Royaume du Maroc
Ministère de l'Education Nationale, du Préscolaire et des Sports
Centre National de l'Évaluation et des Examens

Examen National du Brevet de Technicien Supérieur
Session de Mai 2022
- Sujet -

Filière :	**Développement des Systèmes d'Information - DSI -**	**Durée :**	**4 heures**
Épreuve :	**Développement des Applications Informatiques - DAI -**	**Coefficient :**	**45**

CONSIGNES

- ✓ **Le sujet comporte 3 dossiers ;**
- ✓ **Chaque dossier (ou sous dossier) doit être traité dans une feuille séparée.**

Barème de notation

Dossier 1 : Demandes de maintenance		**22 points**
Sous-dossier 1-1 : Structure de données	11 points	
Sous-dossier 1-2 : Architecture client/serveur	11 points	
Dossier 2 : Attribution des demandes de maintenance		**10 points**
Dossier 3 : Création des demandes de maintenance		**08 points**
Total		**40 points**

- ✓ **Il sera pris en considération la qualité de la rédaction lors de la correction.**
- ✓ **Aucun document n'est autorisé.**

ÉTUDE DE CAS : GPM GESTION DU PROCESSUS DE MAINTENANCE

Le système de gestion de processus de maintenance (**GPM**) est un progiciel qui permet d'assurer la gestion des demandes et le suivi des opérations de maintenance. Le suivi d'une opération peut être sous trois **aspects** : technique, budgétaire et organisationnel. Le processus est composé de plusieurs objets : services, lignes ateliers, machines, équipements, sous-ensembles, pièces, etc. La gestion de ce processus est assurée à partir de terminaux implantés dans des bureaux techniques, ateliers, magasins et bureaux d'approvisionnement. Le scénario du processus de maintenance se fait comme suit :

- À travers une plateforme Web, le client génère sa demande de maintenance. Il peut éventuellement la suivre via la même plateforme.
- Le responsable de maintenance consulte les demandes des clients en utilisant une application Desktop. Il attribue la demande de maintenance à l'agent technique disponible selon sa spécialité.
- L'agent technique prend en charge les demandes reçues et gère ses interventions. Cette gestion est assurée à l'aide d'une application client/serveur.
- Après chaque intervention, le client peut commenter et exprimer sa satisfaction.

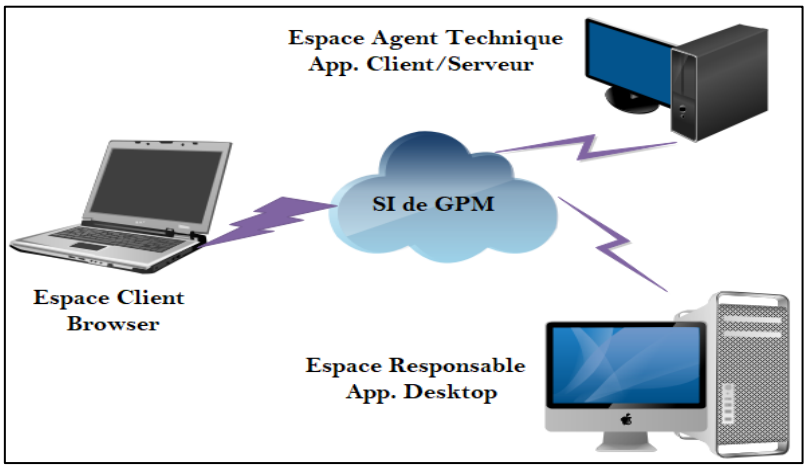


Figure 1 : Schéma structurel des services de maintenance

DOSSIER I : DEMANDES DE MAINTENANCE *(22 points)*

❖ Sous Dossier 1-1 : STRUCTURE DE DONNÉES *(11 points)*

Une demande de maintenance peut être constituée de plusieurs opérations. Une opération concerne le matériel ou le logiciel. Ce processus est modélisé par le diagramme de classes suivant :

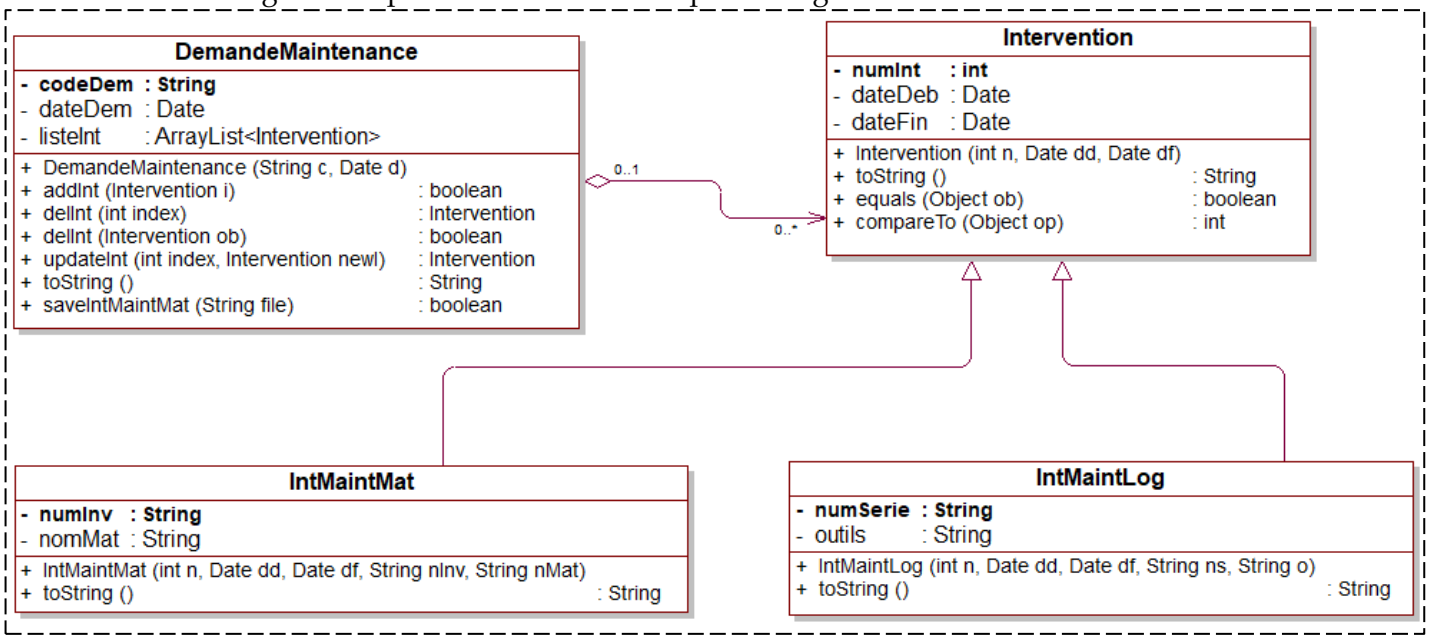


Figure 2 : Diagramme de classes

1. Implémentation de la classe « **Intervention** » :

```
public class Intervention implements Serializable, Comparable {
   private int numInt;
   private Date dateDeb , dateFin;

   public Intervention(...) {...}

   @Override
   public boolean equals(Object ob) { ... }

   @Override
   public String toString() { ... }

    @Override
   public int compareTo() { ... }
}
```

a) Définir un constructeur avec **3** arguments pour initialiser les attributs de cette classe. Ce constructeur lève une exception personnalisée que l'on nomme **« DatesIntInvalides »** *(à définir)* lorsque la date début est supérieure à celle de la fin de l'opération. ***(1 pt)***

b) Redéfinir la méthode **« equals »** qui compare deux interventions selon leurs numéros. ***(0,75 pt)***

c) Redéfinir la méthode **« compareTo »** qui compare deux interventions selon leurs numéros. ***(0,75 pt)***

d) Redéfinir la méthode **« toString »**, afin de retourner une chaine porteuse d'information sur une intervention sous la forme : ***(1 pt)***

```
Numéro : xxxx, Date début : jj/mm/aaaa, Date fin : jj/mm/aaaa
```

2. Implémentation de la classe **« IntMaintMat »** : Intervention de maintenance matériel.

```
public class IntMaintMat extends Intervention {
   private String numInv, nomMat;

   public IntMaintMat(...) { ... }

     @Override
   public String toString() { ... }
}
```

a) Définir un constructeur de **5** arguments pour initialiser tous les attributs de cette classe. ***(1 pt)***

b) Donner la définition de la méthode **« toString »**, afin de retourner une chaine porteuse d'informations sur une intervention de maintenance matériel sous-forme : ***(1 pt)***

```
Numéro :xxxx, Date début :jj/mm/aaaa, Date de fin :jj/mm/aaaa
Intervention de Maintenance Matériel [ numInv : xxxx, nomMat : xxxx]
```

3. Implémentation de la classe **« DemandeMaintenance »** :

```
public class DemandeMaintenance{
   private String codeDem;
   private Date dateDem;
   private ArrayList<Intervention> listeInt;
   public DemandeMaintenance(…) { … }
  public boolean addInt(Intervention op){… }
  public Intervention delInt(int index){ … }
  public boolean delInt(Intervention i){ … }
  public Intervention updateInt(int index, Intervention newI) { … }
  @Override
  public String toString(){ … }
  public boolean saveIntMaintMat(String file){ … }
}
```

Donner le code des méthodes suivantes :

a) Un constructeur avec **2** arguments qui initialise les **2** premiers attributs de la classe **« DemandeMaintenance »** et instancie la collection **« listeInt »** ; ***(0,5 pt)***

b) **addInt(…)** : permet d'ajouter à la collection une nouvelle intervention à condition qu'elle soit unique et retourne l'état de l'opération ; ***(1 pt)***

c) **delInt(int)** : permet de supprimer une intervention de la collection en se basant sur son index et de retourner l'intervention qui vient d'être supprimée. La vérification de la validité de l'index est indispensable ; ***(0,5 pt)***

d) **delInt(Intervention ..)** : permet de supprimer une intervention de la collection en se basant sur un objet **« Intervention »** et de retourner l'état de la suppression ; ***(0,5 pt)***

e) **updateInt(…)** : permet de modifier une intervention déjà existante (*la vérification de la validité de l'index est indispensable*) et de retourner l'intervention avant sa modification ; ***(0,5 pt)***

f) **toString ()** : retourne une chaine de caractères porteuse de toutes les informations d'une demande de maintenance sous-forme : ***(1 pt)***

```
Code Demande : xxxx, Date Demande : jj/mm/aaaa a comme interventions :
1- Numéro:xxxx, Date début :jj/mm/aaaa, Date fin :jj/mm/aaaa

 Intervention de Maintenance Matériel [ numInv : xxxx, nomMat : xxxx]
2- ...
```

g) **saveIntMaintMat (String f)** : permet de sauvegarder toutes les interventions de la maintenance matériel dans un fichier d'objet; ***(1,5 pt)***

❖ Sous Dossier 1-2 : ARCHITECTURE CLIENT/SERVEUR *(11 points)*

Afin qu'un agent de maintenance puisse consulter les interventions d'une demande de maintenance, une architecture client/serveur est mise en place. Son schéma est représenté par la figure suivante :

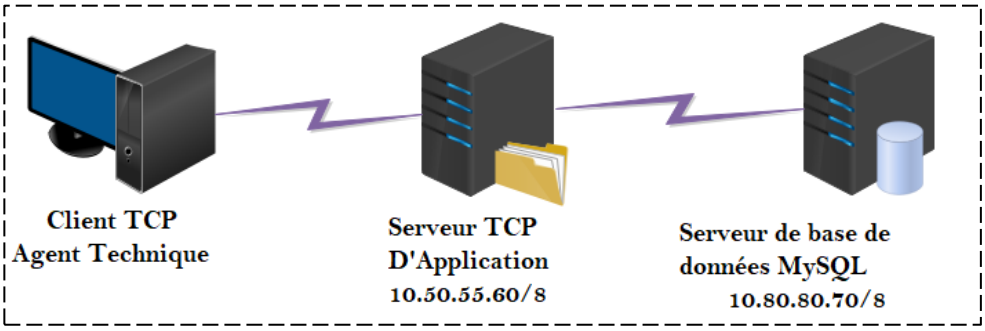


Figure 3 : Architecture client/serveur

Une partie du **MLDR** de la base donnée, implémentée dans le serveur de base de données MySQL, est donnée ci-après :

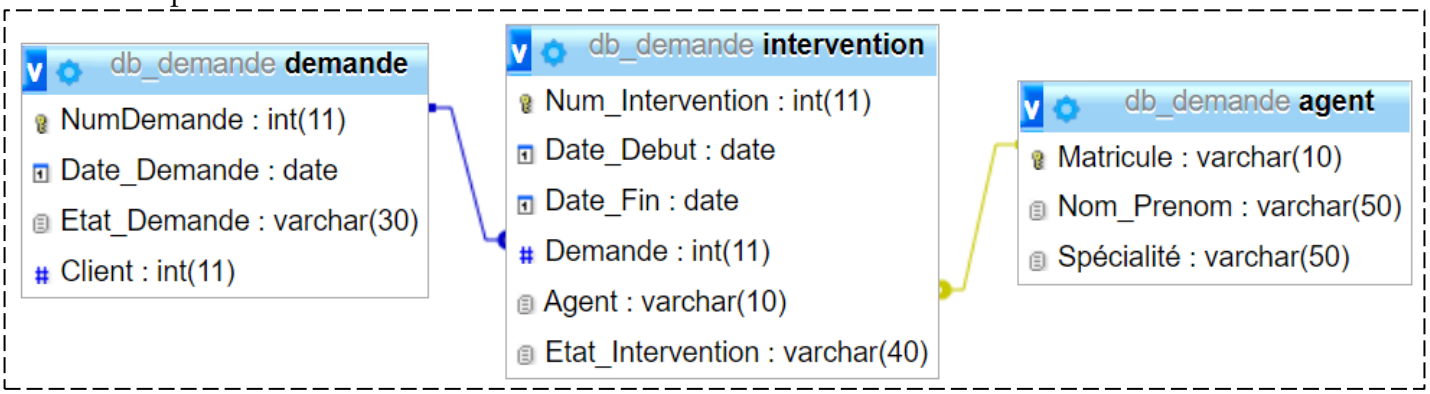


Figure 4 : Partie de MLDR de la base de données du travail

1. S'agit-il d'une architecture **2** tiers ou **3** tiers ? justifier votre réponse. *(0,25 pt)*
2. Préciser la classe d'adressage du serveur d'application. *(0,25 pt)*
3. Donner le masque de réseau correspondant à cette adresse. *(0,25 pt)*
4. Combien de hôtes peut-il y avoir dans ce réseau ? *(0,25 pt)*
5. Architecture serveur :

   Les classes de cette architecture sont :

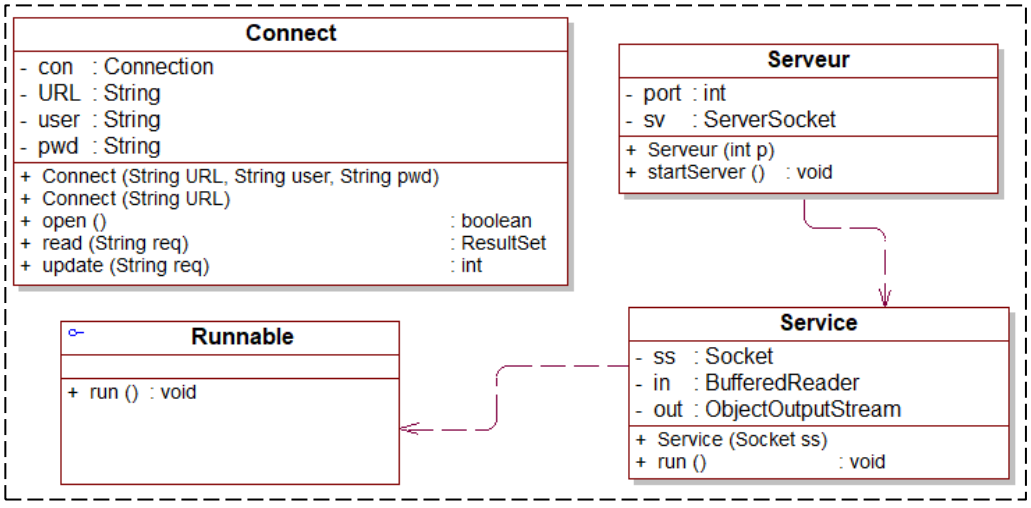


Figure 5 : Classes Serveur

5.1. Classe **: « Connect » :** permet la connexion à la base de données.

Implémenter la classe **« Connect »** qui contient : ***(2 pts)***

- **4** attributs ;
- Un constructeur avec **3** arguments qui initialise les **3** attributs parmi **4** ;
- Un constructeur avec **1** argument qui initialise l'attribut **url** et affecte **user** et **pwd** par **null** ;
- Une méthode **open** qui ouvre une nouvelle connexion avec la base de données et qui retourne l'état de cette opération ;
- Une méthode d'exécution des requêtes de lecture :
- Une méthode d'exécution des requêtes de mise à jour.

```
public class Connect {
   private Connection con;
   private String URL,user,pwd ;
   public Connect (…)  { …… }
   public Connect(String URL)   { …… }
   public boolean open()  { …… }
   public ResultSet read(String req) { …… }
   public int update(String req) { …… }
}
```

5.2. Classe **« Service »** du serveur : ***(3 pts)***

```
public class Service implements Runnable{
   private Socket ss;
   private ObjectOutputStream out;
   private BufferdReader in;

   public Service(...) { ... }
       @Override
   public void run(){
      int numDem;
      ArrayList<Intervention> t;
      Connect con=new Connect(...); [1]
      if(! con.open()) return;
      while(true) {
        Try {
           numDem=Integer.parseInt(in.readLine());
            [2]...
            Thread.sleep(2);
          } catch(Exception ex) { System.out.println(ex.getMessage()); break;}
         }
        }
}
```

Implémenter cette classe en complétant les 2 méthodes suivantes :

- ✓ Un constructeur avec un argument qui initialise l'attribut de type **Socket** et qui prépare les objets de communication (**in** et **out**) ;
- ✓ La méthode **« run »** qui :
  - Établit une connexion avec le serveur de base de données. On rappelle que **:** **URL="jdbc:mysql ://10.80.80.70 :3306/db_maintenance"** **user="admin" et pwd="admin"**
  - Envoie le numéro d'une demande au serveur de base de données pour récupérer les interventions de cette demande en préparant un tableau dynamique d'interventions et envoyer au client ce tableau.

5.3. Classe **« Serveur »** du serveur : ***(2 pts)***

```
public class Serveur {
  private ServerSocket sv;
  private int port;
  public Serveur(...) { ... } [1]
   @Override
   public void startServer(){
      Thread tache;
      Socket s;
      Service ss;
      while(true)  {
         Try {
           s = ...;      [2]
           ss= ...;      [3]
           tache = ...   [4]
           tache.start(); }
          catch(Exception ex) {
              System.out.println(ex.getMessage()); break;}
         }   }
}
```

Implémenter cette classe, ***en complétant les 4 parties demandées***, qui contient :

- ✓ Un constructeur avec un argument qui initialise le port d'écoute et instancie l'objet de type **ServerSocket** ;
- ✓ La méthode **« startServer »** qui permet d'accepter une demande d'un client, préparer un nouveau service et créer un nouveau **Thread**.

6. Architecture client :

La classe de cette architecture est :

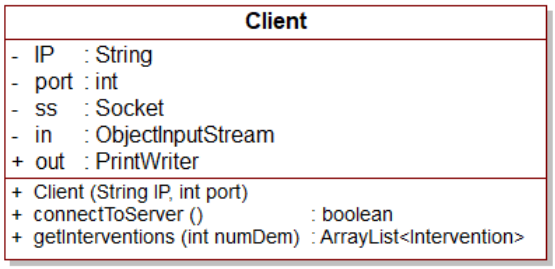


Figure 5 : Classe client

```
public class Client{
   private String IP;    private int port ;
   private ObjectInputStream in;  private PrintWriter out;
   private Socket ss ;
   public Client(...) { ... }
   public boolean connectToServer(){... }
   public ArrayList<Intervention> getInterventions(int numDem){ ... }

     //--------------------code de Test client--
   public static void main(String args[]{
     Client cl= new Client("10.50.55.60",3000);
     if(!cl.connect())
           System.out.println("Echec de connexion au serveur");
      else
            System.out.println(cl.getInterventions(40));  }
}
```

Implémenter cette classe qui contient : ***(3 pts)***

- ✓ Un constructeur avec deux arguments qui initialise les **2** premiers attributs ;
- ✓ La méthode **« connectToServer »** qui instancie le socket **service** en préparant les deux objets de communication (**in** et **out**) et retourne l'état de connexion ;
- ✓ La méthode **« getInterventions »** qui envoie le numéro d'une demande de maintenance et qui reçoit et retourne une collection **ArrayList** des interventions.

DOSSIER II : ATTRIBUTION DES DEMANDES DE MAINTENANCE *(10 points)*

Afin de gérer l'affectation des demandes de maintenance, le responsable de maintenance utilise une application Desktop développée sous Microsoft Visual Basic. Les données sont stockées dans une base de données SQL-Server. Les caractéristiques de ce serveur sont :

- Nom du serveur : **srv_resp** ; Nom de la base de données : **db_maintenance** ;
- Authentification de Windows.

Le **MLDR** de cette base de données est le suivant :

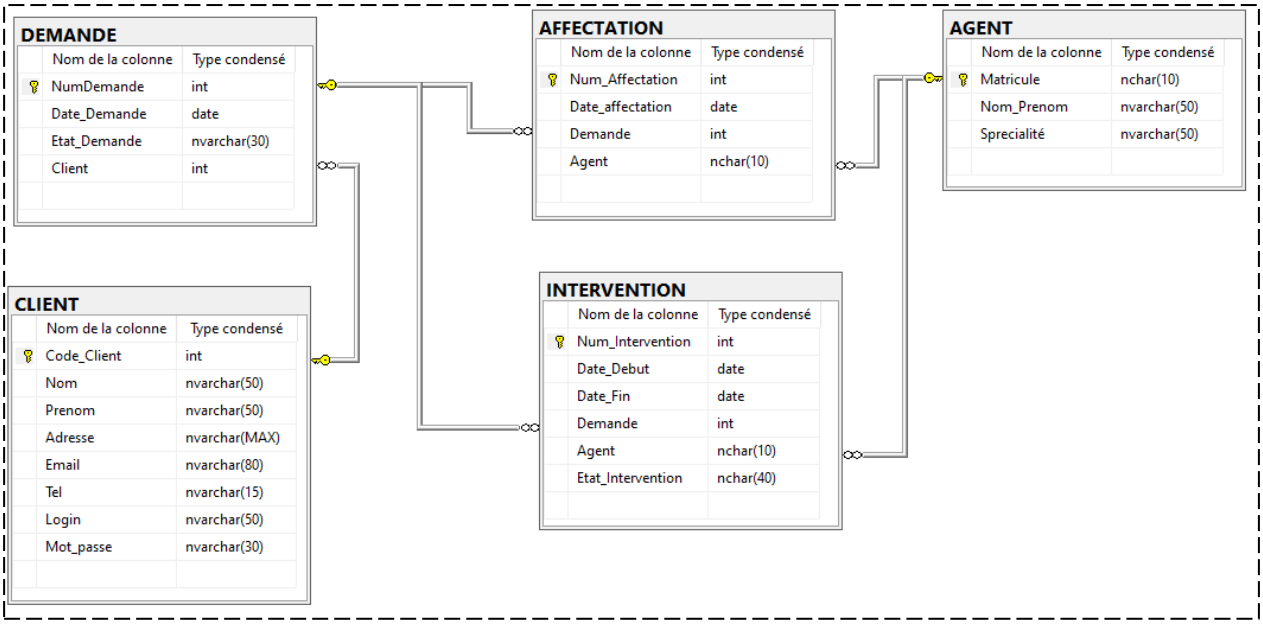


Figure 6 : Modèle logique de données relationnel

Informations complémentaires

- Le champs **«Etat_Demande»** de la table **«DEMANDE»** peut avoir l'une des valeurs suivantes : *En cours, Affectée, Traitée, Annulée.*
- Le champs **«Etat_Intervention»** de la table **«INTERVENTION»** peut avoir l'une des valeurs suivantes : *En cours, reportée, réalisée.*
- Le champs **«Num_Intervention»** s'incrémente automatiquement.

TRAVAIL DEMANDÉ

Consigne : Veuillez indiquer le mode de connexion utilisé (*mode connecté ou non connecté*). Toutes les questions doivent être traitées avec le mode de connexion choisi.

1. Dans le module principal **« Md1.bas »** :

1.1. Déclarer les objets de connexion et les bibliothèques nécessaires pour exploiter ces objets. ***(1 pt)***

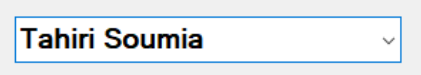
1.2. Créer la fonction « **se_Connecter()** » qui établit une connexion avec le serveur de base de données et retourne «**1**» si la connexion est réussie et «**-1**» dans le cas échéant. ***(2 pts)***

```
Public function se_Connecter() As Integer
    ...
End function
```

2. Créer la fonction « **se_Loguer()** » qui prend en argument le login et le mot de passe du client. Si les données sont correctes la fonction retourne le nom complet du client (*nom et prénom*) ou le message « *Rien* » dans le cas échéant. ***(1,5 pt)***

```
Public function se_Loguer(...) As String
    ...
End function
```

3. Écrire le code de la procédure « **remplir_Clients()** » qui permet de remplir l'objet **ComboBox** nommé « **Cmb_Client** » par les noms et prénoms des clients. ***(1 pt)***




```
Private Sub remplir_Clients()
    ...
End Sub
```

4. Écrire le code de la procédure « **lister_Interventions()** » qui affiche les interventions d'une demande dont la date de début est comprise entre deux dates données. La liste doit être affichée dans l'objet **DataGridView** nommé « **DGV_Liste** » comme suit : ***(2 pts)***

	N° Intervention	Date début	État	Nom d'agent
▶				

```
Private Sub lister_Interventions(ByVal NDemande As Integer, ByVal D1 As Date, ByVal D2 As Date)
    ...
End Sub
```

5. Écrire le code de la procédure « **changer_etat_demande** » qui modifie l'état d'une demande donnée. La procédure prend en argument le numéro de la demande et son nouvel état. ***(1 pt)***

```
Public Sub changer_etat_demande(ByVal NumDemande As Integer, ByVal New_Etat As String)
    ...
End Sub
```

6. Écrire le code de la procédure « **ajouter_Affectation** » qui permet d'insérer une nouvelle affectation. La procédure prend en argument le N° de la demande et le matricule d'agent. Le N° d'affectation est incrémenté par le SGBDR et la date d'affection est celle du système d'application. ***(1,5 pt)***

```
Public Sub ajouter_Affectation(……)
    ...
End Sub
```

DOSSIER III : CRÉATION DES DEMANDES DE MAINTENANCE *(8 points)*

Afin de créer les demandes de maintenance, le client utilise une application Web. Les enregistrements sont stockés dans la base de données **« db_demande »** implémentée sous MYSQL dont une partie de son **MLDR** est la suivante :

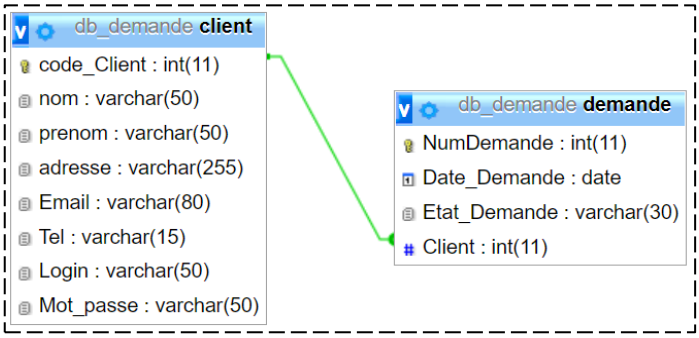


Figure 7 : Partie du Modèle logique de données relationnel

- Nom du serveur : **srv_declaration** ;
- Nom de la base de données : **db_demande** ;
- Nom d'utilisateur : **« user_dep »** et son mot de passe est : user@dep@

La racine du site Web est organisée sous formes de sous dossiers comme suit :

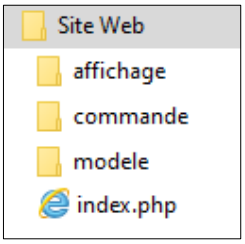


Figure 8 : structure de dossier racine du site Web

Informations complémentaires :

- Le dossier « **affichage** » contient les fichiers PHP permettant l'affichage des données sur le navigateur ;
- Le dossier « **commande** » contient les fichiers PHP permettant le traitement des données ;
- Le dossier « **modele** » contient les fichiers PHP de gestion de la base de données (*connexion*).
- Le fichier **« index.php »** est le fichier principal vu qu'il reçoit des requêtes par **GET/POST** et appelle le fichier convenable du dossier « **commande** ».

TRAVAIL DEMANDÉ

Consigne : Vous devez utiliser un seul mode de programmation : procédural ou orienté objet.

1. Dans le dossier **« modele »**, on crée le fichier **« connecter.php »** qui contient la fonction **« connexion() »** permettant la connexion à la base de données **« db_demande »** en récupérant son identifiant ;
   Écrire le code de la fonction **« connexion() »**. ***(1 pt)***

2. Dans le dossier **« commande »**, on crée les fichiers suivants :
   - **« lectureD.php »** contient la fonction **« recuperer_demandes() »** qui lit les demandes d'un client donné.
   - **« lectureC.php »** contient la fonction **« lire_tous_clients() »** permettant de lire tous les clients.
   - **« ecritureD.php »** contient la fonction **« ajouter_demande() »** qui ajoute une nouvelle demande d'un client.

2.1. On considère la fonction **« lire_tous_clients() »** écrite par 2 méthodes distinctes. Expliquer le code d'une seule des deux méthodes suivantes : ***(2,5 pts)***

Méthode 1 : Programmation procédurale

```
include("./modele/connecter.php") ;                    // ..... [1]
function lire_tous_clients( ) {
   $c = connexion() ;                                  // ..... [2]
   $sql=" select * from client Order by Nom, Prenom " ;  // ..... [3]
   $res = mysqli_query($c,$sql) ;                      // ..... [4]
   $tab = array() ;                                    // ..... [5]
   while($row = mysqli_fetch_row($res)) {
      $tab[] = $row ;                                  // ..... [6]
   }
   include("./affichage/affClients.php") ;
    // appelle le fichier « affClients.php » du dossier « affichage » pour afficher, sous-forme de page web,
   // tous les clients du tableau $tab
}
```

Méthode 2 : PDO

```
include("./modele/connecter.php") ;                    // ..... [1]
function lire_tous_clients( ) {
   $c = connexion() ;                                  // ..... [2]
   $sql=" select * from client Order by Nom, Prenom " ;  // ..... [3]
   $tab = $c->query($sql)->fetchAll(PDO::FETCH_NUM) ;  // ..... [4]
   include("./affichage/affClients.php") ;
    // appelle le fichier « affClients.php » du dossier « affichage » pour afficher, sous-forme de page web,
   // tous les clients du tableau $tab
}
```

2.2. Écrire le code de la fonction **« recuperer_demandes »** qui lit toutes les demandes d'un client donné et appelle le fichier **« afficheDemandes.php »**. Son prototype est le suivant : ***(2,5 pts)***

```
include("./modele/connecter.php") ;
function recuperer_demandes( $idClient) {
     $cd = $idClient ;
      …
}
```

3. Dans le dossier **« affichage »**, on dispose des fichiers suivants : ***(2 pts)***

- **« afficheClients.php »** : qui affiche tous les clients ;
- **« afficheDemandes.php »** : qui affiche les demandes d'un client donné.

Voici un aperçu de la page d'affichage des demandes d'un client :

Liste des demandes du client ayant id=1

Numéro	Date Demande	Etat
1	2021-12-03	traitée
2	2022-01-13	En cours

Figure 9 : Exemple de page d'affichage des demandes d'un client

Reprendre et compléter le code du fichier **« afficheDemandes.php »** :

```
<html> <head> … </head>
  <body>
    <h1> Liste des demandes du client ayant ID : <?php [1] à compléter ?> </h1>
    <table> <thead> <tr>
         <th>Numéro</th> <th>Date Demande</th> <th>État</th>
      </tr> </thead>
      <tbody>
        < ?php
          foreach($tab  as $row) {
               [2] à compléter
          }    ?>  </tbody>
     </table>
  </body>
</html>
```

% Bon courage %