Filière :	Développement des Systèmes d'Information (DSI)	Durée :	4 heures
Épreuve :	Développement des Applications informatiques – DAI -	Coefficient :	45

Dossier 1 : Gestion des branchements électriques. *(14pts)*

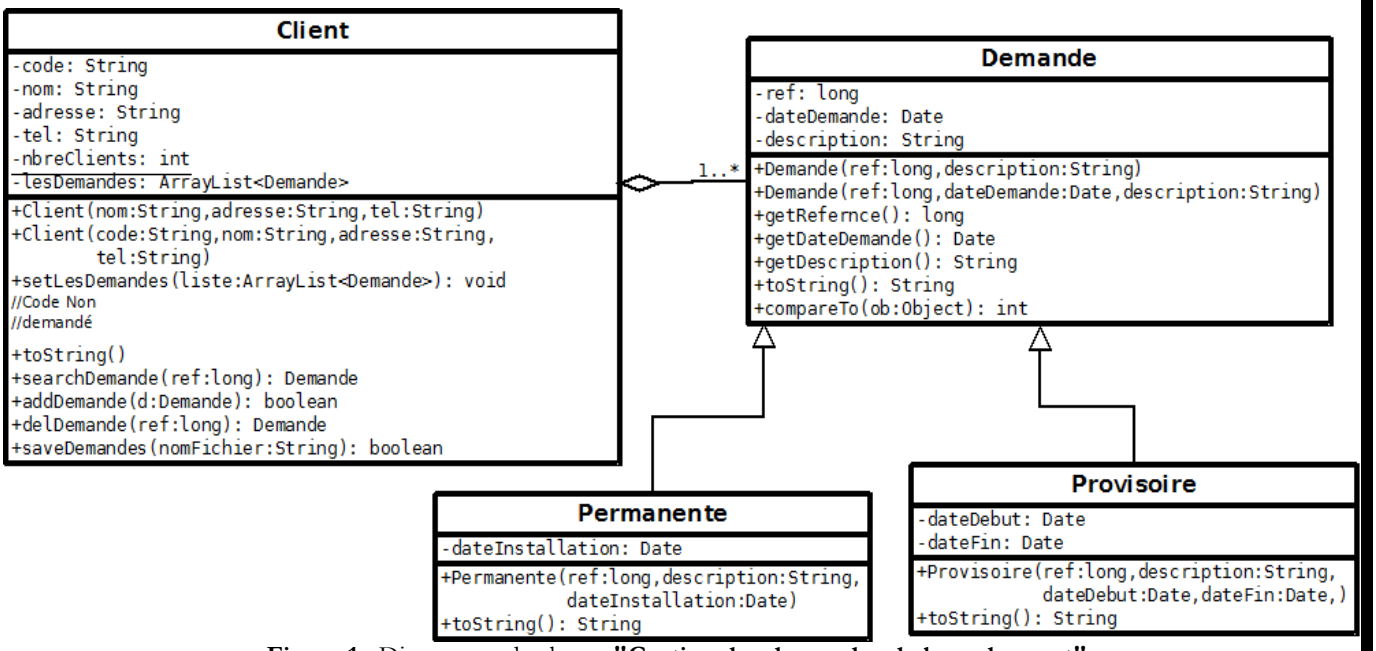


Figure 1 : Diagramme de classes **"Gestion des demandes de branchement"**

1. Créer la classe**« Demande »**qui implémente les deux interfaces « ***Serializable*** », « ***Comparable*** » ***et*** qui contient : ***(0,5 pt)***

- Deux constructeurs, le premier avec deux paramètres (référence et description), la date de la demande sera celle du système, et le deuxième avec trois paramètres qui permet d'initialiser tous les attributs. ***(1 pt)***
- La méthode « **toString** », afin de retourner une chaîne porteuse d'informations sur une demande, la chaîne aura la forme suivante : ***(0,5 pt)***

  Référence : xxxx, Description : xxxx, Date de la demande : jj/mm/aaaa.
- Les accesseurs pour tous les attributs de la classe. ***(0,5 pt)***
- La méthode **compareTo** compare deux demandes par leurs **références**. ***(1 pt)***

```
import java.io.Serializable;
import java.util.Date;

public class Demande implementsSerializable,Comparable  {
private long ref;
private Date dateDemande ;
private String description;
  // constructeur avec deux paramètres (référence et description)
publicDemande(long ref,String description) {
this.ref = ref;
this.description = description;
this.dateDemande=new Date();
}
```

```
//constructeur avec trois paramètres,
   public Demande(long ref, Date dateDemande, String description) {
this.ref = ref;
this.dateDemande = dateDemande;
this.description = description;
   }
   // La méthode « toString »
@Override
public String toString() {
SimpleDateFormatsdf =new SimpleDateFormat("dd/MM/yyyy") ;
return "Reference:" + ref + ", Description :" + description +", Date de demande :" +
sdf.format(dateDemande) ;
   }
   // Les accesseurs des tous les attributs de la classe
public long getRef() {
return ref;
   }
public Date getDateDemande() {
returndateDemande;
   }
public String getDescription() {
return description;
}
// La méthode compareTo
   @Override
   public intcompareTo(Object o) {
      Demande d= (Demande)o;
if(ref>d.ref)
return 1;
else if(ref<d.ref)
return -1;
else
            return 0;
        }
}
```

2. Donner le code de la classe d'exception**ErreurDate**qui permet de récupérer un message d'erreur.

(1 pt)

```
public class ErreurDateextendsException{

   public ErreurDate(String message) {
super(message);
   }

}
```

3. Implémenter la classe **« Permanente»**qui contient : *(0,5 pt)*

- Un constructeur avec trois paramètres qui génère l'exception**ErreurDate**si la date de la demande (date du système) est supérieur à la date d'installation. Le message d'erreur est : "**Erreur de date** ". *(1 pt)*
- La méthode « **toString** » permettant de retourner une chaîne sous la forme suivante : *(1 pt)*
  Référence : xxxx, Description : xxxx, Date de la demande : jj/mm/aaaa, Date d'installation : jj/mm/aaaa.

```
import java.util.Date;
public class Permanente extendsDemande{
private Date dateInstallation;
 // Un constructeur avec trois paramètres qui génère l'exception ErreurDate
   public          Permanente(long          ref,Stringdescription,DatedateInstallation)
throwsErreurDate  {
super(ref, description);
if(dateInstallation.before(new Date()))
throw new ErreurDate("Erreur de date" );
this.dateInstallation = dateInstallation;
   }
   // La méthode « toString »
@Override
public String toString() {
SimpleDateFormatsdf =new SimpleDateFormat("dd/MM/yyyy") ;
return super.toString() + "Date d'installation=" + sdf.format(dateInstallation) ;
}
}
```

NB : Le code de la classe Provisoire est déjà implémenté.

4. Implémenter la classe **« Client »**qui contient : *(0,5 pt)*

- Un constructeur avec trois paramètres permettant d'initialiser : *(1 pt)*
  - Le nom, l'adresse et le téléphone.
  - L'attribut***nbreClients*** est un attribut de classe qui permet de compter le nombre de clients instanciés.
  - *Le* ***Code*** est composé du nom du clientsuivi du numéro de l'ordre de création de ce client.
  - L'instanciation de la collection **lesDemandes** de type **ArrayList**.

NB : on suppose que le constructeur de quatre paramètres est déjà implémenté.

- La méthode « **toString** » permet de retourner une chaîne sous la forme suivante : *(1 pt)*

Code:xxxx, Nom : xxxx, Adresse : xxxx, Tel : xxxx
Les demandes :
Référence : xxxx, Date de la demande : jj/mm/aaaa, Date d'installation : jj/mm/aaaa
…… …… …… …… ……
Référence : xxxx, Date de la demande : jj/mm/aaaa, Date de début : jj/mm/aaaa, Date de fin : jj/mm/aaaa
…… …… …… …… ……

- La méthode **addDemande(Demande d)** : permet d'ajouter unenouvelle demande,donnée en paramètre, à notre collectionet de retourner l'état de l'opération. *(1 pt)*
- La méthode **searchDemande(long ref)** : permet de rechercher et retourner une demande en se basant sur sa référence, si cette demande n'existe pas la méthode retourne nul. *(1 pt)*

- La méthode **delDemande(long ref)** : permet de supprimer unedemande de la collection en se basant sur sa référence et de retournerla demande qui vient d'être supprimée. *(1 pt)*
- La méthode **saveDemandes(String nomFichier)** : permet de sauvegarder, dans un fichier d'objets, les demandes permanentesde ce client. *(1,5 pt)*

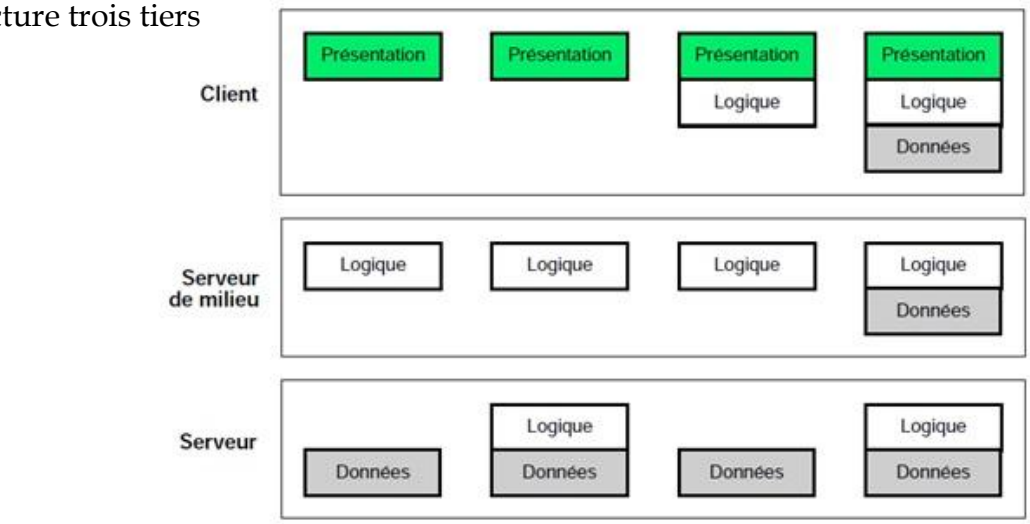
```
import java.io.*;
importjava.util.*;
public class Client {
private String nom,code,adresse,tel;
privatestaticintnbrClient=0;
privateArrayList<Demande>lesDemandes;
  // Un constructeur avec trois paramètres
public Client(String nom, String adresse, String tel) {
nbrClient++;
this.nom = nom;
this.code=nom +nbrClient;
this.adresse = adresse;
      this.tel = tel;
lesDemandes=new ArrayList();
   }
  //  Un constructeur avec quatre paramètres
public Client(String nom, String code, String adresse, String tel) {
this.nom = nom;
this.code = code;
this.adresse = adresse;
this.tel = tel;
lesDemandes=new ArrayList();
   }
  // toString
@Override
   public String toString() {
String chaine = "Nom :" + nom + ", Code :" + code + ", Adresse :" + adresse + ", Tel :" +
tel + "\n Les demandes :  \n";
for(Demande d : lesDemandes)
        chaine+=d.toString()+"\n";
      return chaine;
   }
// méthodeaddDemande
   public booleanaddDemande(Demande d){
      return lesDemandes.add(d);
}
// méthodesearchDemande
   public Demande searchDemande(long ref) {
for(Demande d : lesDemandes)
if(d.getRef()== ref)
return d;
return null;   }
//méthode delDemande
```

```
   public Demande delDemande(long ref){
Demande d=searchDemande(ref);
if(d==null)
return null;
else
returnlesDemandes.remove(lesDemandes.indexOf(d));
}
   // méthode  saveDemandes
   public booleansaveDemandes(String nomFichier) {
try {
ObjectOutputStreamob=new ObjectOutputStream(new
        FileOutputStream(nomFichier));
for(Demande d : lesDemandes)
if(d instanceof Permanente)//d.getClass().getName().equals("Permanente")
{ob.writeObject(d);ob.flush();}
ob.close();
return true;
      } catch (IOException ex) {
return false;}
   }
}
```

Dossier2 : Consultation des demandes des branchements. *(8 pts)*

1. Préciser le type d'architecture client/serveur adopté par ce système et donner le modèle **GartnerGroup** correspondant. *(1 pt)*

Architecture trois tiers



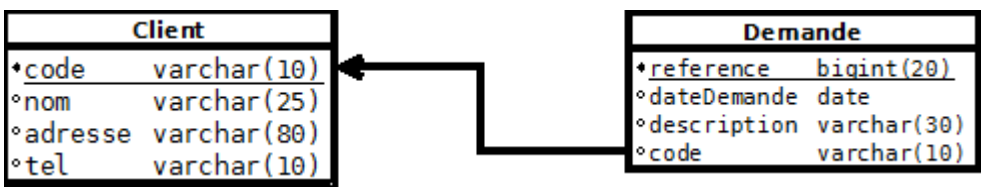


Figure 2 : MLD **"BD_Branchement"**

2. Donner le code de l'interface « **`IntServiceBranchement`**» implémentée par la classe « **ServeurDistant** »afin d'assurer le partage des méthodes de cette classe. ***(1 pt)***

```
public interface IntServiceBranchementextendsRemote{
   public ArrayList<Demande>lireDemandes(String code) throwsRemoteException;
public void lireClients() throws RemoteException;
}
```

3. Compléter la classe « **`ServeurDistant`** » par l'implémentationdes méthodes suivantes :

- Un constructeur permettant d'instancier l'attribut « **liste** » des clients et d'établir une connexion à la base de données MYSQL. ***(1 pt)***
  - ✓ URL:`jdbc:mysql://Administrateur:3306/bd_branchement`
  - ✓ User :***root***
  - ✓ ***Password :***DSI2018.

  En cas d'échec de connexion, afficher un message d'erreur.
- La méthode « **`lireDemandes`** » permet de retourner une collection des demandes dont lecode client est donné en paramètre. ***(1 pt)***
- La méthode « **`lireClients`** » permet de stocker tous les clientsde la table client de la base de données dans l'attribut « **liste** ». Pour chaque client de cette collection, on doit récupérer ses demandes en les sauvegardant dans l'attribut « **lesDemandes** » de l'objet client. ***(1 pt)***

```
importjava.rmi.*;
importjava.sql.*;
importjava.util.*;
public                                                                    class
ServeurDistantextendsUnicastRemoteObjectimplementsIntServiceBranchement{
  privateArrayList<Client>liste;
  privateConnectioncnx;
//constructeur
publicServeurDistant() throwsRemoteException {
super();
  liste=new ArrayList<Client>();
  try {
  cnx=DriverManager.getConnection("jdbc:mysql://Administrateur:3306/bd_brancheme
nt","root","DSI2018");
  } catch (Exception e) {
          e.printStackTrace();    }
}
// La méthode « lireDemandes »
publicArrayList<Demande>lireDemandes(String code) throws RemoteException {
  String req="select * from demande where code=?";
  Demande d;
  ArrayList<Demande> demandes=new ArrayList<Demande>();
  try {
          PreparedStatementst=cnx.prepareStatement(req);
          st.setString(1, code);
          ResultSetrs=st.executeQuery();
          while(rs.next()){
```

```
                d=new Demande(rs.getLong(1),rs.getString(3),rs.getDate(2));
                demandes.add(d);
        }
        returndemandes;
  } catch (Exception e) {
        e.printStackTrace();
        return null;
  }
}
// La méthode « lireClient »
public void lireClient() throws RemoteException {
  String req="select * from Client";
  Client c;
  try {
        Statement st=cnx.createStatement();
        ResultSetrs=st.executeQuery(req);
        while(rs.next()){
                c=new Client(rs.getString(2), rs.getString(1),
                                rs.getString(3), rs.getString(4));
c.setLesDemandes(lireDemandes( rs.getString(1)));
                liste.add(c);
        }
  } catch (SQLException e) {
        e.printStackTrace();    }
}
}
```

4. Écrire le code du ***serveur RMI*** permettant de démarrer l'annuaire ***rmiregistry***, de créer l'objet distant de la classe « **ClientDistant** » et de publier la référence de cet objet dans l'annuairesous le nom «ob». *(1,5 pt)*

```
public class ServeurRMI {
  public static void main(String[] args) {
        try {
                LocateRegistry.createRegistry(1099);
                ClientDistantclt=new ClientDistant();
                Naming.rebind("rmi://serveurRMI:1099/ob", clt);
        } catch (Exception e) {
                e.printStackTrace();
        }
  }
}
```

5. Écrire le code du ***client RMI*** permettant de : *(1,5 pt)*

- Récupérer la référence de l'objet distant,
- Lire et afficher la liste des demandes d'un client dont le code est : « aloui125 ».

```
public class ClientRMI {
  public static void main(String[] args) {
        try {
              IntServiceBranchement
stub=(IntServiceBranchement)Naming.lookup("rmi://serveurRMI:1099/ob");
              ArrayList<Demande>liste=stub.lireDemandes("aloui125 ");
              for(Demande d: liste)
                    System.out.println(d);
        } catch (Exception e) {
              e.printStackTrace();
        }
  }
}
```

Dossier 3 : Gestion des plannings. *(10 pts)*

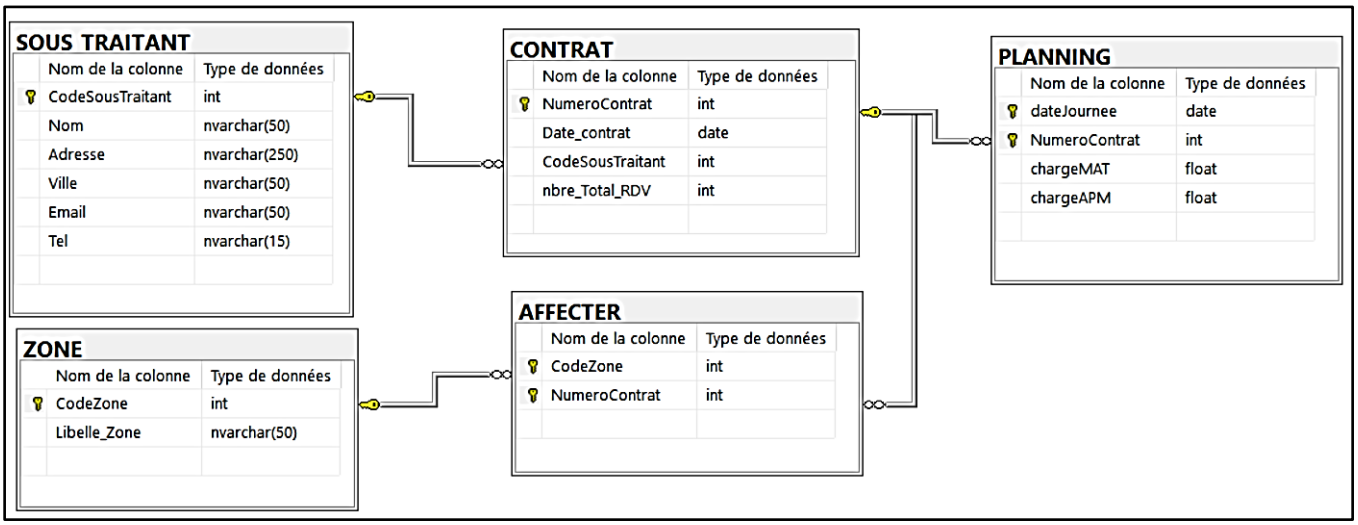


Figure 3 : Extrait du modèle relationnel de la base de données **« DB_Planification »**

1. Déclarer les objets de connexion *(0,5 pt)*

```
Dim CNAsSqlConnection
```

2. Écrire le code de la fonction « **F_Connexion()** » qui retourne **True** si la connexion au serveur de base de données est établie avec succès ou **False** dans le cas échéant. Gérer les exceptions(*nom du serveur : Srv-MSEPE*). *(1 pt)*

Signature de la fonction

```
Public Function F_Connexion() As Boolean
  Try
CN = New SqlConnection("Initial Catalog=DB_Planification ; Data source= Srv-RADEEF ;
Integrated Security=true")
CN.Open()
     Return true
  Catch ex As Exception
           Return false
  End Try
End Function
```

3. Écrire le code de la procédure « **Ps_Affecter()** » qui prend deux arguments (*Code de la zone* et *le numéro de contrat*) puis enregistre les données dans la table « **AFFECTER** ». La procédure doit, tout d'abord, vérifier :

- L'existence du numéro de contrat dans la table « **CONTRAT** » ; ***(0,5 pt)***
- L'existence du code zone dans la table « **ZONE** » ; ***(0,5 pt)***
- L'unicité de la clé composée (**codezone** et **NumeroContrat**) dans la table « **AFFECTER** ». ***(1 pt)***
- Insérer le nouveau enregistrement ***(1 pt)***

Signature de la procédure

```
Public Sub Ps_Affecter(ByValcodeZone As Integer, ByValNumeroContratAsInteger)
Dim cmd1 As New SqlCommand("SELECT count(*) FROM Contrat WHERE
NumeroContrat="+NumeroContrat.ToString, CN)
Dim n As Integer = cmd1.ExecuteScalar()
If n=0 then
MsgBox("Numero de contrat inexistent")
Exit Sub
End if
Dim cmd2 As New SqlCommand("SELECT count(*) FROM Zone WHERE CodeZone="+CodeZone.ToString,
CN)
 n = cmd2.ExecuteScalar()
If n=0 then
MsgBox("Code zone inexistent")
   Exit Sub
End if
Dim cmd3 As New SqlCommand("SELECT count(*) FROM Affecter WHERE
NumeroContrat="+NumeroContrat.ToString+" and CodeZone="+CodeZone.ToString, CN)
n= cmd3.ExecuteScalar()
If n<>0 then
MsgBox("Identifiant de la nouvelle affectation est déjà utilisé")
Exit Sub
End if
Dim cmd4 As New SqlCommand("INSERT INTO Affecter VALUES( "+CodeZone.ToString+","+
NumeroContrat.ToString+")", CN)
 Cmd4.ExecuteNonQuery()
End Sub
```

4. Écrire le code de la procédure « **Supp_Contrat()** » qui prend en argument le numéro du contrat et qui permet de supprimer:

- les affectations de ce contrat ; ***(0,5 pt)***
- les plannings de ce contrat ; ***(0,5pt)***
- le contrat lui-même. ***(0,5pt)***

Signature de la procédure

```
Public Sub Supp_Contrat(ByValNumeroContrat As Integer)
Dim cmd1 As New SqlCommand("DELETE FROM Affecter WHERE
NumeroContrat="+NumeroContrat.ToString, CN)
 cmd1.ExecuteNonQuery()
Dim cmd2 As New SqlCommand("DELETE FROM Planning WHERE
NumeroContrat="+NumeroContrat.ToString, CN)
 cmd2.ExecuteNonQuery()
Dim cmd3 As New SqlCommand("DELETE FROM Contrat WHERE
NumeroContrat="+NumeroContrat.ToString, CN)
 cmd3.ExecuteNonQuery()
EndSub
```

5. Pour lister les interventions prévues ou planifiées pour un contrat donné, le concepteur a mis en place le formulaire suivant :

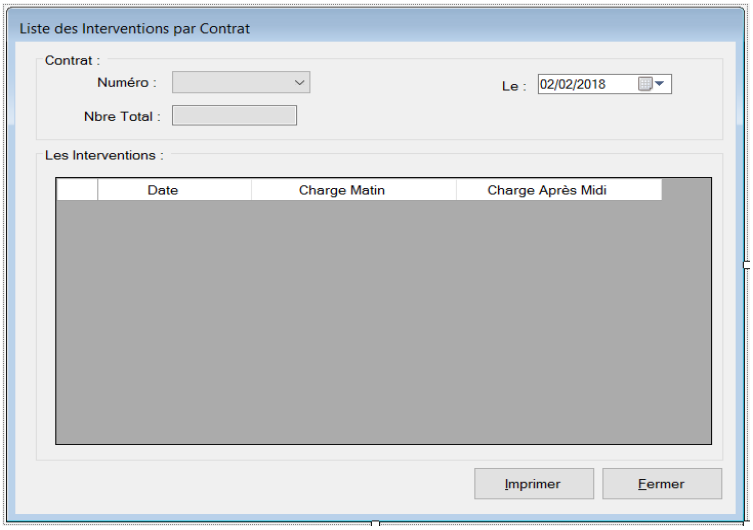


Figure 4 : Formulaire des interventions planifiées par contrat

a. Écrire le code permettant de remplir le **Combobox** « **CmbContrat** » par les numéros des contrats. ***(1 pt)***

Signature de la procédure

```
Public Sub RemplirCombo()
Dim cmd1 As New SqlCommand("SELECT* FROM Contrat,c)
Dim rd As SqlDataReader = cmd1.ExecuteReader()
Dim t As New DataTable
t.load(rd)
rd.close()
CmbContrat.displayMember="NumeroContrat"
CmbContrat.valueMember="NumeroContrat"
CmbContrat.DataSource = t
End Sub
```

b. Écrire le code de la procédure « **Lister_Detail()** » qui liste, dans l'objet **DataGridView** nommé « **DGListe** », les plannings d'un contrat dont les champs sont illustrés dans la figure 5. ***(2 pts)***

Signature de la procédure

```
Public Sub Lister_Detail(ByValNumeroContrat As Integer)
Dim cmd1 As New SqlCommand("SELECT datejournee As Date, chargeMAT As [Charge Matin],
chageAPM As [Charge Après Midi] FROM planning WHERE NumeroContrat="+
NumeroContrat.ToString, CN)
Dim rd As SqlDataReader = cmd1.ExecuteReader()
Dim t As New DataTable
t.load(rd)
rd.close()
DGListe.DataSource = t
EndSub
```

c. Donner le code de la procédure événementielle du comboBox« **CmbContrat** » qui permet de :

- afficher dans la zone texte **txtNbreTotal** le nombre total des rendez-vous du contrat sélectionné dans ce comboBox;
- appeler la procédure « **Lister_Detail()** » pour afficher les planification du contrat sélectionné dans ce comboBox. ***(1 pt)***

Signature de la procédure

```
Public Sub CmbContrat_SelectedIndexChanged(ByVal sender As System.Object, ByVal e As
System.EventArgs)Handles CmbContrat.SelectedIndexChanged
Dim n As Integer = Integer.parse(CmbContrat.SelectedValue.ToString)
Dim cmd1 As New SqlCommand("SELECT nbre_Total_RDV FROM Contrat WHERE NumeroContrat="+
n.ToString, CN)
txtNbreTotal.text=cmd1.ExecutScalar().ToString
Lister_Detail(n)
EndSub
```

Dossier 4 : Suivi à distance des demandes. (8pts)

Le service d'abonnement de la société **MSEPE** souhaite améliorer la gestion des demandes de branchements électriques des clients. Pour ce faire, il propose de mettre en place un site web dynamique pour le suivi à distance de ces demandes. Dans ce contexte, un extrait de la base de données, sous MySQL, nommée «**BD_Demandes**» contient les tables suivantes :

DEMANDE(**refd**, description, dated, **#codec**, **#refop**)
CLIENT(**codec**, civilite, nom, prenom, adresse, ville, mail, tele)
OPERATION(**refop**, detail, type)

Champ	Signification	Type
refd :	Référence de la demande	varchar(25)
description :	Description de la demande	varchar(25)
dated :	Date de la demande	Date
codec :	Code client	varchar(25)
civilite :	Civilité (Mr : 'M', Mme : 'E' , Mlle : 'L')	varchar(25)
tel :	Téléphone	varchar(25)
refop :	Référence de l'opération	varchar(25)
detail :	Détail de l'opération	varchar(25)
type :	Type de la demande (nouvelle :'N', renouvellement :'R', modification :'M')	char(1)

1. Pour que le client accède aux différentes pages de site web, il doit saisir son code dans le formulaire de la page **«index.php»**ou de faire une nouvelle inscription. Le champ code client est validé par la fonction « **valide** ».

Figure 5: Page« index.php »

Le code de la page**«index.php»**est le suivant:

1.a)

```
<script language="JavaScript" >
functionvalide(){
                var tc=document.fcon.code.value;
                if(tc.trim()==""){
                        alert("Entrez un code valide" ) ;
document.fcon.code.focus() ;
                 return false;
                }
                return true;
        }
</script>
```

(1 pt)

```
1.b)<input type='text' size='15' name='code' value=''/>
```
(0,5 pt)

1.c) <input type="submit" value= "Soumettre" name="con" onclick="return valide();"/>
(0,5 pt)

2. On souhaite générer un formulaire permettant d'insérer les coordonnées d'un client dans la base de données si le lien hypertexte de la page index (*Nouvelle inscription*) est actionné.

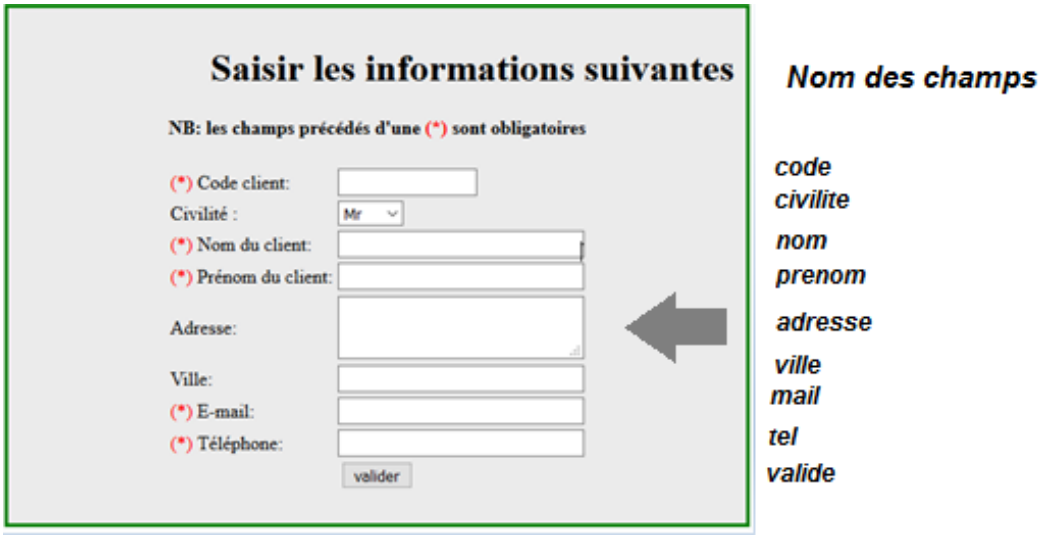


Figure 6: Page « inscription.php»

Le canevas du formulaire ci-dessusest le suivant :

```
<form action='inscriptionPost.php' method='POST' name="fvald">
. .. . . . . . . . . . . . . .
</form></div></center></body></html>
```

Ecrire le script PHP de la page « **inscriptionPost.php** » permettant de :

- Lire le formulaire *(1 pt)*
- Vérifier l'unicité du code client *(1 pt)*
- L'insertion d'un nouveau client *(1 pt)*

```
<?php
$c = $_POST["code"] ;
 $cv = $_POST["cv"] ;
 $n = $_POST["nom"] ;
 $pr = $_POST["prenom"] ;
 $ad = $_POST["adresse"] ;
 $v= $_POST["ville"] ;
 $m = $_POST["mail"] ;
 $t = $_POST["tele"] ;
$con=mysqli_connect("DB_SERVER","root", "DSI2018","BD_Demandes");
$res=mysqli_query($con, "SELECT * FROM CLIENT WHERE codec='$c'");
if(mysqli_num_rows($res)!=0){
echo"<script>alert('Code déjà utilisé') ;</script>" ;
}else{
$req ="insert into client values( '$c','$cv','$n','$pr','$ad','$v','$m','$t')";
mysqli_query($con, $req);
}
?>
```

3. Le bouton « **soumettre** » de la page « **index.php** » nous renvoie vers la page «**rech.php**» (*figure8*). Cette page démarre une session puis enregistreles variables de la session (code_client, nom, prenom) dans le cas où le code existe dans la table client et affiche le contenu de la page de la figure8. Dans le cas contraire, elle nous redirige vers la page « **index.php** ».

Figure 7 : Page « rech.php »

Le canevas de la page « **rech.php** » est le suivant :

```
<html>
<head><meta charset="UTF-8">
<title>Gestion des demandes de branchements électriques</title>
<link href="styles.css" rel="stylesheet">
</head>
<body>
<?php
. . . . . .
?></body></html>
```

Ecrire le code PHP qui permet de :

- Récupérer le code client de la page«**index.php**». *(0,5pt)*
- Vérifier l'existence d'un client ayant le code récupéré précédemment. *(1pt)*
- Si le code existe, il doit mémoriser dans une session les champs **: code client**, **nom**, **prénom** et **civilité**. Il doit inclure le fichier « **affiche.php** » déjà existant. *(1 pt)*
- Dans le cas contraire, il doit nous rediriger vers la page «**index.php**». *(0,5pt)*

```
<?php
$c=$_POST["code"];
 $con=mysqli_connect("DB_SERVER","root", "DSI2018","BD_Demandes") ;
 $req = "SELECT * FROM  clientwhere codec='".$c."'";
$rs = mysqli_query($con, $req);
  if($row =  mysqli_fetch_row($rs)){
session_name("user");
session_start();
$_SESSION['civilite']=$row[1];
$_SESSION['codec']=$row[0];
$_SESSION['nom']=$row[2];
$_SESSION['prenom']=$row[3];
include_once("affiche.php");
}else{
header("location:index.php");
}?>
```