République Algérienne Démocratique et Populaire

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université Aboubekr BELKAÏD - TLEMCEN

Faculté des Sciences

Département de Informatique

TITRE

POLYCOPIÉ BASE DE DONNÉES AVANCÉES 1

-TRAVAUX DIRIGÉES-

Adressé aux étudiants niveau : Master 1 (SIC)

Domaine : Informatique

Filière : Informatique

Spécialité : Système d'Information et de connaissance

Etabli Par :

EL YEBDRI Zeyneb

Année : 2023

Avant-propos

Le présent polycopié **travaux dirigés de bases de données avancées 1** a été rédigé par Dr. Zeyneb EL YEBDRI, Maître de conférences à l'université de AbouBekr Belkaid, Tlemcen, et membre du laboratoire LRIT.

Ce polycopié de travaux dirigés s'adresse aux étudiants de master, plus particulièrement aux étudiants option SIC (système d'information et de connaissance) module Bases de données avancées 1, ainsi les étudiants d'option RSD (Réseaux et Systèmes Distribués), GL (Génie Logiciel) et MID (Modèles Intelligents et Décision) de la filière informatique au département d'informatique de l'université de Tlemcen.

Ainsi, le document est un support de référence très utile pour les étudiants qui veulent préparer des concours nationaux pour l'accès à la formation de 3ème cycle.

Les séries proposées requièrent connaissance des concepts du modèle relationnel, le langage SQL, les notions de contrainte d'intégrité pour une bonne compréhension du présent document.

Son contenu représente une suite logique de la matière Base de Données assurée pour les étudiants de deuxième année licence (L2). Dans ce cours du L2 Informatique, les étudiants ont appris à :

- Manipuler un Système de gestion de base de données
- Créer une base de données (des requêtes SQL LDD[1])
- Mettre à jour une base de données : insertion, modification, suppression (des requêtes SQL MLD [2])

[1] Langage de Définition de données
[2] Langage de Manipulation de données

- Interrogation d'une base de données (des requêtes SQL LID[3])

De ce fait, ce support de TDs avec des exercices corrigés a pour but d'étudier les notions avancées sur les bases de données relationnelles à savoir : Gestion des utilisateurs pour assurer un accès sécurisé aux bases de données, programmation en PL/SQL. Ainsi, Il permet d'acquérir des fonctionnalités avancées des bases de données relationnelles à savoir les procédures stockées et les déclencheurs.

Le polycopié présente 5 séries de TD :

1. **Série de TD 1 : Gestion d'accès aux Bases de données ;**
2. **Série de TD2 : Les vues ;**
3. **Série de TD 3 : PL/SQL ;**
4. **Série de TD 4 : Procédures Stockées ;**
5. **Série de TD 5 : Les déclencheurs.**

[3] Langage d'Interrogation de données

Table des matières

Table des matières

Liste des Figures

Liste syntaxes

Série de TD1 : Gestion d'accès aux bases de données

1. Objectif de la série

Cette série a comme objectif d'appliquer des requêtes SQL du langage de contrôle de données (DCL) qui permettent la protection des données contre les accès non autorisés. A partir de cette série, l'étudiant aura la possibilité de :

- ✓ Créer des utilisateurs
- ✓ Donner des privilèges à des utilisateurs
- ✓ Révoquer des privilèges
- ✓ L'intérêt de création de profile
- ✓ Utilisation et création de rôle

2. Rappel de cours

2.1. Principe de base

La partie contrôle de données (**DCL** : **Data Control Language**) comprend toute la partie gestion des utilisateurs, droits d'utilisation des tables. Pour accéder à un SGBD, il faut disposer d'un compte, donc d'un mot de passe et un ensemble de droits (privilèges) identifiés par le SGBD.

Tout objet (Table ou vue) a un utilisateur créateur (owner). Ce dernier possède tous les droits (consultation, modification et suppression) sur cet objet. Les autres utilisateurs n'ont aucun droit sur cet objet, à moins que le créateur ne le leur accorde explicitement.

L'administrateur a la possibilité de créer des utilisateurs en leurs attribuant un ensemble de privilèges et un profile.

2.2. Création d'un utilisateur

Un utilisateur est défini par un nom d'utilisateur, un mot de passe, un ensemble de privilèges et un profil.

Syntaxe :

```
CREATE USER utilisateur
IDENTIFIED { BY password | EXTERNALLY }
[ DEFAULT TABLESPACE tablespace]
[ TEMPORARY TABLESPACE tablespace]
[ QU0TA { integer| UNLIMITED } ON tablespace] ...
[ PROFILE profil ][PASSWORD EXPIRE] [ACCOUNT {LOCK | UNLOCK}]
```

Syntaxe 1 : création d'utilisateur

2.3. Création de profile

Un profil permet à l'administrateur d'une base de contrôler la consommation des ressources systèmes et des mots de passes de chaque utilisateur, tels que : le nombre de sessions (connexion) simultanées par utilisateur, durée d'inactivité, durée totale de connexion, durée de vie du mot de passe (jours).

2.4. Types de privilèges

La gestion des droits d'accès aux données par les utilisateurs s'effectue par les commandes SQL :

- GRANT : pour donner des privilèges à un utilisateur
- REVOKE : pour supprimer des privilèges à un utilisateur

On distingue deux types de privilèges :

1. **Les privilèges systèmes** : permettent à un utilisateur d'agir sur les définitions des objets de la base. Par exemple : autoriser un utilisateur à créer des tables, des vues, des index, …
    a) **Syntaxe accorder privilèges systèmes** :

```
GRANT privilèges TO bénéficiaire [WITH ADMIN OPTION]
```

Syntaxe 2: Accorder privilèges systèmes

- Le ***bénéficiaire*** peut être soit un utilisateur ou un groupe d'utilisateurs, soit tous les utilisateurs (PUBLIC)

- L'option ***WITH ADMIN OPTION*** permet d'accorder un privilège système avec le droit de le transmettre

b) Syntaxe retirer privilèges systèmes :

```
REVOKE privilèges FROM bénéficiaire
```

Syntaxe 3: retirer privilèges systèmes

2. **Les privilèges objets** : permet d'exécuter une action particulière sur une table, vue, fonction, séquence, procédure, package d'un schéma. Par exemple : autoriser un utilisateur à consulter des tables, permettre à un utilisateur de faire des mises à jour, des Références à des tables …

a) Syntaxe accorder privilèges objets

```
GRANT privilèges ON objet TO bénéficiaire
[WITH GRANT OPTION]
```

Syntaxe 4 : Accorder privilèges objets

- Les ***privilèges*** qui peuvent être passés sont :
  - soit ALL (tous les privilèges)
  - soit une liste de privilèges parmi :
    - SELECT
    - INSERT
    - UPDATE [(liste de colonnes)]
    - DELETE
- Le ***bénéficiaire*** peut être soit un utilisateur ou un groupe d'utilisateurs, soit tous les utilisateurs (PUBLIC)

- L'option ***WITH GRANT OPTION*** permet d'accorder un privilège sur un objet avec le droit de le transmettre

- Aucun utilisateur ne peut octroyer un privilège sur un objet qu'il ne détient pas : soit c'est un objet se trouvant dans son schéma, soit il a reçu le privilège avec l'option WITH GRANT OPTION

b) Syntaxe retirer privilèges objets :

```
REVOKE privilèges ON objet FROM bénéficiaire
```

Syntaxe 5: Retirer privilèges systèmes

2.5. Utilisation de Rôle

Un rôle est le regroupement de privilèges qui peuvent être attribués, soit à un utilisateur soit à un autre rôle. Il peut contenir à la fois les deux types de privilèges : système et objet. Pour créer un rôle, il faut avoir le privilège "CREATE ROLE"

Syntaxe :

```
CREATE ROLE nom_role
```

Syntaxe 6: Création de rôle

3. Exercices

3.1. Exercice1

On considère trois utilisateurs **Rayane**, **Imane** et **Meriem** ayant chacun respectivement un compte ***Ray***, ***Ima***, ***Meri*** sur la base de données BDDSIC. Nous résumons dans le tableau ci-dessous les objets que possède chaque utilisateur dont chacun souhaite gérer l'accès à ses propres objets :

Utilisateurs	Compte	Propriétaires
Rayane	**Ray**	TabA, TabB, TabC
Meriem	**Meri**	TabM1,TabM2 ,VueM1
Imane	**Ima**	Tab

Donner les commandes qui permettent d'attribuer les privilèges qui permettent de définir la politique de gestion des droits d'accès en se basant sur les hypothèses suivantes :

- **Rayane** souhaite que :
  - ✓ TabA soit visible pour Meriem et Imane mais que Meriem puisse mettre à jour ses données avec le droit de le transmettre.
  - ✓ TabB soit privée (aucune action sur cette table)
  - ✓ TabC soit accessible en lecture pour tout le monde (tous les utilisateurs du SGBD)
- **Meriem** souhaite que :
  - ✓ TabM1 soit modifiable (insertion, mise à jour et suppression) par Rayane et Imane.
  - ✓ TabM2 soit accessible en lecture pour tout le monde sauf pour Imane.
  - ✓ VueM1 soit visible uniquement pour Rayane.

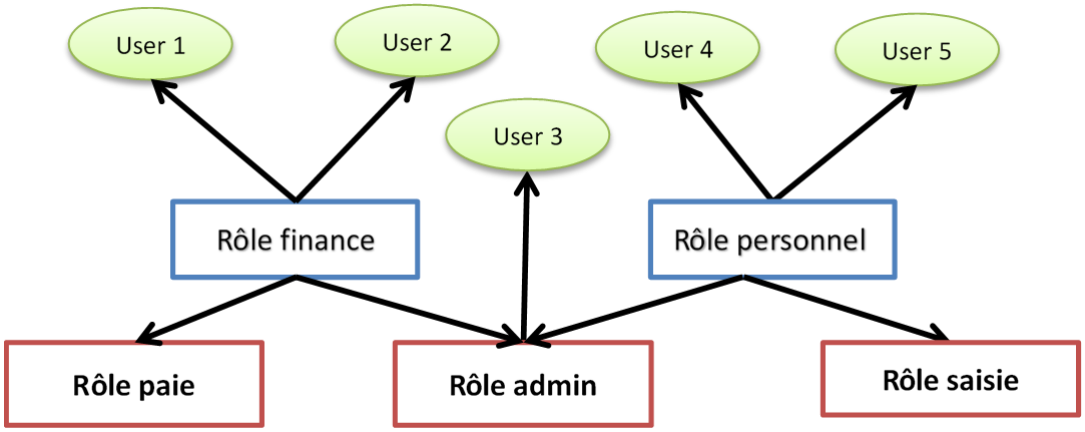
❖ **Imane** souhaite que :

- ✓ les attributs AT1 et AT2 soient modifiables par tous les utilisateurs de la base
- ✓ les attributs AT3 et AT4 soient modifiables uniquement par Meriem.

.

3.2. Exercice2

Soit la figure suivante :



On suppose que les users sont déjà créés. Les privilèges des rôles admin , paie et saisie sont donnés comme suit:

- Le Rôle admin : CREATE role, CREATE user, CREATE profile
- Le Rôle saisie: SELECT, DELETE, INSERT, UPDATE sur la table compte
- Le Rôle paie : SELECT, INSERT, UPDATE salaire sur la table compte

- Le rôle finance rassemble les privilèges accordés à admin et paie
- Le rôle Personnel rassemble les privilèges accordés à admin et saisie

1) Créer l'ensemble des rôles
2) Attribuer les privilèges aux rôle associés (admin, saisie, paie) puis personnel et finance
3) Attribuer les rôles aux users
4) Créer l'utilisateur superuser avec mot de passe super et lui octroyer les deux rôles personnel et finance et en lui permettant de transmettre ces droits
5) Retirer le droit de modification du champ SoldeCpt dans la table compte pour l'user 1

6) On veut limiter l'accès à la base pour user1 et user2, en précisant qu'une seule session est autorisée. Que faut-il faire ?

3.3. Exercice3

Soit le schéma relationnel suivant :

Client(NumCli, CINCli, NomCli, AdrCli, TelCli, NbCptE)
Operation(NumOp, TypeOP, MtOp, NumCpt*, DateOp)
Compte(NumCpt, SoldeCpt, TypeCpt, NumCli*)

1) Créer deux users : Mohamed et Karim. Les mots de passe des deux users sont « m » et « k » respectivement
2) Donner à l'utilisateur Mohamed le droit de création de table avec le droit de le transmettre.
3) Supposons que Mohamed qui a créer la table Client. Donner l'ordre SQL pour permettre à Karim de mettre à jour la table Client sachant que karim pourra aussi donner ce droit à un autre utilisateur.
4) Créer le rôle : « rAGENT » qui peut consulter les tables Client, Operation et Compte.
5) Affecter ce rôle à tous les utilisateurs.
6) Retirer le droit de suppression sur la table Client donné pour karim
7) La table opération, est une table critique, sa manipulation doit être bien sécurisée. Seul l'agent responsable des retraits (utilisateur : caissier) a le droit d'y accéder. Proposer une solution pour donner l'accès à cette table que pour cet utilisateur et juste pendant les heures de travail par exemple entre 8h00 et 16h00 et en dehors des jours de weekend?

4. Solution des exercices

4.1. Solution Exercice1

La gestion de ces tables sera comme suit :

Rayane souhaite que :

- ✓ TabA soit visible pour Meriem et Imane mais que Meriem puisse mettre à jour ses données avec le droit de le transmettre.

```
grant SELECT on  TabA to Meri, Ima;
grant UPDATE salaire on  TabA to Meri;
```

- ✓ TabB soit privée (aucune action sur cette table)

```
REVOKE ALL PRIVILEGES ON TabB FROM public;
```

- ✓ TabC soit accessible en lecture pour tout le monde (tous les utilisateurs du SGBD)

```
GRANT SELECT ON TabC TO PUBLIC;
```

Meriem souhaite que :

- ✓ TabM1 soit modifiable (insertion, mise à jour et suppression) par Rayane et Imane.

```
GRANT INSERT, UPDATE, DELETE ON TabM1 TO Ray, Ima;
```

- ✓ TabM2 soit accessible en lecture pour tout le monde sauf pour Imane.

```
GRANT SELECT ON TabM2 TO PUBLIC ;
REVOKE SELECT ON TabM2 FROM Ima ;
```

- ✓ VueM1 soit visible uniquement pour Rayane.

```
GRANT SELECT ON VueM1 TO Ray;
```

Imane souhaite que :

- ✓ les attributs AT1 et AT2 soient modifiables par tous les utilisateurs de la base

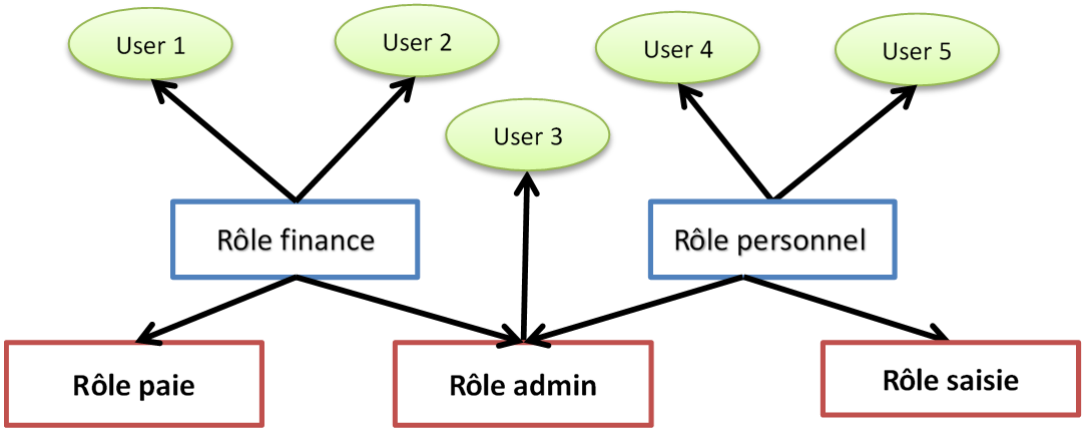
```
GRANT UPDATE(AT1 , AT2) ON Tab TO PUBLIC;
```

- ✓ les attributs AT3 et AT4 soient modifiables uniquement par Meriem.

```
GRANT UPDATE(AT3 , AT4) ON Tab TO Meri;
```

4.2. Solution Exercice2

Soit la figure suivante :



On suppose que les users sont déjà créés. Les privilèges des rôles admin , paie et saisie sont donnés comme suit:

- Le Rôle admin : CREATE role, CREATE user, CREATE profile
- Le Role saisie: SELECT, DELETE, INSERT, UPDATE sur la table compte
- Le Role paie : SELECT, INSERT, UPDATE salaire sur la table compte
- Le role finance rassemble les privilèges accordés à admin et paie
- Le role Personnel rassemble les privilèges accordés à admin et saisie

1) Créer l'ensemble des roles

```
CREATE role admin ; CREATE role saisie ; CREATE role paie ;
 CREATE  role finance ;
CREATE role personnel ;
```

2) Attribuer les privilèges aux rôle associés (admin, saisie, paie) puis personnel et finance

```
grant CREATE role, CREATE user, CREATE profile to admin ;
grant SELECT, DELETE, INSERT, UPDATE on compte to saisie ;
grant SELECT, INSERT, UPDATE(salaire) on  compte to paie ;
```

```
Grant paie, admin to finance ;
Grant admin, saisie to personnel ;
```

3) Attribuer les rôles aux users :

```
Grant finance to user1;
Grant finance  to user2;
Grant Personnel to user4;
Grant Personnel  to user4;
Grant admin to user3 ;
```

4) Créer l'utilisateur superuser avec mot de passe super et lui octroyer les deux rôles personnel et finance et en lui permettant de transmettre ces droits

```
CREATE USER superuser IDENTIFIED BY super ;
GRANT personnel , finance to superuser  WITH ADMIN OPTION;
```

5) retirer le droit de modification du champ SoldeCpt dans la table compte pour l'user 1

```
REVOKE UPDATE(SoldeCpt) on compte FROM user1
```

6) On veut limiter l'accès à la base pour user1 et user2, en precisant qu'une seule session est autorisée. Que faut t-il faire ?

Créer un nouveau profile, puis l'affecter à user1 et user2, en lançant une modification sur la définition de ces users

```
ALTER USER user1 PROFILE limitConnxion_user;
```

4.3. Solution Exercice3

Soit le schéma relationnel suivant :

Client(NumCli, CINCli, NomCli, AdrCli, TelCli, NbCptE)
Operation(NumOp, TypeOP, MtOp, NumCpt*, DateOp)
Compte(NumCpt, SoldeCpt, TypeCpt, NumCli*)

1) Créer deux users : Mohamed et Karim. Les mots de passe des deux users sont « m » et « k » respectivement

```
CREATE USER Mohamed IDENTIFIED BY m;
CREATE USER Karim IDENTIFIED BY k;
```

2) Donner à l'utilisateur Mohamed le droit de création de table avec le droit de le transmettre.

```
Grant CREATE table to mohamed with admin option
```

3) Supposons que Mohamed qui a créer la table Client. Donner l'ordre SQL pour permettre à Karim de mettre à jour la table Client sachant que karim pourra aussi donner ce droit à un autre utilisateur.

```
Grant INSERT, UPDATE, DELETE on client to karim
 with grant option
```

4) Créer le rôle : « rAGENT » qui peut consulter les tables Client, Operation et Compte.

```
CREATE role ragent ;
Grant SELECT on client to ragent ;
Grant SELECT on operation to ragent
Grant SELECT on compte to ragent
```

5) Affecter ce rôle à tous les utilisateurs.

```
Grant ragent to PUBLIC ;
```

6) Retirer le droit de suppression sur la table Client donné pour karim

```
Revoke DELETE on client FROM karim
```

7) La table opération, est une table critique, sa manipulation doit être bien sécurisée. Seul l'agent responsable des retraits (utilisateur : caissier) a le droit d'y accéder. Proposer une solution pour restreindre l'accès à cette table que pour cet agent et juste pendant les heures de travail par exemple entre 8h00 et 16h00 et en dehors des jours de weekend?

   a) Retirer le droit d'accès à la table opération

```
REVOKE ALL PRIVILEGES ON operation FROM PUBLIC;
```

   b) Créer l'utilisateur caissier par l'administrateur de la base:

```
CREATE user caissier identified by c ;
```

c) Créer une vueparexmpel « OperationTravail » avec les conditions citée dans la question

```
CREATE OperationTravail AS
SELECT * FROM operation
WHERE TO_CHAR(SYSDATE, 'HH24') BETWEEN '08' AND '16'
AND TO_CHAR(SYSDATE,'DAY') NOT IN ( 'SAMEDI', 'VENDREDI')
WITH CHECK OPTION;
```

d) Ensuite nous attribuons le droit d'accès à cette vue seulement pour l'utilisateur caissier

```
GRANT ALL ON OperationTravail TO caissier;
```

Série de TD2 : LES VUES

1. Objectifs de la série

Cette série a comme objectif de savoir créer et utiliser des vues avec ses différentes options. A partir de cette série, l'étudiant aura la possibilité de :

- ✓ Créer et interroger des vues
- ✓ Restructurer "virtuellement" une base
- ✓ Restreindre la visibilité des données (on ne donne accès qu'à la vue)
- ✓ Rendre les requêtes complexes à simple

2. Rappels de cours

2.1. Définition

Une vue est une définition logique d'une relation, sans stockage de données, obtenue par interrogation d'une ou plusieurs tables de la base de données.

2.2. Création d'une vue

Pour créer une vue, il faut utiliser la syntaxe suivante :

```
CREATE [OR REPLACE] VIEW <nom_vue> [( nv_nom_col)*]
AS sous-requete
[WITH CHECK OPTION [CONSTRAINT nom_contrainte]]
[WITH READ ONLY];
```

Syntaxe 7: Création d'une vue

- ✓ **[OR REPLACE] :** Permet de remplacer une vue existante
- ✓ **[WITH CHECK OPTION [CONSTRAINT nom_contrainte]]:** si on l'ajoute, signifie que l'insertion et modifications dans cette vue est validé **seulement** si le tuple résultant est sélectionné par la vue
- ✓ **[WITH READ ONLY] :** interdit les MAJ

2.3. Appel d'une vue :

L'appel d'une vue se fait de la même manière qu'on fait appel à une table, à savoir:

```
SELECT ... FROM <nom_vue> WHERE ...

INSERT INTO  <nom_vue> Values …

UPDATE <nom_vue> SET  … WHERE  …

DELETE FROM  <nom_vue>
```

2.4. Supprimer une vue

Pour supprimer une vue :

```
DROP VIEW <nom_vue>
```

2.5. Renommer une vue

```
RENAME Ancien_Nom TO Nouveau_Nom ;
```

3. Exercices

3.1. Exercice 1

Soit le schéma relationnel suivant :

Concours(CodeC, IntituléC, DateC, nbrePoste)
Candidat(NumC, NomC, PrenomC, DN, Cumul, #Concours)
Epreuve(CodeE, IntituléE, Coef, #Concours)
Ep_Can(#CodeE, #NumC, Note)

Cumul: représente la somme des notes coef des épreuves.*

Coef : représente le coefficient de l'épreuve

Note : représente la note attribuée au candidat dans une épreuve

Questions :

1) Créer la vue qui permet de visualiser seulement les concours entre année 2010 et 2020 et dont le nombre de poste est supérieur à 3.
2) Donner la requête qui permet de consulter le contenu de cette vue
3) Modifier le nombre de postes du concours N°13 à 3 via la vue « Concours2000 » ; Est-ce que la modification a été validée ?
4) Proposer une solution qui permet de contrôler ce genre de manipulation
5) Créer la vue CandidatsMoySup10 (NumC, NomC, PrenomC, DN, Cumul, Concours) qui récupère les candidats ayant une moyenne supérieure à 10.
6) Créez la vue CandidatMoyenne qui affiche le nom et le prénom de chaque candidat ainsi que la moyenne qu'il a obtenu dans toutes les épreuves, triés par ordre décroissant des moyennes. Cette vue doit être en lecture seule
7) Créez la vue EpreuveCandidats qui affiche le code et l'intitulé de chaque épreuve ainsi que le nombre de candidats qui ont passé cette épreuve
8) Donner la requête qui permet d'extraire l'Epreuve ainsi le concours ou il y avait le plus de candidat en utilisant la vue précédente

3.2. Exercice 2

Soit le schéma relationnel suivant :

Employe (ENO, NomEmp, Prof, DateEmb, Sal, DNO#)

Depart (DNO, NomDep, Dir#)

Questions

1) Définissez une vue nommée EMP_Ingenieur contenant les Noms des employés, Profession, Date d'embauche et Salaire des employés.
2) Supprimer cette vue du dictionnaire de la base de données.
3) Créer la vue StatDep (NumDep, NomDep, NbreEmp, SUMSalaire) qui permet d'afficher le nombre des employés et la somme des salaires pour chaque département
4) Créer la vue SalaireAnnuel qui permet de lister les employés avec leurs salaires annuels
5) Définissez la vue qui permet d'afficher le résultat suivant :

NomEmp	***Prof***	***DateEmb***	***Sal***
Saidi	Juriste	01.03.2000	45000
Dib	Ingénieur	01.01.2023	45000
Boukli	Comptable	01.03.2010	45000

6) Définissez une vue qui permet d'afficher le même résultat mais avec des noms de colonnes personnalisés. (**NomEmp** devient « NomEmploye», **Prof** devient « Profession », **DateEmb** devient « DateEmbauche » et **Sal** devient « Salaire »
   a) C'est possible de modifier la table ***Emp*** à travers cette vue ? Pourquoi?

   b) C'est possible d'ajouter un nouveau employé dans la table Emp à travers cette vue ? Pourquoi ?

3.3. Exercice 3

Soit le schéma relationnel suivant :

Virus (code_virus, nom_virus, famille_virus)

Symptome (code_symp, nom_symp, fiche_descriptive)

Virus_Symptome (code_virus #, code_symp #, probabilité)

Vaccin (code_vaccin, nom_vaccin, code_virus #, année_découverte)

Questions

1) Noms des vaccins et années découvertes pour lesquels les symptômes du virus sont la perte du goût et de l'odorat
2) Donner le nombre de vaccins qui ont comme symptômes du virus : la perte du goût et de l'odorat à travers la vue
3) Soit la vue suivante :

```
CREATE VIEW Vaccin_2021
AS SELECT Code_Virus, Nom_Vaccin FROM Vaccin
WHERE Année_Découverte ='2021'
WITH CHECK OPTION ;
```

   a) Quelle est le rôle de la clause WITH CHECK OPTION ?
   b) Est ce qu'elle est utile dans cette requête ? Pourquoi ?

4. Solution des exercices

4.1. Solution Exercice 1

Soit le schéma relationnel suivant :

Concours(CodeC, IntituléC, DateC, nbrePoste)
Candidat(NumC, NomC, PrenomC, DN, Cumul, #Concours)
Epreuve(CodeE, IntituléE, Coef, #Concours)
Ep_Can(#CodeE, #NumC, Note)

Réponses

1) Créer la vue qui permet de visualiser seulement les concours entre année 2010 et 2020 et dont le nombre de poste est supérieur à 3.

```
CREATE VIEW Concours2000 AS
SELECT * FROM Concours
WHERE DateC BETWEEN '01/01/2010' AND '01/01/2020' and nbrePoste>3;
```

2) Donner la requête qui permet de consulter le contenu de cette vue

```
SELECT * FROM Concours2000;
```

3) Modifier le nombre de poste du concours N°13 à 3 via la vue « Concours2000 » ; Est-ce que la modification a été validé ?

```
UPDATE Concours2000
SET nbrePoste = 3
WHERE CodeC = 13;
```

OUI la modification a été validée, bien que cet enregistrement ne va pas apparaitre à travers cette vue.

4) Proposer une solution qui permet de contrôler ce genre de manipulation

Il faut ajouter la clause WITH CHECK OPTION en modifiant la définition de la vue avec OR REPLACE

```
CREATE OR REPLACE VIEW Concours2000 AS
SELECT * FROM Concours
WHERE DateC BETWEEN '01/01/2010' AND '01/01/2020'
and nbrePoste>3
WITH CHECK OPTION;
```

5) Créer la vue CandidatsMoySup10 (NumC, NomC, PrenomC, DN, Cumul, Concours) qui récupère les candidats ayant une moyenne supérieure à 10.

Solution1

```
CREATE VIEW CandidatsMoySup10 AS
SELECT NumC, NomC, PrenomC, DN, Cumul, Concours
FROM Candidat
WHERE Cumul / (SELECT SUM(Coef) FROM Epreuve WHERE
Epreuve.Concours = Candidat.#Concours) > 10;
```

OU

Solution2

```
CREATE VIEW CandidatsMoySup10 as SELECT NumC, Nom, PrenomC, DN,
Cumul, Candidat.Concours
FROM Candidat,Epreuve
WHERE epreuve.Concours = Candidat.Concours
GROUP BY NumC, Nom, Cumul, Candidat.Concours
HAVING  Cumul /SUM(Coef)  > 10;
```

Pour insérer des lignes via cette vue, c'est possible dans la 1ère solution, par contre la 2ème, il y a une jointure entre Candidat et Epreuve, d'où on aura un problème de

« non key-preserved table ». Et bien sur l'insertion doit satisfaire la condition de la vue.

6) Créez la vue CandidatMoyenne qui affiche le nom et le prénom de chaque candidat ainsi que la moyenne qu'il a obtenu dans toutes les épreuves, triés par ordre décroissant des moyennes. Cette vue doit être en lecture seule

```
CREATE VIEW CandidatMoyenne AS
SELECT NomC, PrenomC, SUM(Note * Coef)/SUM(Coef) AS MOYCand
FROM Candidat natural JOIN Ep_Can natural JOIN Epreuve
GROUP BY Candidat.NumC, NomC, PrenomC
ORDER BY MOYCand DESC
WITH READ ONLY;
```

7) Créez la vue EpreuveCandidats qui affiche le code et l'intitulé de chaque épreuve ainsi que le nombre de candidats qui ont passé cette épreuve

```
CREATE VIEW EpreuveCandidats AS
SELECT CodeE, IntituléE, COUNT(NumC) AS NbrCandidats
FROM Epreuve NATURAL JOIN Ep_Can
GROUP BY Epreuve.CodeE, IntituléE
```

8) Donner la requête qui permet d'extraire l'Epreuve ainsi le concours ou il y avait le plus de candidat en utilisant la vue précédente

```
SELECT ec.CodeE, ec.IntituléE, ec.CodeC, ec.IntituléC,
ec.NbCandidats
FROM EpreuveCandidats ec
WHERE ec.NbCandidats = (SELECT MAX(NbCandidats) FROM
EpreuveCandidats)
```

4.2. Solution Exercice 2

Soit le schéma relationnel suivant :

Employe (ENO, NomEmp, Prof, DateEmb, Sal, DNO#)

Depart (DNO, NomDep, Dir#)

Réponses

1) Définissez une vue nommée EMP_Ingenieur contenant les Noms des employés, Profession , Date Embauche et Sal des employés .

```
CREATE VIEW EMP_Ingenieur as
SELECT NomEmp, Prof, DateEmb, Sal
FROM employe
WHERE Prof='Ingénieur' ;
```

2) Supprimer cette vue du dictionnaire de la base de données.

```
DROP VIEW EMP_Ingenieur;
```

3) Créer la vue StatDep (NumDep, NomDep, NbreEmp, SUMSalaire) qui permet d'afficher le nombre des employés et la somme des salaires pour chaque département

```
CREATE VIEW StatDep (NumDep, NomDep, NbreEmp, SUMSalaire) as
SELECT DNO, NomDep, COUNT(*), SUM(Sal)
FROM Depart Natural join Employe
GROUP BY DNO, NomDep;
```

4) Créer la vue SalaireAnnuel qui permet de lister les employés avec leurs salaires annuels

```
CREATE VIEW SalaireAnnuel AS
SELECT ENO, NomEmp, Prof, DateEmb, salary*12 AS annuel
FROM employe ;
```

5) Définissez la vue qui permet d'afficher le résultat suivant :

NomEmp	*Prof*	*DateEmb*	*Sal*
Saidi	Juriste	01.03.2000	45000
Dib	Ingénieur	01.01.2023	45000
Boukli	Comptable	01.03.2010	45000

```
CREATE VIEW EMP_S AS
SELECT NomEmp, Prof, DateEmb, Sal
FROM   Employe
WHERE  Sal = 45000 ;
```

6) Définissez une vue qui permet d'afficher le même résultat mais avec des noms de colonnes personnalisés. (**NomEmp** devient « NomEmploye», **Prof** devient « Profession », **DateEmb** devient « DateEmbauche » et **Sal** devient « Salaire »

```
CREATE VIEW EMP_S_2 (NomEmploye, Profession, DateEmbauche
Salaire) AS
SELECT  NomEmp, Prof, DateEmb, Sal FROM Employe
WHERE Sal = 45000
```

7) C'est possible de modifier la table Emp à travers cette vue ? Pourquoi?

Oui, parce que c'est une vue simple.

8) C'est possible d'ajouter un nouveau employé dans la table Emp à travers cette vue ? Pourquoi ?

Non, car on n'a pas attribué une valeur à la clé primaire. C'est possible dans le cas ou cette opération (affectation d'une valeur à la clé primaire) est effectuée par le SGBD ou une autre solution (par exemple un Trigger)

4.3. Solution Exercice 3

Soit le schéma relationnel suivant :

Virus (code_virus, nom_virus, famille_virus)

Symptome (code_symp, nom_symp, fiche_descriptive)

Virus_Symptome (code_virus #, code_symp #, probabilité)

Vaccin (code_vaccin, nom_vaccin, code_virus #, année_découverte)

Réponses

1) Noms des vaccins et année découverte pour lesquels les symptômes du virus sont la perte du goût et de l'odorat

```
CREATE VIEW Vacc_Symp AS (SELECT nom_vaccin, année_découverte
FROM symptome, virus_symptome, vaccin
WHERE  virus_symptome.code_symp= symptome. code_symp and
virus_symptome.code_virus=vaccin.code_virus and nom_symp='pertre
du goût' )
INTERSECT
(SELECT nom_vaccin, année_découverte
FROM symptome, virus_symptome, vaccin
WHERE virus_symptome.code_symp = symptome.code_symp and
virus_symptome.code_virus=vaccin.code_virus and nom_symp='perte-
odorat');
```

2) Donner le nombre de vaccins qui ont comme symptômes du virus : la perte du goût et de l'odorat à travers la vue

```
SELECT count(*) FROM Vacc_Symp;
```

Dans ce cas, l'intérêt de la vue est de réduire la complexité d'une requête SQL

3) Soit la vue suivante :

```
CREATE VIEW Vaccin_2021
AS SELECT Code_Virus, Nom_Vaccin FROM Vaccin
WHERE Année_Découverte ='2021'
WITH CHECK OPTION ;
```

4) Quelle est le rôle de WITH CHECK OPTION ?

Permet d'insérer et modifier seulement si le tuple résultant est sélectionné par la vue, donc il permet de bloquer l'ajout de nouveaux vaccins découverts hors l'année 2021

5) Est ce qu'il est utile dans cette requête ? Pourquoi ?

Non il n'est pas utile.

Puisque le champ Année_Découverte n'est pas parmi les champs de la vue, donc aucun enregistrement ne va être inséré via cette vue. Pour que WITH CHECK OPTION joue son role dans cet exemple, il faut ajouter le champ Année_Découverte dans le SELECT

Remarque :

On peut garder la même définition de la vue, mais l'insertion via cette vue n'est garantie qu'à travers un déclencheur sur cette vue.

Série de TD 3 : PL/SQL

1. Objectif de la série

Cette série a comme objectif de connaitre le langage PL/SQL (Procedural Langage/Structured Query Langage). Il permet de combiner les avantages d'un langage de programmation classique qui offre les structures algorithmiques (les instructions conditionnelles et répétitives, les sous-programmes...) avec les possibilités de manipulation de données offertes par SQL. A partir de cette série, l'étudiant aura la possibilité de :

- ✓ Ecrire des blocs PL/SQL anonymes
- ✓ Ajouter les éléments essentiels de programmation tels que l'utilisation de variables, les conditions (SI – SINON), les boucles (TANT QUE) mais aussi les fonctions et les procédures afin de réaliser des traitements plus complexes
- ✓ Intégrer une requête d'interrogation (SELECT) dans un programme PL/SQL
- ✓ Inclure des commentaires dans le code
- ✓ Fonctions SQL dans PL/SQL
- ✓ Créer et manipuler des curseurs
- ✓ Traiter des erreurs et des exceptions ;

2. Rappels de cours

2.1. Syntaxe d'un programme PL/SQL

```
DECLARE
 -- section de déclarations
 -- section optionnelle
 …
BEGIN
 -- traitement, avec éventuelles directives SQL
 -- section obligatoire
 …
EXCEPTION
 -- gestion des erreurs retournées par le SGBDR
 -- section optionnelle
 …
END;
```

Syntaxe 8: structure d'un programme PL/SQL

- Déclaration des variables et constants sont faites dans la section DECLARE d'un bloc PL/SQL

- Pour référencer un type existant : utiliser %**TYPE** et %**ROWTYPE**
  - Type d'une autre variable

```
VCode Varchar(23);
CodeEnseignant VCode%TYPE;
```

  - Type d'attribut d'une table

```
nom-variable nom-table.nom-attribut%TYPE;

VCodeEtud Etudiant.CodeEtud%TYPE;
```

  - Type des n-uplets d'une table

```
nom-variable nom-table%ROWTYPE;

VEtud Etudiant%ROWTYPE ;
```

  - Type d'un curseur

```
nom-variable nom-curseur%ROWTYPE;

CURSOR C1 IS SELECT CodeEtu, nom, prénoms FROM
Etudiant ;
V1 C1%ROWTYPE ;
```

2.2. Affectation

- Affectation simple :

```
i := 0;
n := n + 1;
```

- Affectation à partir d'une lecture de valeurs de la base de données

```
SELECT listeAttributs INTO listeVariables    FROM...

SELECT listeAttributs INTO nomStructure      FROM…
```

2.3. Structures de contrôle

- Structure Conditionnelles
  - If then else

```
IF condition1 THEN instructions1;
[ELSIF condition2 THEN instructions3;]
[ELSE instructions2;]
END IF;
```

  - Case when

```
CASE variable
WHEN expression1 THEN instructions1;
WHEN expression2 THEN instructions2; …
[ELSE instructionsj;]
END CASE;
```

2.4. Structures Répétitives

- While

```
WHILE <condition> LOOP
 <instructions> ;
END LOOP;
```

- Loop

```
LOOP
 instructions ;
EXIT WHEN condition ;
  instructions
END LOOP;
```

- For

```
FOR <variable> IN <value> .. <value> LOOP
  <instructions> ;
END LOOP;
```

2.5. Curseurs

L'accès direct : ne prend qu'un seul tuple, afin de récupérer plusieurs tuples , on utilise **un curseur**. Il parcourt un par un chaque tuple résultat. Chaque tuple récupéré pourra être mis dans une variable. Quatre étapes sont nécessaires pour utiliser un curseur:

1) Déclaration du curseur
2) Ouverture du curseur
3) Traitement des lignes du résultat
4) Fermeture du curseur

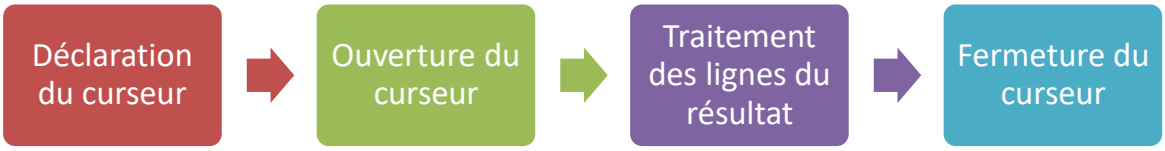


Figure 1 : Étapes d'utilisation d'un curseur

- **La déclaration d'un curseur :** Se fait dans la section DECLARE d'un bloc PL/SQL. On a deux types de déclaration de curseur
  - Curseur **sans** paramètre

```
CURSOR nom-curseur IS requête;
```

- Curseur **avec** paramètre

```
CURSOR nom-curseur (nomparam type-p [:= val-défaut], ...)
IS requête;
```

- **L'ouverture du curseur** : Alloue un espace mémoire au curseur

```
OPEN nom-curseur;
```

ou

```
OPEN nom-curseur(liste-par-effectifs);
```

- **Traitement des lignes** : Autant d'instructions FETCH que de lignes résultats

```
FETCH nom-curseur INTO liste-variables;
```

ou

```
FETCH nom-curseur INTO nom-enregistrement;
```

Afin de traiter les lignes d'un curseur, il faut utiliser une **boucle** pour traiter chaque ligne retournée par ce curseur.

- **Fermeture du curseur** : permet de libérer le curseur

```
CLOSE nom-Curseur ;
```

Nous pouvons résumer dans la figure suivante, le déroulement d'exécution d'un curseur

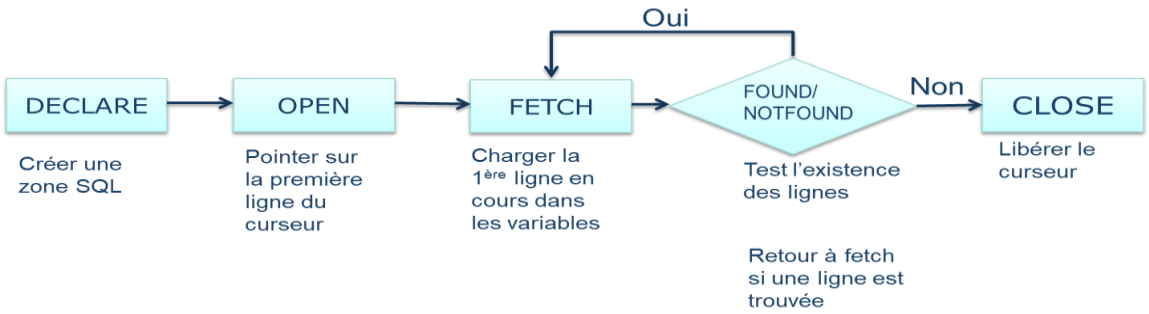


Figure 2 : déroulement d'exécution d'un curseur

2.6. Gestion des erreurs

La gestion des erreurs (EXCEPTION). Nous distinguons deux types d'exceptions :

1) prédéfinie par Oracle
2) définie par le programmeur.

- **Utilisation d'une exception prédéfinie :** sont des exceptions générées par le moteur du système (division par zéro, connexion non établie, table

inexistante, privilèges insuffisants, mémoire saturée, espace disque insuffisant, ...)

```
DECLARE
…
BEGIN
…
EXCEPTION
WHEN NO_DATA_Found THEN traitement ;
…
WHEN Too_Many_Rows THEN traitement ;
WHEN OTHERSTHEN traitement ; --optionnel
END;
```

- **Utilisation d'une exception définie par le programmeur :** PL/SQL permet à l'utilisateur de définir ses propres exceptions. Pour se faire, nous effectuons les opérations suivantes :
  - Déclaration : sont déclarées dans la section DECLARE
  - Déclenchement : sont déclenchées explicitement dans la section BEGIN par l'instruction **RAISE**
  - Traitement : dans la section **EXCEPTION**, référencer le nom défini dans la section DECLARE.

```
DECLARE
...
Nom_anomalie EXCEPTION;
BEGIN
instructions ;
IF (condition_anomalie) THEN RAISE Nom_anomalie ;
END IF;
...
EXCEPTION
WHEN Nom_anomalie THEN (traitement);
END ;
```

3. Exercices

3.1. Exercice 1

Soit le schéma relationnel de la base de données d'une école privée :

L'école 'TlemForm' veut gérer ses différentes formations dans une base de données. Le schéma de la base de données est le suivant :

FORMATIONS (Id_Form, Titre_Form, Duree_Form)
FORMATEURS (Id_F, Nom_F, Adresse_F, Statut, Salaire)
SESSIONS (id_Session, Date_Session, id_Form#, id_F#)

Statut : Permanent ou Vacataire

Questions :

1) Ecrire un bloc PL/SQL qui permet de calculer et d'afficher le nombre de formateurs qui ont statut « Permanent ». Une exception sera générée si aucun formateur n'est permanent, en affichant message : « Pas de formateur permanents »
2) Ecrire un bloc PL/SQL qui permet d'afficher le nom et le statut du formateur num = 3.

```
NOM = SAIDANI
STATUT = Permanent
```

3) Etendre la question précédente en lisant le numéro dynamiquement. Appliquer un critère de recherche saisi à l'exécution, en utilisant « &nomvariable » dans la requête SELECT
4) On veut afficher l'ensemble des formations animés par les formateurs, en précisant date Début et date fin sessions (Calculer à partir de la durée) de chaque formation. Ci-dessous le résultat attendu :

```
- 7910: UNIX--> 2fois
**********les Détails de cette formation**********
*- DU 08/02/23   AU  12/02/23
*- DU 21/02/23   AU  25/02/23

- 8000: SQLServer--> 1fois
**********les Détails de cette formation**********
*- DU 21/02/23   AU  26/02/23
```

5) Afficher les formateurs qui ont fait une formation. Utilisez une procédure locale pour tester chaque formateur s'il a fait une formation. S'il n'a fait aucune formation, supprimer le de la table formateur.
   Ci-dessous un exemple d'affichage :

```
Aucune Formations pour DIB, il est supprimé avec succès
Le formateur: CHABANI a fait des formations
Le formateur: MOUSSOUL a fait des formations
Le formateur: SAIDANI a fait des formations
Le formateur: SLIMANI a fait des formations
```

3.2. Exercice 2

Soit le schéma relationnel :

Employe(NEmp, Nom, grade, Salaire)
Client(NumCli, CINCli, NomCli, AdrCli,TelCli)
Operation(NumOp, TypeOP, MtOp, NumCpt*, DateOp, Nemp*)
Compte(NumCpt, SoldeCpt, TypeCpt, NumCli*)

TypeCpt : courant ou épargne
TypeOP: (v) versement ou (r) retrait

Questions

Écrivez un programme PL/SQL qui permet de:

1) Récupérer le plus grand montant versé (MtOp) et le plus petit puis afficher :

   « Le montant le plus grand versé est : montantmax, et le plus petit est montantmin»

2) Afficher la liste de tous les employés avec nombre d'opération effectués, si le nombre est supérieur à 20 alors augmenter son salaire par 10% :

   « L'employé sonNOM qui a le grade nomGrade a fait nbreop operations »

3) Calculer la moyenne des trois plus grands salaires des employés
4) Déterminer le nombre de clients qui ont effectué une opération de dépôt au cours des dernières 24 heures

3.3. Exercice 3

Soit le schéma relationnel concernant gestion des occupants d'immeuble

Immeuble (NumImmeuble, DateConstr, NomProp)
Appart (NumImmeu*, NumApp, Type, Superficie, Etg)

Personne (NumOcc, Nom, Age, Prof)
Occupant (NImmeuble*, NumApp*, NumOccup*, DateO)

1) Afficher les immeubles qui existent dans la base
   - **Num immeuble : B1**
   - **Num immeuble : B2**

2) Afficher les appartements de chaque immeuble de la manière suivante :
   - **Num immeuble : B1**
     - **Num Appartement : B1/11**
     - **Num Appartement : B1/22**
   - **Num immeuble : B2**
   - **……**
   -

3) Donner le code PL/SQL permettant d'afficher tous les personnes occupant un appartement de la manière suivante :

   Num immeuble : 212
   Num Appartement : 212/2
   - **Hamdi Karim**
   - **Amari Amina**

4. Solution des exercices

4.1. Solution Exercice 1

Soit le schéma relationnel de la base de données d'une école privée:
L'école 'TlemForm' veut gérer ces différentes formations dans une base de données. Le schéma de la base de données est le suivant :

FORMATIONS (Id_Form, Titre_Form, Duree_Form)
FORMATEURS (Id_F, Nom_F, Adresse_F, Statut, Salaire)
SESSIONS (id_Session, Date_Session, id_Form#, id_F#)

Statut : Permanent ou Vacataire

Questions :

1) Ecrire un bloc PL/SQL qui permet de calculer et d'afficher le nombre de formateurs qui ont statut « Permanent ». Une exception sera généré si aucun formateur est permanent, en affichant message : « Pas de formateur permanents »

```
DECLARE
        vEname FORMATEURS.NOM_F%TYPE;
        vStatut FORMATEURS.STATUT%TYPE;
        vnbre integer;
        zero EXCEPTION;
BEGIN
        SELECT count(*)  into vnbre
        FROM formateurs
        WHERE statut = 'Permanent';
        if vnbre= 0 then RAISE zero; END if;

    DBMS_OUTPUT.PUT_LINE('le nombre de formateurs permanent est =
' || vnbre);

    EXCEPTION
    WHEN zero THEN DBMS_OUTPUT.PUT_LINE('PAS DE FORMATEUR
PERMANENTS');
END;
```

2) Ecrire un bloc PL/SQL qui permet d'afficher le nom et le statut du formateur num = 3

```
NOM = SAIDANI
STATUT = Permanent
```

3) Etendre la question précédente en lisant le numéro dynamiquement. Appliquer un critère de recherche saisi à l'exécution, en utilisant « &nomvaribale » dans la requête SELECT

```
ACCEPT num PROMPT 'Entrer n° de formateur: '

declare
        vEname FORMATEURS.NOM_F%TYPE;
```

```
        vStatut FORMATEURS.STATUT%TYPE;
    BEGIN
        SELECT NOM_F , STATUT  into vEname, vStatut
        FROM formateurs
        WHERE id_F = &vnum;   --& : permet de lire un numero en moment
de l'execution
    DBMS_OUTPUT.PUT_LINE('NOM = ' || vEname);
    DBMS_OUTPUT.PUT_LINE('STATUT = ' || vStatut);
   END;
```

4) On veut afficher l'ensemble des formations animés par les formateurs, en précisant date début et date fin sessions (Calculer à partir de la durée) de chaque formation. Ci-dessous le résultat attendu :

```
- 7910: UNIX--> 2fois
**********les Détails de cette formation**********
*- DU 08/02/23   AU  12/02/23
*- DU 21/02/23   AU  25/02/23

- 8000: SQLServer--> 1fois
**********les Détails de cette formation**********
*- DU 21/02/23   AU  26/02/23
```

```
DECLARE
CURSOR c1 is SELECT ID_Form, TITRE_FORM, count(*) as nbrefois
FROM sessions natural join formations group by ID_Form,
TITRE_FORM;


Cursor cdetail(IDF Formations.ID_Form%TYPE) is
SELECT date_Session,Titre_form, date_Session + duree_form as
dateFin
FROM Sessions natural join  formations WHERE id_form = IDF  order
by date_session;
BEGIN
for cform in c1 loop
 DBMS_OUTPUT.PUT_LINE('- ' || cform.id_form ||':
'||cform.TITRE_FORM ||'--> ' || cform.nbrefois ||'fois');
```

```
 DBMS_OUTPUT.PUT_LINE('**********les Détails de cette
formation**********');
  for df in cdetail(cform.ID_Form) loop
  DBMS_OUTPUT.PUT_LINE('*- DU '||df.date_session||'   AU  ' ||
df.dateFin);
  END loop;
  DBMS_OUTPUT.PUT_LINE('');
END loop;


END;
```

5) Afficher les formateurs qui ont fait une formation. En utilisant une procedure locale, tester chaque formateur s'il a fait une formation. S'il n'a fait aucune formation, supprimer le de la table formateur.

Ci-dessous un exemple d'affichage :

```
Aucune Formations pour DIB, il est supprimé avec succès
Le formateur: CHABANI a fait des formations
Le formateur: MOUSSOUL a fait des formations
Le formateur: SAIDANI a fait des formations
Le formateur: SLIMANI a fait des formations
```

```
DECLARE -début du programme principale
 Cursor CF IS SELECT id_f, nom_F FROM FORMATEURS;
--Corp de la procedure locale
PROCEDURE leformateur(NF formateurs.id_f%TYPE, NomF
FORMATEURS.NOM_F%TYPE) is
 n integer;
 BEGIN
 SELECT COUNT(*) INTO n  FROM SESSIONS  WHERE ID_F = NF;
if n=0 THEN
  DELETE FROM FORMATEURS WHERE ID_F = NF;
  DBMS_OUTPUT.PUT_LINE ('Le formateur: '||NomF||' est supprimé
avec succès');
else
DBMS_OUTPUT.PUT_LINE ('Le formateur: '||NomF||' a fait des
formations');
END if;
END ; -- fin de la procedure locale
```

```
BEGIN  -- programme principale
for ligne in CF loop
--appel de la procedure
leformateur(ligne.id_F, ligne.nom_f);
END loop;
END;
```

4.2. Solution Exercice 2

Soit le schéma relationnel :

Employe(NEmp, Nom, grade, Salaire)

Client(NumCli, CINCli, NomCli, AdrCli,TelCli, Nbcpt)

Operation(NumOp, TypeOP, MtOp, NumCpt*, DateOp, Nemp*)

Compte(NumCpt, SoldeCpt, TypeCpt, NumCli*)

TypeCpt : courant ou épargne
TypeOP: (v) versement ou (r) retrait

1) Récupérer le plus grand montant versé (MtOp) et le plus petit puis afficher :

« Le montant le plus grand versé est : montantmax, et le plus petit est montantmin»

```
DECLARE
Vmin Operation.MtOp%TYPE ;
Vmax Operation.MtOp%TYPE ;
tableVide EXCEPTION;
n integer;
BEGIN
SELECT count(*) FROM Operation WHERE type ='v';

IF n=0  THEN  Raise tabeVide; END IF;

SELECT min(MtOp), max(MtOp) INTO Vmin, Vmax FROM Operation WHERE
TypeOp = 'v';

DBMS_OUTPUT.PUT_LINE('Le montant le plus grand versé est : '
||Vmax || ' , et le plus petit est ' || Vmin) ;
```

```
EXCEPTION
WHEN tableVide Then DBMS_OUTPUT.PUT_LINE('Table vide !!!!!');
END ;
```

2) Afficher la liste de tous les employés avec nombre d'opération effectués, si le nombre est supérieur à 20 alors augmenter son salaire par 10% :

« L'employé sonNOM qui a le grade nomGrade a fait nbreop operations »

```
DECLARE
CURSOR C IS
SELECT EMPLOYE.Nemp, Nom, grade, count(*) as nbr  FROM OPERATION
NATURAL JOIN EMPLOYE GROUP BY EMPLOYE.Nemp, Nom, grade;

BEGIN
FOR e IN C LOOP
DBMS_OUTPUT.PUT_LINE('L''employé ' || e.Nom ||' qui a le grade '
||nomGrade || ' a fait ' || e.nbr  ' operations' );

IF e.nbr >20 THEN UPDATE employe
                  SET salaire = salaire + (salaire *0,1)
                  WHERE Nemp= e.Nemp;
END IF ;

END LOOP ;

END ;
```

3) Écrivez un programme PL/SQL qui calcule la moyenne des trois plus grands salaires des employés

```
DECLARE
CURSOR c IS SELECT salaire FROM EMPLOYE order by salaire desc;
somme NUMBER := 0;
```

```
sal EMPLOYE.salaire%TYPE;
BEGIN
OPEN c;
FOR i IN 1..3 LOOP
FETCH c into sal;
somme := somme + sal;
END LOOP;
CLOSE c;
DBMS_OUTPUT.PUT_LINE('Moyenne des 3 plus grands salaires de cette
banque: ' ||somme/3);
END;
```

4) Déterminer le nombre de clients qui ont effectué une opération de versement au cours des dernières 24 heures

```
DECLARE nbr integer;
BEGIN
SELECT COUNT(DISTINCT NumCli) INTO nbr FROM Operation o WHERE
o.TypeOP = 'v' AND o.DateOp >= SYSDATE - 1;
DBMS_OUTPUT.PUT_LINE('Nombre de clients ayant effectué une
opération de dépôt dans les dernières 24 heures : ' || nbr);
END;
```

4.3. Solution Exercice 3

Soit le schéma relationnel concernant gestion des occupants d'immeuble

Immeuble (NumImmeuble, DateConstr, NomProp)

Appart (NumImmeu*, NumApp, Type, Superficie, Etg)

Personne (NumOcc, Nom, Age, Prof)

Occupant (NImmeuble*, NumApp*, NumOccup*, DateO)

1) Afficher les immeubles qui existent dans la base

- **Num immeuble : B1**
- **Num immeuble : B2**

```
DECLARE
CURSOR C1 is SELECT NumImmeuble FROM immeuble;
```

```
NumIm immeuble.NumImmeuble%TYPE;


BEGIN
For c in C1 LOOP
DBMS_OUTPUT.PUT_LINE('- Num immeuble :  ' ||c.NumIm);
END ;
```

2) Afficher les appartements de chaque immeuble de la manière suivante :

- **Num immeuble : B1**
  - **Num Appartement : B1/11**
  - **Num Appartement : B1/22**
- **Num immeuble : B2**
- **……**
-

```
DECLARE
CURSOR C1 is SELECT distinct NumImmeuble FROM Appart;
CURSOR C2(num immeuble.NumImmeuble%TYPE) is SELECT NumApp FROM
Appart WHERE numimmeuble = num;



BEGIN
For c in C1 LOOP
DBMS_OUTPUT.PUT_LINE('- Num immeuble :  ' ||c.NumIm);


For V in C2(c.NumIm) LOOP
   DBMS_OUTPUT.PUT_LINE('Num Appartement: '||c.NumIm||'/'
||V.NumApp);
END loop;


END loop;
END ;
```

3) Donner le code PL/SQL permettant d'afficher tous les personnes occupant un appartement de la manière suivante :

Num immeuble : 212

Num Appartement : 212/2

- Hamdi Karim
- Amari Amina

```
DECLARE
CURSOR C1 is SELECT distinct NumImmeuble FROM Appart;


CURSOR C2(num immeuble.NumImmeuble%TYPE) is SELECT NumApp FROM
Appart WHERE numimmeuble = NUM;


CURSOR C3(a occupant.numapp%type, i occupant.NImmeuble%type) is
SELECT nom FROM personne , occupant
WHERE NumOccup = NumOcc
and numapp = a and NImmeuble = i;
BEGIN
For C in C1 LOOP
DBMS_OUTPUT.PUT_LINE('Num immeuble :  ' ||C.NumIm);


For V in C2(NumIm) LOOP
DBMS_OUTPUT.PUT_LINE('Num Appartement : '||NumIm ||'/'
||V.NumApp);


    For CC in C3(V.NumApp , C.NumIm)LOOP
        DBMS_OUTPUT.PUT_LINE('- '||CC.nom);
    END loop;


END LOOP;
END LOOP;
END;
```

Série de TD 4 : Les procédures stockées

1. Objectif de la série

Cette série a comme objectif d'utiliser le langage PL/SQL pour créer des procédures stockées.. C'est des procédures directement stockées dans la base de données après compilation. A partir de cette série, l'étudiant aura la possibilité de :

- ✓ Créer des procédures et des fonctions et les exécuter
- ✓ Créer des packages
- ✓ Réutiliser des procédures (appel dans des boucles)

2. Introduction, rappels

Les procédures stockées sont des fonctions SQL précompilées stockées de manière permanente dans la base de données et exécutées par le SGBD. Autrement dit, c'est des blocs PL/SQL nommés. Les procédures (fonctions) permettent de réduire le trafic sur le réseau (les procédures sont locales sur le serveur), du coup, mettre en œuvre une architecture client/serveur de procédures et rendre indépendant le code client de celui des procédures.

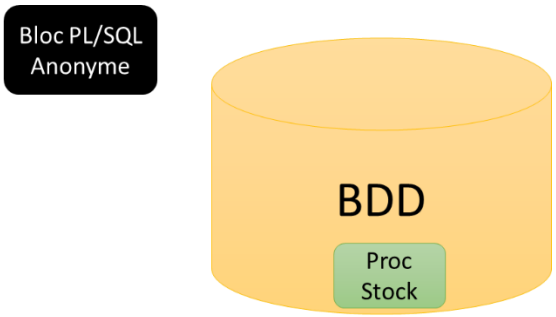


Figure 3 : Bloc PL/SQL anonyme VS Procédure stockées

2.1. Syntaxe procédure

```
CREATE PROCEDURE [OR REPLACE] nomProcedure (param1, param2…) IS
-- Déclarations de variables (pas de clause DECLARE)
BEGIN
-- Instructions PL/SQL
[EXCEPTION
-- Gestion des exceptions]
END;
```

Syntaxe 9: Création de procédure

2.2. Syntaxe Fonction

```
CREATE FUNCTION nomfonction (param1, param2…) RETURN type-retour
IS
-- Déclarations de variables (pas de clause DECLARE)
BEGIN
-- Instructions PL/SQL
RETURN valeurRetour;
[EXCEPTION
-- Gestion des exceptions]
END;
```

Syntaxe 10: création de fonction

2.3. Syntaxe package

Ensemble de types, curseurs, variables et sous-programmes inter reliés et stockés ensemble. Un package est subdivisé en deux parties :

– Spécification : interface (déclarations publiques),

– Corps : déclarations privées et code.

```
-- Définition de la spécification
CREATE [OR REPLACE] PACKAGE nom_paquetage AS
[-- Définition de types publics]
[-- Déclaration de curseurs publics]
[-- Déclaration de variables globales publiques (à éviter !)]
[-- Déclaration de sous-programmes publics]
END;
```

```
-- Définition du corps (optionnelle)
CREATE [OR REPLACE] PACKAGE BODY nom_paquetage AS
[-- Définition de types privés]
[-- Spécification de curseurs publics et privés]
[-- Déclaration de variables globales privées (à éviter !)]
[-- Spécification de sous-programmes publics et privés]
END;
```

Syntaxe 11: création de package

3. Exercices

3.1. Exercice 1

Soit le schéma relationnel :

Employe(NEmp, Nom, grade, Salaire)

Client(NumCli, CINCli, NomCli, AdrCli,TelCli, Nbcpt)

Operation(NumOp, TypeOP, MtOp, NumCpt*, DateOp, Nemp*)

Compte(NumCpt, SoldeCpt, TypeCpt, NumCli*)

1) Créer la procédure AugSalaire qui permet de modifier le salaire d'un employé à partir des paramètres suivant : le numéro de l'employé et le montant à ajouter au salaire de cet employé.
2) Modifier les salaires de tous les employés en utilisant la procedure AugSalaire
3) Créer la fonction qui prend en paramètre le numéro de l'employé et retourne le nombre d'opération effectués par cet employé.
4) Créer une fonction « SoldeTotal » qui reçoit le numéro de client, et retourne la somme totale de ses comptes.
5) Créer une fonction « SoldeTotalCompte » qui admet deux paramètres : numéro de client et type de compte, retourne le solde total d'un seul type de compte choisi.
6) Créer une procédure stockée **MajSoldeCpt(vNumCpt, vmtOp, vTypeOP)** qui met à jour le solde d'un compte en fonction de l'opération effectuée
7) créer un package qui permet de rassembler les procédures et fonctions précédentes, en plus utiliser des variable globales, ces dernières contiennent le nombre des clients, la moyenne des soldes.

   Nous allons prendre seulement les deux premières procédures stockées à savoir : ModifSalaire et NbreOp. Le même principe sera pour les autres.

3.2. Exercice 2

Soit le schéma relationnel suivant qui porte sur la gestion des concours de doctorat:

Concours(CodeC, IntituléC, DateC, nbrePoste)
Candidat(NumC, NomC, PrenomC, DN, Cumul, #Concours)
Epreuve(CodeE, IntituléE, Coef, #Concours)
Ep_Can(#CodeE, #NumC, Note)

Cumul: représente la somme des notes coef des épreuves.*

Coef : représente le coefficient de l'épreuve

Note : représente la note attribuée au candidat dans une épreuve

Afin de faciliter la tâche aux utilisateurs, et d'alléger la programmation coté client, nous allons créer l'ensemble des procédures et fonctions suivantes :

1) Un concours n'est autorisé seulement si le nombre des candidats pour ce concours est 2 fois le nombre de poste ouverts.
2) Créer un programme PL/SQL qui parcourt chaque concours, et calcul le nombre de candidats, si le nombre dépasse la condition afficher :

NomConcours : Nombre de candidat requis n'est pas atteint
Sinon : NomConcours : Nombre de candidat atteint , **NomConcours** autorisé**.**

3) Grace a un programme PL/SQL ajouter les concours non autorisé dans une table ReffusConcours qui va contenir ces concours , puis supprimer les de la table Concours
4) On désire connaitre le nombre de sujet pour chaque concours. Créer une fonction qui permet de calculer le nombre d'épreuve pour un concours donné en paramètre (CodeC).
5) Créer une fonction qui permet de calculer le nombre de candidat dans un concours donné en paramètre (CodeC).
6) Créer une Procédure InfoSujetConcours () qui va utiliser les deux fonctions précédentes pour afficher les informations des concours de la manière suivante :
   - **NomConcours** : se déroule le **DateC**, le nombre de Sujet est : **NbreSujet**.

7) À la fin de chaque année universitaire, le ministère désire avoir des informations concernant le niveau moyen des doctorants dans chaque épreuve, de chaque concours. L'information demandée est la suivante : donner pour chaque épreuve, son code, intitulé, le concours pour lequel il a été fait, ainsi le niveau moyen de cette épreuve :
   - si la moyenne des notes dépasse 15, alors niveau : fort,
   - si la moyenne des notes est compris entre 11 et 15, alors niveau : moyen
   - et niveau : faible sinon.

   Proposer une fonction qui facilite cette tache, en spécifiant la requête qui permet d'utiliser et d'exploiter cette fonction.

   a. La fonction InfoNiveau
   b. La requête qui permet de faire cet affichage

3.3. Exercice 3

Soit le schéma relationnel suivant qui représente les données relatives à la coupe du monde de football qui se déroulera à Qatar. La relation Equipe contient toutes les équipes qui vont participer à cette coupe du monde. Dans la relation match nous enregistrons tous les matchs des deux équipes en précisant le stade où va se dérouler ce match. La relation réservation consiste à enregistrer l'ensemble des réservations pour chaque match.

Equipe (Equipe, NbrePoint)
Match (NumMatch, Equipe1#, Equipe2#, DateMatch, Stade#, ScoreEqui1, ScoreEqui2)
Stade(CodeStade, NomStade, Lieu, Capacité)
Reservation(Nreservation, Match#, nbrePlace)

Questions

1) Donner la fonction NbreMatch qui permet de retourner le nombre de match programmé dans le Stade Al Bayt.
2) Faites les modifications nécessaires pour prendre le nom du stade comme paramètre.
3) Créer la procedure MAJPoint (leMatch Match%ROWTYPE) qui permet de mettre à jour le Nombre de points d'une équipe par rapport au résultat du match de la façon suivante : Lorsqu'une équipe gagne un match, elle reçoit 3 points, lorsqu'elle fait match nul, elle reçoit 1 point, et bien sûr une défaite ne rapporte aucun point.
4) Une réservation doit se faire avant 2 jours de la date du match. Créer une fonction droitreservJour(Nmatch Match.NumMatch%type) qui permet de retourner une valeur booléenne , True si oui, False sinon
5) Créer une procédure AjouterReserv(NMatch , NbrePlace) qui permet d'ajouter une réservation tout en utilisant la fonction droitreserv. Sachant que l'attribut Nreservation est autoincrement.

4. Solution des exercices

4.1. Solution Exercice 1

Soit le schéma relationnel :

Employe(NEmp, Nom, grade, Salaire)
Client(NumCli, CINCli, NomCli, AdrCli,TelCli, Nbcpt)
Operation(NumOp, TypeOP, MtOp, NumCpt*, DateOp, Nemp*)
Compte(NumCpt, SoldeCpt, TypeCpt, NumCli*)

1) Créer la procédure AugSalaire qui permet de modifier le salaire d'un employé à partir des paramètres suivant : le numéro de l'employé et le montant à ajouter au salaire de cet employé.

```
CREATE OR REPLACE PROCEDURE AugSalaire(p_empno Employe.NEmp%TYPE,
p_montant Number) IS
V_sal_actuel Employe.Salaire%type;
Err_sal_NULL EXCEPTION;
BEGIN
SELECT Salaire INTO v_sal_actuel FROM Employe
WHERE NEmp = p_empno and p_empno is not null;

IF v_sal_actuel IS NULL THEN RAISE Err_sal_NULL;
END IF;

UPDATE Employe SET Salaire = Salaire + p_montant
WHERE NEmp = p_empno;
EXCEPTION
WHEN Err_sal_NULL THEN
DBMS_OUTPUT.PUT_LINE('salaire non saisie pour cet employé')
END;
END;
```

2) Modifier les salaires de tous les employés en utilisant la procedure AugSalaire

```
CREATE procedure ModifTOUTSalaire() is
CURSOR employes is SELECT NEmp FROM employe;
```

```
BEGIN
For emp in employes LOOP
ModifSalaire(emp.Nemp, 10000);
END LOOP;
END;
```

3) Créer la fonction qui prend en paramètre le numéro de l'employé et retourne le nombre d'opérations effectuées par cet employé.

```
CREATE OR REPLACE FUNCTION NbreOp(p_empno Employe.NEmp%TYPE)
return
integer is
n integer;
BEGIN
SELECT COUNT(*) INTO n FROM Operation WHERE Nemp = p_empno;
return  n;
END ;
```

4) Créer une fonction « SoldeTotal » qui reçoit le numéro de client, et retourne la somme totale de ses comptes.

```
 CREATE OR REPLACE FUNCTION SoldeTotal(p_NCli Client.NumCli%TYPE)
return number is
T_Sold number;
Err_aucun_client EXCEPTION;
Nbr integer ;
BEGIN

SELECT count(*) into Nbr FROM compte WHERE NumCli= p_NCli;
If Nbr = 0 Then raise  Err_aucun_client ; END if ;
SELECT Sum(SoldeCpt) INTO T_Sold FROM Compte
WHERE NumCli= p_NCli;
return T_Sold;
EXCEPTION
WHEN Err_aucun_client THEN
          RAISE_APPLICATION_ERROR(-20020, 'N° client ' ||
TO_CHAR(p_NCli) || ' Non Trouvé') ;;
```

```
END ;
```

5) Créer une fonction « SoldeTotalCompte » qui admet deux paramètres : numéro de client et type de compte, retourne le solde total d'un seul type de compte choisi.

```
CREATE OR REPLACE FUNCTION SoldeTotalCompte(p_NCli
Client.NumCli%TYPE, p_typeCpt Compte.TypeCpt%TYPE )RETURN number
is
T_Sold number;
Err_aucun_client EXCEPTION;

BEGIN
SELECT count(*) into nbr FROM compte WHERE NumCli= p_NCli;
If nbr = 0 Then raise  Err_aucun_client ; END if ;

if p_typeCpt= 'cc' then --cc :compte courant, un client peut avoir
qu'un seul
SELECT SoldeCpt INTO T_Sold FROM Compte
WHERE NumCli= p_NCli
And TypeCpt = 'cc '
END if;

if p_typeCpt= 'ce ' then  --ce : compte epargne, un client peut avoir
plusieurs
SELECT Sum(SoldeCpt) INTO T_Sold FROM Compte
WHERE NumCli= p_NCli
And TypeCpt = 'ce ';
END if;
return  T_Sold;

EXCEPTION
WHEN Err_aucun_client THEN
          RAISE_APPLICATION_ERROR(-20020, 'N° client ' ||
TO_CHAR(p_NCli) || ' Non Trouvé') ;
```

```
END ;
```

6) Créer une procédure stockée **MajSoldeCpt(vNumCpt, vmtOp, vTypeOP)** qui met à jour le solde d'un compte en fonction de l'opération effectuée

```
CREATE PROCEDURE MajSoldeCpt(vNumCpt Compte.NumCpt%type , vmtOp
Operation.MtOp%type,  vTypeOP Operation.TypeOP %type)
AS
vSoldeCpt number; TypeOP_Incorrect EXCEPTION;
BEGIN
   If ((vTypeOP<>'R') and (vTypeOP<>'V')) then Raise
TypeOP_Incorrect; END if;


   SELECT SoldeCpt INTO vSoldeCpt  FROM Compte WHERE NumCpt =
vNumCpt;
If   vTypeOP='R' then vSoldeCpt = vSoldeCpt - vmtOp;
Elseif vTypeOP='V'  then
 vSoldeCpt = vSoldeCpt + vmtOp;
END if;


   UPDATE Compte SET SoldeCpt = vSoldeCpt WHERE NumCpt = vNumCpt;
   INSERT INTO Operation (NumCpt, TypeOP, MtOp, DateOp) VALUES
(vNumCpt, vTypeOP, vmtOp, SYSDATE);
Exception
WHEN TypeOP_Incorrect then DBMS_OUTPUT.PUT_LINE('Type operation
incorrect');


END;
```

7) créer un package qui permet de rassembler les procédures et fonctions précédentes, en plus utiliser des variable globales, ces dernières contiennent le nombre des clients, la moyenne des soldes.

Nous allons prendre seulement les deux premières procédures stockées à savoir : ModifSalaire et NbreOp. Le même principe sera pour les autres.

1. Créer la partie spécification du package

```
CREATE OR REPLACE PACKAGE pkg_Bnq AS
-- Déclaration variables Globales
G_nb_emp number(3);
G_nb_cli number(2);
G_moy_Solde number(2);

-- Définition de la spécification des Fonctions / Procédures
Globales
PROCEDURE ModifSalaire(p_empno Employe.NEmp%TYPE, p_montant
Number) ;
FUNCTION NbreOp(p_empno Employe.NEmp%TYPE)  return
integer;

END;
```

2. Créer la partie corps du package

```
CREATE OR REPLACE PACKAGE BODY Pkg_Bnq AS

-- Définition des corps des Fonctions / Procédures Globales
PROCEDURE ModifSalaire(p_empno Employe.NEmp%TYPE, p_montant
Number)is
V_sal_actuel Employe.Salaire%type;
Err_sal_NULL EXCEPTION;
BEGIN
SELECT Salaire INTO v_sal_actuel FROM Employe
WHERE NEmp = p_empno and p_empno is not null;

IF v_sal_actuel IS NULL THEN RAISE Err_sal_NULL;
END IF;

UPDATE Employe SET Salaire = Salaire + p_montant
WHERE NEmp = p_empno;
EXCEPTION
WHEN Err_sal_NULL THEN
DBMS_OUTPUT.PUT_LINE('salaire non saisie pour cet employé');
END;
```

```
END;

FUNCTION NbreOp(p_empno Employe.NEmp%TYPE)  return
Integer IS
n integer;
BEGIN
SELECT COUNT(*) INTO n FROM Operation WHERE Nemp = p_empno;

return  n;

END ;
BEGIN
--Initialiser la variable G_nb_emp
SELECT count(*)into G_nb_emp
FROM Employe;
--Initialiser la variable G_nb_cli
SELECT count(*)into G_nb_cli
FROM Client;
--Initialiser la variable G_moy_Solde
SELECT avg(SoldeCpt)into G_moy_Solde
FROM Compte;
END;
```

Utilisation du package

```
BEGIN
DBMS_OUTPUT.PUT_LINE ('Nombre des operations de lemploye: 123 est'
|| TO_CHAR(PKg_Bnq.NbreOp(123)));
DBMS_OUTPUT.PUT_LINE ('la moyenne des soldes des clients est' ||
TO_CHAR(PKg_Bnq. G_moy_Solde));

--Pour modifier le salaire de l'employé 123
PKg_Bnq.ModifSalaire(123, 55000) ;
…
END;
```

4.2. Solution Exercice 2

Soit le schéma relationnel suivant qui porte sur la gestion des concours de doctorat:

Concours(CodeC, IntituléC, DateC, nbrePoste)

Candidat(NumC, NomC, PrenomC, DN, Cumul, #Concours)

Epreuve(CodeE, IntituléE, Coef, #Concours)

Ep_Can(#CodeE, #NumC, Note)

Cumul: représente la somme des notes coef des épreuves.*

Coef : représente le coefficient de l'épreuve

Note : représente la note attribuée au candidat dans une épreuve

Afin de faciliter la tâche aux utilisateurs, et d'alléger la programmation coté client, nous allons créer l'ensemble des procédures et fonctions suivantes :

1) Un concours n'est autorisé seulement si le nombre des candidats pour ce concours est 2 fois le nombre de poste ouverts.
2) Créer un programme PL/SQL qui parcourt chaque concours, et calcul le nombre de candidats, si le nombre dépasse la condition afficher :

NomConcours : Nombre de candidat requis n'est pas atteint
Sinon : NomConcours : Nombre de candidat atteint , **NomConcours** autorisé**.**

```
DECLARE
cursor C1 is SELECT codec,NBREPOSTE  FROM concours;
n integer;
BEGIN
for c in c1 loop
SELECT count(*) into n FROM candidat WHERE CONCOURS = c.CODEC;
if   c.NBREPOSTE > n*2 then DBMS_OUTPUT.PUT_LINE(c.CODEC ||' :
Nombre de candidat requis n"est pas atteint');
else DBMS_OUTPUT.PUT_LINE(c.CODEC ||' : Nombre de candidat atteint
, Concours autorisé');
END if;
END LOOP;
END;
```

3) Grace a un programme PL/SQL ajouter les concours non autorisé dans une table ReffusConcours qui va contenir ces concours , puis supprimer les de la table Concours

```
DECLARE
cursor C1 is SELECT codec,DATEC, NBREPOSTE  FROM concours;
n integer;
BEGIN
for c in c1 loop
SELECT count(*) into n FROM candidat WHERE CONCOURS = c.CODEC;
if   c.NBREPOSTE > n*2 then DBMS_OUTPUT.PUT_LINE(c.CODEC ||' :
Nombre de candidat requis n"est pas atteint');
INSERT INTO REFFUSCONCOURS VALUES (c.codec,c.datec, c.NBREPOSTE);

DBMS_OUTPUT.PUT_LINE('supprimer les candidats');
DELETE FROM candidat WHERE CONCOURS=c.codec;

DBMS_OUTPUT.PUT_LINE('supprimer le concours');
DELETE FROM CONCOURS WHERE CODEC = c.codec;
END IF;
END LOOP;
END;
```

On désire connaitre le nombre de sujet pour chaque concours.

4) Créer une fonction qui permet de calculer le nombre d'épreuve pour un concours donné en paramètre (CodeC).

```
CREATE or replace FUNCTION NBRE_EP(codeC concours.codec%type)
return integer is
n integer;
BEGIN
SELECT count(*) into n FROM EPREUVE WHERE CONCOURS = CODEC;
return n;
END;
```

5) Créer une fonction qui permet de calculer le nombre de candidat dans un concours donné en paramètre (CodeC).

```
CREATE or replace FUNCTION NBRE_candidat(codeC
concours.codec%type) return integer is
n integer;
BEGIN
SELECT count(*) into n FROM candidat WHERE CONCOURS = CODEC;
 return n;
END;
```

6) Créer une Procédure InfoSujetConcours () qui va utiliser les deux fonctions précédentes pour afficher les informations des concours de la manière suivante :

- **NomConcours** : se déroule le **DateC**, le nombre de Sujet est : **NbreSujet**.

```
CREATE or replace procedure infosujetconcours()  is
declare
cursor C1 is SELECT codec,DATEC, NBREPOSTE  FROM concours;
n integer;
n1 integer;
BEGIN
for c in c1 loop
n1 := NBRE_EP(c.codec);


DBMS_OUTPUT.PUT_LINE('  le nombre epreuve de '|| c.codec ||'  est
' || TO_CHAR(n1) );
DBMS_OUTPUT.PUT_LINE('  le nombre CANDIDAT de '|| c.codec ||'  est
' || NBRE_candidat(c.codec) );
DBMS_OUTPUT.PUT_LINE(c.codec ||': se déroule le ' || c.datec ||' ,
le nombre de sujet est: ' || NBRE_EP(c.codec)  *
NBRE_candidat(c.codec));
END LOOP;
END;
```

7) À la fin de chaque année universitaire, le ministère désire avoir des informations concernant le niveau moyen des doctorants dans chaque épreuve, de chaque concours. L'information demandée est la suivante : donner pour chaque épreuve, son code, intitulé, le concours pour lequel il a été fait, ainsi le niveau moyen de cette épreuve :

- si la moyenne des notes dépasse 15, alors niveau : fort,
- si la moyenne des notes est compris entre 11 et 15, , alors niveau : moyen
- et niveau : faible sinon.

Proposer une fonction qui facilite cette tache, en spécifiant la requête qui permet d'utiliser et d'exploiter cette fonction.

a. La fonction InfoNiveau

```
CREATE OR REPLACE FUNCTION InfoNiveau (num_epreuve varchar) RETURN
varchar
IS
  moynote NUMBER;
BEGIN
  SELECT avg(note) INTO moynote
  FROM epca

  WHERE codee=num_epreuve;
    IF (moynote > 15) THEN
    RETURN('FORT');
  ELSIF (moynote BETWEEN 11 AND 15) THEN
    RETURN('MOYEN');
  ELSE
   RETURN('FAIBLE');
  END IF;
END ;
```

b. La requête qui permet de faire cet affichage est :

```
SELECT CODEE,INTITULE, CONCOURS , INFONIVEAU(CODEE)
FROM epreuve ep;
```

4.3. Solution Exercice 3

Soit le schéma relationnel suivant qui représente les données relatives à la coupe du monde de football qui se déroulera à Qatar. La relation Equipe contient toutes les équipes qui vont participer à cette coupe du monde. Dans la relation match nous enregistrons tous les matchs des deux équipes en précisant le stade où va se dérouler ce match. La relation réservation consiste à enregistrer l'ensemble des réservations pour chaque match.

Equipe (Equipe, NbrePoint)
Match (NumMatch, Equipe1#, Equipe2#, DateMatch, Stade#, ScoreEqui1, ScoreEqui2)
Stade(CodeStade, NomStade, Lieu, Capacité)
Reservation(Nreservation, Match#, nbrePlace)

Questions

1) Donner la fonction NbreMatch qui permet de retourner le nombre de match programmé dans le Stade Al Bayt.

```
CREATE FUNCTION NbreMatch() retrurn integer
Is
N integer;
BEGIN
SELECT count(*) into N FROM Match natural join Stade WHERE
NomStade = 'Al Bayt';
Return N;
END;
```

2) Faites les modifications nécessaires pour prendre le nom du stade comme paramètre.

```
CREATE FUNCTION NbreMatch(NomStade Stade.NomStade%Type) retrurn
integer
Is
N integer;
BEGIN
SELECT count(*) into N FROM Match natural join Stade WHERE
NomStade = NomStade;
Return N;
```

```
END;
```

3) Créer la procedure MAJPoint (leMatch Match%ROWTYPE) qui permet de mettre à jour le Nombre de points d'une équipe par rapport au résultat du match de la façon suivante : Lorsqu'une équipe gagne un match, elle reçoit 3 points, lorsqu'elle fait match nul, elle reçoit 1 point, et bien sûr une défaite ne rapporte aucun point.

```
CREATE procedure MAJPoint (leMatch Match%ROWTYPE) is
BEGIN
If leMatch.ScoreEqui1> leMatch.ScoreEqui2 then
UPDATE Equipe
Set  NbrePoint= NbrePoint +3
WHERE Equipe= leMatch. Equipe1;
END if;
If leMatch.ScoreEqui2> leMatch.ScoreEqui1 then
UPDATE Equipe
Set  NbrePoint= NbrePoint +3
WHERE Equipe= leMatch.Equipe2;
END if;

If leMatch.ScoreEqui1= leMatch.ScoreEqui2 then
UPDATE Equipe
Set  NbrePoint= NbrePoint +1
WHERE Equipe= leMatch.Equipe1;

UPDATE Equipe
Set  NbrePoint= NbrePoint +1
WHERE Equipe= leMatch.Equipe2;
END if;
END;
```

4) Une réservation doit se faire avant 2 jours de la date du match. Créer une fonction droitreservJour(Nmatch Match.NumMatch%type) qui permet de retourner une valeur booléenne , True si oui, False sinon

```
CREATE OR REPLACE FUNCTION droitreserv(Nmatch Match.codec%type)
return boolean
Is

Reserver Boolean :=true;
vDateMatch date;
diff number;
BEGIN
SELECT DateMatch INTO vDateMatch FROM Match WHERE NumMatch =
Nmatch;
diff :=(sysdate - vDateMatch );
If diff> 2 then
Reserver:=True ;
else
Reserver:=false;
END if;
return Reserver;
END;
```

5) Créer une procédure AjouterReserv(NMatch , NbrePlace) qui permet d'ajouter une réservation tout en utilisant la fonction droitreserv. Sachant que l'attribut Nreservation est autoincrement.

```
CREATE Procedure ajouterReserv(NumMatch Match.codec%type ,
NbrePlace integer)  is
Match.codec%type
BEGIN
if droitreserv(NumMatch) then
INSERT Into Reservation(Match, nbrePlace) Values(NumMatch,
NbrePlace);
```

```
DBMS_OUTPUT.PUT_LINE( 'Réservation du match faite avec succès');
Else
DBMS_OUTPUT.PUT_LINE( 'Désolé, dépassement de délais de
réservation');


END if;
END;
```

Série de TD 5 : Les déclencheurs (Triggers)

1. Objectif de la série

Cette série a comme objectif de créer des déclecheurs tout en utilisant du code pl/sql ou des procédures ou fonctions stockées. A partir de cette série, l'étudiant aura la possibilité de :

- ✓ Créer des déclencheurs globaux et lignes
- ✓ Assurer des contraintes que le SGBD ne peut les assurer
- ✓ Réutiliser des procédures stockées dans des triggers
- ✓ Créer des déclencheurs sur une vue

2. Rappel de cours

2.1. Définition

Un Déclencheur (trigger) est un programme dont l'exécution est ***déclenchée*** par un événement (INSERT, UPDATE, DELETE), il n'est pas appelé ***explicitement*** par une application contrairement aux procédures stockées.

2.2. Syntaxe

```
CREATE [OR REPLACE] TRIGGER nomDeclencheur
{BEFORE | AFTER | INSTEAD OF}
{DELETE | INSERT | UPDATE [OF colonne 1, …] [OR …]}
ON {nomTable | nomVue}
[FOR EACH ROW]
[WHEN conditionSupplementaire]
{[DECLARE …]
BEGIN
…
[EXCEPTION …]
END;
```

Syntaxe 12: création de déclencheur

2.3. Type de déclencheurs

Il existe deux types de triggers :

- ➢ **les triggers de table (STATEMENT)** : sont déclenchées une seule fois pour l'instruction INSERT, UPDATE ou DELETE, même si elle traite plusieurs lignes d'un coup.

- **les triggers de ligne (ROW)** : se déclenchent individuellement pour chaque ligne de la table affectée par le trigger. Si l'option FOR EACH ROW est spécifiée, c'est un trigger ligne, sinon c'est un trigger de table. Pour les triggers lignes, on peut introduire une restriction sur les lignes à l'aide d'une expression logique SQL : c'est la clause WHEN.

2.4. Option BEFORE/AFTER

Elle précise le moment quand ORACLE déclenche le trigger.

- Un trigger " BEFORE " s'exécutent avant que l'action considérée soit exécutée, ce qui permet d'empêcher ou de modifier les valeurs du tuple de cette action.
- Un trigger " AFTER " ne pourra plus modifier le tuple considéré et agira seulement sur d'autres tuples.

• les triggers AFTER row sont plus efficaces que les BEFORE row parce qu'ils ne nécessitent pas une double lecture des données

2.5. Manipulation des anciennes et nouvelles valeurs dans les triggers

Pour les triggers de type "FOR EACH ROW", il est possible d'avoir accès à la valeur ancienne et la valeur nouvelle grâce aux mots clés OLD et NEW

```
:old.nom_colonne

:new.nom_colonne
```

- Si l'instruction de déclenchement du trigger est INSERT, seule la nouvelle valeur a un sens.
- Si l'instruction de déclenchement du trigger est DELETE, seule l'ancienne valeur a un sens.

2.6. Les prédicats conditionnels INSERTING, DELETING et UPDATING

Quand un trigger comporte plusieurs instructions de déclenchement (par exemple INSERT OR DELETE OR UPDATE) sur une table, on peut utiliser des prédicats conditionnels (INSERTING, DELETING et UPDATING) pour exécuter des blocs de code spécifiques pour chaque instruction de déclenchement.

```
CREATE TRIGGER ...
BEFORE INSERT OR UPDATE ON employe
.......
BEGIN
......
```

```
IF INSERTING THEN ..... END IF;
IF UPDATING THEN ..... END IF;

......
END;
```

3. Exercices

3.1. Exercice 1

Soit le schéma relationnel suivant décrivant un système de réservations de places des spectacles :

SPECTACLES(nospectacle, nom, durée, placeOffertes, placeLibres, tarif)
CLIENT(noClient, nom, Depense)
RESERVATION(nospectacle#, noClient#, nbrePlaceReserve, dateSpectacle)

Depense est le cumul des réservations

Assurer via des déclencheurs :

1) Les places libres d'un nouveau spectacle sera le même que places offertes
2) Mettre à jour (insertion et suppression)
   a) le champ places libres de chaque réservation
   b) le champ Dépense
3) Empêcher une réservation si le nombre de place libre est atteint

3.2. Exercice 2

Concours(CodeC, IntituléC, DateC, nbrePoste)
Candidat(NumC, NomC, PrenomC, DN, Cumul, #Concours)
Epreuve(CodeE, IntituléE, Coef, #Concours)
Ep_Can(#CodeE, #NumC, Note)

Cumul: représente la somme des notes coef des épreuves.*
Coef : représente le coefficient de l'épreuve
Note : représente la note attribuée au candidat dans une épreuve

1) Créer un déclencheur qui prend comme code d'épreuve : le code du concours +3 lettres du nom d'intitulé.

   Exemple: code concours = SIAD , intituléE = Algorithmique

   → CodeE : SIAD_Alg ;

   La fonction **`SUBSTR(chaine, debut [,longueur] )`**

   Ou

`SUBSTRING(chaine, debut, longueur)` *: retourne la chaîne de caractère "chaine" en partant de la position définie par "début" et sur la longueur*

2) Un concours ne doit pas avoir plus de 3 épreuves.

   Remarque : utiliser la fonction « NBRE_EP » créée dans la série 4, Exercice2

3) Créer un déclencheur qui met à jour le champ cumul après chaque saisie d'une note.

4) Soit la vue suivante :

```
CREATE VIEW ConcoursBis as
SELECT Concours, IntituléC, NumC, NomC, PrenomC
FROM Concours natural join Candidat WHERE DateC ='11/02/2023'
```

   Donner le trigger qui rend l'insertion possible via cette vue

3.3. Exercice 3

L'école '**TlemForm'** veut gérer ces différentes formations dans une base de données. La base de données contient des données portant sur des formations du domaine Informatique, le schéma de la base de données est le suivant :

FORMATIONS (Id_Form, Titre_Form, Duree_Form, type, nbre_fois)
FORMATEURS (Id_F, Nom_F, Adresse_F, Statut, Salaire)
SESSIONS (id_Session, Date_Session, id_Form, id_F#)
APPRENANT(Id_App, Nom_App, Prénom_App, Niveau)
INSCRIPTION(Id_App#, id_Formation#, date_inscr)

Créer les déclencheurs qui permettent de :

1) Mettre le nom du formateur toujours en majuscule
2) Mettez à jour automatiquement le champ nbre_fois. Ce champ contient le nbre de fois qu'une formation est programmée. Proposer une solution avec déclencheur global et ligne
3) Chaque apprenant doit être inscrit à la formation "informatique générale" (existe dèjà dans la table formations). Cad chaque apprenant ajouté, il sera inscrit automatiquement dans la formation « informatique général »
4) Seulement 10 apprenants sont autorisés dans une formation, sauf pour la formation "informatique générale"

5) On veut historiser les différentes opérations effectuées (modification et suppression) sur la table apprenant. Créer la table **CtrlMAJ(NomUser, Evt, dateMAJ)** et enregistrer l'utilisateur et l'opération effectuée. L'attribut « Evt » porte bien sur les valeurs suivantes : UPDATE ou DELETE. Il faut prendre en considération le cas de modification de l'attribut « niveau ».

4. Solution des exercices

4.1. Solution Exercice 1

Soit le schéma relationnel suivant décrivant un système de réservations de places des spectacles :

SPECTACLES(nospectacle, nom, durée, placeOffertes, placeLibres, tarif)
CLIENT(noClient, nom, Depense)
RESERVATION(nospectacle#, noClient#, nbrePlaceReserve, dateSpectacle)

Depense est le cumul des réservations

Assurer via des déclencheurs :

1) Les places libres d'un nouveau spectacle sera le même que places offertes

```
CREATE TRIGGER T1
before INSERT on spectacle
FOR EACH ROW
BEGIN
 :new.placeLibres =:new.placeOffertes;
END ;
```

2) Mettre à jour (insertion et suppression)

   a) le champ places libres de chaque réservation

```
CREATE TRIGGER T2
After INSERT or DELETE on reservation
FOR EACH ROW
BEGIN
 IF INSERTING then
```

```
UPDATE spectacles  SET placeLibres = placeLibres - :new.
nbrePlaceReserve
WHERE  nospectacle = :new.nospectacle;
END IF
IF DELETING then
 UPDATE spectacles
 SET placeLibres = placeLibres + :old.nbrePlaceReserve
 WHERE  nospectacle = :old.nospectacle;
END IF;
END;
```

b) le champ Dépense

```
CREATE TRIGGER T3
After INSERT or DELETE on reservation
FOR EACH ROW
DECLARE
t tariff%type;
BEGIN
 IFINSERTING then
SELECT tarif INTO t FROM spectacles WHERE nospectacle =
:new.nospectacle;
UPDATE client SET depense = depense +(t*:new. nbrePlaceReserve)
WHERE noClient =:new. noClient;
END if;

IF DELETING then
  SELECT tarif INTO t FROM spectacles WHERE nospectacle =
:old.nospectacle;
UPDATE client SET depense = depense -(t*:old. nbrePlaceReserve)
WHERE noClient =:old. noClient;
END IF;
END;
```

OU le tout (a+b) dans un seul déclencheur

```
CREATE TRIGGER T2
```

```
After INSERT or DELETE on reservation
FOR EACH ROW
DECLARE
t tariff%type;
BEGIN
 IFINSERTING then
UPDATE spectacles  SET placeLibres = placeLibres - :new.
nbrePlaceReserve
WHERE  nospectacle = :new.nospectacle;

SELECT tarif INTO t FROM spectacles WHERE nospectacle =
:new.nospectacle;
UPDATE client SET depense = depense +(t*:new. nbrePlaceReserve)
WHERE noClient =:new. noClient;
END IF;

IF DELETING then
   UPDATE spectacles
   SET placeLibres = placeLibres + :old.nbrePlaceReserve
   WHERE  nospectacle = :old.nospectacle;

--recupérer le tarif du spectacle
   SELECT tarif INTO t FROM spectacles
   WHERE nospectacle = :old.nospectacle;

   UPDATE client
   SET depense = depense -(t*:old. nbrePlaceReserve)
   WHERE noClient =:old. noClient;

END IF;
END;
```

3) Empêcher une réservation si le nombre de place libre est atteint

```
CREATE TRIGGER InterdireReserv
before INSERT on RESERVATION
```

```
FOR EACH ROW
Declare nbrLibre Integer;
BEGIN
Select placeLibres into nbrLibre from SPECTACLES where nospectacle
= :new.nospectacle ;
If :new.nbrePlaceReserve> :new.placeLibres then
RAISE_APPLICATION_ERROR(-20999,'Pas de place pour ce spectacle')
END;
```

4.2. Solution Exercice 2

Concours(CodeC, IntituléC, DateC, nbrePoste)

Candidat(NumC, NomC, PrenomC, DN, Cumul, #Concours)

Epreuve(CodeE, IntituléE, Coef, #Concours)

Ep_Can(#CodeE, #NumC, Note)

Cumul: représente la somme des notes* coef des épreuves.
Coef : représente le coefficient de l'épreuve
Note : représente la note attribuée au candidat dans une épreuve

1) Créer un déclencheur qui prend comme code d'épreuve : le code du concours +3 lettres du nom d'intitulé.

Exemple: code concours = SIAD , intituléE = Algorithmique

→ CodeE : SIAD_Alg ;

La fonction **`SUBSTR(chaine, debut [,longueur] )`**

Ou

`SUBSTRING(chaine, debut, longueur)` *: retourne la chaîne de caractère "chaine" en partant de la position définie par "début" et sur la longueur*

```
CREATE or replace TRIGGER codeepreuve
before INSERT on Epreuve
FOR EACH ROW
BEGIN
  :new.CodeE := :new.Concours ||'_'|| SUBSTR(:new. IntituléE,
                              1, 3);
END;
```

2) Un concours ne doit pas avoir plus de 3 épreuves.

Remarque : utiliser la fonction « NBRE_EP » créée dans la série 4, Exercice2

```
CREATE or replace trigger limite3ep
before INSERT on epreuve
FOR EACH ROW
BEGIN
 IF NBRE_EP(:new.Concours)> 3 THEN RAISE_APPLICATION_ERROR(-
20999,'pas plus de 3 APP'); END IF;
END;
```

3) Créer un déclencheur qui met à jour le champ cumul après chaque saisie d'une note.

```
CREATE or replace TRIGGER MAJ_cumul
after INSERT on epca
FOR EACH ROW
declare
coeff integer;

BEGIN
SELECT coef into coeff FROM epreuve WHERE codee= :new.codee;
UPDATE CANDIDAT
set CUMUL = CUMUL + (:new.note * coeff)
WHERE NUMC = :new.numc;
END;
```

4) Soit la vue suivante :

```
CREATE VIEW ConcoursBis as
SELECT Concours, IntituléC, NumC, NomC, PrenomC
FROM Concours natural join Candidat WHERE DateC ='11/02/2023'
```

Donner le trigger qui rend l'insertion possible via cette vue

```
CREATE trigger trVue
Instead of INSERT on ConcoursBis
FOR EACH ROW
Declare n integer ;
BEGIN
SELECT count(*) into n FROM councours WHERE codeC= :new.concours;
If n= 0 then -- verifier si le concours n'existe pas , il faut
l'ajouter
```

```
INSERT INTO concours (CodeC, IntituléC, DateC) values (:new.
Concours, :new. IntituléC, '11/02/2023')
END if ;
INSERT INTO candidat (NumC, NomC, PrenomC, #Concours) values (
:new. NumC, :new. NomC, :new. PrenomC, :new. Concours) ;
END;
```

4.3. Solution Exercice 3

L'école '**TlemForm'** veut gérer ces différentes formations dans une base de données. La base de données contient des données portant sur des formations du domaine Informatique, le schéma de la base de données est le suivant :

FORMATIONS (Id_Form, Titre_Form, Duree_Form, type, nbre_fois)
FORMATEURS (Id_F, Nom_F, Adresse_F, Statut, Salaire)
SESSIONS (id_Session, Date_Session, id_Form, id_F#)
APPRENANT(Id_App, Nom_App, Prénom_App, Niveau)
INSCRIPTION(Id_App#, id_Formation#, date_inscr)

Créer les déclencheurs qui permettent de :

1) Mettre le nom du formateur toujours en majuscule

```
CREATE or replace trigger t1
before INSERT on formateurs
FOR EACH ROW
BEGIN
:new.nom_F := UPPER(:new.nom_F);
END;
```

2) Mettez à jour automatiquement le champ nbre_fois. Ce champ contient le nbre de fois qu'une formation est programmée. Proposer une solution avec déclencheur global et ligne

- Avec déclencheur Ligne

```
CREATE OR REPLACE TRIGGER INCNBREFOIS
AFTER INSERT on sessions

FOR EACH ROW
```

```
BEGIN
 UPDATE FORMATIONS set NBRE_FOIS  = NBRE_FOIS +1 WHERE ID_form =
:new.id_form;


END;
```

- Avec déclencheur global

```
CREATE TRIGGER MAJ
AFTER INSERT OR UPDATE OR DELETE ON sessions
BEGIN
UPDATE FORMATIONS  f
SET  NBRE_FOIS  = (SELECT count(*) FROM sessions s
                   WHERE f.id_form = s.id_form);
END;
```

3) Chaque apprenant doit être inscrit à la formation "informatique générale" (existe dèjà dans la table formations). Cad chaque apprenant ajouté, il sera inscrit automatiquement dans la formation « informatique général »

```
CREATE or replace trigger inscrinfo
after INSERT on apprenant


FOR EACH ROW
DECLARE codeformation integer;
BEGIN
SELECT ID_FORM form into codeformation FROM formations WHERE
titre_form = 'informatique général';


INSERT into INSCRIPTION (Id_App, id_Formation, date_inscr)
values(:new.id_app, f, sysdate);


END;
```

4) Seulement 10 apprenants sont autorisés dans une formation, sauf pour la formation "informatique générale"

```
CREATE or replace trigger APPRENANT10
```

```
before INSERT on inscription


FOR EACH ROW
DECLARE n integer; codeformation integer;
BEGIN
--Récupérer le code de la formation de informatique générale
SELECT ID_FORM form into codeformation FROM formations WHERE
titre_form = 'informatique général';
--seulemnt s'il est différent de ce code qu'on vérifie si le
nombre dépasse 10
if :new.num_form <> codeformation then
 SELECT COUNT(*) into n FROM inscription WHERE num_form=
:new.num_form;
 IF n = 10 THEN raise_application_error(-20999,'pas plus de 10
APP'); END IF;
ENDIF;


END;
```

5) On veut historiser les différentes opérations effectuées (modification et suppression) sur la table apprenant. Créer la table **CtrlMAJ(NomUser, Evt, dateMAJ)** et enregistrer l'utilisateur et l'opération effectuée. L'attribut « Evt » porte bien sur les valeurs suivantes : UPDATE ou DELETE. Il faut prendre en considération le cas de modification de l'attribut « niveau ».

```
CREATE OR REPLACE TRIGGER TRIG_DELETE_OR_UPDATE
AFTER DELETE OR UPDATE ON APPRENANT
FOR EACH ROW
BEGIN
IF UPDATING ('Niveau') THEN
INSERT into CtrlMAJ values(USER, 'UPDATE',sysdate);
DBMS_OUTPUT.PUT_LINE('Le Niveau est Modifié');
END IF ;
IF UPDATING THEN
INSERT into CtrlMAJ values(USER, 'UPDATE',sysdate);
```

```
DBMS_OUTPUT.PUT_LINE('Modifié Avec Succès');
END IF;

IF DELETING THEN
INSERT into CtrlMAJ values(USER, 'DELETE',SYSDATE);
DBMS_OUTPUT.PUT_LINE('Supprimé Avec Succès ');
END IF;
END;
```

REFERENCES BIBLIOGRAPHIQUES

1. Thomas Connolly, Carolyn Begg. Systèmes de bases de données : approche pratique de la conception, de l'implémentation et de l'administration. Eyrolles, 2005.
2. Mohamed Nemiche, Administration du Système Oracle 10g, 2013
3. Christian Soutou. SQL pour Oracle. Editions Eyrolles, 2013.
4. George Gardarin. Bases de données. Eyrolles, 2003.
5. Ian ABRAMSON, Michael ABBEY, Michael COREY, Oracle 10g: Notions Fondamentales, Oracle Press, 2004.
6. Mohamed Fadhel SAAD , PL/SQL sous Oracle 12c : Guide du développeur, 2016.
7. Anne-Sophie LACROIX, Jérôme Gabillaud, Oracle 12c Programmez avec SQL et PL/SQL : Exercices et corrigés, 2014

FACULTÉ DES SCIENCES

Faculty of Sciences

كلية العلوم

Faculté des Sciences – Tidjani HADDAM

Tél.: 043 21 63 70 / Tél&Fax: 043 21 63 68 / 043 21 63 71

Site Web: www.fs.univ-tlemcen.dz

Email : vdrpg.facscience@gmail.com