R2.02 : développement d'application avec IHM
Feuille TD-TP n° 7

Programmation d'applications avec IHM exploitant une base de données MySql (Mise en œuvre avec Qt et c++ des mécanismes généraux de programmation)

Objectifs *:*
Découverte des principes de mise en œuvre d'une application événementielle en C++ exploitant une base de données MySql. Codage en c++ de requêtes SQL de sélection simples, de requêtes de création de tables et d'insertions d'enregistrements.

Activité 1 : Modèle objet et première approche d'une application graphique Qt accédant à une base de données

Créer un nouveau projet d'application Qt graphique nommé 1-testSelectBDD que vous enregistrerez dans votre espace de travail (cf. dossier R202DevApplisAvecIHM ou équivalent). Nommez votre classe testSelectBDD et une fois le projet créé, remarquez dans l'entête de la classe graphique **TestSelectBDD** la déclaration de propriété privée Ui::TestSelectBDD *ui; Que signifie cette déclaration ?

On se propose de procéder de la même façon pour référencer dans notre application Qt la base de données que nous allons manipuler. Pour ce faire, vous devez modifier l'entête de la classe pour obtenir :

```
private:
    Ui::TestSelectBDD *ui;
    Database *db;
```

On se propose maintenant de créer la classe que nous venons de référencer. Créer une nouvelle classe nommée Database :

Analysez le fichier database.h :

```
1  #ifndef DATABASE_H
2  #define DATABASE_H
3
4  #include <QSqlDatabase>
5
6  #define DATABASE_NAME "BD_Nodenot_Lakartxela"    // ici indiquer le nom du DSN permettant d'accéder à la BD
7  #define CONNECT_TYPE  "QODBC"                    // permet d'indiquer que l'on veut utiliser le driver odbc
8
9  class Database
10 {
11
12 public:
13     Database();
14     bool openDataBase();
15     void closeDataBase();
16
17
18
19 private:
20     QSqlDatabase mydb;
21 };
22
23 #endif // DATABASE_H
```

Pour adapter ce code, vérifiez que la base de données correspondante est bien toujours accessible sur lakartxela. Vérifiez que le DSN est bien déclaré dans les sources de données ODBC64 de votre PC.

Compte tenu des spécifications de la ma classe QSqlDatabase, proposez le code des méthodes openDatabase() et closeDatabase() à mettre dans votre fichier database.cpp

Notez que la classe QSqlDatabase apparaît comme étant non reconnue par Qt ; pour qu'elle le soit, il vous faut modifier le projet et ajouter sql en ligne 1 de votre projet Qt :

```
QT      += core gui sql
```

Il nous reste à écrire le corps de notre classe **TestSelectBDD**. Pour ce faire, il faut tout d'abord initialiser votre propriété privée db déclarée dans l'entête de cette classe. Cette initialisation consiste à instancier la classe Database puis à ouvrir la base de donnée associée à cette classe.

En utilisant la classe QSqlQuery, créez ensuite une requête Sql permettant de récupérer tous les références d'articles, descriptifs, prixHT ainsi que le libellé de la catégorie des différents articles.

Selon-vous, sur la base de cette requête, que donnera l'exécution des lignes suivantes :

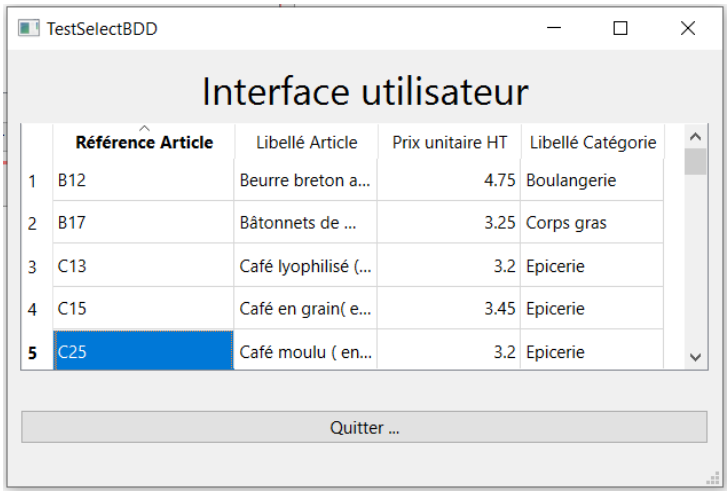
```
les lignes suivantes permettront d'afficher uniquement la reference article, le libellé et le prix (les 3 premieres propriétés)
for(int i = 0; query.next(); i++){
        qDebug() << query.value(0) << query.value(1) << query.value(2) << Qt::endl;

}
```

Finalisez le programme afin de vérifier que vous obtenez les résultats attendus.

Activité 2 : Création d'une IHM avec Qt exploitant les résultats d'une requête Sql

Sur la base des résultats précédents, on souhaite maintenant créer l'interface graphique ci-dessous :



Pour ce faire, vous utiliserez un composant graphique de type tableWidget dans lequel vous créerez les 4 entêtes de colonnes. Les différentes lignes seront créées dans le code de votre constructeur **TestSelectBDD** à l'aide de la méthode setItem de la classe tableWidget.

A l'aide du designer de Qt, concevez cette interface et complétez votre code (signaux notamment) pour obtenir le programme attendu. Votre interface permet-elle de trier sur l'un ou l'autre des colonnes ?

Activité 3

On souhaite créer une nouvelle application capable de créer une table Villes dans votre base de données personnelle sur lakartxela si cette table n'existe pas déjà, puis d'y insérer un certain nombre d'enregistrements avant d'en afficher le contenu dans l'interface graphique de l'application.

La requête SQL permettant de créer la structure de la table Villes est la suivante :

```
create table Villes (
        codepostal varchar(5) primary key, nomVille varchar(30), nbHabitants int(10)
                )
```

1- En partant du même fichier database.h que celui de l'activité 2, ajoutez une méthode publique : bool **restoreDataBase**().

2- On veut écrire ensuite le code de cette méthode qui vérifiera d'abord que la base de données existe avant de tenter de créer la table Villes.

Si la base de données n'existe pas, le programme l'indiquera en mode debug. De même, si la base existe et que la table Villes existe déjà, le programme l'indiquera en mode debug.

Au contraire, si la base existe et que la table Villes n'existait pas, une méthodes insertVilles() de notre classe Database sera appelée pour insérer les enregistrements dans la table.

Sachant que la méthode insertVilles() sera codée comme suit, on vous demande d'analyser /commenter les lignes importantes du code fourni :

```
bool Database::insertVilles()
{
    QSqlQuery query;
    if (!query.exec("insert into Villes values('64000', 'Pau', 70000);"))
            return false;

    QString insertions="insert into Villes values(?, ?, ?);";
    query.prepare(insertions);
    QVariantList codesPostaux;
    codesPostaux << "33000" << "32000" ;
    query.addBindValue(codesPostaux);

    QVariantList nomsVilles;
    nomsVilles << "Bordeaux" << "Auch";
    query.addBindValue(nomsVilles);

    QVariantList nbHabitants;
    nbHabitants << 300000 << 25000;
    query.addBindValue(nbHabitants);

    if (!query.execBatch())
            return false;
    return true;
}
```

3- Après l'avoir codée, vérifiez que votre application est bien capable de créer ces enregistrements dans le table Villes, si nécessaire.
Vous testerez les différents cas de figure à savoir :

- Base de données inexistante ou non accessible (via le DSN fourni ; ou à cause d'un pb réseau/wifi)
- Table Villes inexistante (qu'il faut donc créer avant d'y ajouter des enregistrements)
- Table Villes existante (ce qui peut générer des problèmes si on essaie de rajouter des enregistrements existants).

4- Après avoir analysé le code ci-dessous, ajoutez la méthode ci-dessous à votre classe et exploitez-la dans votre programme :

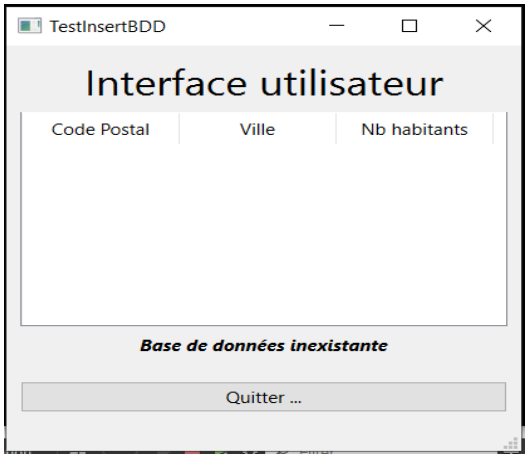
```
bool DataBase::insertIntoTableVilles(const QVariantList &data)
{
    QSqlQuery query;
    QString insertions="insert into Villes "
                       "     values(:cpVille, :nomVille, :nbHabitants);";
    query.prepare(insertions);
    query.bindValue(":cpVille", data[0].toInt());
    query.bindValue(":nomVille", data[1].toString());
    query.bindValue(":nbHabitants", data[2].toInt());

    if(!query.exec()){
        qDebug() << "Erreur lors de l'insertion : ";
        qDebug() << query.lastError().text();
        return false;
        }
    else
        return true;
}
```

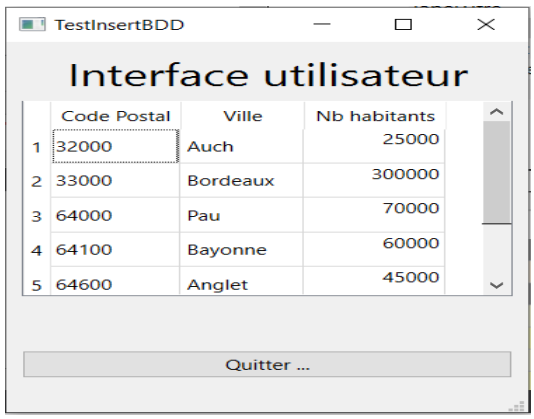
Activité 4

On souhaite maintenant utiliser l'application obtenue pour maintenant la doter d'une interface graphique digne de ce nom.

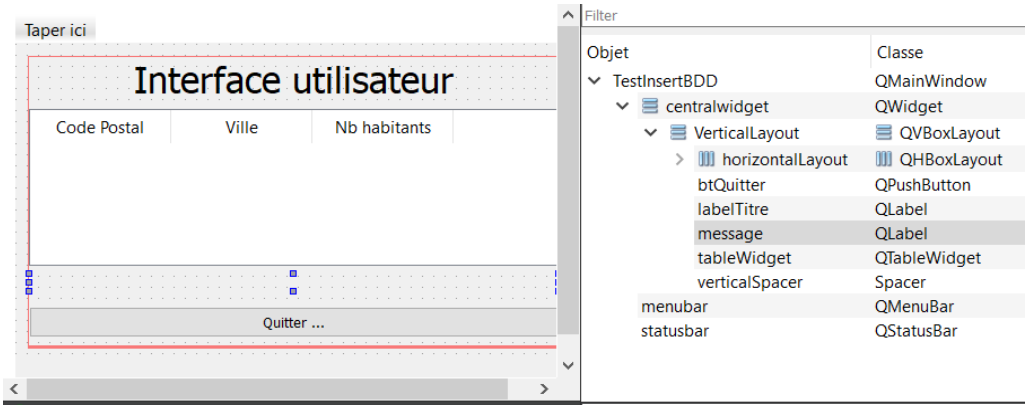
Si l'on constate un problème lors de la connexion à la base de données, l'interface ci-dessous sera utilisée.



Dans les autres cas, l'application affichera l'interface graphique ci-dessous.



A cette fin, vous utiliserez un composant graphique de type tableWidget pour obtenir le résultat attendu et pourrez par exemple structurer votre interface graphique comme suit :



1- Utilisez le designer pour créer votre application graphique selon les besoins précisés puis testez son fonctionnement. Pour ce faire, n'oubliez pas les signaux et slots nécessaires à son bon fonctionnement.