GSB – Documentation technique

1. La structure de données

```sql
use gsb;

SET default_storage_engine = InnoDb;

create table departement  (
    code varchar(2) primary key,
    nom  varchar(50) not null unique
);

create table ville (
    id int auto_increment primary key,
    idDepartement varchar(2),
    idCommune char(5),
    nom varchar(75),
    codePostal char(5)
);

create table famille
(
    id      varchar(3) primary key,
    libelle varchar(80) not null
);

create table medicament
(
    id               varchar(10) PRIMARY KEY,
    nom              varchar(25)  not null,
    composition      varchar(255) not null,
    effets           varchar(255),
    contreIndication varchar(255),
    idFamille        varchar(3)   not null references famille
);
```

Table	Action	Lignes	Type	Interclassement	Taille	Perte
departement	Parcourir Structure Rechercher Insérer Vider Supprimer	96	InnoDB	utf8mb3_general_ci	32,0 kio	-
echantillon	Parcourir Structure Rechercher Insérer Vider Supprimer	5	InnoDB	utf8mb3_general_ci	32,0 kio	-
famille	Parcourir Structure Rechercher Insérer Vider Supprimer	20	InnoDB	utf8mb3_general_ci	16,0 kio	-
levisiteur	Parcourir Structure Rechercher Insérer Éditer Supprimer	~0	Vue	---	-	-
medicament	Parcourir Structure Rechercher Insérer Vider Supprimer	28	InnoDB	utf8mb3_general_ci	32,0 kio	-
mesechantillons	Parcourir Structure Rechercher Insérer Éditer Supprimer	~0	Vue	---	-	-
mespraticiens	Parcourir Structure Rechercher Insérer Éditer Supprimer	~0	Vue	---	-	-
mesvilles	Parcourir Structure Rechercher Insérer Éditer Supprimer	~0	Vue	---	-	-
mesvisites	Parcourir Structure Rechercher Insérer Éditer Supprimer	~0	Vue	---	-	-
motif	Parcourir Structure Rechercher Insérer Vider Supprimer	5	InnoDB	utf8mb3_general_ci	16,0 kio	-
praticien	Parcourir Structure Rechercher Insérer Vider Supprimer	107	InnoDB	utf8mb3_general_ci	80,0 kio	-
specialite	Parcourir Structure Rechercher Insérer Vider Supprimer	44	InnoDB	utf8mb3_general_ci	16,0 kio	-
typepraticien	Parcourir Structure Rechercher Insérer Vider Supprimer	5	InnoDB	utf8mb3_general_ci	16,0 kio	-
ville	Parcourir Structure Rechercher Insérer Vider Supprimer	35 269	InnoDB	utf8mb3_general_ci	2,5 Mio	-
visite	Parcourir Structure Rechercher Insérer Vider Supprimer	18	InnoDB	utf8mb3_general_ci	96,0 kio	-
visiteur	Parcourir Structure Rechercher Insérer Vider Supprimer	65	InnoDB	utf8mb3_general_ci	32,0 kio	-
16 tables	Somme	~35 662	MyISAM	utf8mb3_general_ci	2,9 Mio	0 o

Les déclencheurs

```
create trigger avantAjoutVisite before insert on Visite
for each row
begin
    # Vérification du praticien
    if not exists(select 1 from mesPraticiens where id = new.idPraticien) then
        SIGNAL sqlstate '45000' set message_text = 'Ce praticien n''est pas dans le secteur du visiteur';
    end if;

    # vérification du motif
    if not exists(select 1 from motif where id = new.idMotif) then
        SIGNAL sqlstate '45000' set message_text = 'Ce motif n''existe pas';
    end if;

    # dateEtHeure > à aujourd'hui + 1 heure
    if new.dateEtHeure < now() + interval 1 hour then
        SIGNAL sqlstate '45000' set message_text = 'Il n''est pas possible de programmer un rendez-vous dans moins d''une heure';
    end if;

    # dateEtHeure < à aujourd'hui + 2 mois
    if new.dateEtHeure > now() + interval 2 month then
        SIGNAL sqlstate '45000' set message_text = 'Une visite ne peut être programmée plus de 2 mois à l''avance';
    end if;

    # pas le dimanche
    if WeekDay(new.dateEtHeure) = 6 then
        SIGNAL sqlstate '45000' set message_text = 'Une visite ne peut pas se dérouler un dimanche';
    end if;

    # entre 8 et 19 heures
    if time(new.dateEtHeure) not between time('08:00:00') and time('19:00:00') then
        SIGNAL sqlstate '45000' set message_text = 'Une visite doit se situer entre 8 et 19 heures';
    end if;

    # pas de médicaments présentés ni de bilan à la création
    if new.premierMedicament is not null or new.secondMedicament is not null or new.bilan is not null then
        SIGNAL sqlstate '45000' set message_text =
                'le bilan et les médicaments présentés ne peuvent être renseignés lors de la programmation d''une visite';
    end if;
```

```
create trigger avantMajVisite before update on Visite
for each row
begin
    -- Les champs non modifiables
    if new.id != old.id then
        SIGNAL sqlstate '45000' set message_text = 'L''identifiant d''une visite n''est pas modifiable ';
    end if;
    if new.idPraticien != old.idPraticien then
        SIGNAL sqlstate '45000' set message_text = 'Le praticien visité n''est pas modifiable ';
    end if;
    if new.idVisiteur != old.idVisiteur then
        SIGNAL sqlstate '45000' set message_text = 'Le visiteur n''est pas modifiable ';
    end if;
    -- Si la visite est close (bilan déjà renseigné) plus aucun champ ne doit être modifiable
    if old.bilan is not null then
        SIGNAL sqlstate '45000' set message_text = 'Aucune modification n''est possible sur une visite clôturée ';
    end if;
    -- si la date est modifiée, elle doit respecter les 4 règles : entre 8 et 19 heures hors dimanche, dans les deux mois et séparée d'au moin

    if new.dateEtHeure != old.dateEtHeure then
           # dans une heure et dans 2 mois
           if new.dateEtHeure < now() + interval 1 hour  then
                SIGNAL sqlstate '45000' set message_text = 'Il n''est pas possible de reprogrammer un rendez-vous dans moins d''une heure';
            end if;
           if new.dateEtHeure > now() + interval 2 month then
                SIGNAL sqlstate '45000' set message_text = 'Une visite ne peut être remise à plus de deux mois';
            end if;
            # pas le dimanche
            if WeekDay(new.dateEtHeure) = 6 then
                SIGNAL sqlstate '45000' set message_text = 'La nouvelle date proposée tombe un dimanche';
            end if;
           end if;
                # entre 8 et 19 heures
    if time(new.dateEtHeure) not between time('08:00:00') and time('19:00:00') then
        SIGNAL sqlstate '45000' set message_text = 'Une visite doit se situer entre 8 et 19 heures';
    end if;
    # au moins deux heures entre chaque visite
    if exists(select 1
              from mesVisites
              where abs(timestampdiff(minute, new.dateEtHeure, dateEtHeure) < 120)) then
        SIGNAL sqlstate '45000' set message_text = 'Il faut au moins deux heures d''écart entre deux visites';
    end if;
```

```sql
create trigger avantAjoutEchantillon before insert on Echantillon
for each row
begin

    -- Vérifier si le bilan de la visite est renseigné
    if (select bilan from visite where id = new.idVisite) is null then
        signal sqlstate '45000'
            set message_text = 'Impossible d\'ajouter un échantillon : le bilan de la visite n\'est pas renseigné.';
    end if;

    -- Vérifier le nombre total d'échantillons pour cette visite
    if (select count(*) from echantillon where idVisite = new.idVisite) >= 10 then
        signal sqlstate '45000'
            set message_text = 'Impossible d\'ajouter cet échantillon : le nombre total de médicaments dépasse 10.';
    end if;

    -- Vérifier que la quantité est supérieure à 0
    if new.quantite <= 0 then
        signal sqlstate '45000'
            set message_text = 'La quantité doit être supérieure à 0.';
    end if;
end;
```

Les procédures

```sql
drop procedure if exists ajouterRendezVous;
CREATE PROCEDURE ajouterRendezVous(_idPraticien INT, _idMotif INT, _dateEtHeure DATETIME, OUT _idVisite INT)
    sql security definer
begin
    -- Vérification du praticien
    if not exists(select 1 from mespraticiens where id = _idPraticien) then
        SIGNAL sqlstate '45000' set message_text = 'Ce praticien n''est pas dans le secteur du visiteur.';
    end if;

    -- vérification du motif
    if not exists(select 1 from motif where id = _idMotif) then
        SIGNAL sqlstate '45000' set message_text = 'Ce motif n''existe pas.';
    end if;

    -- dateEtHeure > à aujourd'hui + 1 heure
    if _dateEtHeure < date_add(now(), interval 1 hour) then
        SIGNAL sqlstate '45000' set message_text =
                'Il n''est pas possible de programmer un rendez-vous dans moins d''une heure.';
    end if;

    -- dateEtHeure < à aujourd'hui + 2 mois
    if _dateEtHeure > date_add(now(), interval 2 month) then
        SIGNAL sqlstate '45000' set message_text = 'Une visite ne peut être programmée plus de 2 mois à l''avance.';
    end if;

    -- pas le dimanche
    IF DAYOFWEEK(_dateEtHeure) = 1 THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Une visite ne peut pas se dérouler un dimanche.';
    END IF;

     # entre 8 et 19 heures
    if time(_dateEtHeure) not between time('08:00:00') and time('19:00:00') then
        SIGNAL sqlstate '45000' set message_text = 'Une visite doit se situer entre 8 et 19 heures';
    end if;

   # au moins deux heures entre chaque visite
    if exists(select 1 from mesvisites where abs(timestampdiff(minute, _dateEtHeure, dateEtHeure)) < 120) then
        SIGNAL sqlstate '45000' set message_text = 'Il faut au moins deux heures d''écart entre deux visites';
    end if;
```

```sql
drop procedure if exists modifierRendezVous;
CREATE PROCEDURE modifierRendezVous(_idVisite int, _dateEtHeure datetime)
    sql security definer
begin
    -- vérification de l'identifiant de la visite qui doit faire partie des visites du visiteur connecté
    if not exists(select 1 from mesvisites where id = _idVisite) then
        SIGNAL sqlstate '45000' set message_text = 'Cette visite ne vous concerne pas';
    end if;
    -- la visite ne doit pas être close : bilan renseigné


    # dans au moins 1 heure
    if _dateEtHeure < now() + interval 1 hour then
        SIGNAL sqlstate '45000' set message_text =
                'Il n''est pas possible de reprogrammer un rendez-vous dans moins d''une heure';
    end if;
    # au plus tard dans deux mois
    if _dateEtHeure > now() + interval 2 month then
        SIGNAL sqlstate '45000' set message_text = 'Une visite ne peut être remise à plus de deux mois';
    end if;

    # pas un dimanche
    if WeekDay(_dateEtHeure) = 6 then
        SIGNAL sqlstate '45000' set message_text = 'La nouvelle date proposée tombe un dimanche';
    end if;

    # entre 8 et 19 heures
    if time(_dateEtHeure) not between time('08:00:00') and time('19:00:00') then
        SIGNAL sqlstate '45000' set message_text = 'Une visite doit se situer entre 8 et 19 heures';
    end if;

    # au moins deux heures entre chaque visite
    if exists(select 1 from mesvisites where abs(timestampdiff(minute, _dateEtHeure, dateEtHeure)) < 120) then
        SIGNAL sqlstate '45000' set message_text = 'Il faut au moins deux heures d''écart entre deux visites';
    end if;

    -- les autres contrôles sont à la charge du déclencheur avantMajVisite
    -- modification de la date dans la table visite
    update visite
    set dateEtHeure = _dateEtHeure
    where id = _idVisite;
end;
```

```sql
-- supprimer une visite
drop procedure if exists supprimerRendezVous;

create procedure supprimerRendezVous(_idVisite INT)
    sql security definer
begin
    -- vérification de l'identifiant de la visite qui doit faire partie des visites du visiteur connecté
    if not exists(select 1 from mesvisites where id = _idVisite) then
        SIGNAL sqlstate '45000' set message_text = 'Cette visite ne vous concerne pas';
    end if;

    -- suppression
    delete from visite where id = _idVisite;
```

```sql
-- enregistrer le bilan de la visite
drop procedure if exists enregistrerBilanVisite;
CREATE PROCEDURE enregistrerBilanVisite(_idVisite INT, _bilan TEXT, _premierMedicament VARCHAR(10), _secondMedicament VARCHAR(10)) sql security definer
begin
    -- vérification de l'identifiant de la visite qui doit faire partie des visites du visiteur connecté
    if not exists(select 1 from gsb.mesvisites where id = _idVisite) then
        signal sqlstate '45000' set message_text = 'L''identifiant de la visite n''est pas valide';
    end if;

    -- le paramètre _bilan qui doit être renseigné
    if _bilan is null then
        signal sqlstate '45000' set message_text = 'Le bilan doit être renseigné';
    end if;

    -- vérification du premier médicament
    if not exists(select 1 from gsb.medicament where id = _premierMedicament) then
        signal sqlstate '45000' set message_text = 'le médicament n''est pas valide';
    end if;

    -- le paramètre _premierMedicament doit être renseigné et correspondre à un médicament
    if _premierMedicament is null or not exists(select 1 from medicament where id = _premierMedicament) then
        signal sqlstate '45000' set message_text = 'Le premier médicament doit être renseigné';
    end if;

    -- modification
    update visite
        set bilan            = _bilan,
            premierMedicament = _premierMedicament,
            secondMedicament  = _secondMedicament
    where id = _idVisite;
end;
```

```sql
-- Supprimer les anciens échantillon d'une visite
drop procedure if exists supprimerEchantillon;
CREATE PROCEDURE supprimerEchantillon(_idVisite INT)
    sql security definer
begin
    -- vérification sur _idVisite
    if _idVisite is null or trim(_idVisite) = '' then
        SIGNAL sqlstate '45000' set message_text = 'L\'identifiant de la visite n\'est pas transmis';
    end if;

    if _idVisite not regexp '^[0-9]+$' then
        SIGNAL sqlstate '45000' set message_text = 'L\'identifiant de la visite n\'est pas valide';
    end if;
    -- vérification de l'identifiant de la visite qui doit faire partie des visites du visiteur connecté
    if not exists(select 1 from mesVisites where id = _idVisite) then
        SIGNAL sqlstate '45000' set message_text = 'L\'identifiant de la visite n\'existe pas';
    end if;

    -- suppression
    delete from echantillon where idVisite = _idVisite;

end;
```

Les vues

```sql
drop view if exists leVisiteur;

create view leVisiteur(id, nomPrenom, idDepartement) as
    select id, concat(nom, ' ', prenom), idDepartement
    from visiteur
     where concat(visiteur.id, '@localhost') = user();

-- récupérer les praticiens gérés par le visiteur connecté
-- une praticien qui réside dans le département x (2 premiers caractères de son code postal) est géré par le visiteur affecté au département x (idDepartement)
drop view if exists mesPraticiens;

create view mesPraticiens as
    Select id, nom, prenom, rue, codePostal, ville, telephone, email, idType, idSpecialite
    from praticien
    where substring(codePostal, 1, 2) = (select idDepartement from leVisiteur);

-- récupérer les villes (nom et code postal) situées dans le département géré par le visiteur connecté
drop view if exists mesVilles;
create view mesVilles (nom, codePostal)  as
    select nom, codePostal
    from ville
    where idDepartement = (select idDepartement from leVisiteur)
    order by nom;

-- récupérer les visites du visiteur connecté
drop view if exists mesVisites;
create view mesVisites as
    select id, dateEtHeure, bilan, idMotif, idPraticien, premierMedicament, secondMedicament
    from visite
    where idVisiteur = (select id from leVisiteur)
    order by dateEtHeure;

-- récupérer les échantillons fournis lors des visites pour le visiteur connecté
drop view if exists mesEchantillons;
create view mesEchantillons as
    select idVisite, idMedicament, quantite
    from echantillon
    where idVisite in (select id from mesVisites)
    order by idVisite, idMedicament;
```

2. La classe Passerelle

```csharp
namespace GSB
{
    7 références
    static class Passerelle
    {
        private static MySqlConnection cnx;

        // Vérifier les paramètres de connexion et alimente l'objet globale leVisiteur
        1 référence
        static public bool seConnecter(string login, string mdp, out string message) {
            string chaineConnexion = $"Data Source=localhost;Database=gsb; User Id={login}; Password={mdp}";
            cnx = new MySqlConnection(chaineConnexion);
            bool ok = true;
            message = null;

            try {
                // etablit une connexion saut si une connexion existe déjà
                cnx.Open();

            } catch (MySqlException e) {
                ok = false;
                if (e.Message.Contains("Accès refusé")) {
                    message = "Vos identifiants sont incorrects.";
                } else {
                    message = "Problème lors de la tentative de connexion au serveur.\n";
                    message += "Prière de contacter le service informatique";
                }
            } catch (Exception e) {
                message = e.ToString();
                ok = false;
            }
```

```
        if (ok) {
            // récupération des informations sur le visiteurs depuis la vue leVisiteur
            MySqlCommand cmd = new MySqlCommand("Select nomPrenom from leVisiteur;", cnx);
            try {
                Globale.nomVisiteur = cmd.ExecuteScalar().ToString();
            } catch (MySqlException e) {
                message = "Erreur lors de la récupération de vos paramètres \n";
                message += "Veuillez contacter le service informatique\n";
                ok = false;
            }
        }
        if (ok) message = "Visiteur authentifié";
        return ok;
    }

    // se déconnecter
    1 référence
    static public void seDeConnecter() => cnx.Close();
```

```
    // chargement des données de la base dans les différentes collections statiques de la classe Globale
    // dans cette méthode pas de bloc try catch car aucune erreur imprévisible en production ne doit se produire
    // en cas d'erreur en développement il faut laisser faire le debogueur de VS qui va signaler la ligne en erreur
    // le chargement des données concernant tous les visiteurs (médicament, type praticien, specialite, motif) ne doit être fait qu'une fois
    // si elles sont déja chargées on ne les recherche pas.
    // le chargement des données spécifiques au visiteur connecté doit se faire à chaque fois en vidant les anciennes données
    1 référence
    static public void chargerDonnees()
    {
            MySqlCommand cmd = new MySqlCommand();
            cmd.Connection = cnx;
            MySqlDataReader curseur;

            // chargement des données générales si l'application vient d'être lancée : les médicaments, les types de praticiens, les spécialités
            if (Globale.lesMedicaments.Count == 0)
            {
                // chargement des motifs de visite dans la collection lesMotifs de la classe Globale
                cmd.CommandText = "Select id, libelle from Motif order by libelle;";
                curseur = cmd.ExecuteReader();
                while (curseur.Read()) Globale.lesMotifs.Add(new Motif(curseur.GetInt32("id"), curseur.GetString("libelle")));
                curseur.Close();

                // Chargement des types de praticien dans la collection lesTypes de la classe Globale

                cmd.CommandText = "Select id, libelle from TypePraticien order by libelle;";
                curseur = cmd.ExecuteReader();
                while (curseur.Read()) Globale.lesTypes.Add(new TypePraticien(curseur.GetString("id"), curseur.GetString("libelle")));
                curseur.Close();

                // Chargement des Spécialités dans la collection lesSpecialites de la classe Globale
                cmd.CommandText = "Select id, libelle from Specialite order by libelle;";
                curseur = cmd.ExecuteReader();
                while (curseur.Read()) Globale.lesSpecialites.Add(new Specialite(curseur.GetString("id"), curseur.GetString("libelle")));
                curseur.Close();
```

```
        // Chargement des familles de médicaments dans le dictionnaire lesFamilles de la classe Globale
        cmd.CommandText = "Select id, libelle from Famille order by libelle;";
        curseur = cmd.ExecuteReader();
        while (curseur.Read())
        {
            string id = curseur.GetString("id");
            Globale.lesFamilles.Add(id, new Famille(id, curseur.GetString("libelle")));
        }
        curseur.Close();

        // chargement des médicaments dans la collection lesMedicaments de la classe Globale
        cmd.CommandText = "SELECT id, nom, composition, effets, contreIndication, idFamille FROM medicament order by nom;";
        curseur = cmd.ExecuteReader();
        while (curseur.Read())
        {
            string id = curseur.GetString("id");
            string nom = curseur.GetString("nom");
            string composition = curseur.GetString("composition");
            string effet = curseur.GetString("effets");
            string contreIndication = curseur.GetString("contreIndication");
            string idFamille = curseur.GetString("idFamille");
            Globale.lesMedicaments.Add(new Medicament(id, nom, composition, effet, contreIndication, Globale.lesFamilles[idFamille]));
        }
        curseur.Close();
    }

    // chargement des données spécifiques au visiteur connecté (réalisé à chaque connexion d'un visiteur)

    // vider les anciennes données : nécessaire si une autre visiteur était connecté avant
    Globale.mesPraticiens.Clear();
    Globale.mesVisites.Clear();
    Globale.mesVilles.Clear();
```

```
// chargement des villes gérées par le visiteur dans la collection mesVilles de la classe Globale
cmd.CommandText = "Select nom, codePostal from mesVilles";
curseur = cmd.ExecuteReader();
while (curseur.Read()) Globale.mesVilles.Add(new Ville(curseur.GetString("nom"), curseur.GetString("codePostal")));
curseur.Close();

// chargement des praticiens à partir de la vue afin d'alimenter la collection mesPraticiens de la classe Globale
// il faut récupérer l'objet TypePraticien et éventuellement l'objet Specialite
cmd.CommandText = "SELECT id, nom, prenom, rue, codePostal, ville, email, telephone, idType, idSpecialite FROM mespraticiens;";
curseur = cmd.ExecuteReader();
while (curseur.Read())
{
    int id = curseur.GetInt32("id");
    string nom = curseur.GetString("nom");
    string prenom = curseur.GetString("prenom");
    string rue = curseur.GetString("rue");
    string codePostal = curseur.GetString("codePostal");
    string ville = curseur.GetString("ville");
    string email = curseur.GetString("email");
    string telephone = curseur.GetString("telephone");
    string idType = curseur.GetString("idType");
    string idSpecialite = curseur.IsDBNull(9) ? null : (string)curseur["idSpecialite"];
    // Récupération des objets TypePraticien et Specialite
    TypePraticien t = Globale.lesTypes.Find(x => x.Id == idType);
    Specialite s = null;
    if (idSpecialite != null)
    {
        s = Globale.lesSpecialites.Find(x => x.Id == idSpecialite);
    }
    Globale.mesPraticiens.Add(new Praticien(id, nom, prenom, rue, codePostal, ville, email, telephone, t, s));
}
curseur.Close();
```

```
// chargement des visites du visiteur connecté à partir de la vue afin d'alimenter la collection mesVisites
cmd.CommandText = "SELECT id, dateEtHeure,  idMotif, idPraticien, bilan, premierMedicament, secondMedicament from mesvisites;";
curseur = cmd.ExecuteReader();
while (curseur.Read())
{
    int idVisite = curseur.GetInt32("id");
    int idPraticien = curseur.GetInt32("idPraticien");
    int idMotif = curseur.GetInt32("idMotif");

    DateTime dateEtHeure = curseur.GetDateTime("dateEtHeure");
    // récupération des objets liés
    Motif m = Globale.lesMotifs.Find(x => x.Id == idMotif);
    Praticien p = Globale.mesPraticiens.Find(x => x.Id == idPraticien);
    // création de l'objet visite
    Visite uneVisite = new Visite(idVisite, p, m, dateEtHeure);
    // si le bilan est enregistré
    if (!curseur.IsDBNull(4))
    {
        // initialisation du bilan et du premierMedicament
        string bilan = curseur.GetString("bilan");
        string idPremierMedicament = curseur.GetString("premierMedicament");
        Medicament premierMedicament = Globale.lesMedicaments.Find(x => x.Id == idPremierMedicament);
        Medicament secondMedicament = null;
        // initialisation éventuelle du second médicament s'il est renseigné
        if (!curseur.IsDBNull(6))
        {
            string idSecondMedicament = curseur.GetString("secondMedicament");
            secondMedicament = Globale.lesMedicaments.Find(x => x.Id == idSecondMedicament);
        }
        uneVisite.enregistrerBilan(bilan, premierMedicament, secondMedicament);
        // le chargement des échantillons s'effetue globalement à la fin
    }
    Globale.mesVisites.Add(uneVisite);
}
curseur.Close();
```

```
// chargement de la synthèse des échantillons distribués par le visiteur
cmd.CommandText = "SELECT idVisite, idMedicament, quantite from mesEchantillons";
curseur = cmd.ExecuteReader();
while (curseur.Read())
{
    // récupération des données du curseur
    int idVisite = curseur.GetInt32("idVisite");
    string idMedicament = curseur.GetString("idMedicament");
    int quantite = curseur.GetInt32("quantite");
    // récupération de l'objet visite correspondant
    Visite v = Globale.mesVisites.Find(x => x.Id == idVisite);
    // récupération de l'objet medicament
    Medicament m = Globale.lesMedicaments.Find(x => x.Id == idMedicament);
    // ajout d'un échantillon
    v.ajouterEchantillon(m, quantite);
}
curseur.Close();
```

```
/// <summary>
///     Ajout d'une nouvelle visite
/// </summary>
/// <param name="idPraticien"></param>
/// <param name="idMotif"></param>
/// <param name="uneDate"></param>
/// <param name="uneHeure"></param>
/// <param name="message"></param>
/// <returns>identifiant de la nouvelle visite ou 0 si erreur lors de la création</returns>
1 référence
static public int ajouterRendezVous(int idPraticien, int idMotif, DateTime uneDate, out string message)
{
    int idVisite = 0;
    message = string.Empty;
    MySqlCommand cmd = new MySqlCommand()
    {
        Connection = cnx,
        CommandText = "ajouterRendezVous",
        CommandType = CommandType.StoredProcedure,
    };
    cmd.Parameters.AddWithValue("_idPraticien", idPraticien);
    cmd.Parameters.AddWithValue("_idMotif", idMotif);
    cmd.Parameters.AddWithValue("_dateEtHeure", uneDate);
    // paramètre en sortie
    cmd.Parameters.Add("_idVisite", MySqlDbType.Int32);
    cmd.Parameters["_idVisite"].Direction = ParameterDirection.Output;
    try
    {
        cmd.ExecuteNonQuery();
        idVisite = (Int32)cmd.Parameters["_idVisite"].Value;
    }
    catch (MySqlException e)
    {
        message += e.Message;
    }
    return idVisite;
}
1 référence
```

```
1 référence
static public bool supprimerRendezVous(int idVisite, out string message)
{
    message = string.Empty;
    MySqlCommand cmd = new MySqlCommand()
    {
        Connection = cnx,
        CommandText = "supprimerRendezVous",
        CommandType = CommandType.StoredProcedure,
    };
    cmd.Parameters.AddWithValue("_idVisite", idVisite);
    try
    {
        cmd.ExecuteNonQuery();
        return true;
    }
    catch (MySqlException e)
    {
        message += e.Message;
        return false;
    }
}
```

```
1 référence
static public bool modifierRendezVous(int idVisite, DateTime uneDate, out string message)
{
    bool success = false;
    message = string.Empty;
    MySqlCommand cmd = new MySqlCommand()
    {
        Connection = cnx,
        CommandText = "modifierRendezVous",
        CommandType = CommandType.StoredProcedure,
    };

    // Ajout du paramètre idVisite pour identifier le rendez-vous à modifier
    cmd.Parameters.AddWithValue("_idVisite", idVisite);
    cmd.Parameters.AddWithValue("_dateEtHeure", uneDate);

    try
    {
        cmd.ExecuteNonQuery();
        success = true;
    }
    catch (MySqlException e)
    {
        message += e.Message;
    }
    return success;
}
```

```
1 référence
static public bool enregistrerBilan(Visite uneVisite, out string message)
{
    message = string.Empty;
    // Déclaration d'un transaction
    MySqlTransaction uneTransaction = cnx.BeginTransaction();


    MySqlCommand cmd = new MySqlCommand();
    cmd.Connection = cnx;
    // attacher la commande à la transcation
    cmd.Transaction = uneTransaction;

    // enregistrement au niveau de la table visite
    cmd.CommandType = CommandType.StoredProcedure;
    cmd.CommandText = "enregistrerBilanVisite";
    cmd.Parameters.AddWithValue("_idVisite", uneVisite.Id);
    cmd.Parameters.AddWithValue("_bilan", uneVisite.Bilan);
    cmd.Parameters.AddWithValue("_premierMedicament", uneVisite.PremierMedicament.Id);
    cmd.Parameters.AddWithValue("_secondMedicament", uneVisite.SecondMedicament is null ? null : uneVisite.SecondMedicament.Id);

    try
    {
        cmd.ExecuteNonQuery();
    }
    catch (MySqlException e)
    {
        message += e.Message;
        // annuler la transaction
        uneTransaction.Rollback();
        return false;
    }

    // suppression préalable des anciens échantillons
    cmd.CommandType = CommandType.StoredProcedure;
    cmd.CommandText = "supprimerEchantillon";
    cmd.Parameters.Clear();
    cmd.Parameters.AddWithValue("_idVisite", uneVisite.Id);
    try
```

```
// enregistrements au niveau des echantillons
cmd.CommandType = CommandType.StoredProcedure;
cmd.CommandText = "ajouterEchantillon";
cmd.Parameters.Clear();
cmd.Parameters.AddWithValue("_idVisite", uneVisite.Id);
cmd.Parameters.Add("_idMedicament", MySqlDbType.VarChar, 10);
cmd.Parameters.Add("_quantite", MySqlDbType.Int32);
foreach (KeyValuePair<Medicament, int> unCouple in uneVisite)
{
    Medicament unMedicament = unCouple.Key;
    int quantite = unCouple.Value;
    cmd.Parameters["_idMedicament"].Value = unMedicament.Id;
    cmd.Parameters["_quantite"].Value = quantite;

    try
    {
        cmd.ExecuteNonQuery();
    }
    catch (MySqlException e)
    {
        message += e.Message;
        uneTransaction.Rollback();
        return false;
    }
}
// validation de la transaction
uneTransaction.Commit();
return true;
```

3. La classe Famille

```
// ------------------------------------------
// Nom du fichier : famille.cs
// Objet : classe famille
// Auteur : M. Verghote
// ------------------------------------------

using System;
using System.Collections.Generic;

namespace lesClasses {
    [Serializable]
    6 références
    public class Famille {
        // attribut
        2 références
        private List<Medicament> lesMedicaments { get; }

        // Propriétés automatiques
        1 référence
        public string Id { get; set; }
        1 référence
        public string Libelle { get; set; }


        // Constructeur
        1 référence
        public Famille(string id, string libelle)
            => (Id, Libelle, lesMedicaments) = (id, libelle, new List<Medicament>());

        // méthode
        1 référence
        public void ajouterMedicament(Medicament unMedicament) {
            lesMedicaments.Add(unMedicament);
        }

    }
}
```

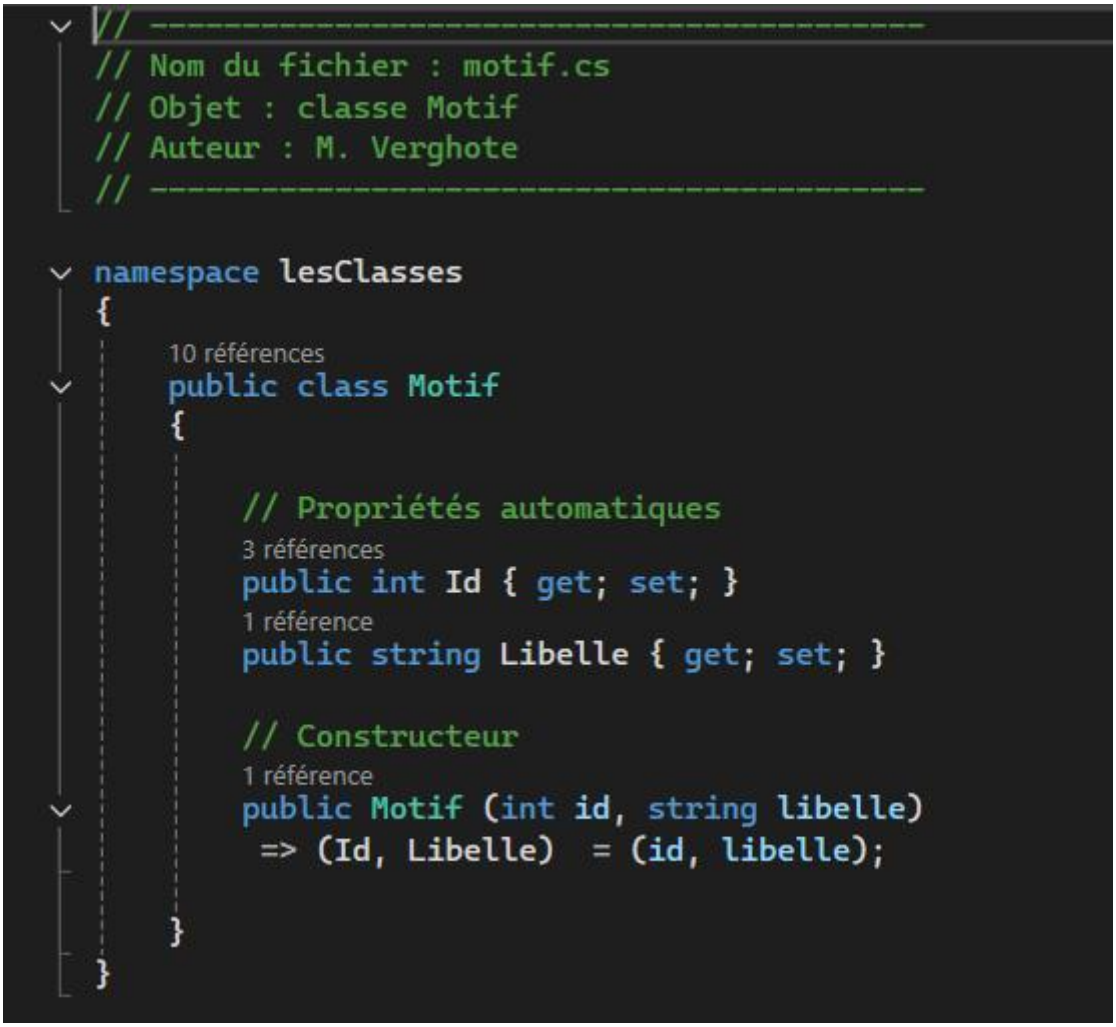
4. La classe Medicament

```
namespace lesClasses
{
    [Serializable]
    34 références
    public class Medicament : IComparable<Medicament>
    {
        // la classe doit implémenter une méthode de comparaison car les objets Medicamement
        // seront utilisés comme clé d'accès dans un dictionnaire  (panier des médicaments distribués)
        0 références
        public int CompareTo(Medicament o) => Nom.CompareTo(o.Nom);

        // Propriétés automatiques
        8 références
        public string Id { get; set; }
        5 références
        public string Nom { get; set; }
        1 référence
        public string Composition { get; set; }
        1 référence
        public string Effets { get; set; }
        1 référence
        public string ContreIndication { get; set; }
        2 références
        public Famille LaFamille { get; set; }

        // Constructeur
        1 référence
        public Medicament (string id, string nom, string composition, string effets, string contreIndication, Famille famille)
        {
            Id = id;
            Nom = nom;
            Composition = composition;
            Effets = effets;
            ContreIndication = contreIndication;
            LaFamille = famille;
            // mise à jour de l'association bidirectionnelle : une famille contient la collection des médicaments associés
            LaFamille.ajouterMedicament(this);

        }
        0 références
        public override string ToString() => Id;
```

5. La classe Ville

```
// ----------------------------------------
// Nom du fichier : ville.cs
// Objet : classe ville
// Auteur : M. Verghote
// ----------------------------------------

using System;

namespace lesClasses
{
    [Serializable]
    4 références
    public class Ville
    {

        // Constructeur
        1 référence
        public Ville (string nom, string code)
        => (Code, Nom) = (code, nom);

        // Propriétés automatiques
        1 référence
        public string Nom { get; set; }
        1 référence
        public string Code { get; set; }

    }
}
```

6. La classe Motif



```
// ------------------------------------------
// Nom du fichier : motif.cs
// Objet : classe Motif
// Auteur : M. Verghote
// ------------------------------------------

namespace lesClasses
{
    10 références
    public class Motif
    {

        // Propriétés automatiques
        3 références
        public int Id { get; set; }
        1 référence
        public string Libelle { get; set; }

        // Constructeur
        1 référence
        public Motif (int id, string libelle)
         => (Id, Libelle)  = (id, libelle);

    }
}
```