


Objectif du TP : Gestion des Stagiaires en SQL

Ce TP a pour objectif d'explorer et de manipuler une base de données SQL dédiée à la gestion des stagiaires. Il permet de mettre en pratique plusieurs concepts fondamentaux de SQL, notamment :

- ☑ **La création et la gestion des bases de données**
- ☑ **La création et modification des tables**
- ☑ **L'insertion, la mise à jour et la suppression de données**
- ☑ **L'utilisation des procédures stockées et fonctions**
- ☑ **L'optimisation des requêtes avec des triggers et transactions**


1. Affichage des bases de données disponibles

```
SHOW DATABASES;
```

 Cette commande affiche toutes les bases de données présentes sur le serveur SQL.






2. Choisir et utiliser la base de données `sql-tp`

```
USE `sql-tp`;
```

 Cette commande sélectionne la base de données `sql-tp` pour y effectuer des opérations.


3. Création de la table `stagiaires`

```
CREATE TABLE IF NOT EXISTS stagiaires (  
    matricule INT AUTO_INCREMENT PRIMARY KEY,  
    nom VARCHAR(50) NOT NULL,  
    prenom VARCHAR(50) NOT NULL,  
    date_naissance DATE NOT NULL,  
    adresse VARCHAR(255) NOT NULL,  
    ville VARCHAR(10) DEFAULT 'Temara'  
);
```

-  **Matricule** : Identifiant unique et auto-incrémenté
 -  **Nom & Prénom** : Champs obligatoires
 -  **Date de naissance** : Doit être renseignée
 -  **Adresse** : Obligatoire
 -  **Ville** : Par défaut "Temara"
-

4. Insertion de stagiaires

```
INSERT INTO stagiaires (nom, prenom, date_naissance, adresse, ville)
VALUES ('Najibi', 'Aymane', '2004-02-03', 'El ghazali', 'Temara');
```

 Ajoute un stagiaire dans la table.

5. Création d'une procédure pour récupérer un stagiaire par son ID

```
DELIMITER //
CREATE PROCEDURE getStagInfo(IN id INT)
BEGIN
    SELECT * FROM stagiaires WHERE matricule = id;
END //
DELIMITER ;
```

 Permet d'afficher un stagiaire en fonction de son ID.

6. Appel de la procédure `getStagInfo`

```
CALL getStagInfo(2);
```

 Affiche les informations du stagiaire ayant l'ID 2.


7. Suppression de la procédure après son utilisation

```
DROP PROCEDURE IF EXISTS getStagInfo;
```

 Supprime la procédure si elle existe déjà.

8. Affichage des tables existantes

```
SHOW TABLES;
```

 Liste les tables de la base de données actuelle.

9. Description de la structure de la table `stagiaires`

```
DESCRIBE stagiaires;
```

🔗 Affiche les colonnes et types de données de la table.

🔴 10. Suppression de la table `stagiaires` si elle existe

```
DROP TABLE IF EXISTS stagiaires;
```

⚠ Supprime complètement la table `stagiaires`.

📄 11. Création d'une procédure pour ajouter un stagiaire

```
DELIMITER //  
CREATE PROCEDURE AddS(IN p_nom VARCHAR(50), IN p_prenom VARCHAR(50), IN  
p_date_naissance DATE, IN p_adresse VARCHAR(255), IN p_ville VARCHAR(10))  
BEGIN  
    INSERT INTO stagiaires (nom, prenom, date_naissance, adresse, ville)  
    VALUES (p_nom, p_prenom, p_date_naissance, p_adresse, p_ville);  
END //  
DELIMITER ;
```

🔗 Permet d'ajouter un stagiaire dynamiquement.

📊 12. Affichage des stagiaires enregistrés

```
SELECT * FROM stagiaires;
```

👁 Affiche tous les stagiaires de la table.

✖ 13. Suppression d'un stagiaire spécifique

```
DELETE FROM stagiaires WHERE matricule = 1;
```

🔗 Supprime le stagiaire ayant le matricule `1`.

⚠ 14. Désactivation du mode sécurisé

```
SET SQL_SAFE_UPDATES = 0;
```

🔔 Permet de modifier ou supprimer toutes les lignes sans restriction.

🗑️ 15. Suppression de toutes les données

```
DELETE FROM stagiaires;
```

⚠️ Vide complètement la table `stagiaires`.

🔒 16. Réactivation du mode sécurisé

```
SET SQL_SAFE_UPDATES = 1;
```

🔒 Rétablit la protection contre les suppressions involontaires.

🗑️ 17. Création d'une procédure pour supprimer un stagiaire

```
DELIMITER //  
CREATE PROCEDURE deleteStagiaire(IN id INT)  
BEGIN  
    DELETE FROM stagiaires WHERE matricule = id;  
END //  
DELIMITER ;
```

✂️ Supprime un stagiaire en fonction de son ID.

✎ 18. Mise à jour de la date de naissance d'un stagiaire

```
DELIMITER //  
CREATE PROCEDURE updateDateNaissance(IN id INT, IN new_date DATE)  
BEGIN  
    UPDATE stagiaires SET date_naissance = new_date WHERE matricule = id;  
END //  
DELIMITER ;
```

📅 Modifie la date de naissance d'un stagiaire.

19. Compter les stagiaires par ville

```
DELIMITER //
```


```
CREATE PROCEDURE countStagiairesByCity(IN city_name VARCHAR(10))
```

```
BEGIN
```

```
    SELECT COUNT(*) AS total_stagiaires FROM stagiaires WHERE ville = city_name;
```

```
END //
```

```
DELIMITER ;
```

 Retourne le nombre de stagiaires d'une ville donnée.

20. Ville avec le plus de stagiaires

```
DELIMITER //
```

```
CREATE PROCEDURE cityWithMostStagiaires()
```

```
BEGIN
```

```
    SELECT ville, COUNT(*) AS total_stagiaires
```

```
    FROM stagiaires
```

```
    GROUP BY ville
```

```
    ORDER BY total_stagiaires DESC
```

```
    LIMIT 1;
```

```
END //
```

```
DELIMITER ;
```


 Retourne la ville avec le plus grand nombre de stagiaires.

21. Suppression et recréation de la base de données

```
DROP DATABASE IF EXISTS gestion_stagiaires;
```

```
CREATE DATABASE gestion_stagiaires;
```

```
USE gestion_stagiaires;
```

 Réinitialise complètement la base `gestion_stagiaires`.

22. Création de la table `stagiaires`

```
CREATE TABLE stagiaires (
```

```
    matricule INT PRIMARY KEY AUTO_INCREMENT,
```

```
    nom VARCHAR(50) NOT NULL,
```

```
    prenom VARCHAR(50) NOT NULL,
```

```
    date_naissance DATE NOT NULL,
```

```
    ville VARCHAR(50) NOT NULL
```

```
);
```

🔗 Crée une table avec les informations des stagiaires.

📝 23. Insertion de données de test

```
INSERT INTO stagiaires (nom, prenom, date_naissance, ville) VALUES
('Dupont', 'Jean', '2000-05-10', 'Paris'),
('Martin', 'Sophie', '1999-07-20', 'Lyon'),
('Durand', 'Paul', '1998-12-15', 'Marseille');
```

🔗 Ajoute trois stagiaires fictifs.

🔍 24. Sélection des stagiaires

```
SELECT * FROM stagiaires;
```

🔗 Affiche la liste des stagiaires.

+ 25. Création d'une procédure d'ajout de stagiaire

```
DELIMITER //
CREATE PROCEDURE addStagiaire(IN nom VARCHAR(50), IN prenom VARCHAR(50), IN
date_naissance DATE, IN ville VARCHAR(50))
BEGIN
    INSERT INTO stagiaires (nom, prenom, date_naissance, ville) VALUES (nom,
prenom, date_naissance, ville);
END //
DELIMITER ;
```

🔗 Ajoute un stagiaire en passant des paramètres.

🍰 26. Fonction pour calculer l'âge

```
DELIMITER //
CREATE FUNCTION getAge(id INT) RETURNS INT DETERMINISTIC
BEGIN
    DECLARE age INT;
    SELECT TIMESTAMPDIFF(YEAR, date_naissance, CURDATE()) INTO age FROM stagiaires
WHERE matricule = id;
    RETURN age;
```

```
END //
DELIMITER ;
```

📅 Calcule l'âge d'un stagiaire.

⚠ 27. Vérification de la date de naissance

```
DELIMITER //
CREATE TRIGGER check_date_naissance BEFORE INSERT ON stagiaires
FOR EACH ROW
BEGIN
    IF NEW.date_naissance > CURDATE() THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'La date de naissance ne peut
pas être dans le futur !';
    END IF;
END //
DELIMITER ;
```

🚫 Empêche l'insertion d'une date future.

🔴 28. Transaction pour supprimer tous les stagiaires

```
DELIMITER //
CREATE PROCEDURE deleteAllStagiaires()
BEGIN
    START TRANSACTION;
    DELETE FROM stagiaires;
    COMMIT;
END //
DELIMITER ;
```

⚠ Supprime tous les stagiaires et confirme l'opération.