLOSE age\_cursor;

age\_moy := age\_total / num;

DBMS\_OUTPUT.PUT\_LINE('Age moyen: ' || TO\_CHAR(age\_moy));

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('Pas de données dans la table');

END;

La déclaration de la variable "age\_total" a été corrigée pour inclure le symbole ":" après le nom de la variable.

La déclaration de la variable "age\_moy" a été corrigée pour inclure le symbole ":" après le nom de la variable.

La déclaration de la variable "num" a été corrigée pour inclure le type de données "NUMBER(5)".

La variable "temp" a été déclarée séparément pour stocker temporairement les valeurs de la colonne "age" pendant la boucle FETCH.

Les instructions OPEN et CLOSE ont été ajoutées pour ouvrir et fermer le curseur "age\_cursor".

La condition d'arrêt de la boucle FETCH a été corrigée pour utiliser l'opérateur NOTFOUND plutôt que FOUND.

La clause EXCEPTION a été ajoutée pour gérer le cas où la table "candidat" est vide.

Le code récupère toutes les valeurs de la colonne "age" dans la table "candidat" à l'aide d'un curseur et les stocke temporairement dans la variable "temp". Il utilise ensuite une boucle pour ajouter toutes les valeurs ensemble et compter le nombre total de valeurs. Enfin, il calcule la moyenne des âges en divisant la somme totale par le nombre de valeurs et affiche le résultat à l'écran à l'aide de la fonction DBMS\_OUTPUT.PUT\_LINE. Si la table "candidat" est vide, un message "Pas de données dans la table" sera affiché à l'écran.