【Experimental Name】 Journal Management System Coding and Testing

【Purpose】

Master the common programming techniques, the key techniques of database operation, and the key techniques of interface realization.

database connection technology

1. SQLite

Considering the nature of our courses and the computer configuration of each team member, we finally decided to use the *Sqlite* database. *Sqlite* is a relatively small local database, which is suitable for storing some software configuration parameters or small amount of data. At the same time, *Qt* itself has its own *Sqlite* driver, so we don't need to configure the environment, just use the relevant class library directly.

2. reference header file

In the class definition that needs to use *SQL* , refer to the relevant header files , for example:

```
#include <QSqlDatabase>
#include <QSqlError>
```

3. build database

Check connection, add database driver, set database name, user name, password

```
QSqlDatabase database;
if (QSqlDatabase::contains("qt_sql_default_connection"))
{
database = QSqlDatabase::database("qt_sql_default_connection");
}
else
{
database = QSqlDatabase::addDatabase("QSQLITE");
database.setDatabaseName("MyDataBase.db");
database.setUserName("XingYeZhiXia");
database.setPassword("123456");
}
```

1) *QSqlDatabase object* is established , and subsequent operations will use this

object.

2) The if statement is used to check whether the specified connection (*connection*) exists . The connection name specified here is *qt_sql_default_connection* , which is the Qt default connection name. In actual use, this name can be chosen arbitrarily.

3) If it is judged that the connection already exists, then *the QSqlDatabase::contains()* function returns true. At this point, enter the first branch, *QSqlDatabase::database()* returns the connection ; if the connection does not exist, enter *the else* branch, you need to create a connection and add a database.

4) In the first line of *the else branch,* the parameter *QSQLITE of addData base()* is the driver name corresponding to *SQLite , which cannot be ch anged* , and it should be noted that the second parameter of *addDataba se() is omitted, and the default parameter of the second parameter is th e above* The mentioned *Qt* default connection name *qt_sql_default_connection* .

   If you need to use a custom connection name (this is the case if the pr ogram needs to process multiple database files), you should add a secon d parameter, for example :

database = QSqlDatabase::addDatabase("QSQLITE", "my_sql_connection);

the my_sql_connection connection exists in another place , you should us e *if (QSqlDatabase::contains("my_sql_connection"))* .

(4) In the second line of the *else branch, the parameter of setDatabaseN ame()* is the database file name. If the database does not exist, it will b e created automatically during subsequent operations; if it already exists, subsequent operations will be performed on the existing database.

(5) In the next two lines of the *else branch, set the user name and pas sword.* Username and password can be taken at will or omitted.

4. open database

Use open() to open the database and determine whether it is successful. Note that when checking whether the connection exists in the first step, if the connection exists, the database will be opened by default when the connection is returned .

```
if (!database. open())
{
qDebug() << "Error: Failed to connect database." << database.lastError();
}
else
{
// do something
}
```

If the opening is successful, enter *the else* branch , and all operations on the database need to be performed in *the else* branch .

5. close database

After the database operation is complete, it is best to close it , that is, use *database.close()* .

一、 key database operations

1. Create user information form

Description: Create a table named *userInfo* The order is : id - username - password - permission

CREATE TABLE userInfo (userid int PRIMARY KEY, name VARCHAR, passwd VARCHAR, level VARCHAR)

2. Insert user information

Explanation: The rightmost column level indicates the authority, ' 0 ' indicates the administrator, ' 1 ' indicates the normal user

insert into userInfo values (1001,'admin','123456','0')
insert into userInfo values (1002,'jym','987654','1')

3. Find user information

Description: Find the student whose user id is 1001 ; find the user information whose identity is the administrator

*select * from userInfo where userid=1001*
*select * from userInfo where level='0'*

4. Create Borrowing Information Form

Description: Create a table named VISITINFO to record information about borrowing

CREATE TABLE VISITINFO (strSFBH VARCHAR PRIMARY KEY, strName VARCHAR, strDes VARCHAR, strStartTIME VARCHAR, strEndTime VARCHAR, strOper VARCHAR)

5. Find Journal Information

Description: Find the borrowing information of the user whose ID number is 1 001 ; find the number of periodicals he is borrowing

*Select * from VISITINFO where strSFBH=1001*
Select strDes from VISITINFO where strSFBH=1001

6. update user information

Description: Change the password of the user whose id is 1 001 to ' 9 87654' ; change the identity of the user whose id is 1 002 to an administrator, and change the name to ' a dmin02'

Update userInfo set passwd='987654' where userid=1001
Updata userInfo set name='admin02', level='0' where userid=1002

【Detailed code】

【' User Login ' module】

```
//system login module
#include "login.h"
#include "ui_login.h"
#include <QMessageBox>
#include <QUrl>
#include <QDesktopServices>
#include <QDir>
#include <QDebug>
#include <QMenu>
#include <QDateTime>
#include "api/mydatabase.h"
#include "api/define.h"
#include <QSqlError>
float opacity1 = 0.0, opacity2 = 1.0;
Login ::Login(QWidget * parent ) :
QDialog(parent),
ui( new Ui::Login)
{

ui->setupUi( this );
ui->cBox_account->setFixedHeight(26);
ui->lineEdit_passwd->setFixedHeight(26);
    UserInfoStu ::userlevel= "0" ;
init();
}

Login ::~Login()
{
    delete ui;
}
void Login ::init()
{
setWindowTitle( "Journal Management System" );
setWindowIcon(QIcon( ":/image/top_img1.png" )); //Set application icon

configWindow(); //UI interface settings to border, minimize, maximize button
init_sql(); //Initialize the interface password, the initial value of the account
}

void Login ::configWindow()
{
```

```
QString top_img_path= ":/image/top_img1.png" ;
QImage top_img;
top_img.load(top_img_path);;

}
void Login ::init_sql()
{
    MyDatabase ::getInstance().open();
QSqlQuery q;

    //Create a table named userInfo in the order: Username Password Permissions
QString sql_create_table = "CREATE TABLE userInfo (userid int PRIMARY KEY, name VARCHAR,
passwd VARCHAR, level VARCHAR)" ;
q. prepare(sql_create_table);
    if (!q.exec())
{
qDebug()<< "creater table error" ;
}
q.exec( "insert into userInfo values (1001,'admin','123456','0')" ); //0 means
administrator
q.exec( "insert into userInfo values (1002,'jym','123456','1')" ); //1 means normal user
q.exec( "select * from userInfo" );

    while (q. next())
{
QString userID = q.value(0).toString();
ui->cBox_account->addItem(userID);
}
ui->cBox_account->setCurrentIndex(0);



}

void Login ::on_btn_login_clicked()
{
    if (ui->cBox_account->currentText().isEmpty() ||
ui->lineEdit_passwd->text().isEmpty()){
QMessageBox::warning( this ,tr( "Warning" ),tr( "Please enter your username and password!" ),
"OK" );
}
    else
{
        int is_use_exist_flag = 0; // determine whether the user exists
```

```
        int is_use_nampwd_check_flag = 0; //Check whether the username and password match
get_user_info();

QSqlQuery query;
qDebug() << "database open success login!" ;
query.exec( "select * from userInfo" );
        while (query. next())
{
QString userID = query. value(0). toString();
QString passwd = query.value(2).toString();

            if (userID == UserInfoStu::userID){
is_use_exist_flag = true ; //user exists
                if (passwd == UserInfoStu::passwd){
is_use_nampwd_check_flag = true ; //username and password match
                    this- >accept();

QDateTime current_date_time = QDateTime::currentDateTime();
QString current_date = current_date_time.toString( "yyyy-MM-dd hh:mm:ss" );
QString sql_log_table = "CREATE TABLE LOGSINFO (OPERTIME VARCHAR PRIMARY KEY,OPER
VARCHAR,OPERCONTENT VARCHAR,OPERESULT VARCHAR,OPEROTHER VARCHAR)" ;
query. prepare(sql_log_table);
query.exec();
query. prepare( "insert into LOGSINFO values(?,?,?,?,?)" );
query.bindValue(0, current_date);
query.bindValue(1, userID);
query.bindValue(2, "login" );
query.bindValue(3, "Successful login" );
query. bindValue(4, "" );
                    if (!query.exec())
{
qDebug()<<query.lastError()<< "inset LOGSINFO error" ;
                        return ;
}

}
}
}

        if (is_use_exist_flag == false )
{
QMessageBox::information( this ,tr( "Prompt" ),tr( "User does not exist!" ), "OK" );
}
        else
{
```

```
            if (is_use_nampwd_check_flag == false )
{
QMessageBox::warning( this ,tr( "Warning" ),tr( "User password error!" ), "OK" );
}
            return ;
}
MyDatabase::getInstance().close();
}
}
void Login::get_user_info()
{
UserInfoStu::userName. clear();
UserInfoStu::userID = ui->cBox_account->currentText();
UserInfoStu::passwd. clear();
UserInfoStu::passwd = ui->lineEdit_passwd->text();
MyDatabase::getInstance().open();
QSqlQuery query;
QString select_sql;
select_sql= "select * from userInfo where userid =?" ;
query. prepare(select_sql);

query.addBindValue(UserInfoStu::userID );
    if (!query.exec())
{
        return ;
}
QString strLevel;
QString strUserName;
    while (query. next())
{
strLevel = query. value(3). toString();

}
    // QString sql_create_table = "CREATE TABLE userInfo (userid int PRIMARY KEY,name
VARCHAR,passwd VARCHAR, level VARCHAR)";
    UserInfoStu ::userlevel. clear();
    UserInfoStu ::userlevel = strLevel;

    UserInfoStu ::passwd. clear();
    UserInfoStu ::passwd = ui->lineEdit_passwd->text();
}

void Login::on_pushButton_3_clicked()
{
close();
```

```
}
```

【’ Add ’ operation module】

```
#include "insertgoods.h"
#include "ui_insertgoods.h"
#include "api/mydatabase.h"
#include <QSqlQuery>
#include <QDebug>
#include <QSqlError>
#include <QDateTime>
#include <QMessageBox>
#include <QFileDialog>
InsertGoods ::InsertGoods(QWidget * parent ):
QDialog(parent),
ui( new Ui::InsertGoods)
{
ui->setupUi( this );
    this ->setWindowTitle( "Journal Information" );
ui->comboBox->addItem( "Academic Journal" );
ui->comboBox->addItem( "General Journal" );
ui->comboBox->addItem( "Industry Journal" );
ui->comboBox->addItem( "Search Journal" );
ui->comboBox->setCurrentIndex(0);
connect(ui->comboBox,SIGNAL(currentIndexChanged( int )), this ,SLOT(comboxchange())); //
setWindowIcon(QIcon( ":/image/top_img1.png" )); //Set application icon


}

InsertGoods ::~InsertGoods()
{
    delete ui;
}
void InsertGoods ::comboxchange()
{

}

void InsertGoods ::setVisibleInsert( bool isDis )
{
```

```
    if ( isDis == true )
{
ui->pushButton->setVisible( true );
ui->pushButton_3->setVisible( false );
}
    else
{
ui->pushButton->setVisible( false );
ui->pushButton_3->setVisible( true );
}
}
void InsertGoods ::setVisibleChange( bool isDis )
{
    if ( isDis == true )
{
ui->pushButton_3->setVisible( true );
ui->pushButton->setVisible( false );
}
    else
{
ui->pushButton->setVisible( true );
ui->pushButton_3->setVisible( false );
}
}

void InsertGoods ::on_pushButton_clicked()
{
QString strXSBH = ui->lineEdit->text(); //
QString strSSBH = ui->lineEdit_2->text(); //
QString strNAME = ui->lineEdit_3->text(); //
QString strSex = ui->comboBox->currentText(); //
QString strYXBH = ui->lineEdit_4->text(); //
QString strTEL = ui->lineEdit_5->text(); //
QString strSFZ = ui->lineEdit_6->text(); //
QString strContent = ui->lineEdit_7->text(); //
    if (strXSBH== "" )
{
QMessageBox::warning( this ,tr( "Warning" ),tr( "Please enter the journal number (CN
number)!" ), "OK" );
        return ;
}
QString sql_task_table = "CREATE TABLE QKINFO (XSBH VARCHAR PRIMARY KEY,SSBH VARCHAR, NAME
VARCHAR,SEX VARCHAR,YXBH VARCHAR,TELPHONE VARCHAR,SFZH VARCHAR,OPER VARCHAR,OPERTIME
VARCHAR,CONTENT VARCHAR)" ;
    MyDatabase ::getInstance().open();
```

```
QSqlQuery q;
q. prepare(sql_task_table);
    if (!q.exec())
{

}
q. prepare( "insert into QKINFO values(?,?,?,?,?,?,?,?,?)" ) ;
q. bindValue(0, strXSBH);
q. bindValue(1, strSSBH);
q. bindValue(2, strNAME);
q. bindValue(3, strSex);
q. bindValue(4, strYXBH);
q. bindValue(5, strTEL);
q. bindValue(6, strSFZ);
q. bindValue(7, UserInfoStu :: userID );

QDateTime oper_date_time = QDateTime::currentDateTime();
QString stroper_date_time = oper_date_time.toString( "yyyy-MM-dd hh:mm:ss" );
q. bindValue(8, stroper_date_time);
q. bindValue(9, strContent);
    bool success=q.exec();
    if (!success)
{
qDebug()<<q. lastError();
QMessageBox::warning( this ,tr( "Warning" ),tr( "The journal number is unique, please enter
a different journal number" ), "OK" );
        return ;
}
QString sql_log_table = "CREATE TABLE LOGSINFO (OPERTIME VARCHAR PRIMARY KEY,OPER
VARCHAR,OPERCONTENT VARCHAR,OPERESULT VARCHAR,OPEROTHER VARCHAR)" ;
q. prepare(sql_log_table);
q. exec();
q. prepare( "insert into LOGSINFO values(?,?,?,?,?)" );
q. bindValue(0, stroper_date_time);
q.bindValue(1, UserInfoStu ::userID);
strContent = "Add journal information with journal number " +strXSBH+ "" ; //
q.bindValue(3, "Added successfully" );
q. bindValue(4, "" );
    if (!q.exec())
{
        return ;
}
QMessageBox::information( this ,tr( "Prompt" ),tr( "Added successfully!" ), "OK" );

emit changeLogListView();
```

```
emit changeListView();
    this- >close();
emit changeLogListView();
emit changeListView();
    this- >close();
    // }
}

void InsertGoods::on_pushButton_2_clicked()
{
    this- >close();
}




void InsertGoods::on_pushButton_3_clicked()
{
QString strXSBH = ui->lineEdit->text(); //
QString strSSBH = ui->lineEdit_2->text(); //
QString strNAME = ui->lineEdit_3->text(); //
QString strSex = ui->comboBox->currentText(); //
QString strYXBH = ui->lineEdit_4->text(); //
QString strTEL = ui->lineEdit_5->text(); //
QString strSFZ = ui->lineEdit_6->text(); //
QString strContent = ui->lineEdit_7->text(); //Remarks
    bool isOpen = MyDatabase ::getInstance().open();
QSqlQuery q;
QString update_sql= "update QKINFO set SSBH=?, NAME = ?, SEX = ?, YXBH = ?, TELPHONE = ?,
SFZH = ?, CONTENT = ? where XSBH=?" ;

q. prepare(update_sql);
q. bindValue(0, strSSBH);
q. bindValue(1, strNAME);
q. bindValue(2, strSex);
q. bindValue(3, strYXBH);
q. bindValue(4, strTEL);
q. bindValue(5, strSFZ);

q. bindValue(6, strContent);
q.bindValue(7,strXSBH);

    bool success=q.exec();
    if (!success)
{
```

```
qDebug()<<q. lastError();
QMessageBox::warning( this ,tr( "Warning" ),tr( "Failed to change journal information,
please confirm whether there is this journal" ), "OK" );
        return ;
}
QString sql_log_table = "CREATE TABLE LOGSINFO (OPERTIME VARCHAR PRIMARY KEY,OPER
VARCHAR,OPERCONTENT VARCHAR,OPERESULT VARCHAR,OPEROTHER VARCHAR)" ;
q. prepare(sql_log_table);
q. exec();
QDateTime oper_date_time = QDateTime::currentDateTime();
QString stroper_date_time = oper_date_time.toString( "yyyy-MM-dd hh:mm:ss" );
q. prepare( "insert into LOGSINFO values(?,?,?,?,?)" );
q. bindValue(0, stroper_date_time);
q.bindValue(1,UserInfoStu::userID);
strContent = "Change the journal information with the journal number " +strXSBH+ "" ; //
q. bindValue(2, strContent);
q.bindValue(3, "Change journal information successfully" );
q. bindValue(4, "" );
    if (!q.exec())
{
        return ;
}
emit changeLogListView();
emit changeListView();
    this- >close();
}
```

【' Delete ' operation module】

```
#include "deleteddialog.h"
#include "ui_deleteddialog.h"
#include "api/mydatabase.h"
#include <QSqlQuery>
#include <QDebug>
#include <QSqlError>
#include <QDateTime>
#include <QMessageBox>
deleteDialog ::deleteDialog(QWidget * parent ) :
QDialog(parent),
ui( new Ui::deleteDialog)
{
ui->setupUi( this );
    this ->setWindowTitle( "Delete journal information" );
```

```
}

deleteDialog ::~deleteDialog()
{
    delete ui;
}


void deleteDialog ::on_pushButton_clicked()
{
QString str = ui->lineEdit->text();
    if (str== "" )
{
QMessageBox::warning( this ,tr( "Warning" ),tr( "Please enter the journal number (CN number)!" ), "OK" );
}
    MyDatabase ::getInstance().open();
QSqlQuery query;
    // QString sql_task_table = "CREATE TABLE QKINFO (XSBH VARCHAR PRIMARY KEY, SSBH VARCHAR, NAME VARCHAR, SEX VARCHAR, YXBH VARCHAR, TELPHONE VARCHAR, SFZH VARCHAR, OPER VARCHAR, OPERTIME VARCHAR, CONTENT VARCHAR)";
QString select_sql= "select * from QKINFO where XSBH =?" ;
query. prepare(select_sql);
query. addBindValue(str);
    bool hasData = false ;
    if (!query.exec())
{
        return ;
}
    while (query. next())
{
hasData = true ;
}
    if (!hasData)
{
QMessageBox::warning( this ,tr( "Warning" ),tr( "No information about this journal!" ), "OK" );
        return ;
}
    if (QMessageBox::No==QMessageBox::information(NULL, "Prompt" , "Cannot be restored after deletion, are you sure to delete?" , QMessageBox::Yes | QMessageBox::No, QMessageBox::No))
{
        return ;
}
```

```
QString delete_sql= "delete from QKINFO where XSBH =?" ;
query. prepare(delete_sql);
query. addBindValue(str);
    if (!query.exec())
{
qDebug()<<query.lastError()<< "delete GOODSINFO error" ;
}
QString sql_log_table = "CREATE TABLE LOGSINFO (OPERTIME VARCHAR PRIMARY KEY,OPER
VARCHAR,OPERCONTENT VARCHAR,OPERESULT VARCHAR,OPEROTHER VARCHAR)" ;
query. prepare(sql_log_table);
query.exec();
query. prepare( "insert into LOGSINFO values(?,?,?,?,?)" );
QDateTime oper_date_time = QDateTime::currentDateTime();
QString stroper_date_time = oper_date_time.toString( "yyyy-MM-dd hh:mm:ss" );
query.bindValue(0, stroper_date_time);
query.bindValue(1,UserInfoStu::userID);
QString strContent = "The periodical information whose periodical number is " + str + " is
deleted by " +UserInfoStu::userID + "" ;
query.bindValue(2,strContent);
query.bindValue(3, "Deleted successfully" );
query. bindValue(4, "" );
    if (!query.exec())
{
        return ;
}
QMessageBox::information( this ,tr( "Prompt" ),tr( "Delete journal information
successfully!" ), "OK" );
emit changeLogListView();
emit changeListView();
      this- >close();
}

void deleteDialog::on_pushButton_2_clicked()
{
    this- >close();
}
```