Qt pour Android : Base de données SQLite

> Version PDF du document http://tvaira.free.fr/dev/qt-android/qt-android-base-donnees-sqlite.html
> Thierry Vaira <tvaira@free.fr>
> 2019 (rev. 1.0)

L'application à développer devra utiliser la technique dite *embedded SQL* : les instructions en langage SQL seront incorporées dans le code source d'un programme écrit dans un autre langage (ici le C++ sous Qt pour Android).

Lire : Activité bases de données

SQLite est un moteur de base de données relationnelle accessible par le langage SQL et intégrée dans chaque appareil Android.

Remarques :

- Si l'application crée une base de données, celle-ci est par défaut enregistrée dans le répertoire : `/data/APP_NAME/databases/DATABASE_NAME`.
- Il est possible ensuite de récupérer la base de données créée à partir de l'émulateur `adb` :

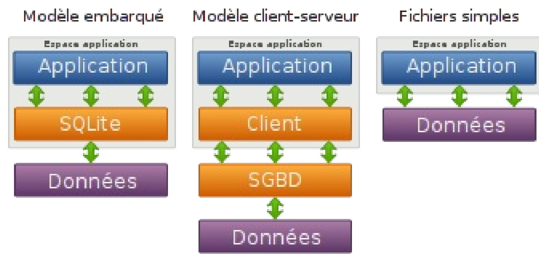
```
$ adb -d shell "run-as com.example.tv.APP_NAME cat /data/com.example.tv.APP_NAME/databases/DATABASE_NAME" > bd.sqlite
```

On peut ensuite l'ouvrir avec le *plugin* **SQLite Manager de Firefox** ou avec l'application `sqlitebrowser` ou `sqliteman` sous Linux.

Il existe aussi un exemple pour une base de données MySQL.

SQLite

SQLite est une bibliothèque écrite en C qui propose un moteur de base de données relationnelle accessible par le langage SQL. Contrairement aux serveurs de bases de données traditionnels, comme MySQL ou PostgreSQL, sa particularité est de ne pas reproduire le schéma habituel client-serveur mais d'être directement intégrée aux programmes. L'intégralité de la base de données (déclarations, tables, index et données) est stockée dans un fichier indépendant de la plateforme. SQLite est le moteur de base de données le plus distribué au monde, grâce à son utilisation dans de nombreux logiciels grand public comme Firefox, Skype, Google Gears, dans certains produits d'Apple, d'Adobe et de McAfee et dans les bibliothèques standards de nombreux langages comme PHP ou Python. De par son extrême légèreté (moins de 300 Kio), il est également très populaire sur les systèmes embarqués, notamment sur la plupart des smartphones modernes : l'iPhone ainsi que les systèmes d'exploitation mobiles Symbian et Android l'utilisent comme base de données embarquée.



API Qt

Qt fournit de nombreuses classes pour la gestion des base de données. Il faudra activer le module dans son fichier de projet `.pro` pour pouvoir accéder aux classes :

```
...
QT += sql
...
```

On utilisera alors la classe `QSqlDatabase` qui permet la connexion à une base de données.

Lien : [La classe QSqlDatabase Qt5 (en)]

Et ensuite la classe `QSqlQuery` pour exécuter des requêtes SQL :

Lien : [La classe QSqlQuery Qt5 (en)]

Déploiement

On peut assurer le déploiement de la base de données liée à l'application dans son fichier de projet `.pro` :

- Android (*apk*) :

```
...
deployment.files += mabase.sqlite
deployment.path = /assets/db
INSTALLS += deployment
```

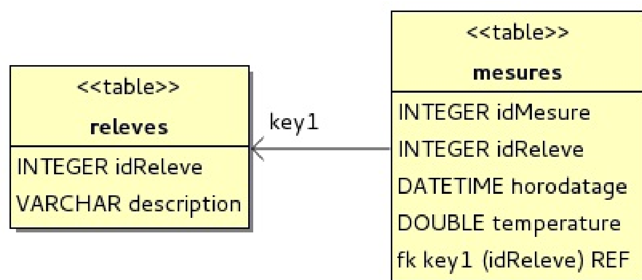
- Desktop (*build*) :

```
...
# copie la base de données dans le dossier build
CONFIG += file_copies
COPIES += bd
bd.files = mabase.sqlite
bd.path = $$OUT_PWD/
bd.base = $$PWD/
```

Exemple

Pour l'exemple, on va créer une base de données `mabase.sqlite` comprenant 2 tables :

- la table `releves` qui contiendra la description des relevés
- la table `mesures` qui contiendra les mesures horodatées de températures pour chaque relevé




```
pragma foreign_keys = on;

CREATE TABLE releves (
        idReleve INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL ,
        description VARCHAR(255) NULL
);

CREATE TABLE mesures (
        idMesure INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL ,
        idReleve INTEGER NOT NULL ,
        horodatage DATETIME,
        temperature DOUBLE NULL ,
        CONSTRAINT fk_mesures_1 FOREIGN KEY (idReleve) REFERENCES releves (idReleve) ON DELETE CA
SCADE
);
```

Pour les tests, on va insérer quelques mesures :

```
INSERT INTO releves(idReleve,description) VALUES(1,'Relevé de la serre 1');
INSERT INTO mesures(idReleve,horodatage,temperature) VALUES(1,'2017-04-01 08:00:00',35.23);
INSERT INTO mesures(idReleve,horodatage,temperature) VALUES(1,'2017-04-01 08:30:00',35.1);
INSERT INTO mesures(idReleve,horodatage,temperature) VALUES(1,'2017-04-01 09:00:00',34.45);
INSERT INTO mesures(idReleve,horodatage,temperature) VALUES(1,'2017-04-01 09:30:00',35.02);
INSERT INTO mesures(idReleve,horodatage,temperature) VALUES(1,'2017-04-01 10:00:00',35.53);
INSERT INTO mesures(idReleve,horodatage,temperature) VALUES(1,'2017-04-01 10:30:00',35.24);
INSERT INTO mesures(idReleve,horodatage,temperature) VALUES(1,'2017-04-01 11:00:00',35.25);
INSERT INTO mesures(idReleve,horodatage,temperature) VALUES(1,'2017-04-01 11:30:00',35.7);
INSERT INTO mesures(idReleve,horodatage,temperature) VALUES(1,'2017-04-01 12:00:00',35.61);
INSERT INTO mesures(idReleve,horodatage,temperature) VALUES(1,'2017-04-01 12:30:00',35.65);
INSERT INTO mesures(idReleve,horodatage,temperature) VALUES(1,'2017-04-01 13:00:00',35.75);
INSERT INTO mesures(idReleve,horodatage,temperature) VALUES(1,'2017-04-01 13:30:00',36.03);
INSERT INTO mesures(idReleve,horodatage,temperature) VALUES(1,'2017-04-01 14:00:00',36.1);
INSERT INTO mesures(idReleve,horodatage,temperature) VALUES(1,'2017-04-01 14:30:00',36.05);
INSERT INTO mesures(idReleve,horodatage,temperature) VALUES(1,'2017-04-01 15:00:00',36.33);
INSERT INTO mesures(idReleve,horodatage,temperature) VALUES(1,'2017-04-01 15:30:00',36.5);

INSERT INTO releves(idReleve,description) VALUES(2,'Relevé de la serre 2');
INSERT INTO mesures(idReleve,horodatage,temperature) VALUES(2,'2017-04-01 08:00:00',35.1);
INSERT INTO mesures(idReleve,horodatage,temperature) VALUES(2,'2017-04-01 08:30:00',35.15);
INSERT INTO mesures(idReleve,horodatage,temperature) VALUES(2,'2017-04-01 09:00:00',35.25);
INSERT INTO mesures(idReleve,horodatage,temperature) VALUES(2,'2017-04-01 09:30:00',35.05);
INSERT INTO mesures(idReleve,horodatage,temperature) VALUES(2,'2017-04-01 10:00:00',35.35);
INSERT INTO mesures(idReleve,horodatage,temperature) VALUES(2,'2017-04-01 10:30:00',35.24);
INSERT INTO mesures(idReleve,horodatage,temperature) VALUES(2,'2017-04-01 11:00:00',35.65);
INSERT INTO mesures(idReleve,horodatage,temperature) VALUES(2,'2017-04-01 11:30:00',35.7);
INSERT INTO mesures(idReleve,horodatage,temperature) VALUES(2,'2017-04-01 12:00:00',35.71);
```

```
INSERT INTO mesures(idReleve,horodatage,temperature) VALUES(2,'2017-04-01 12:30:00',35.75);
INSERT INTO mesures(idReleve,horodatage,temperature) VALUES(2,'2017-04-01 13:00:00',35.65);
INSERT INTO mesures(idReleve,horodatage,temperature) VALUES(2,'2017-04-01 13:30:00',35.93);
INSERT INTO mesures(idReleve,horodatage,temperature) VALUES(2,'2017-04-01 14:00:00',36.1);
INSERT INTO mesures(idReleve,horodatage,temperature) VALUES(2,'2017-04-01 14:30:00',36.15);
INSERT INTO mesures(idReleve,horodatage,temperature) VALUES(2,'2017-04-01 15:00:00',36.4);
INSERT INTO mesures(idReleve,horodatage,temperature) VALUES(2,'2017-04-01 15:30:00',36.35);
```

Et quelques requêtes SQL que l'on utilisera ensuite dans le code :

```
// la liste des relevés
SELECT description FROM releves ORDER BY releves.description ASC
```

	description
1	Relevé de la serre 1
2	Relevé de la serre 2

```
// Les 5 dernières mesures du relevé 1
SELECT * FROM (SELECT mesures.horodatage, mesures.temperature FROM mesures INNER JOIN releves ON releves.idReleve = mesures.idReleve WHERE releves.description = 'Relevé de la serre 1' ORDER BY mesures.horodatage DESC LIMIT 5) tmp ORDER BY horodatage ASC LIMIT 5
```

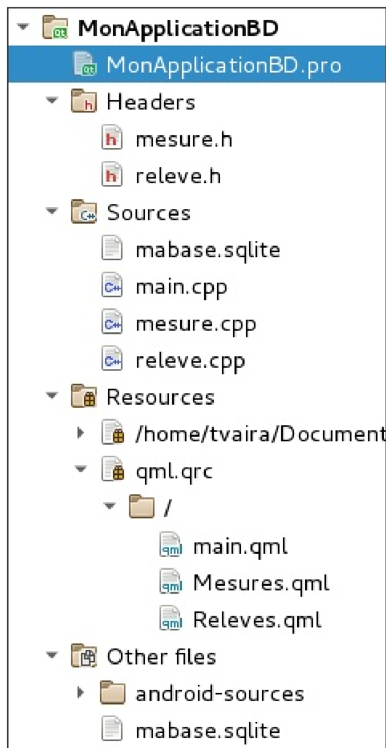
	horodatage	temperature
1	2017-04-01 13:30:00	36.03
2	2017-04-01 14:00:00	36.1
3	2017-04-01 14:30:00	36.05
4	2017-04-01 15:00:00	36.33
5	2017-04-01 15:30:00	36.5

```
// La moyenne des 5 dernières mesures du relevé 1
SELECT AVG(temperature) FROM (SELECT mesures.horodatage, mesures.temperature FROM mesures INNER JOIN releves ON releves.idReleve = mesures.idReleve WHERE releves.description = 'Relevé de la serre 1' ORDER BY mesures.horodatage DESC LIMIT 5) tmp ORDER BY horodatage ASC LIMIT 5
```

	AVG(temperature)
1	36.202

Projet Qt

On crée un projet Qt QML Quick :



Au final, le fichier `.pro` sera le suivant :

```
QT += quick quickcontrols2 sql
CONFIG += c++11

HEADERS += \
    releve.h \
    mesure.h

SOURCES += main.cpp \
    releve.cpp \
    mesure.cpp

RESOURCES += qml.qrc \
             icons/MonApplicationBD/index.theme \
             $$files(icons/*.png, true)

unix:!macx:
{
    android:
    {
        DISTFILES += \
            android-sources/AndroidManifest.xml
        ANDROID_PACKAGE_SOURCE_DIR = $$PWD/android-sources
        # déploie la base de données avec l'apk
        deployment.files += mabase.sqlite
        deployment.path = /assets/db
        INSTALLS += deployment
    }
    !android:
    {
        # copie la base de données dans le dossier build
        CONFIG += file_copies
```

```
        COPIES += bd
        bd.files = mabase.sqlite
        bd.path = $$OUT_PWD/
        bd.base = $$PWD/
    }
}
```

Interactions C++/QML

On a ajouté une classe `Releve` qui aura la charge d'ouvir la base de données `mabase.sqlite`, d'effectuer les 3 requêtes SQL et de fournir les résultats à l'IHM.

La classe `Releve` hérite de `QObject` afin de béníficier des mécanismes Qt :

```
#ifndef RELEVE_H
#define RELEVE_H

#include <QObject>

class Releve : public QObject
{
    Q_OBJECT

public:
    explicit Releve(QObject *parent = nullptr);

private:

signals:

public slots:
};

#endif // RELEVE_H
```

On instanciera un objet `Releve` que l'on rendra accessible à partir de l'IHM décrite dans `qmain.qml`. Cela sera fait dans le fichier `main.cpp` avec `setContextProperty()` :

```
#include <QGuiApplication>
#include <QIcon>
#include <QQmlApplicationEngine>
#include <QQmlContext>

#include "releve.h"

int main(int argc, char *argv[])
{
    QGuiApplication::setApplicationName("MonApplicationBD");
    QCoreApplication::setAttribute(Qt::AA_EnableHighDpiScaling);
    QGuiApplication app(argc, argv);

    QIcon::setThemeName("MonApplicationBD");

    QQmlApplicationEngine engine;
    engine.rootContext()->setContextProperty("Releve", new Releve());
    engine.load(QUrl(QStringLiteral("qrc:/main.qml")));
    if (engine.rootObjects().isEmpty())
```

```
        return -1;

    return app.exec();
}
```

Dans `qmain.qml`, on accèdera à notre objet avec l'identifiant `Releve`.

L'interaction entre la classe C++ `Releve` et l'IHM `main.qml` sera basée sur 3 mécanismes :

- les propriétés qui se déclarent avec la macro `Q_PROPERTY` en C++
- les appels de méthodes qui seront déclarées avec le préfixe `Q_INVOKABLE` en C++
- les signaux qui seront connectés avec l'élément `Connections` en QML

On ajoutera 4 propriétés :

- `erreurConnexion` : pour gérer l'erreur d'ouverture de la base de données
- `listeReleves` : pour récupérer la liste des relevés de la table `releves`
- `mesures` : les mesures d'un relevé sous la forme d'une liste de `Mesure`
- `moyenne` : la moyenne de l'ensemble des `mesures`

Les 3 dernières propriétés ne sont accessibles qu'en lecture auxquelles on associera un accesseur pour `READ`.

```
Q_PROPERTY(bool erreurConnexion MEMBER erreurConnexion NOTIFY erreurChanged)
Q_PROPERTY(QStringList listeReleves READ getReleves NOTIFY listeRelevesChanged)
Q_PROPERTY(QVariant mesures READ getMesures NOTIFY mesuresUpdated)
Q_PROPERTY(QString moyenne READ getMoyenne NOTIFY moyenneUpdated)
```

On pourra par exemple accéder à la propriété `moyenne` directement en QML comme ceci :

```
labelMoyenne.text = "Moyenne : " + Releve.moyenne + " °C";
```

Côté C++, il faudra définir l'accesseur `getMoyenne()` qui permettra de lire (READ) la propriété du même nom :

```
QString Releve::getMoyenne()
{
    return moyenne;
}
```

On créera 3 méthodes publiques appelables depuis QML :

- `lireReleve()` et `lireMoyenneReleve()` qui recevront en paramètres le nom du relevé et le nombre des dernières mesures désirées (0 signifiera toutes les mesures du relevé)
- `executerRequete()` qui recevra en paramètre une requête SQL de type `INSERT`, `UPDATE` ou `DELETE`

```
Q_INVOKABLE bool lireReleve(QString releve, int nb=0);
Q_INVOKABLE void lireMoyenneReleve(QString releve, int nb=0);
Q_INVOKABLE bool executerRequete(QString requete);
```

Côté QML, on pourra apeller une de ces méthodes de la manière suivante :

```
Button {
        id: purgerReleve
        enabled: true;
        text: "Purger le relevé"
        onClicked: {
            Releve.executerRequete("DELETE FROM releves WHERE releves.description = '" + choixRel
eve.currentText + "'");
            //...
        }
}
```

On aura aussi besoin des sigaux suivants :

```
signals:
    void erreurChanged();
    void listeRelevesChanged();
    void mesuresUpdated();
    void mesuresErreur();
    void moyenneUpdated();
...
```

En QML, il est possible de connecter un *slot* à un *signal* de `target` en le préfixant avec `on` comme ceci :

```
Connections {
        target: Releve
        onMoyenneUpdated: {
            labelMoyenne.text = "Moyenne : " + Releve.moyenne + " °C";
            labelMoyenne.color = "#0000FF"
            labelMoyenne.visible = true
        }
        onMesuresErreur: {
            labelMoyenne.text = "Aucune mesure pour ce relevé !";
            labelMoyenne.color = "#FF0000"
            labelMoyenne.visible = true
            listeMesures.visible = false
        }
}
```

On ajoute les membres privés suivants :

```
private:
    QSqlDatabase db;
    bool erreurConnexion;
    QStringList releves;
    QList<QObject*> mesures;
    QString moyenne;
...
```

Au final, la déclaration de la classe `Releve` est la suivante :

```
#ifndef RELEVE_H
#define RELEVE_H

#include <QObject>
```

```cpp
#include <QString>
#include <QtSql/QtSql>
#include <QSqlDatabase>

#define NOM_BD QString("mabase.sqlite")

class Releve : public QObject
{
    Q_OBJECT
    Q_PROPERTY(bool erreurConnexion MEMBER erreurConnexion NOTIFY erreurChanged)
    Q_PROPERTY(QStringList listeReleves READ getReleves NOTIFY listeRelevesChanged)
    Q_PROPERTY(QVariant mesures READ getMesures NOTIFY mesuresUpdated)
    Q_PROPERTY(QString moyenne READ getMoyenne NOTIFY moyenneUpdated)

public:
    explicit Releve(QObject *parent = nullptr);
    virtual ~Releve();

    Q_INVOKABLE bool lireReleve(QString releve, int nb=0);
    Q_INVOKABLE void lireMoyenneReleve(QString releve, int nb=0);
    Q_INVOKABLE bool executerRequete(QString requete);
    QStringList getReleves();
    QVariant getMesures();
    QString getMoyenne();

private:
    QSqlDatabase db;
    bool erreurConnexion;
    QStringList releves;
    QList<QObject*> mesures;
    QString moyenne;

    bool recuperer(QString requete, QStringList &donnees);
    bool recuperer(QString requete, QVector<QStringList> &donnees);
    bool recuperer(QString requete, QString &donnee);
    bool copier(QFile &sfile, QFile &dfile);
    bool remplacer(QFile &sfile, QFile &dfile);
    bool estBDPresente(QString BD);

signals:
    void erreurChanged();
    void listeRelevesChanged();
    void mesuresUpdated();
    void mesuresErreur();
    void moyenneUpdated();

public slots:
};

#endif // RELEVE_H
```

Base de données SQLite

Dans le constructeur de la classe `Releve`, on assurera l'ouverture de la base de données SQLite.

On précisera tout d'abord avec la méthode statique `addDatabase()` le type `QSQLITE`.

Sous Android, il faudra préalablement copier la base de données depuis l'*apk* ("assets:/db") à la racine de

l'application puis lui donner les droits.

Remarque : si l'application Android est déjà installée, la base de données ne sera pas “re-copier”. Si vous souhaitez réinstaller la base de données, vous devez soit désinstaller l'application Android soit appeler la méthode `remplacer()` qui supprimera d'abord le fichier `mabase.sqlite` avant de le copier.

Ensuite, on fixera le nom de la base de données avec `setDatabaseName()` et on l'ouvrira avec `open()`.

```
Releve::Releve(QObject *parent) : QObject(parent), erreurConnexion(false)
{
    db = QSqlDatabase::addDatabase("QSQLITE");

    QFile sfile(QString("assets:/db") + QString("/" + NOM_BD));
    QFile dfile(QString("./" + NOM_BD));

    copier(sfile, dfile);
    // ou :
    //remplacer(sfile, dfile);

    if(estBDPresente(QString("./") + NOM_BD))
    {
        db.setDatabaseName(QString("./") + NOM_BD);
        db.open();
        erreurConnexion = false;
        QSqlQuery r;
        r.exec("pragma foreign_keys = on;");
    }
    else
    {
        erreurConnexion = true;
    }

    emit erreurChanged();
}

bool Releve::copier(QFile &sfile, QFile &dfile)
{
    if (sfile.exists())
    {
        return sfile.copy(dfile.fileName());
    }
    return false;
}

bool Releve::remplacer(QFile &sfile, QFile &dfile)
{
    bool retour;

    // supprime le fichier destination
    if (sfile.exists())
    {
        if (dfile.exists())
        {
            retour = dfile.remove();
            if(!retour)
            {
                return false;
            }
```

```
        }
    }

    return copier(sfile, dfile);
}

bool Releve::estBDPresente(QString BD)
{
    QFile fichier(BD);

    return fichier.exists();
}
```

Ici, le signal `erreurChanged()` est utile pour prévenir l'utilisateur si un problème d'ouverture a eu lieu. Dans ce cas, on affichera une boîte de dialoge avec un message d'erreur :

```
...
onErreurChanged: {
        if (Releve.erreurConnexion)
        {
            messageErreur.text = qsTr("Problème d'ouverture de la base de données !")
            erreurDialog.open()
        }
}

...

Dialog {
        id: erreurDialog
        x: (parent.width - width) / 2
        y: (parent.height - height) / 2
        standardButtons: Dialog.Close
        title: "Erreur"
        Label {
            id: messageErreur
            text: ""
        }
}
```

© http://tvaira.free.fr

Pour exécuter des requêtes SQL, on aura besoin d'un objet `QSqlQuery`. Il faudra distinguer les requêtes SQL `SELECT`, qui retournent des résultats, des requêtes `INSERT`, `UPDATE` ou `DELETE` qui ne produisent pas de résultats en retour.

On passera par des méthodes privées :

- `executerRequete()` qui recevra en paramètre une requête SQL de type `INSERT`, `UPDATE` ou `DELETE`
- `recuperer()` qui recevra en paramètre une requête SQL de type `SELECT` et un type pour stocker les données sélectionnées par la requête. On surchargera la méthode pour les types suivants : `QString`, `QStringList` et `QVector<QStringList>`.

```
bool Releve::executerRequete(QString requete)
{
    QSqlQuery r;

    if(db.isOpen())
    {
        bool retour = r.exec(requete);
        return retour;
    }
    return false;
}

bool Releve::recuperer(QString requete, QStringList &donnees)
```

```
{
    QSqlQuery r;
    bool retour;

    if(db.isOpen())
    {
        retour = r.exec(requete);
        if(retour)
        {
            // pour chaque enregistrement
            while ( r.next() )
            {
                // on stocke l'enregistrement dans le QStringList
                donnees << r.value(0).toString();
            }
            return true;
        }
        else
        {
            return false;
        }
    }
    else
        return false;
}

bool Releve::recuperer(QString requete, QVector<QStringList> &donnees)
{
    QSqlQuery r;
    bool retour;
    QStringList data;

    if(db.isOpen())
    {
        retour = r.exec(requete);
        if(retour)
        {
            // pour chaque enregistrement
            while ( r.next() )
            {
                // on récupère sous forme de QString la valeur de tous les champs sélectionnés
                // et on les stocke dans une liste de QString
                for(int i=0;i<r.record().count();i++)
                    data << r.value(i).toString();

                // on stocke l'enregistrement dans le QVector
                donnees.push_back(data);

                // on efface la liste de QString pour le prochain enregistrement
                data.clear();
            }
            return true;
        }
        else
        {
            return false;
        }
    }
    else
        return false;
```

```
}

bool Releve::recuperer(QString requete, QString &donnee)
{
    QSqlQuery r;
    bool retour;

    if(db.isOpen())
    {
        retour = r.exec(requete);
        if(retour)
        {
            // on se positionne sur l'enregistrement
            r.first();

            // on vérifie l'état de l'enregistrement retourné
            if(!r.isValid())
            {
                return false;
            }

            // on récupère sous forme de QString la valeur du champ
            if(r.isNull(0))
            {
                return false;
            }
            donnee = r.value(0).toString();
            return true;
        }
        else
        {
            return false;
        }
    }
    else
        return false;
}
```

L'utilisation des méthodes `recuperer()` se fera à partir des méthodes publiques suivantes :

- `getReleves()` qui récupera la liste des relevés
- `lireReleve()` qui recevra en paramètres le nom du relevé et le nombre des dernières mesures désirées (0 signifiera toutes les mesures du relevé) et qui fabriquera la liste des `Mesure`
- `lireMoyenneReleve()` qui recevront en paramètres le nom du relevé et le nombre des dernières mesures à prendre en compte dans le calcul de la moyenne (0 signifiera toutes les mesures du relevé) et qui affectera l'attribut `moyenne`

```
QStringList Releve::getReleves()
{
    if(!erreurConnexion)
    {
        releves.clear();
        recuperer("SELECT description FROM releves ORDER BY releves.description ASC", releves);
    }

    return releves;
```

```
}

bool Releve::lireReleve(QString releve, int nb)
{
    if(erreurConnexion)
        return false;

    QVector<QStringList> relevesMesures;
    QString requete;

    if(nb > 0)
    {
        // les nb dernières mesures
        requete = "SELECT * FROM (SELECT mesures.horodatage, mesures.temperature FROM mesures INN
ER JOIN releves ON releves.idReleve = mesures.idReleve WHERE releves.description = '" + releve +
"' ORDER BY mesures.horodatage DESC LIMIT " + QString::number(nb) + ") tmp ORDER BY horodatage AS
C LIMIT " + QString::number(nb);
    }
    else
    {
        // toutes les mesures du relevé
        requete = "SELECT mesures.horodatage,mesures.temperature FROM mesures INNER JOIN releves
ON releves.idReleve = mesures.idReleve WHERE releves.description = '" + releve + "' ORDER BY mesu
res.horodatage ASC";
    }

    qDeleteAll(mesures);
    mesures.clear();

    if(recuperer(requete, relevesMesures))
    {
        for(int i=0;i<relevesMesures.count();i++)
        {
            Mesure *m = new Mesure(QDateTime::fromString(relevesMesures.at(i).at(0), "yyyy-MM-dd
HH:mm:ss"), relevesMesures.at(i).at(1).toDouble(), this);
            mesures.append(m);
        }

        if(mesures.count() > 0)
        {
            emit mesuresUpdated();
            return true;
        }
        else
        {
            emit mesuresErreur();
        }
    }
    else
    {
        qDebug() << Q_FUNC_INFO;
        emit mesuresErreur();
    }
    return false;
}

void Releve::lireMoyenneReleve(QString releve, int nb)
{
    if(erreurConnexion)
        return;
```

```
    QString requete;

    if(nb > 0)
    {
        // la moyenne des nb dernières mesures
        requete = "SELECT AVG(temperature) FROM (SELECT mesures.horodatage, mesures.temperature FROM mesures INNER JOIN releves ON releves.idReleve = mesures.idReleve WHERE releves.description = '" + releve + "' ORDER BY mesures.horodatage DESC LIMIT " + QString::number(nb) + ") tmp ORDER BY horodatage ASC LIMIT " + QString::number(nb);
    }
    else
    {
        // la moyenne de toutes les mesures du relevé
        requete = "SELECT AVG(mesures.temperature) FROM mesures INNER JOIN releves ON releves.idReleve = mesures.idReleve WHERE releves.description = '" + releve + "' ORDER BY mesures.horodatage ASC";
    }

    if(recuperer(requete, moyenne))
    {
        emit moyenneUpdated();
    }
}
```

L'accès au relevé de `Mesure` se fera par l'accesseur de la propriété `listeReleves` :

```
QVariant Releve::getMesures()
{
    return QVariant::fromValue(mesures);
}
```

On créera une nouvelle classe `Mesure` pour stocker une mesure qui est caractérisée par :

- une température (un `double`)
- et son horodatage (un `QDateTime` que l'on retournera sous la forme d'un `QString`)

```
#ifndef MESURE_H
#define MESURE_H

#include <QObject>
#include <QDateTime>

class Mesure : public QObject
{
    Q_OBJECT
    Q_PROPERTY(double temperature READ getTemperature NOTIFY mesure)
    Q_PROPERTY(QString horodatage READ getHorodatage NOTIFY mesure)

public:
    explicit Mesure(QDateTime horodatage, double temperature, QObject *parent = nullptr);

    double getTemperature() const;
    QString getHorodatage() const;

private:
    QDateTime horodatage;
```

```
    double temperature;

signals:
    void mesure();

public slots:
};

#endif // MESURE_H
```

L’IHM

L’interface utilisateur sera décrite en QML avec Qt Quick Controls 2.

On utilisera un `AppWindow` dans `main.qml` :

```
import QtQuick 2.11
import QtQuick.Window 2.3
import QtQuick.Controls 2.2
import QtQuick.Layouts 1.3

ApplicationWindow {
    id: window
    title: qsTr("Mon ApplicationBD")
    width: Screen.desktopAvailableWidth
    height: Screen.desktopAvailableHeight
    property bool erreur: Releve.erreurConnexion
    visible: true

    onErreurChanged: {
        if (Releve.erreurConnexion)
        {
            messageErreur.text = qsTr("Problème d'ouverture de la base de données !")
            erreurDialog.open()
        }
    }
    header: ToolBar {
        RowLayout {
            spacing: 20
            anchors.fill: parent
            Label {
                id: titre
                text: qsTr("Mon ApplicationBD")
                font.pixelSize: 20
                elide: Label.ElideRight
                horizontalAlignment: Qt.AlignHCenter
                verticalAlignment: Qt.AlignVCenter
                Layout.fillWidth: true
            }
            ToolButton {
                id: toolButton2
                icon.name: "menu"
                onClicked: menu.open()
            }
        }
    }
    Menu {
        id: menu
```

```
        x: parent.width - width
        transformOrigin: Menu.TopRight
        MenuItem {
            id: about
            text: "À propos"
            onTriggered: {
                aPropos.open()
            }
        }
    }
    Dialog {
        id: aPropos
        modal: true
        focus: true
        title: "À propos"
        x: (window.width - width) / 2
        y: window.height / 6
        width: Math.min(window.width, window.height) / 3 * 2
        contentHeight: message.height
        Label {
            id: message
            width: aPropos.availableWidth
            text: "Exemple d'utilisation d'une Base de données SQLite en Qt Quick Controls 2."
            wrapMode: Label.Wrap
            font.pixelSize: 12
        }
    }
    Dialog {
        id: erreurDialog
        x: (parent.width - width) / 2
        y: (parent.height - height) / 2
        standardButtons: Dialog.Close
        title: "Erreur"
        Label {
            id: messageErreur
            text: ""
        }
    }
    footer: Label {
            width: parent.width
            horizontalAlignment: Qt.AlignRight
            padding: 10
            text: qsTr("© http://tvaira.free.fr")
            font.pixelSize: 14
            font.italic: true
    }
    Releves
    {
        id: pageReleve
    }
}
```

L'affichage du relevé se fera dans `Releve.qml`. On listera les noms de relevés dans un `ComboBox` et on ajoutera la possibilité de choisir le nombre de dernières mesures que l'on souhaite.

On complètera la GUI avec trois boutons :

- "Annuler" : pour masquer l'affichage des résultats

- “Voir le relevé” : pour afficher les mesures
- “Purger le relevé” : pour supprimer un relevé et ses mesures

On positionnera les éléments avec `Column` et `Row`.

```
import QtQuick 2.9
import QtQuick.Window 2.2
import QtQuick.Controls 2.2
import QtQuick.Layouts 1.3

Page {
    width: parent.width
    padding: 10

    Connections {
        target: Releve
        onMoyenneUpdated: {
            labelMoyenne.text = "Moyenne : " + Releve.moyenne + " °C";
            labelMoyenne.color = "#0000FF"
            labelMoyenne.visible = true
        }
        onMesuresErreur: {
            labelMoyenne.text = "Aucune mesure pour ce relevé !";
            labelMoyenne.color = "#FF0000"
            labelMoyenne.visible = true
            listeMesures.visible = false
        }
    }
    Column {
        id: colonnePage
        width: parent.width
        spacing: 15
        Row {
            Label {
                id: introduction
                wrapMode: Label.Wrap
                text: "Relevés de mesures de températures dans les serres"
            }
        }
        Row {
            ComboBox {
                id: choixReleve
                width: introduction.width
                model: Releve.listeReleves
                enabled: true;
            }
        }
        Row {
            spacing: 20
            Label {
                wrapMode: Label.Wrap
                text: "Nb dernières mesures :"
                anchors.verticalCenter: parent.verticalCenter
            }
            SpinBox {
                id: limiteNbMesures
                enabled: choixReleve.count > 0 ? true : false;
                value: 5
                editable: true
```

```
            }
        }
        Row {
            spacing: 20
            Button {
                id: annuleReleve
                enabled: choixReleve.count > 0 ? true : false;
                text: "Annuler"
                onClicked: {
                    listeMesures.visible = false
                    labelMoyenne.visible = false
                }
            }
            Button {
                id: voirReleve
                enabled: choixReleve.count > 0 ? true : false;
                text: "Voir le relevé"
                onClicked: {
                    if(Releve.lireReleve(choixReleve.currentText, limiteNbMesures.value))
                    {
                        listeMesures.visible = true
                        Releve.lireMoyenneReleve(choixReleve.currentText, limiteNbMesures.value);
                    }
                }
            }
            Button {
                id: purgerReleve
                enabled: choixReleve.count > 0 ? true : false;
                text: "Purger le relevé"
                onClicked: {
                    Releve.executerRequete("DELETE FROM releves WHERE releves.description = '" +
choixReleve.currentText + "'");
                    listeMesures.visible = false
                    labelMoyenne.visible = false
                }
            }
        }
        Row {
                Mesures
                {
                    id: listeMesures
                    visible: false
                    width: colonnePage.width //onglets.width
                    height: Screen.desktopAvailableHeight * 0.45
                    color: "#cfcfcf"
                }
        }
        Row {
            Label {
                id: labelMoyenne
                visible: false
                anchors.margins: 5
                wrapMode: Label.Wrap
                text: ""
            }
        }
    }
}
```

L'affichage des mesures se fera dans `Mesures.qml`. On utilisera un `ListView`.

`ListView` permet une vue en liste des éléments fournis par un `model`, ici notre relevé accessible par la propriété `mesures`. L'affichage des éléments de la liste sera pris en charge par un `delegate`. Le délégué fournit un modèle définissant chaque élément instancié par la `ListView`. Le type `ItemDelegate` utilisé ici est l'élément de vue standard.

La `ListView` étant défilable dans la vue en fonction du nombre de mesures, on ajoutera un `ScrollIndicator`.

```
import QtQuick 2.9
import QtQuick.Window 2.2
import QtQuick.Controls 2.2
import QtQuick.Layouts 1.3
import QtQuick.Controls.Material 2.1

Rectangle {
    ListView {
        id: listeMesures
        anchors.fill: parent
        spacing: 10
        anchors.margins: 5
        model: Releve.mesures
        delegate:
            ItemDelegate {
            width: parent.width
            height: colonne.implicitHeight
            Column {
                id: colonne
                padding: 5
                Text { text: model.modelData.horodatage; }
                Text { text: '<b>Température : ' + model.modelData.temperature + ' °C</b>'; font.
italic: true }
            }
        }
        focus: true
        clip: true
        ScrollIndicator.vertical: ScrollIndicator { }
    }
}
```

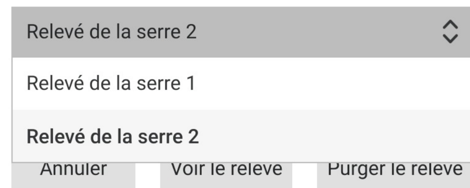
Captures d'écran

On obtient :

100% 14:56

Mon ApplicationBD

Relevés de mesures de températures dans les serres



Annuler | Voir le relevé | Purger le relevé

Le 01/04/2017 à 13:30
Température : 35.93 °C

Le 01/04/2017 à 14:00
Température : 36.1 °C

Le 01/04/2017 à 14:30
Température : 36.15 °C

Le 01/04/2017 à 15:00
Température : 36.4 °C

Le 01/04/2017 à 15:30
Température : 36.35 °C

Moyenne : 36.186 °C

© http://tvaira.free.fr

Mon ApplicationBD

Relevés de mesures de températures dans les serres

Relevé de la serre 2

Nb dernières mesures : − 5 +

Annuler | Voir le relevé | Purger le relevé

Le 01/04/2017 à 13:30
Température : 35.93 °C

Le 01/04/2017 à 14:00
Température : 36.1 °C

Le 01/04/2017 à 14:30
Température : 36.15 °C

Le 01/04/2017 à 15:00
Température : 36.4 °C

Le 01/04/2017 à 15:30
Température : 36.35 °C

Moyenne : 36.186 °C

© http://tvaira.free.fr

Code source

Lien : MonApplicationBD.zip

Voir aussi

Qt fournit le module Qt Charts pour dessiner des graphiques.

Voir : Qt pour Android : dessiner des graphiques

73% 14:24

Mon ApplicationBDCharts

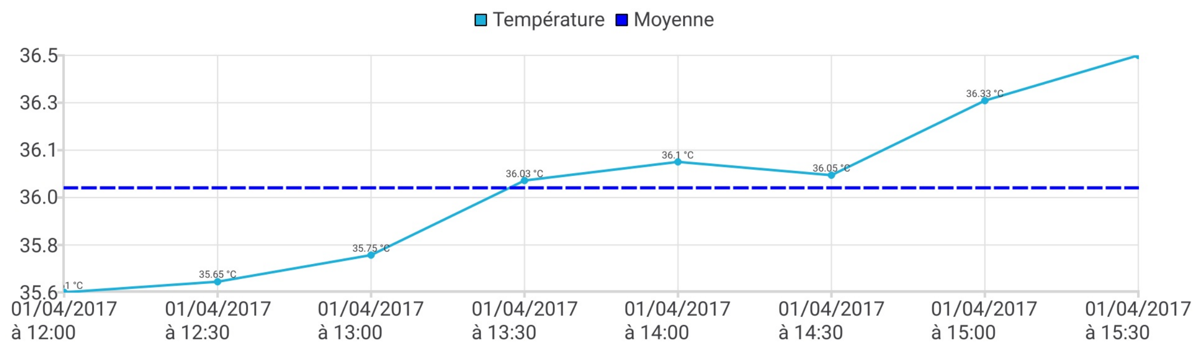
Relevés de mesures de températures dans les serres

Relevé de la serre 1

Nb dernières mesures : − 8 +

Liste Graphique

Annuler | Voir le relevé | Purger le relevé



Moyenne : 36.0025 °C

© http://tvaira.free.fr

Il est aussi possible d'utiliser une base de données **MySQL**.

Voir : [Qt pour Android : Base de données MySQL]

[http://tvaira.free.fr/]