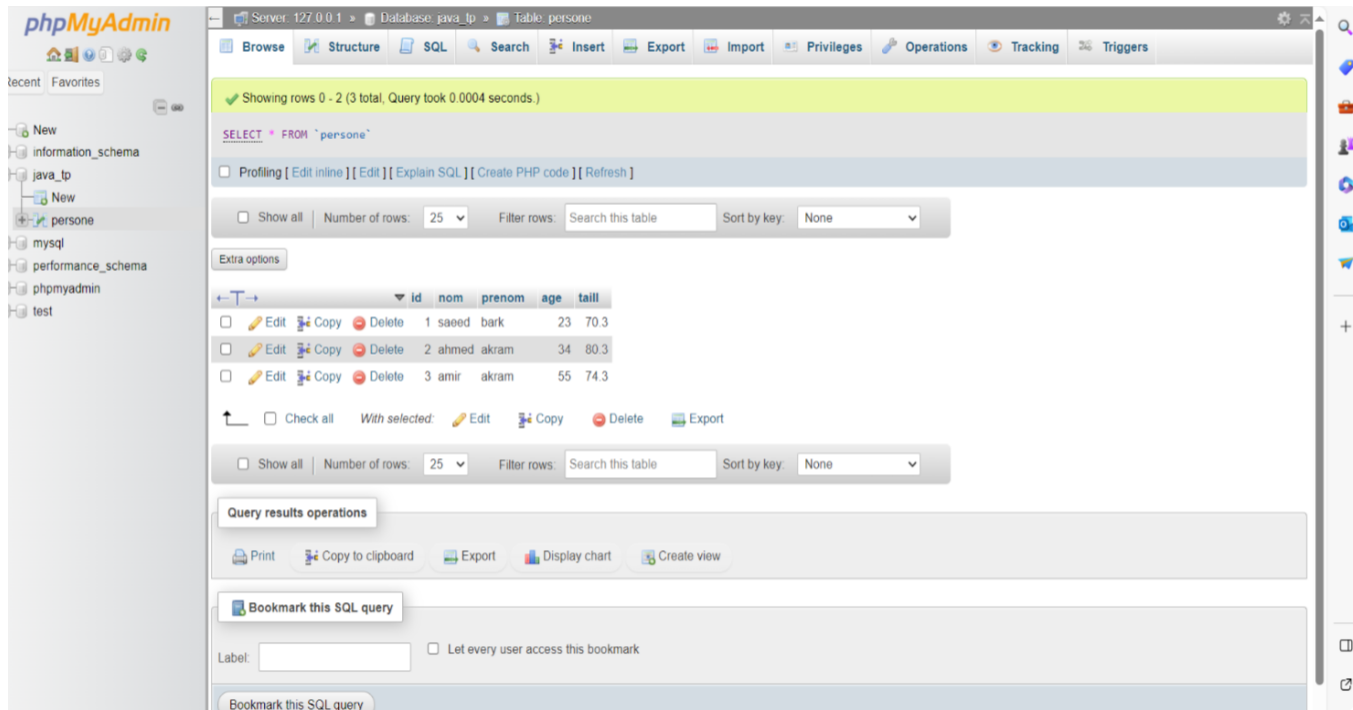


Saeed bark saeed bin gawhar C16430

Cheikh Brahim Moctar C16705

M1 SI



Server: 127.0.0.1 » Database: tp\_java » Table: persone

Showing rows 0 - 2 (3 total, Query took 0.0004 seconds.)

SELECT \* FROM `persone`

Number of rows: 25 Filter rows: Search this table Sort by key: None

	id	nom	prenom	age	taille
<input type="checkbox"/>	1	saeed	bark	23	70.3
<input type="checkbox"/>	2	ahmed	akram	34	80.3
<input type="checkbox"/>	3	amir	akram	55	74.3

Query results operations

Bookmark this SQL query

### Ce Explication de la base de données tp\_java

la base de données tp\_java est une base de ,D'après l'image que vous avez envoyée Cette table contient les ."données MySQL qui contient une table appelée "persone informations suivantes sur les personnes :

Un numéro d'identification unique pour chaque personne : **idnom** •

Le prénom de la personne : **prenom** •

L'âge de la personne : **age** •

La taille de la personne : **taille** •

chacune ,L'image montre également que la table contient trois lignes de données correspondant à une personne différente.

### Voici quelques informations supplémentaires sur la base de données tp\_java :

Le type de base de données est MySQL. •

Le nom de la base de données est "tp\_java". •

Le nom de la table est "persone". •

age" et "taille"." , "prenom" , "idnom" : La table contient quatre colonnes •

La table contient trois lignes de données. •

```

import java.sql.*;

public class ConnectBD {

    Run | Debug
    public static void main(String[] args) {

        try {

            String url = "jdbc:mysql://localhost/java_tp";
            String user = "root";
            String password = "";

            Connection connection = DriverManager.getConnection(url, user, password);

            Statement statement = connection.createStatement();

            System.out.println(x:"Connected");

            ResultSet result = statement.executeQuery(sql:"SELECT * from persone");

            while (result.next()) {
                System.out.println(result.getObject(columnIndex:1).toString());

                System.out.println(x:"|");

                System.out.println(result.getObject(columnLabel:"age").toString());

            }

            statement.executeUpdate(sql:"INSERT INTO `persone` (`id`, `nom`, `prenom`, `age`, `taille`) VALUES (NULL, 'amir', 'akram', '55', '74.1");

```

```

5         System.out.println(x:"|");
6
7         System.out.println(result.getObject(columnLabel:"age").toString());|
8     }
9
10    statement.executeUpdate(sql:"INSERT INTO `persone` (`id`, `nom`, `prenom`, `age`, `taille`) VALUES (NULL, 'amir', 'akram', '
11
12    // Do something with the result set if needed
13    System.out.println(x:"ok");
14
15    // Close resources
16    result.close();
17    statement.close();
18    connection.close();
19
20 } catch (Exception error) {
21     error.printStackTrace();
22 }
23
24 }
25
26

```

```

3 public class ConnectBD {
4     public static void main(String[] args) {
5
6         Statement statement = connection.createStatement();
7
8         System.out.println(x:"Connected");
9
10        ResultSet result = statement.executeQuery(sql:"SELECT * from persone");
11
12        while (result.next()){
13            System.out.println(result.getObject(columnIndex:1).toString());
14
15            System.out.println(x:"|");
16
17            System.out.println(result.getObject(columnLabel:"age").toString());
18        }
19
20        Personne persone = new Personne(nom:"amir", prenom:"akram", age:55, taille:74.3);
21
22        insertPersonne(connection, persone);
23
24
25        // Do something with the result set if needed
26        System.out.println(x:"ok");
27    }
28 }

```

Ce code Java se connecte à une base de données MySQL, exécute une requête SELECT pour récupérer des données à partir d'une table nommée "persone", puis affiche les résultats. Ensuite, il exécute une requête INSERT pour ajouter une nouvelle ligne à cette table. Voici ce que chaque partie du code fait :

### 1. **\*\*Import des classes nécessaires\*\* :**

```

```java
import java.sql.*;
```

```

Cette ligne importe toutes les classes du package `java.sql`, qui contient les classes nécessaires pour interagir avec les bases de données via JDBC (Java Database Connectivity).

### 2. **\*\*Définition de la classe principale\*\* :**

```

```java
public class ConnectBD {
```

```

Cette classe contient la méthode `main` qui est le point d'entrée du programme.

### 3. **\*\*Connexion à la base de données\*\* :**

```

```java
String url = "jdbc:mysql://localhost/java_tp";

String user = "root";

```

```
String password = "";
```

```
Connection connection = DriverManager.getConnection(url, user, password);
```

```
...
```

Ces lignes définissent l'URL de connexion à la base de données MySQL, le nom d'utilisateur et le mot de passe. Ensuite, elles utilisent `DriverManager.getConnection()` pour établir une connexion à la base de données.

#### 4. **\*\*Création d'une déclaration (Statement)\*\* :**

```
```java
```

```
Statement statement = connection.createStatement();
```

```
...
```

Cette ligne crée un objet `Statement` à partir de la connexion établie, qui sera utilisé pour envoyer des requêtes SQL à la base de données.

#### 5. **\*\*Exécution d'une requête SELECT\*\* :**

```
```java
```

```
ResultSet result = statement.executeQuery("SELECT * from persone");
```

```
...
```

Cette ligne exécute une requête SELECT pour récupérer toutes les lignes de la table "persone" et stocke les résultats dans un objet `ResultSet`.

#### 6. **\*\*Parcours et affichage des résultats\*\* :**

```
```java
```

```
while (result.next()) {
```

```
    System.out.println(result.getObject(1).toString());
```

```
    System.out.println(" | ");
```

```
    System.out.println(result.getObject("age").toString());
```

```
}
```

```
...
```

Ces lignes parcourent les lignes du `ResultSet` et affichent les valeurs des colonnes "id" et "age" pour chaque ligne.

#### 7. **\*\*Exécution d'une requête INSERT\*\* :**

```
``java

statement.executeUpdate("INSERT INTO `persone` (`id`, `nom`, `prenom`, `age`, `taille`) VALUES
(NULL, 'amir', 'akram', '55', '74.3')");

...

```

Cette ligne exécute une requête INSERT pour insérer une nouvelle ligne dans la table "persone".

#### 8. **\*\*Fermeture des ressources\*\*** :

```
``java

result.close();

statement.close();

connection.close();

...

```

Ces lignes ferment respectivement le `ResultSet`, le `Statement` et la connexion à la base de données pour libérer les ressources utilisées.

#### 9. **\*\*Gestion des erreurs\*\*** :

```
``java

} catch (Exception error) {

    error.printStackTrace();

}

...

```

En cas d'erreur pendant l'exécution du code, les détails de l'erreur seront imprimés sur la console.

Cependant, il y a une erreur dans le code. La table est appelée "persone" dans la requête SELECT mais "personne" dans la requête INSERT. Assurez-vous que le nom de la table est correctement orthographié dans les deux requêtes.

```

    }

    private static void insertPersonne(Connection connection, Personne personne) throws SQLException {
        String insertQuery = "INSERT INTO `personne` (`id`, `nom`, `prenom`, `age`, `taille`) VALUES (NULL, ?, ?, ?, ?)";

        PreparedStatement preparedStatement = connection.prepareStatement(insertQuery);

        preparedStatement.setString(parameterIndex:2, personne.getPrenom());
        preparedStatement.setInt(parameterIndex:3, personne.getAge());
        preparedStatement.setDouble(parameterIndex:4, personne.getTaille());

        preparedStatement.executeUpdate();

        preparedStatement.close();
    }
}

```

**Corps de la méthode:**

### 1. Préparation de la requête SQL:

- String insertQuery = "INSERT INTO personne(id,nom,prenom,age,taille) VALUES (NULL, ?, ?, ?, ?)"; :
  - Une chaîne de caractères contenant la requête SQL pour insérer une nouvelle ligne dans la table personne.
  - Les valeurs ? sont des placeholders qui seront remplacés par les valeurs réelles de l'objet personne.

### 2. Création d'un prepared statement:

- PreparedStatement preparedStatement = connection.prepareStatement(insertQuery); :
  - Crée un objet PreparedStatement à partir de la requête SQL, permettant d'exécuter la requête de manière sécurisée et efficace.

### 3. Remplacement des placeholders:

- preparedStatement.setString(2, personne.getPrenom());
- preparedStatement.setInt(3, personne.getAge());
- preparedStatement.setDouble(4, personne.getTaille()); :
  - Remplacent les placeholders ? dans la requête SQL par les valeurs correspondantes de l'objet personne.

### 4. Exécution de la requête:

- preparedStatement.executeUpdate(); : Exécute la requête SQL pour insérer la nouvelle ligne dans la table.

### 5. Fermeture du prepared statement:

- preparedStatement.close(); : Libère les ressources associées au prepared statement.

**En résumé, cette méthode permet d'insérer une nouvelle personne représentée par un objet Personne dans la table personne d'une base de données.**

```
public class Personne {  
  
    private int id;  
  
    private String nom;  
  
    private String prenom;  
  
    private int age;  
  
    private double taille;  
  
  
    // Constructors  
  
    public Personne() {  
  
    }  
  
  
    public Personne(String nom, String prenom, int age, double taille) {  
  
        this.nom = nom;  
  
        this.prenom = prenom;  
  
        this.age = age;  
  
        this.taille = taille;  
  
    }  
  
  
    // Getters and setters  
  
    public int getId() {  
  
        return id;  
  
    }  
  
  
    public void setId(int id) {  
  
        this.id = id;  
  
    }  
  
  
    public String getNom() {  
  
        return nom;  
  
    }  
  
}
```

```
public void setNom(String nom) {  
    this.nom = nom;  
}
```

```
public String getPrenom() {  
    return prenom;  
}
```

```
public void setPrenom(String prenom) {  
    this.prenom = prenom;  
}
```

```
public int getAge() {  
    return age;  
}
```

```
public void setAge(int age) {  
    this.age = age;  
}
```

```
public double getTaille() {  
    return taille;  
}
```

```
public void setTaille(double taille) {  
    this.taille = taille;  
}
```

```
// toString method
```

```
@Override
```

```
public String toString() {
```



```

return "Persone{" +
    "id=" + id +
    ", nom=" + nom + "\" +
    ", prenom=" + prenom + "\" +
    ", age=" + age +
    ", taille=" + taille +
    "}";
}
}

```

Le code que vous m'avez fourni définit une classe nommée `Personne` en Java. Voici une explication en français de chaque élément :

#### Propriétés:

- `private int id;` : Un entier privé représentant l'identifiant unique de la personne.
- `private String nom;` : Une chaîne de caractères privée représentant le nom de la personne.
- `private String prenom;` : Une chaîne de caractères privée représentant le prénom de la personne.
- `private int age;` : Un entier privé représentant l'âge de la personne.
- `private double taille;` : Un nombre décimal privé représentant la taille de la personne en mètres.

#### Constructeurs:

- `public Personne() { }` : Un constructeur public sans argument qui crée une instance de `Personne` avec des valeurs par défaut (probablement 0 pour les valeurs numériques et des chaînes vides).
- `public Personne(String nom, String prenom, int age, double taille) { }` : Un autre constructeur public qui prend le nom, le prénom, l'âge et la taille en argument et initialise les propriétés correspondantes de l'instance créée.

#### Accesseurs et Modificateurs:

- Pour chaque propriété privée, il existe une paire de méthodes publiques :
  - Un **accesseur (getter)**, commençant par `get` et prenant le nom de la propriété avec une majuscule initiale, qui retourne la valeur de la propriété.
  - Un **modificateur (setter)**, commençant par `set` et prenant le nom de la propriété avec une majuscule initiale suivi d'un paramètre du même type que la propriété, qui permet de modifier la valeur de la propriété.

#### Méthode `toString`:

- `@Override public String toString() { }`: Cette méthode redéfinit la méthode `toString` héritée de la classe `Object`. Elle permet de renvoyer une chaîne de caractères représentant l'état de l'objet `Personne`, formatée avec les valeurs de ses propriétés.

En résumé, cette classe représente une personne avec ses attributs de base (nom, prénom, âge, taille) et fournit des moyens pour créer des instances, accéder et modifier ses propriétés, et obtenir une représentation textuelle de son état.