Lycée TOUCHARD WASHINGTON

Section des Techniciens Supérieurs
Cybersécurité Informatique Electronique
Option Informatique et Réseaux

TD1 – Agence Bancaire

Accès à la base de données sous Qt

- *Date : decembre 2024*
- *Version : 3*
- *Référence : TD1 – Agence Bancaire.odt*

1. Objectif

- *Accès aux bases de données avec la bibliothèque **Qt***
- *SQL.*
- *Codage de l'information*

2. Conditions de réalisation

- *Ce fichier contient des liens hypertextes.*
- *Ressources utilisées :*
  - *Un PC sous Linux*
  - *Une base de données est disponible*
  - *Qt-creator*

3. Ressources

Les classes QSqlDatabase et QSqlQuery dans la technologie QT, consulter les sites : https://doc.qt.io/qt-6/qtsql-index.html https://doc.qt.io/qt-6/sql-programming.html et plus particulièrement

https://doc.qt.io/qt-6/qsqldatabase.html et https://doc.qt.io/qt-6/qsqlquery.html

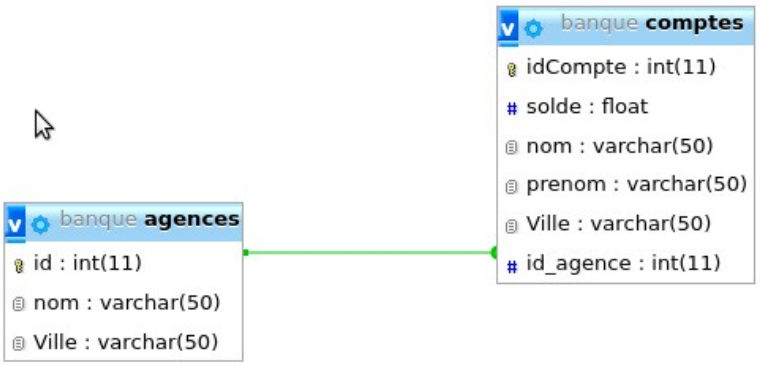
4. Accès à la base de données

. La base de données

Les informations concernant la base de données sont les suivantes:

Adresse	172.18.58.7
Nom de la base de données	TD2_votreLogin
Login	ciel1_votreLogin
Mot de passe	VotreMotDePasse

Votre base de données doit utiliser deux tables nommées "**comptes**" et "**agences**" dont voici les caractéristiques :



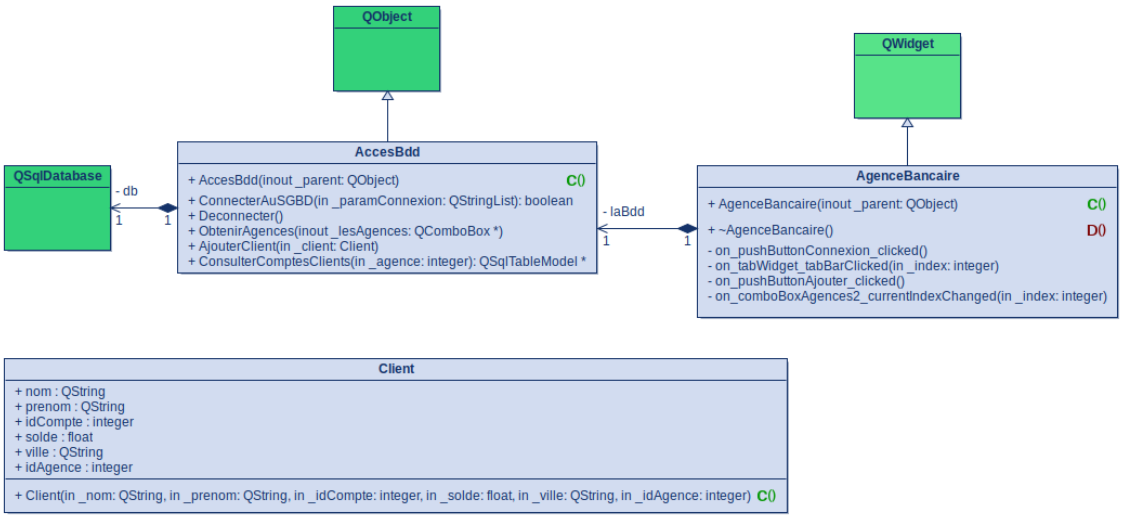
Voici un script SQL permettant de créer les deux tables et de peupler la table **agences**.

```
CREATE TABLE `agences` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `nom` varchar(50) NOT NULL,
  `Ville` varchar(50) NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=4 DEFAULT CHARSET=latin1;

INSERT INTO `agences` (nom, ville) VALUES ('LaBanqueCiel
Bollée','Le Mans'),('LaBanqueCiel République','La Ferté Bernard'),
('LaBanqueCiel Arnage','Arnage');

CREATE TABLE `comptes` (
  `idCompte` int(11) NOT NULL,
  `solde` float NOT NULL,
  `nom` varchar(50) NOT NULL,
  `prenom` varchar(50) NOT NULL,
  `Ville` varchar(50) NOT NULL,
  `id_agence` int(11) NOT NULL,
  PRIMARY KEY (`idCompte`),
  UNIQUE KEY `idCompte` (`idCompte`),
  KEY `id_agence` (`id_agence`),
   CONSTRAINT `comptes_ibfk_1` FOREIGN KEY (`id_agence`) REFERENCES
`agences` (`id`) ON DELETE NO ACTION ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

5. Création du Projet

Créez une application ***Qt*** de type Application avec l'utilisation de **Widget**. Cette application se nomme ***AgenceBancaire***. Modifiez le fichier ***.pro*** pour pouvoir utiliser la gestion de données avec SQL.

Le diagramme de classe de l'application est le suivant :



Les relations de composition seront implémentées de manière dynamique dans les constructeurs respectifs et les ressources libérées dans les destructeurs.

Classe `Client`

- La classe **`Client`** est une structure simplifiée où tous les attributs sont publics.
- Elle inclut un constructeur permettant d'initialiser tous les attributs en une seule opération.

Classe `AccesBdd`

- La classe **`AccesBdd`** est responsable de la gestion des interactions avec la base de données.
- Elle n'interagit jamais directement avec l'utilisateur. Chaque interaction avec la base de données est soumise à une vérification préalable. En cas d'erreur, les messages correspondants sont affichés dans la console de débogage.

Méthodes de la classe `AccesBdd`

```
ConnecterAuSGBD(const QStringList& _paramConnexion)
```

- Cette méthode établit une connexion à la base de données.
- Les paramètres nécessaires à la connexion sont fournis dans une **`QStringList`** passée par référence. Les éléments de la liste doivent être dans l'ordre suivant :
    1. Adresse du serveur
    2. Nom de la base
    3. Nom de l'utilisateur
    4. Mot de passe

```
Deconnecter()
```

- Cette méthode ferme la connexion à la base de données et libère toutes les ressources associées.
- Elle utilise un délai avant de retirer la base de données pour éviter d'éventuelles erreurs liées à la gestion des connexions multiples.

```
ObtenirAgences(QComboBox* _comboAgences)
```

- Cette méthode remplit une liste déroulante (type **`QComboBox`**) avec les noms et identifiants des agences stockées dans la table **`agences`**.

- Le pointeur vers la **QComboBox** cible est passé en paramètre.

```
AjouterClient(const Client& _client)
```

- Cette méthode ajoute un client dans la table **comptes** de la base de données.Les données du client sont fournies via un objet de la classe **Client** passé en paramètre.

```
ConsulterComptesClients(const int _idAgence)
```

- Cette méthode retourne un pointeur vers un objet **QSqlTableModel** contenant les données des comptes clients.
- Les données sont filtrées pour ne concerner que les comptes liés à l'identifiant de l'agence spécifié (**_idAgence**).
- Les résultats sont triés par ordre croissant des noms des clients.

Classe AgenceBancaire

- La classe **AgenceBancaire** gère les interactions avec l'utilisateur via l'interface graphique. Elle contient plusieurs slots connectés aux différents widgets de l'interface pour traiter les événements utilisateur.
- En cas de réussite d'une opération, une boîte de dialogue informative (**QMessageBox::information**) notifie l'utilisateur. En cas d'échec, une boîte de dialogue critique (**QMessageBox::critical**) informe l'utilisateur de l'erreur.

Slot de la classe AgenceBancaire

```
on_pushButtonAjouter_clicked()
```

- Vérifie que tous les champs requis du formulaire sont complétés et qu'une agence est sélectionnée dans la liste déroulante.
- Si ces conditions sont remplies, elle appelle la méthode **AjouterClient** pour ajouter un nouveau client. En cas de champs manquants ou d'absence de sélection d'agence, affiche un message d'erreur approprié.

```
on_comboBoxAgences2_currentIndexChanged(int _index)
```

- **Si l'index est nul (aucune agence sélectionnée)** :

  La table associée (**QTableView**) est réinitialisée avec un modèle nul (**nullptr**). Le champ indiquant le nombre de clients est vidé.

- **Si un autre index est sélectionné (agence valide)** :

  Configure l'affichage de la **QTableView,** les colonnes sont redimensionnées pour occuper tout l'espace disponible (**QheaderView::Stretch**). La colonne contenant les identifiants d'agence est masquée pour plus de lisibilité.

  Elle Appelle la méthode **ConsulterComptesClients** de la classe **AccesBdd**, en passant l'identifiant de l'agence sélectionnée comme paramètre. Le modèle de données retourné est assigné à la table pour afficher les comptes clients de l'agence. Met à jour le champ indiquant le nombre de clients en utilisant la méthode **rowCount()** du modèle retourné.

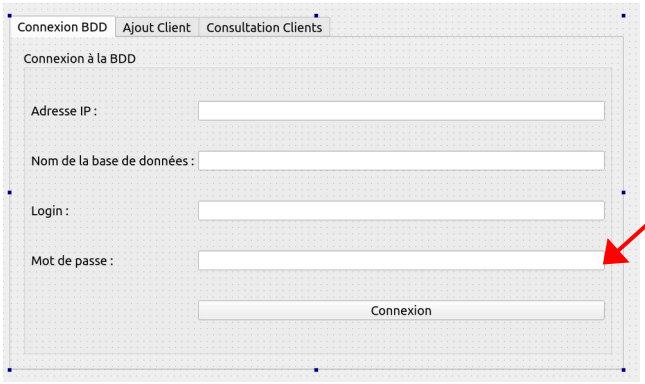
```
on_tabWidget_tabBarClicked(int _index)
```

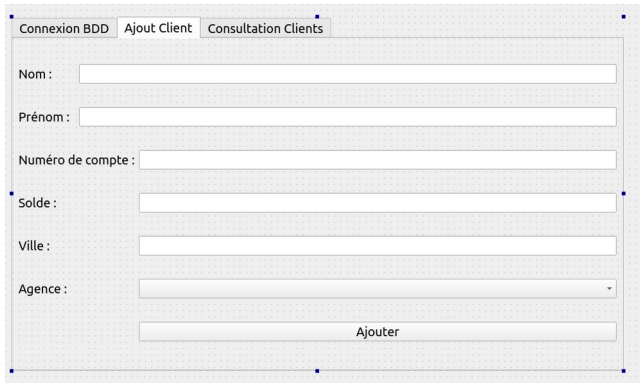
Lorsqu'un utilisateur clique sur les onglets **"Ajout Client"** ou **"Consultations Client"**, la méthode identifie la liste déroulante (de type QComboBox) à compléter en fonction de l'onglet sélectionné.

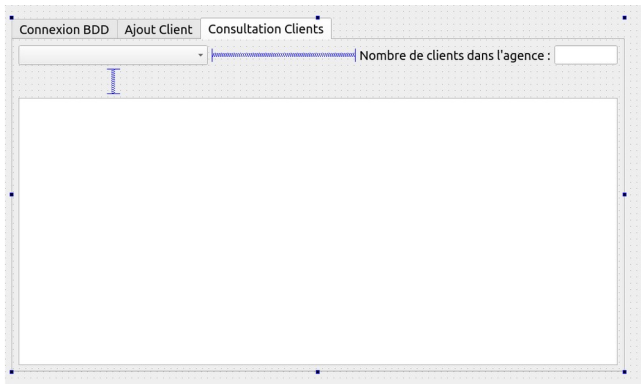
Elle passe le pointeur sur le **QComboBox** correspondant en paramètre à la méthode **AccesBdd::ObtenirAgence()**, qui remplit la liste avec les noms et identifiants des agences disponibles dans la base de données.

Description de l'IHM de l'agence bancaire

L'interface utilisateur se présente sous la forme d'un **QTableWidget** contenant 3 onglets :







Cette zone d'édition pourra prendre la valeur **Password** dans la propriété **echoMode**

Objet	Classe
AgenceBancaire	QWidget
tabWidget	QTabWidget
tabAjoutClient	QWidget
comboBoxAgences1	QComboBox
label_10	QLabel
label_5	QLabel
label_6	QLabel
label_7	QLabel
label_8	QLabel
label_9	QLabel
lineEditCompte	QLineEdit
lineEditNom	QLineEdit
lineEditPrenom	QLineEdit
lineEditSolde	QLineEdit
lineEditVille	QLineEdit
pushButtonAjouter	QPushButton
tabConnexion	QWidget
groupBoxBdd	QGroupBox
LineEditAdresseIP	QLineEdit
label	QLabel
label_2	QLabel
label_3	QLabel
label_4	QLabel
lineEditBdd	QLineEdit
lineEditLogin	QLineEdit
lineEditMdp	QLineEdit
pushButtonConnexion	QPushButton
tabConsultation	QWidget
comboBoxAgences2	QComboBox
horizontalSpacer	Spacer
label_11	QLabel
lineEditNbClients	QLineEdit
tableViewComptes	QTableView
verticalSpacer	Spacer

```
[Bdd]
DataBase=NomDeLabase
HostName=AdresseIPServeur
Login=Utilisateur
Mdp=MotDePasse
```

Fonctionnalités attendues :

Au lancement, de l'interface, seul l'onglet permettant la connexion est actif, les autres dépendent de la connexion à la base de données.

Le bouton Connexion devient Déconnexion et inversement.

Si un fichier **.ini** existe, il permet de compléter les champs de saisie de l'onglet de connexion. Il est créé la première fois et écrasé à chaque connexion réussie à la base de données.

Pour aller plus loin :

Les traitements d'erreur sont gérés maintenant par des exceptions avec la classe **`DataBaseException`** présente dans le cours.