Lecture SQL07
Qt and SQL – Part II

continued

Outline

- Last Time
  - Connecting to MySQL and SQLite
  - Connection Errors
  - Querying the Database from Qt
- **QSqlDatabase** Class
  - Default connection
  - Named connections
- QSqlTableModle/QTableView

QSqlDatabase Class

- Represents a connection to a database
- Connections are known by the connection name not the database it is connected to
- It is possible to have multiple connections to the same database
- If no name is provided, then the default connection is used

Default Connection

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

db.setHostName( "localhost" );    // db on local machine
db.setDatabaseName( "somefilenamehere" );
db.setUserName( "myusername" );
db.setPassword( "mypassword" );

if ( !db.open() )
{
  qDebug() << db.lastError();
  qDebug() << "Error:  Unable to connect due to above error";
}

or

QSqlDatabase  db = QSqlDatabase::addDatabase( "QSQLITE" );

db.setDatabaseName( "vetclinic.db" );

if ( !db.open() )
{
  qDebug() << db.lastError();
  qDebug() << "Error:  Unable to connect due to above error";
}
```

No connection name specified. Only type listed.

Same here.

Named Connections - 1

```
//  MultipleDBases Example
#include <QApplication>
#include <QtSql>
#include <QtDebug>

int main(int argc, char* argv[])
{
    QApplication myApp(argc, argv);

    QSqlDatabase customerdb = QSqlDatabase::addDatabase("QSQLITE", "cdb");
    QSqlDatabase petdb = QSqlDatabase::addDatabase("QSQLITE", "pdb");
    QSqlDatabase accountdb = QSqlDatabase::addDatabase("QSQLITE", "adb");
    QSqlDatabase vetdb = QSqlDatabase::addDatabase("QSQLITE", "vdb");

    customerdb.setDatabaseName("customers.db");
    petdb.setDatabaseName("pets.db");
    accountdb.setDatabaseName("accounts.db");
    vetdb.setDatabaseName("vets.db");

    if ( !(customerdb.open() && petdb.open() && accountdb.open() && vetdb.open()) )
    {
        qDebug() << "Error: unable to open one or more databases";
        return 1;
    }
```

Named Connections - 2

```
QSqlQuery q("SELECT * FROM customers;", customerdb);
if ( !q.isActive() )
{
    qDebug() << q.lastError();
    qDebug() << "Error: query failed";
}
while ( q.next() )
{
    qDebug() << "UID: " << q.value(0).toInt() << "  "
             << q.value(2).toString() << ", " << q.value(1).toString();
}
```

UID: 128 "John" , "Smith"
UID: 324 "John" , "Doe"
UID: 245 "Mark" , "Jones"
UID: 756 "Jane" , "Smith"
UID: 459 "Sara" , "Moore"
UID: 721 "Ralph" , "Parks"

Named Connections - 3

```
QSqlQuery qq("SELECT * FROM pets", petdb);
if ( !qq.isActive() )
{
    qDebug() << qq.lastError();
    qDebug() << "Error: query failed";
}
while ( qq.next() )
{
    qDebug() << "UID: " << qq.value(0).toInt() << "  Type:"
             << qq.value(2).toString() << "   PetName:" << qq.value(1).toString();
}
```

UID: 128 Type: "Dog" PetName: "Spot"
UID: 324 Type: "Dog" PetName: "Rex"
UID: 756 Type: "Cat" PetName: "Tiger"
UID: 756 Type: "Cat" PetName: "Fluffy"
UID: 459 Type: "Bird" PetName: "Tweety"
UID: 721 Type: "Dog" PetName: "Yippy"
UID: 128 Type: "Dog" PetName: "Rover"
UID: 245 Type: "Cat" PetName: "Stripes"
UID: 324 Type: "Dog" PetName: "Cupcake"
UID: 459 Type: "Dog" PetName: "Chewy"

Named Connections - 4

```
    QSqlQuery qqq( accountdb );
    qqq.exec("SELECT * FROM accounts");
    if ( !qqq.isActive() )
    {
        qDebug() << qq.lastError();
        qDebug() << "Error: query failed";
    }
    while ( qqq.next() )
    {
        qDebug() << "UID: " << qqq.value(0).toInt() << "   $" << qqq.value(1).toInt();
    }

    return myApp.exec();
} // End main()
```

UID: 128 $ 0
UID: 756 $ 45
UID: 459 $ 0
UID: 721 $ 10

Querying the Database from Qt With a GUI

UAH

CPE 353

Relations

customers

UID	Last Name	First Name
128	Smith	John
324	Doe	John
245	Jones	Mark
756	Smith	Jane
459	Moore	Sara
721	Parks	Ralph

vets

UID
324
245

accounts

UID	Balance
128	0
756	45
459	0
721	10

pets

UID	Pet Name	Type
128	Spot	Dog
324	Rex	Dog
756	Tiger	Cat
756	Fluffy	Cat
459	Tweety	Bird
721	Yippy	Dog
128	Rover	Dog
245	Stripes	Cat
324	Cupcake	Dog
459	Chewy	Dog

VetClinic Qt/SQL Example - 1

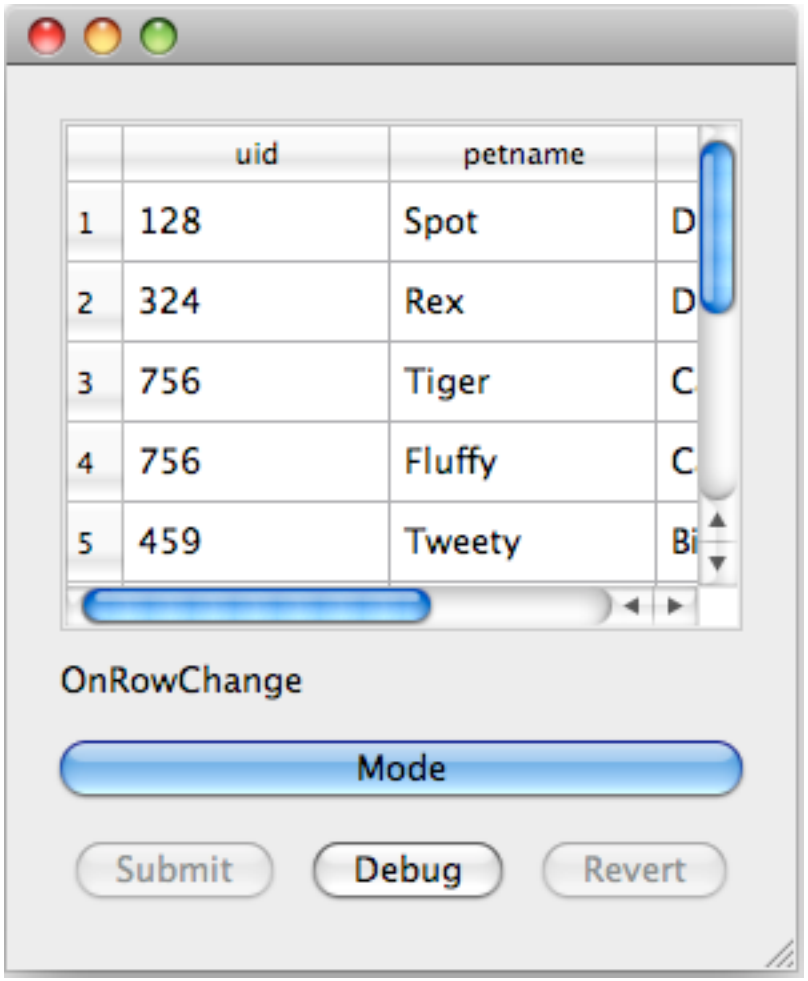
```
#include <QApplication>
#include <QtSql>
#include <QTableView>
#include <QtGui>
#include "dialog.h"
using namespace std;

int main(int argc, char* argv[])
{
    QApplication myApp(argc, argv);

    Dialog d;
    d.show();

    return myApp.exec();

} // End main()
```



VetClinic Qt/SQL Example - 2

```
// dialog.h

#ifndef DIALOG_H
#define DIALOG_H

#include <QtGui>
#include <QtSql>

class Dialog : public QDialog
{
  Q_OBJECT

  public:
    Dialog();
    ~Dialog();

  private:
    QSqlDatabase db;
    QSqlTableModel* model;
    QTableView* view;

    QVBoxLayout* mainlayout;
    QHBoxLayout* hbox;
    QPushButton* submit;
    QPushButton* revert;
    QPushButton* dbug;
    QPushButton* mode;
    QLabel* label;
```

VetClinic Qt/SQL Example - 3

```
// dialog.h - continued



  public slots:
    void DebugLog();
    void SwitchMode();
};

#endif // DIALOG_H
```

VetClinic Qt/SQL Example - 4

```
// dialog.cpp
#include "dialog.h"

Dialog::Dialog()
{
    db = QSqlDatabase::addDatabase("QSQLITE");
    db.setDatabaseName("vetclinic.db");

    if ( !db.open() )
    {
        qDebug() << db.lastError();
        qDebug() << "Error: Unable to connect";
        exit(1);
    }

    model = new QSqlTableModel;
    model->setTable("pets");
    model->select();
    model->setEditStrategy(QSqlTableModel::OnRowChange);

    view = new QTableView;
    view->setModel(model);
```

VetClinic Qt/SQL Example - 5

```
// dialog.cpp - continued

    mainlayout = new QVBoxLayout(this);
    hbox = new QHBoxLayout;
    submit = new QPushButton("Submit");
    submit->setDisabled(true);
    revert = new QPushButton("Revert");
    revert->setDisabled(true);
    dbug = new QPushButton("Debug");
    mode = new QPushButton("Mode");
    label = new QLabel("OnRowChange");

    mainlayout->addWidget(view);
    hbox->addWidget(submit);
    hbox->addStretch();
    hbox->addWidget(dbug);
    hbox->addStretch();
    hbox->addWidget(revert);
    mainlayout->addStretch();
    mainlayout->addWidget(label);
    mainlayout->addStretch();
    mainlayout->addWidget(mode);
    mainlayout->addStretch();
    mainlayout->addLayout(hbox);
```

VetClinic Qt/SQL Example - 6

```
// dialog.cpp - continued

    connect(submit, SIGNAL(clicked()), model, SLOT(submitAll()));
    connect(revert, SIGNAL(clicked()), model, SLOT(revertAll()));
    connect(dbug, SIGNAL(clicked()), this, SLOT(DebugLog()));
    connect(mode, SIGNAL(clicked()), this, SLOT(SwitchMode()));
}

Dialog::~Dialog()
{
    db.close();
}

void Dialog::DebugLog()
{
  QSqlQuery  q("SELECT * FROM pets");
  qDebug() << endl;
  while ( q.next() )
  {
      qDebug() << q.value(0).toInt() << "   " << q.value(1).toString()
               << "   " << q.value(2).toString();
  }
  qDebug() << endl;
}
```

Reverts all pending changes

Submits all pending changes

VetClinic Qt/SQL Example - 7

```
// dialog.cpp - continued
void Dialog::SwitchMode()
{
  if (label->text() == "OnRowChange")
  {
      label->setText("OnFieldChange");
      submit->setDisabled(true);
      revert->setDisabled(true);
      model->setEditStrategy(QSqlTableModel::OnFieldChange);
      qDebug() << endl << "Mode = OnFieldChange" << endl;
  }
  else if (label->text() == "OnFieldChange")
  {
      label->setText("OnManualSubmit");
      submit->setDisabled(false);
      revert->setDisabled(false);
      model->setEditStrategy(QSqlTableModel::OnManualSubmit);
      qDebug() << endl << "Mode = OnManualSubmit" << endl;
  }
  else if (label->text() == "OnManualSubmit")
  {
      label->setText("OnRowChange");
      submit->setDisabled(true);
      revert->setDisabled(true);
      model->setEditStrategy(QSqlTableModel::OnRowChange);
      qDebug() << endl << "Mode = OnRowChange" << endl;
  }
}
```

Integrity Constraints - 1

```
--
-- Vet Clinic Example
--

-- Create and populate the customers table
CREATE TABLE customers
(
  uid INTEGER CHECK( uid > 0 ),
  lastname TEXT,
  firstname TEXT
);

INSERT INTO customers (uid, lastname, firstname) VALUES (128, 'Smith', 'John');
INSERT INTO customers (uid, lastname, firstname) VALUES (324, 'Doe', 'John');
INSERT INTO customers VALUES (245, 'Jones', 'Mark');
INSERT INTO customers VALUES (756, 'Smith', 'Jane');
INSERT INTO customers (lastname, firstname, uid) VALUES ('Moore', 'Sara', 459);
INSERT INTO customers (lastname, firstname, uid) VALUES ('Parks', 'Ralph', 721);


-- Create and populate the accounts table
CREATE TABLE accounts
(
  uid INTEGER CHECK( uid > 0 ),
  balance DECIMAL
);

INSERT INTO accounts (uid, balance) VALUES (128, 0.00);
INSERT INTO accounts (uid, balance) VALUES (756, 45.00);
INSERT INTO accounts (uid, balance) VALUES (459, 0.00);
INSERT INTO accounts (uid, balance) VALUES (721, 10.00);
```

Integrity Constraints - 2

```
-- Create and populate the vets table
CREATE TABLE vets
(
  uid INTEGER CHECK( uid > 0 )
);

INSERT INTO vets (uid) VALUES (324);
INSERT INTO vets (uid) VALUES (245);


-- Create and populate the pets table
CREATE TABLE pets
(
  uid INTEGER CHECK( uid > 0 ),
  petname CHARACTER VARYING,
  type TEXT
);

INSERT INTO pets VALUES (128, 'Spot', 'Dog');
INSERT INTO pets VALUES (324, 'Rex', 'Dog');
INSERT INTO pets VALUES (756, 'Tiger', 'Cat');
INSERT INTO pets VALUES (756, 'Fluffy', 'Cat');
INSERT INTO pets VALUES (459, 'Tweety', 'Bird');
INSERT INTO pets VALUES (721, 'Yippy', 'Dog');
INSERT INTO pets VALUES (128, 'Rover', 'Dog');
INSERT INTO pets VALUES (245, 'Stripes', 'Cat');
INSERT INTO pets VALUES (324, 'Cupcake', 'Dog');
INSERT INTO pets VALUES (459, 'Chewy', 'Dog');
```

Integrity Constraints - 3

- QSqlTableModel/QTableView rejects inputs that violate integrity constraints

VetClinic Qt/SQL IC Example

```
void Dialog::DebugLog()
{
  QSqlQuery  q("SELECT * FROM pets");
  qDebug() << endl;
  qDebug() << model->lastError();
  qDebug() << endl;
  while ( q.next() )
  {
      qDebug() << q.value(0).toInt() << "   " << q.value(1).toString()
               << "   " << q.value(2).toString();
  }
  qDebug() << endl;
}
```

```
QSqlError(19, "Unable to fetch row", "constraint failed")
128 "Spot" "Dog"
324 "Rex" "Dog"
756 "Tiger" "Cat"
756 "Fluffy" "Cat"
459 "Tweety" "Bird"
721 "Yippy" "Dog"
128 "Rover" "Dog"
245 "Stripes" "Cat"
324 "Cupcake" "Dog"
459 "Chewy" "Dog"
```

More Integrity Constraints - 1

```
sqlite3 vetclinic.db
SQLite version 3.4.0
Enter ".help" for instructions
sqlite> .dump
BEGIN TRANSACTION;
CREATE TABLE customers
(
  uid INTEGER CONSTRAINT uid_range CHECK( (9999 > uid) AND (uid > 0) ) ,
  lastname TEXT,
  firstname TEXT,
  UNIQUE(uid)
);
INSERT INTO "customers" VALUES(128,'Smith','John');
INSERT INTO "customers" VALUES(324,'Doe','John');
INSERT INTO "customers" VALUES(245,'Jones','Mark');
INSERT INTO "customers" VALUES(756,'Smith','Jane');
INSERT INTO "customers" VALUES(459,'Moore','Sara');
INSERT INTO "customers" VALUES(721,'Parks','Ralph');
```

Named Constraints

UNIQUE ensures no duplicate values in listed columns

More Integrity Constraints - 2

```
CREATE TABLE accounts
(
  uid INTEGER CONSTRAINT uid_range CHECK( (9999 > uid) AND (uid > 0) ),
  balance DECIMAL
);
INSERT INTO "accounts" VALUES(128,0);
INSERT INTO "accounts" VALUES(756,45);
INSERT INTO "accounts" VALUES(459,0);
INSERT INTO "accounts" VALUES(721,10);
CREATE TABLE vets
(
  uid INTEGER CONSTRAINT uid_range CHECK( (9999 > uid) AND (uid > 0) )
);
INSERT INTO "vets" VALUES(324);
INSERT INTO "vets" VALUES(245);
```

More Integrity Constraints - 3

```
CREATE TABLE pets
(
  uid INTEGER CONSTRAINT uid_range CHECK( (9999 > uid) AND (uid > 0) ),
  petname CHARACTER VARYING,
  type TEXT
);
INSERT INTO "pets" VALUES(128,'Spot','Dog');
INSERT INTO "pets" VALUES(324,'Rex','Dog');
INSERT INTO "pets" VALUES(756,'Tiger','Cat');
INSERT INTO "pets" VALUES(756,'Fluffy','Cat');
INSERT INTO "pets" VALUES(459,'Tweety','Bird');
INSERT INTO "pets" VALUES(721,'Yippy','Dog');
INSERT INTO "pets" VALUES(128,'Rover','Dog');
INSERT INTO "pets" VALUES(245,'Stripes','Cat');
INSERT INTO "pets" VALUES(324,'Cupcake','Dog');
INSERT INTO "pets" VALUES(459,'Chewy','Dog');
COMMIT;
sqlite>
```

More Integrity Constraints - 4

```
sqlite> INSERT INTO vets VALUES(000);
SQL error: constraint failed
sqlite> INSERT INTO vets VALUES(10000);
SQL error: constraint failed
sqlite> INSERT INTO customers VALUES(000,'Nail', 'Rusty');
SQL error: constraint failed
sqlite> INSERT INTO customers VALUES(10000,'Nail', 'Rusty');
SQL error: constraint failed
sqlite> INSERT INTO customers VALUES(128,'Nail', 'Rusty');
SQL error: column uid is not unique
sqlite>
```