Criando aplicações orientadas a banco de dados usando Qt

Introdução

Uma outra característica interessante do Qt é o seu suporte a Structured Query Language (SQL). Usando a arquitetura model/view do Qt e um conhecimento mínimo de SQL é possível criar aplicações poderosas orientadas a de banco de dados. Se você é novato em SQL é possível encontrar bons materiais introdutórios sobre SQL apenas pesquisando na internet. SQL é considerada a língua mais utilizada por sistemas de gerenciamento de banco de dados relacionais e faz parte de qualquer curso de ciência da computação. Em especial, a implementação conhecida como SQLite é útil para aplicações móveis pois é auto-contida, não necessita de servidor ou configuração e tem suporte a transações. SQLite tem sido usada em muitos projetos grandes como o Mozilla Firefox, nos smartphones Symbian, Skype, Solaris, PHP, Mac OS, entre outros, e é provavelmente a a base de dados SQL mais implantada. O suporte a SQLite em Qt está disponível para o desktop, Meego e Symbian.

Neste artigo será demonstrado como é possível integrar recursos de banco de dados SQLite num aplicativo do tipo "contatos" usando a arquitetura model/view. O último SDK Qt Nokia será utilizado neste exemplo.

Se você estiver usando dispositivos como Maemo N900, é necessário instalar o suporte a SQLite para Qt antes de testar esta aplicação no seu dispositivo. Para isso, abra um terminal e digite os seguintes comandos:

```
sudo gainroot
apt-get install libqt4-sql-sqlite
```

sudo gainroot permite executar comandos no seu celular como administrador, devendo ser usado com cuidado.

Para usuários Symbian, o aplicativo **sqlite3.sis** é necessário. Ele pode ser encontrado dentro do SDK em **\NokiaQtSDK\Symbian\sis\sqlite3.sis** ou pronto para a instalação a partir do menu Iniciar (Nokia Qt SDK -> Symbian -> Install SQLite to Symbian).

Se você não quiser usar um dispositivo real, uma alternativa é usar o simulador do Qt, disponível também no SDK. Neste caso, nenhuma instalação adicional é necessária.

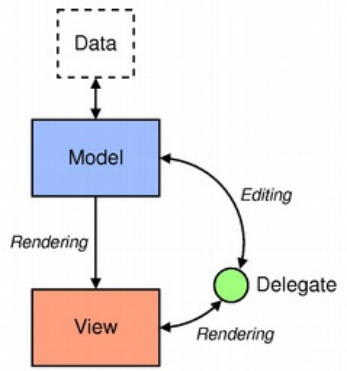
Classes Qt para acessar SQLite

Usando Qt, as classes SQL são divididos em três camadas:

- **Camada de driver**: um conjunto de classes úteis para aqueles que querem criar seu próprio driver para banco de dados ou adicionar alguma personalização aos já existentes.
- **Camada da API SQL**: o acesso básico ao banco de dados, como conexões, erros, manipulação de campos e assim por diante.
- **Camada de interface com o usuário**: um conjunto de classes para ser usado com a arquitetura model/view do Qt.

Somente a segunda e terceira camadas serão utilizados neste artigo, representada pelas seguintes classes disponíveis na arquitetura model/view:

- **QSqlDatabase**: o conector do banco de dados (fonte de dados para o model/view). Como o SQLite é mapeado em um arquivo, a conexão é direta. Para bancos de dados mais sofisticados, parâmetros como login, senha e portas podem ser fornecidos.
- **QSqlTableModel**: usado como modelo, ele irá fornecer uma interface de alto nível para acesso ao banco de dados.
- **QTreeView**: é o objeto de visualização utilizado para mostrar os registros do banco de dados.



[1]

Conectando o banco de dados SQLite a arquitetura model/view do Qt

Como as bases de dados SQLite são mapeados em arquivos, a conexão com o banco de dados consiste apenas em especificar o arquivo a ser utilizado.

(c) Copyright Nokia 2013. All rights reserved.

Naturalmente, o caminho para o arquivo pode ser diferente se você está programando para Meego ou Symbian e você deve prever esta situação com declarações do tipo #ifdef Q_OS_SYMBIAN. Além disso, será preciso criar a estrutura de dados dentro deste arquivo antes de usá-lo para armazenar registros.

No trecho de código seguinte o conector do banco de dados é criado e o banco de dados aberto. Se a estrutura de banco de dados ainda não existir ela será criada.

```
// using SQLITE driver
QSqlDatabase db = QSqlDatabase::addDatabase("QSQLITE");
db.setDatabaseName("path_to_database_file");

if (db.open()) {
    // Try to locate the contacts database.
    // If it is not available create it.
    if (db.tables().indexOf("contacts") == -1) {
        QSqlQuery query(db);
        query.prepare("CREATE TABLE contacts (name VARCHAR(32), phone VARCHAR(16))");
        if (!query.exec())
            qCritical() << query.lastError();
    }
} else {
    qCritical() << db.lastError();
}
```

O próximo passo é associar essa banco de dados à arquitetura model/view. Os métodos **setTable()** (da classe de modelo) e **setModel()** (da classe de visualização) são utilizados para realizar essa tarefa. É necessário chamar **select()** para preencher o modelo com os dados da tabela e o desenvolvedor pode escolher ainda a sua estratégia de edição chamando **setEditEstrategy()**. Neste caso, usando **QSqlTableModel::OnFieldChange**, qualquer mudança será aplicada imediatamente ao banco de dados.

```
// create a new model and use the contacts table
QSqlTableModel *model = new QSqlTableModel(this, db);
model->setTable("contacts");
// All changes to the model will be applied immediately to the database
model->setEditStrategy(QSqlTableModel::OnFieldChange);
// Populate the model with data from table
model->select();
// creating the model/view association
ui->view->setModel(model);
```

Agora o seu aplicativo está pronto para rodar e todos os dados do banco de dados estarão disponíveis para o usuário através do componente de visualização. Além disso, sua aplicação tem suporte para persistência de dados sem nenhuma linha de código, todos os registros serão mantidos entre as execuções sucessivas do programa.

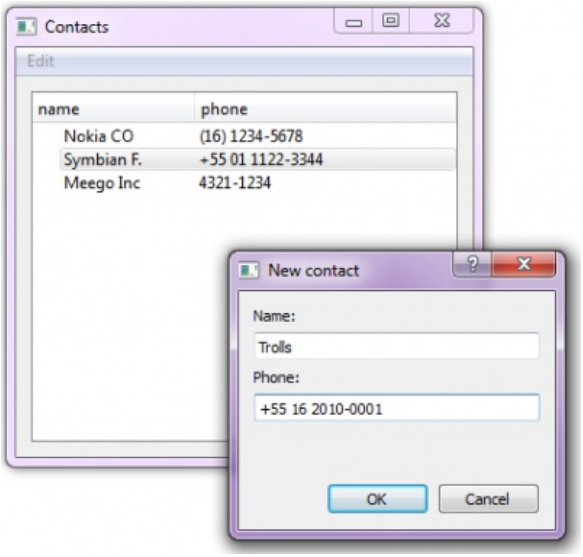
Adicionando registros

Uma vez que os registros podem ser visualizados, a próxima tarefa é criar um diálogo para a coleta de dados dos usuários. O editor de formulários do Qt Creator torna esta tarefa muito fácil e um diálogo simples com apenas dois campos (nome e telefone) é necessário para esta aplicação. Crie este diálogo e as classes relacionadas e adicione ao projeto, criando callbacks para os botões **Cancel** e **Ok**. Para mostrar o diálogo você precisará usar o método **exec()**. Depois de o diálogo ser fechado, você pode verificar se ele tem dados válidos e então adicioná-los ao banco de dados como um novo registo. O método **record()** (do modelo) retornará um registro de SQL que corresponde à sua especificação de banco de dados que você pode preencher esse registro usando o método **setValue()**. Qt properties foram utilizados na classe de diálogo para ajudar a recolher os dados necessários e o método **insertRecord()** irá adicionar este registro ao banco de dados.

```
// Create a new empty dialog and show it using exec()
NewContact dlg(this);
dlg.resetName();
dlg.resetPhone();

if(dlg.exec() == QDialog::Accepted && dlg.name().length() > 0) {
    // create a record from the current model
    QSqlRecord rec = model->record();
    rec.setValue("name", dlg.name());
    rec.setValue("phone", dlg.phone());
    // insert a new record (-1)
    model->insertRecord(-1,rec);
}
```


) Copyright Nokia 2013. All rights reserved.



Excluindo registros

A exclusão de registos é simples uma vez que se tenha o índice do registo. O índice pode ser obtido como um objeto **QModelIndex** através de uma consulta ao componente de visualização. Este índice é então usado em uma chamada do método **removeRow()** (do modelo).

```
// get the current index, if any
QModelIndex sample = ui->view->currentIndex();
if (sample.row() >= 0) {
    QMessageBox::StandardButton reply;
    reply = QMessageBox::question(this, tr("Remove contact"),
                                  QString(tr("Remove contact ?")),
                                  QMessageBox::Yes | QMessageBox::No);

    if(reply == QMessageBox::Yes)
        model->removeRow(sample.row()); // remove the current index
}
```

Modificando registros

Depois de criar os métodos "'addRegister() **e** deleteRecord() **é mais fácil gerar o código para o método** editRecord()**. Basicamente, obtenha o índice do registo e preencha o diálogo com informações deste registro. Depois, apresente este diálogo para o usuário e salve as alterações.**

```
// get the current index, if any
QModelIndex sample = ui->view->currentIndex();
if (sample.row() >= 0 ) {
    // copy the current record
    QSqlRecord rec = model->record(sample.row());
    NewContact dlg(this);
    dlg.setName(rec.value("name").toString());
    dlg.setPhone(rec.value("phone").toString());

    if (dlg.exec() == QDialog::Accepted) {
        rec.setValue("name",dlg.name());
        rec.setValue("phone",dlg.phone());
        // save modified data
        model->setRecord(sample.row(),rec);
    }
}
```

Conclusão

Com o Qt é possível criar programas concisos que usam persistência de dadosem bancos de dados SQL em poucas linhas de códigos. Em especial, a abstração para o SQLite provida pelo Qt é realmente simples quando usada em conjunto com a arquitetura model/view do Qt.

© Copyright Nokia 2013. All rights reserved.

O código-fonte

Este programa foi testado em Maemo, Linux e Windows sem modificações. Você pode baixar todo o código fonte a partir deste link.

© Copyright Nokia 2013. All rights reserved.

© Copyright Nokia 2013. All rights reserved.