Rapport Projet BDD

Partie 1:

Question 1:

```
SELECT * FROM Serie;
```

	serieID	TitreSerie	annee	createur	producteur	PaysOrigine	DateCreation	NombreSaison	NombreEpisode	NoteMoyenne	genre
1	1	Serie1	2020-01-01	Auteur1	Producteur1	USA	2020-01-01	1	50	4	Drame
2	2	Serie2	2019-02-01	Auteur2	Producteur2	Canada	2019-02-01	2	100	5	Comédi
3	3	Serie3	2021-03-01	Auteur3	Producteur3	France	2021-03-01	3	110	10	Action

Question 2 :

```
1   SELECT COUNT(DISTINCT PaysOrigine) FROM Serie;
2
```

Results Messages

	(No column name)
1	3

Question 3 :

```
SELECT TitreSerie FROM Serie WHERE PaysOrigine = 'Japon' ORDER BY TitreSerie;
```

	TitreSerie

Question 4 :

```
1   SELECT PaysOrigine, COUNT(*) AS count FROM Serie GROUP BY PaysOrigine;
2
```

Results Messages

	PaysOrigine	count
1	Canada	1
2	France	1
3	USA	1

Question 5 :

```
1   SELECT COUNT(*) FROM Serie WHERE annee BETWEEN '2001-01-01' AND '2015-12-31';
```

Results Messages

	(No column name)
1	0

Question 6 :

```
1  SELECT TitreSerie FROM Serie WHERE genre LIKE '%Comédie%' AND genre LIKE '%Science-Fiction%';
2
```

Results Messages

	TitreSerie

Question 7 :

```
1  SELECT TitreSerie FROM Serie WHERE producteur = 'Spielberg' ORDER BY annee DESC;
```

Results Messages

	TitreSerie

Question 8 :

```
1  SELECT TitreSerie FROM Serie WHERE PaysOrigine = 'USA' ORDER BY NombreSaison ASC;
```

Results Messages

	TitreSerie
1	Serie1

Question 9 :

```
1    SELECT TitreSerie FROM Serie ORDER BY NombreEpisode DESC;
```

Results Messages

	TitreSerie
1	Serie3
2	Serie2
3	Serie1

Question 10 :

```
1    SELECT genre, COUNT(*) AS count FROM Serie WHERE TitreSerie = 'Big Bang Theory' GROUP BY genre;
2
```

Results Messages

	genre	count

Question 11 :

```sql
SELECT TitreSerie FROM Serie WHERE NoteMoyenne < 5 ORDER BY NoteMoyenne;
```

Results Messages

	TitreSerie
1	Serie1

Partie 2:

Contraintes:

1. Essayez de modifier les tables pour ajouter les contraintes suivantes en SQL :
    - La note d'un étudiant doit être comprise entre 0 et 20.
    - Le sexe d'un étudiant doit être dans la liste: 'm', 'M', 'f', 'F' ou Null.
    - Contrainte horizontale : Le salaire de base d'un professeur doit être inférieur au salaire actuel.
    - Contrainte verticale : Le salaire d'un professeur ne doit pas dépasser le double de la moyenne des salaires des enseignants de la même spécialité.

Question 1 :
Pour s'assurer que la note d'un étudiant est comprise entre 0 et 20, on peut utiliser une contrainte CHECK. Cette contrainte garantit que toutes les valeurs entrées dans le champ 'note' seront comprises entre 0 et 20.

Question 2 :
Cette contrainte permet les valeurs 'm', 'M', 'f', 'F', ou NULL pour le champ 'sexe'.

Question 3 :
ALTER TABLE professeurs
ADD CONSTRAINT chk_salaire_base CHECK (salaire_base < salaire_actuel);

Que constatez-vous ?

- Les contraintes CHECK pour les notes et le sexe fonctionneront comme prévu, empêchant l'insertion de données non valides.

- La contrainte horizontale sur le salaire fonctionnera également, mais elle pourrait entraîner des erreurs si les données existantes ne respectent pas déjà cette règle.
- La contrainte verticale sur le salaire ne peut pas être mise en place directement via une contrainte CHECK et nécessite une approche plus complexe, comme un déclencheur. Cela peut être complexe à mettre en œuvre et pourrait avoir un impact sur les performances, car chaque mise à jour ou insertion nécessiterait de recalculer et de comparer le salaire moyen.

Triggers

Question 1 :
Ce trigger s'activera avant la mise à jour des enregistrements dans la table des professeurs pour s'assurer que le salaire ne diminue pas.

Question 2 :
Ce trigger se déclenchera après chaque insertion, suppression, ou mise à jour dans la table des professeurs pour mettre à jour le nombre de professeurs par spécialité dans la table PROF_SPECIALITE.

Question 3 :
Ce trigger s'activera après la suppression d'un professeur dans la table PROFESSEUR ou la modification de son numéro, et mettra à jour la table CHARGE en conséquence.

Securité :
Question 1 :
Ce trigger s'enclenchera à chaque fois qu'une modification est apportée à la table RÉSULTAT et enregistrera ces modifications dans la table AUDIT_RESULTATS.
Notez que USER_NAME() et CURRENT_TIMESTAMP peuvent varier selon le système de gestion de base de données (SGBD) que vous utilisez.

Question 2 :
Ce trigger limitera les augmentations de salaire à moins de 20% sauf pour l'utilisateur 'GrandChef'.

Tests et Validations

Pour tester ces triggers :

- Pour le trigger d'audit, effectuez diverses opérations (insertion, suppression, mise à jour) sur la table RÉSULTAT et vérifiez que les entrées correspondantes sont correctement créées dans la table AUDIT_RESULTATS.
- Pour le trigger de confidentialité, essayez d'augmenter le salaire d'un professeur de plus de 20% avec un utilisateur autre que 'GrandChef' pour voir si l'erreur est déclenchée.

Fonctions et procédures :
Question 1 :
Cette fonction prendra en paramètre l'identifiant d'un étudiant et retournera la moyenne de ses notes. Cette fonction calcule la moyenne des points dans la table resultats pour l'étudiant spécifié par id_etudiant.

Question 2 :
Cette procédure utilise un curseur pour parcourir tous les étudiants de la table etudiants, calcule leur moyenne à l'aide de la fonction fn_moyenne et détermine la mention correspondante.

Tests et Validations

Après avoir créé ces éléments, nous devons tester la fonction fn_moyenne avec divers identifiants d'étudiants pour vous assurer qu'elle calcule correctement la moyenne. Ensuite, exécutez la procédure pr_resultat pour afficher les moyennes et les mentions de tous les étudiants.