TP accès à une base de données avec Qt5

1. OBJECTIF

Utilisation des classes d'accès aux bases de données avec la bibliothèque Qt5 affichage du contenu d'une table sous forme de tableau.

Les points techniques abordés seront :

- Accès aux bases de données avec **QsqlDatabase**
- Les requêtes en langage SQL avec **QsqlQuery**
- Le design pattern Model/Vue
- Les requêtes SQL avec le modèle **QsqlTableModel**
- La vue **QtableView** pour afficher des données sous forme de tableau.

2. CONDITIONS DE RÉALISATION

Travail individuel sur PC avec QT Creator et phpMyAdmin

Un serveur de base de données est accessible avec phpMyAdmin:

hostname **172.18.58.5**
username **snir**
password **snir**
basename **snirBanque1** ou **snirBanque2**

3. RESSOURCES

Les classes Qt utilisées dans ce TP sont :

QSqlDatabase, http://doc.qt.io/qt-5/qsqldatabase.html

QSqlQuery http://doc.qt.io/qt-5/qsqlquery.html

QsqlTableModel http://doc.qt.io/qt-5/qsqltablemodel.html

QtableView http://doc.qt.io/qt-5/qtableview.html

4. LE BESOIN

Les distributeurs automatiques de billets (DAB) permettent de se connecter à la banque. Ils offrent la possibilité d'interroger un compte pour en connaître le solde, de déposer de l'argent et d'en retirer. Ce sont ces fonctionnalités que nous allons étudier dans un premier temps côté client (sur le DAB) puis côté serveur (à la banque).

4.1. La base de données

Les informations concernant la base de données sont les suivantes:

Nom d'hôte	**172.18.58.5**
Nom de la base	**snirBanque1 ou snirBanque2**
utilisateur	**snir**
Mot de passe	**snir**

La base de données "**snirBanque1**" ou "**snirBanque2**" contient un ensemble de tables et de vues avec des données initiales.

A l'aide de l'outil phpMyAdmin accessible à l'adresse http://172.18.58.5/phpMyAdmin/ (attention au respect des majuscules)

se connecter en tant qu'utilisateur **snir** avec le mot de passe **snir** sur la base de données **snirBanque1 ou snirBanque2**.

1. Combien y-a t'il de tables dans cette base de données ?

2. Combien y-a t'il de vues dans cette base de données ?

3. Quelles sont les relations entre les tables ? Représenter le schéma des tables ?

4. Quelle est la requête exprimée en langage SQL permettant d'obtenir le solde de chaque compte. Le solde d'un compte correspond à la somme des opérations sur ce compte car les opérations correspondant à un dépôts sont positives et les opérations correspondant à un retrait sont négatives. (indication vous devez utiliser **SUM**() et **GROUP BY** dans votre requête)

5. Quelle est la requête (exprimée en langage SQL) permettant d'obtenir la vue vue_compte_courant

Nota vous pouvez vérifier votre résultat en éditant la requête associée à la vue. Pour ce faire cliquer sur la vue **vue_compte_courant** puis sur l'onglet **structure** et enfin sur le crayon **éditer la vue.**

6. Quels sont les utilisateurs autorisés à se connecter à la base de données snirBanque1 ou snirBanque2 ?

7. Quels sont les droits accordés à l'utilisateur snir ?

4.2. Test de la connexion avec Qt

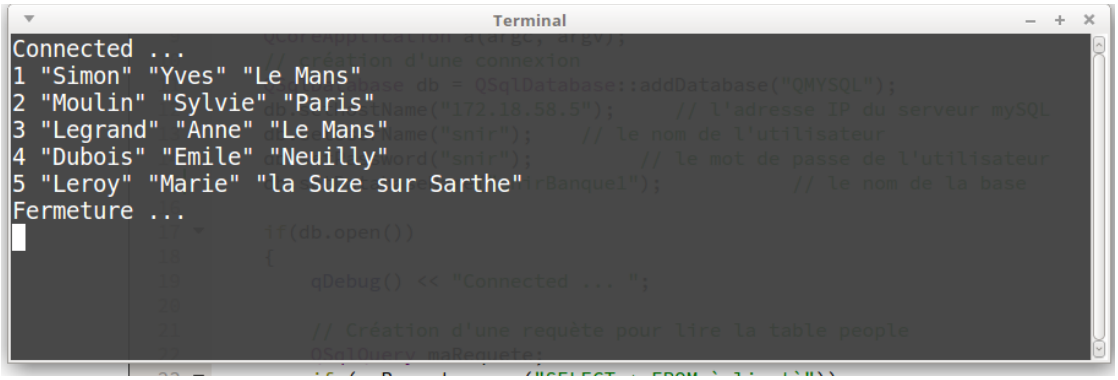
Dans un premier temps on se propose d'écrire un programme simple pour vérifier la connexion avec la base snirBanque1 en tant qu'utilisateur snir.

Le programme est écrit directement dans la fonction main().

Dans votre fichier .pro vous n'oublierez pas d'ajouter la ligne suivante :

```
QT += sql
```

Le visuel attendu est le suivant :



Votre programme affiche dans la console le contenu de la table client, puis insère un nouveau client à votre nom et votre prénom.

1 L'application se connecte

```
QSqlDatabase db = QSqlDatabase::addDatabase("QMYSQL");

db.setHostName("172.18.58.5");     // l'adresse IP du serveur mySQL
db.setUserName("snir");            // le nom de l'utilisateur
db.setPassword("snir");                    // le mot de passe de l'utilisateur
db.setDatabaseName("snirBanque1");         // le nom de la base

if( db.open() ){ . . . }
```

2 Si la connexion est réussie alors l'application exécute la requête

```
QSqlQuery maRequete(db);
if (maRequete.exec("SELECT * FROM `client`"))
{ . . . }
```

Tant qu'il y a des lignes dans le résultat

```
while(maRequete.next())

{
QVariant id = maRequete.value(0);  // 0 première colonne
QVariant nom = maRequete.value("nom");
QVariant prenom = maRequete.value("prenom");
QVariant ville = maRequete.value("ville");
        . . . .     }
```

4 Exemple pour insèrer un nouveau client dans la table

```
QString insRequete =
"INSERT INTO `snirBanque`.`client` ( `nom`, `prenom`, `ville` ) VALUES ( :prenom , :nom, :ville )";


    maRequete.prepare(insRequete);
    maRequete.bindValue(":prenom", "Michel");
    maRequete.bindValue(":nom", "Lafont");
    maRequete.bindValue(":ville", "Versaille");

    if (maRequete.exec()){
      . . . .
    }
```

5 fermeture de la connexion

```
    db.close();
```

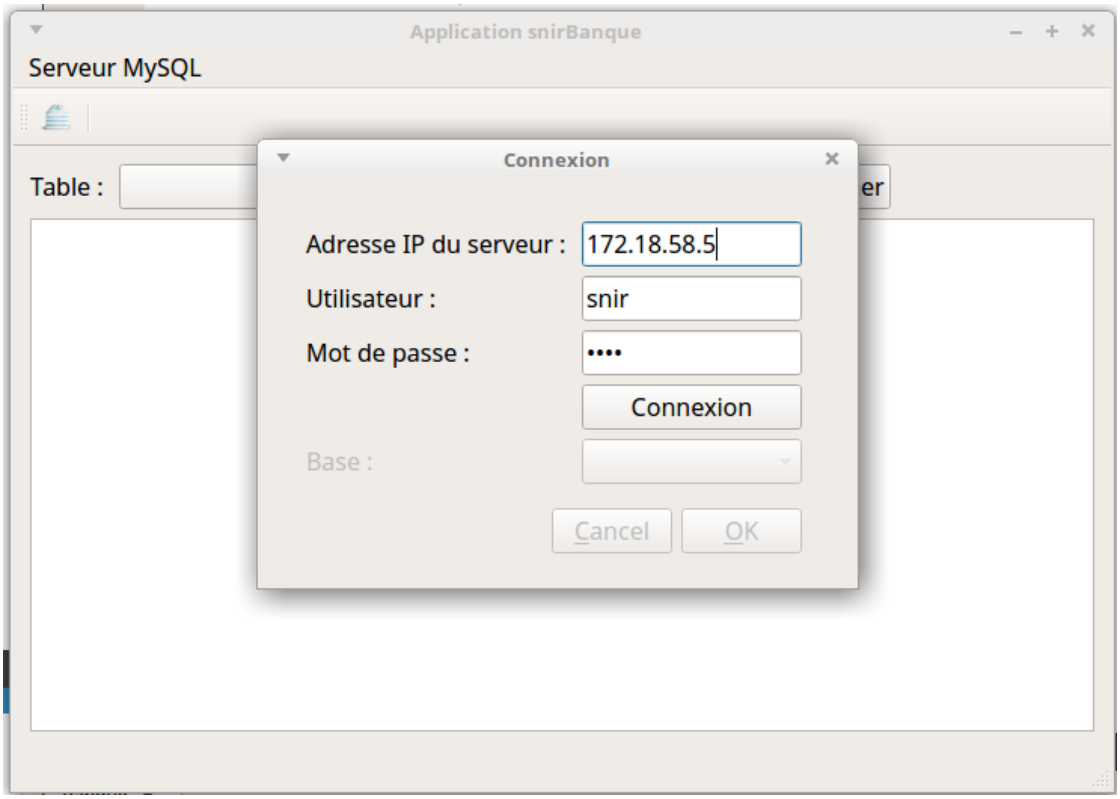
4.3. Création d'une IHM de consultation

L'interface aura l'aspect suivant :

Au lancement, de l'application une fenêtre principale s'affiche :

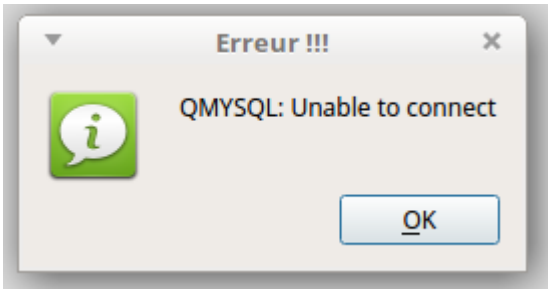
Dans le menu Serveur MySQL l'utilisateur peut cliquer sur connexion

Une boîte de dialogue s'affiche alors pour établir une connexion avec un serveur de données.



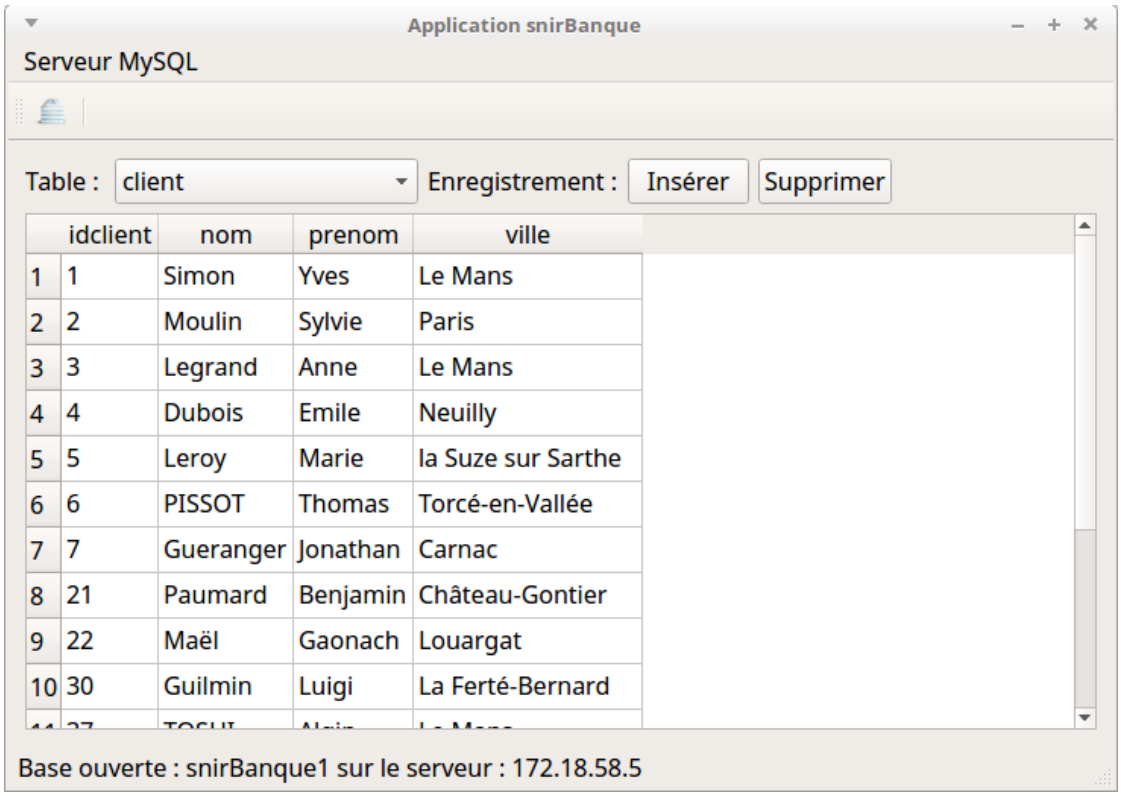
Un clique sur le bouton **Connexion** permet d'ouvrir la session, les bases de données disponibles s'affichent alors dans un combobox .

Si la connexion échoue, un message d'erreur apparaît sous forme de **messageBox** (utilisez la méthode **lastError** pour avoir un message explicite).



L'opérateur peut choisir une base de données parmi celles disponibles et cliquer sur le bouton OK pour valider son choix. La boîte de dialogue se ferme.

Dans la fenêtre principale, la barre d'états affiche le nom de la base et l'adresse du serveur.

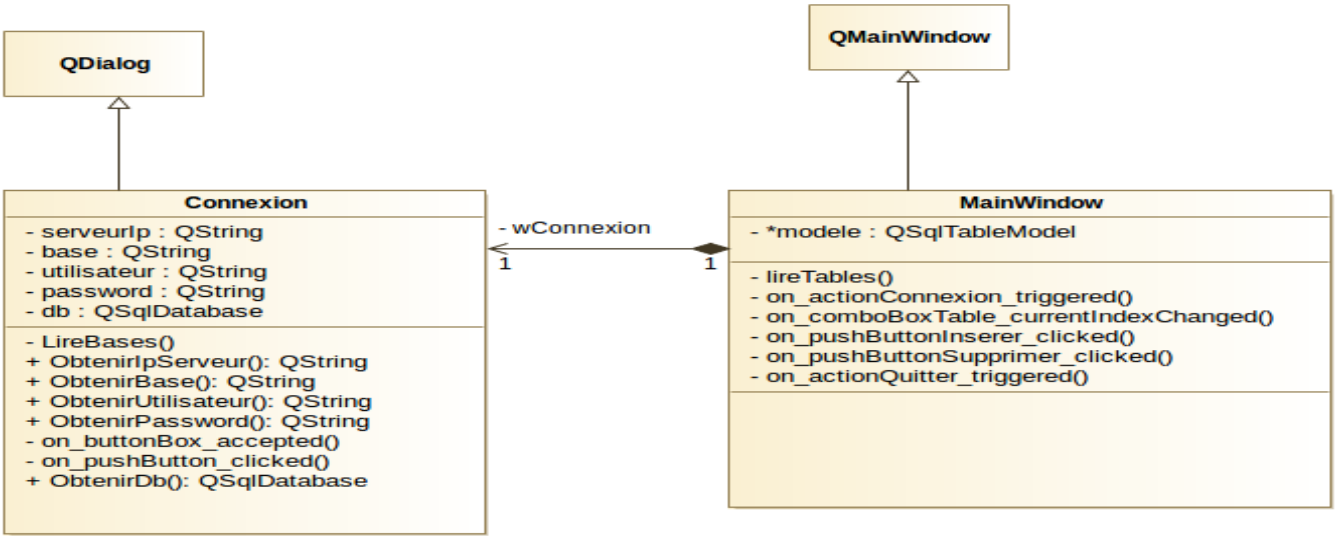


Le combobox table permet de sélectionner une table parmi celles disponibles pour la base choisie.

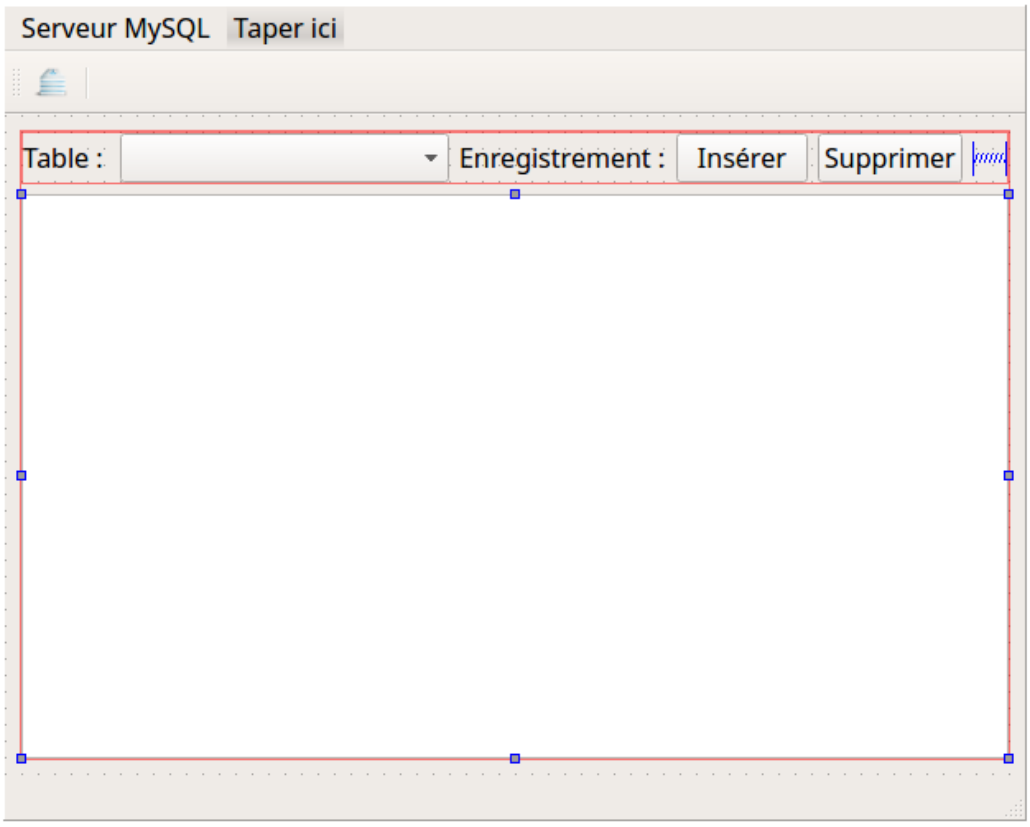
Il est alors possible de modifier un enregistrement en l'éditant dans le tableau.

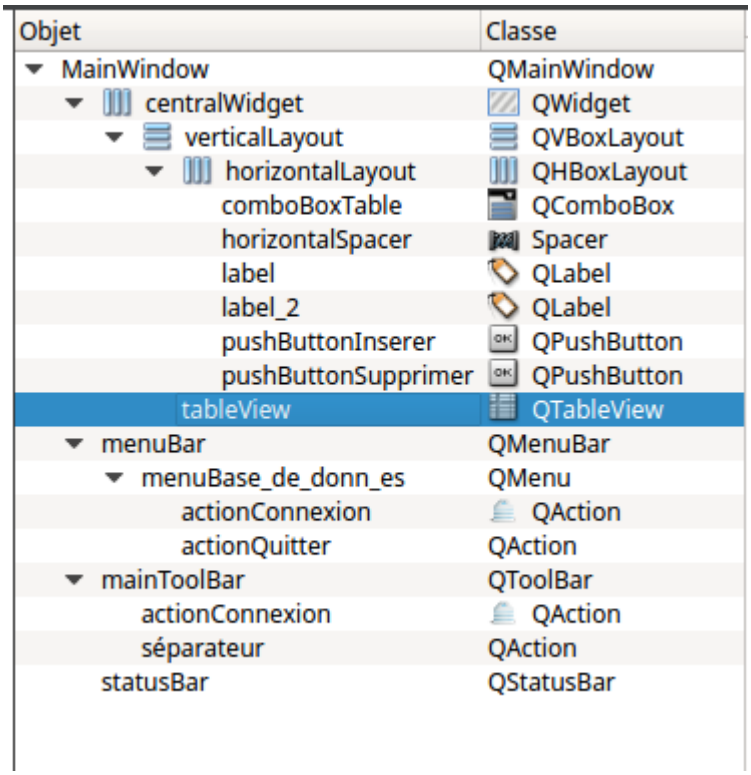
La Boîte de dialogue **Connexion** hérite de **Qdialog** et la **MainWindow** de **QMainWindow** comme le montre le diagramme de classes ci dessus.

4.4. L'IHM de la boîte de dialogue Connexion



4.5. L'IHM de la fenêtre principale





Objet	Classe
MainWindow	QMainWindow
centralWidget	QWidget
verticalLayout	QVBoxLayout
horizontalLayout	QHBoxLayout
comboBoxTable	QComboBox
horizontalSpacer	Spacer
label	QLabel
label_2	QLabel
pushButtonInserer	QPushButton
pushButtonSupprimer	QPushButton
tableView	QTableView
menuBar	QMenuBar
menuBase_de_donn_es	QMenu
actionConnexion	QAction
actionQuitter	QAction
mainToolBar	QToolBar
actionConnexion	QAction
séparateur	QAction
statusBar	QStatusBar

Pour ouvrir la fenêtre de dialoque Wconnexion :

```
// ouvre la fenêtre connexion
Wconnexion->exec();
```

Pour réaliser une requête avec QSqlTableModel :

```
this->modele = new QSqlTableModel(this);

modele->setTable("client");
modele->setFilter("`ville` = 'le mans'");
modele->select();
```

ce qui est équivalent à la requête

```
SELECT * FROM `client` WHERE `ville` = 'le mans'
```

Pour associer le modèle à une vue :

```
ui->tableView->setModel(modele);
```

Pour ajuster la largeur des colonnes au contenu :

```
ui->tableView->resizeColumnsToContents();
```

Fonctionnalités attendues:

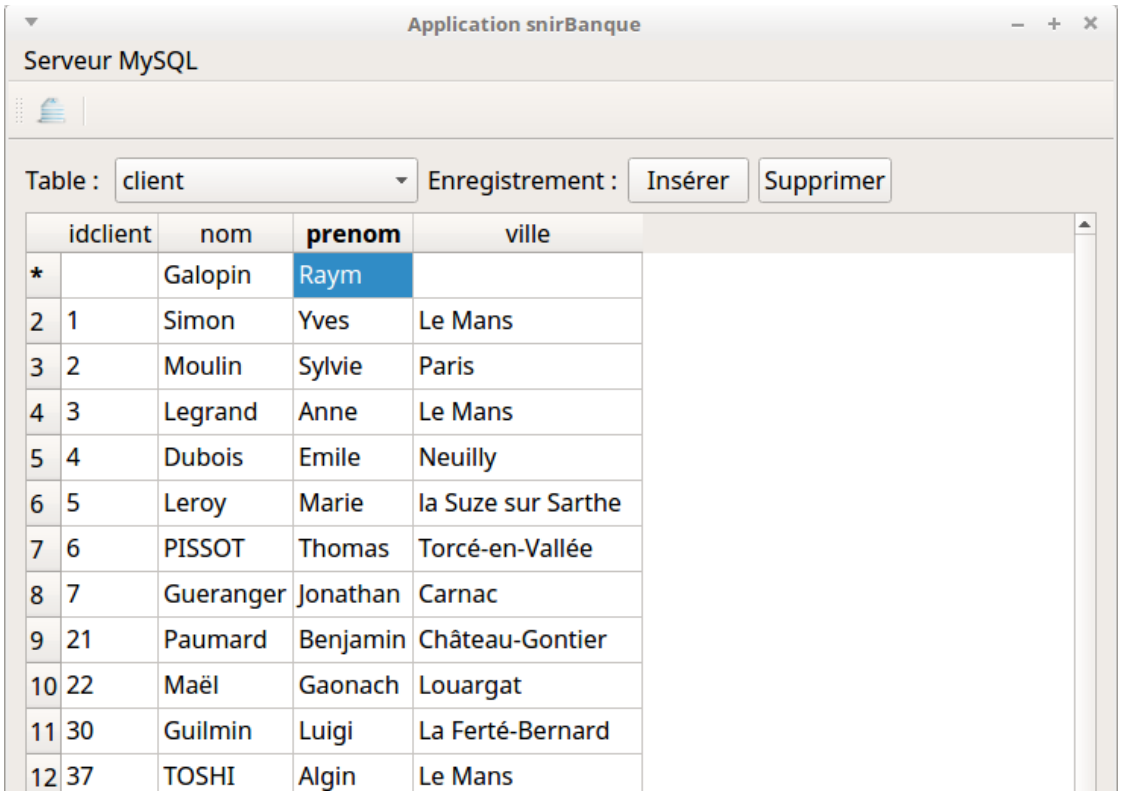
1. Un comboBox permet de sélectionner une table à afficher parmi celles disponibles. La méthode tables() sur l'objet QsqlDatabase permet d'obtenir la

liste des tables présentes dans la base de données sélectionnées.

```
QStringList tables = Wconnexion->db.tables(QSql::AllTables);
// AllTables les tables plus les vues
```

Compléter le comboBox avec la liste des tables obtenues.

2. Un clique sur le bouton **ajouter** permet de créer un enregistrement en ajoutant une ligne dans le modèle associé à la vue. nous appelons insertRows() pour créer une nouvelle ligne (enregistrement) vide.



	idclient	nom	prenom	ville
*		Galopin	Raym	
2	1	Simon	Yves	Le Mans
3	2	Moulin	Sylvie	Paris
4	3	Legrand	Anne	Le Mans
5	4	Dubois	Emile	Neuilly
6	5	Leroy	Marie	la Suze sur Sarthe
7	6	PISSOT	Thomas	Torcé-en-Vallée
8	7	Gueranger	Jonathan	Carnac
9	21	Paumard	Benjamin	Château-Gontier
10	22	Maël	Gaonach	Louargat
11	30	Guilmin	Luigi	La Ferté-Bernard
12	37	TOSHI	Algin	Le Mans

Il est alors possible d'éditer les champs de la table, un appui sur la touche **Enter** du clavier permet de valider la saisie.

La stratégie de modification du modèle sera défini sur OnRowChange

```
modele->setEditStrategy(QSqlTableModel::OnRowChange);
```

3. Un clique sur le bouton **supprimer** permet de supprimer l'enregistrement sélectionné sur la vue.
   Nous appelons removeRow() pour retirer une ligne du modèle.
   removeRow() reçoit le numéro de la ligne à supprimer c'est à dire celle sélectionnée.
   Une fenêtre s'affichera pour demander la confirmation de la suppression.

   Pour connaître le numéro de la ligne sélectionnée sur la vue :

```
ligneSelectionee = ui->tableView->selectionModel()->currentIndex().row();
```