Лекція 6

ПРОГРАМУВАННЯ БАЗ ДАНИХ ІЗ ЗАСТОСУВАННЯМ QT

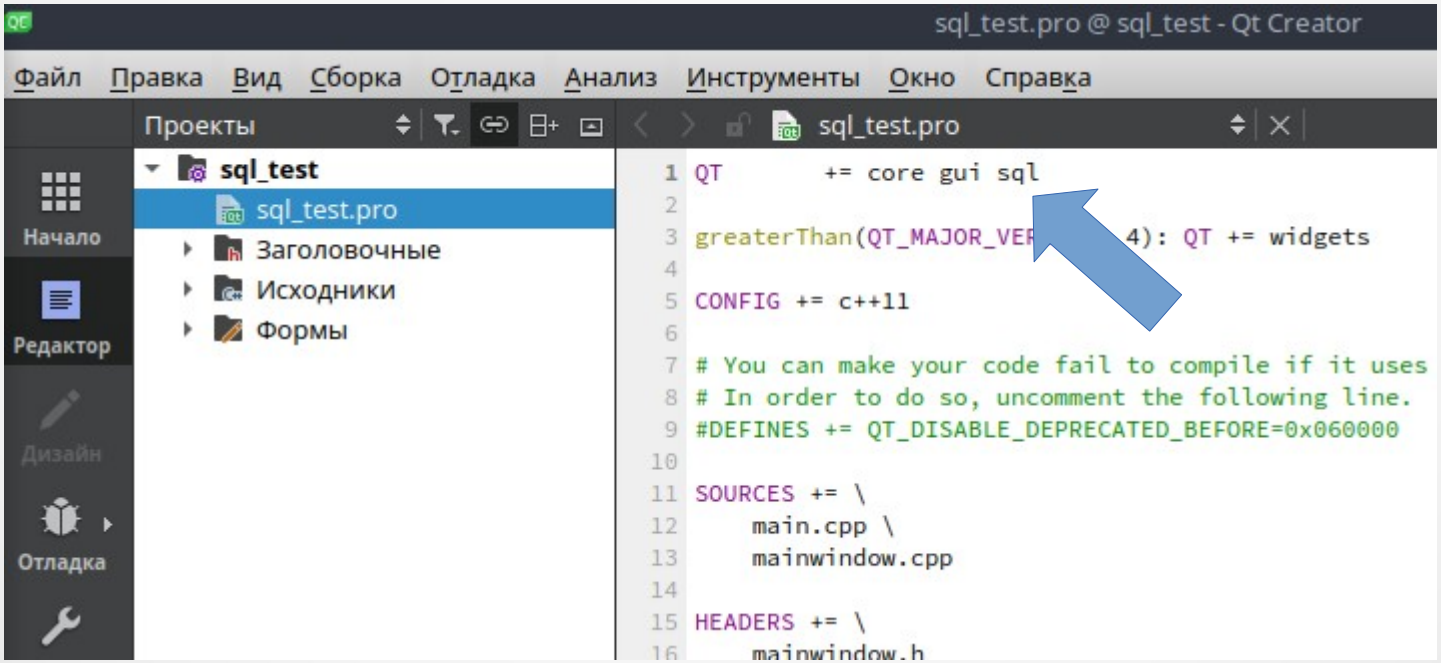
Лекція 6. Програмування баз даних із застосуванням Qt

План

1. Модуль QtSql.
2. З’єднання з БД.
3. Виконання SQL-запитів.
4. GUI для роботи з базами даних.

1. Модуль QtSql

Фреймворк Qt має всі необхідні засоби для роботи з базами даних (БД). Для цього застосовується спеціальний модуль **QtSql**, підключення якого виконується додаванням в файл проєкту Qt (*.pro) відповідного параметру:



Для роботи з класами, що забезпечують роботу з БД, необхідно підключити заголовний файл **QtSql.**

1. Модуль QtSql

Класи модуля QtSql поділяються на три рівні:
1) рівень драйверів;
2) програмний рівень;
3) рівень інтерфейсу користувача.

Класи **першого рівня** реалізують фізичний доступ до БД. Це, наприклад, **QSqlDriver**, **QSqlDriverCreatorBase**, **QSqlDriverPlugin**, **QSqlResult** та інші. Вони в основному реалізують взаємодію Qt з певними типами систем управління базами даних (СУБД).

Класи **другого рівня** (**QSqlDatabase**, **QSqlQuery**, **QSqlError**, **QSqlField**, **QSqlIndex** та **QSqlRecord**) реалізують універсальний програмний інтерфейс взаємодії з БД.

Третій рівень надає моделі для відображення результатів запитів. До цих класів належать: **QSqlQueryModel**, **QSqlTableModel** та **QSqlRelationalTableModel**.

2. З’єднання з БД

Для з’єднання з БД необхідно активізувати відповідний драйвер. З цією метою застосовується клас **QSqlDatabase**, статичний метод **addDatabase()** якого приймає параметр – строку, що містить назву СУБД, яка буде використовуватися для підключення до конкретної БД.

Для підключення до БД також можуть знадобитися чотири наступні параметри:

- **ім’я бази даних** – визначається параметром методу **setDatabaseName()**;
- **логін користувача** – передається до методу **setUserName()**;
- **мережеве ім’я комп’ютера**, на якому розміщена база даних (визначається параметром методу **setHostName()**);
- **пароль** – передається як параметр методу **setPassword()**.

2. З’єднання з БД

Приклад підключення до БД test.db типу **SQLite**.

```
#include <QtSql>


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

db.setHostName("localhost");
db.setDatabaseName("test.db");
db.setUserName("user");
db.setPassword("user");
bool ok = db.open();
//...
```

2. З’єднання з БД

Безпосередньо з’єднання з БД здійснюється методом **open()** класу QSqlDatabase. У разі, якщо при відкритті БД виникне помилка, інформацію про неї можна отримати за допомогою методу **lastError()**, який повертає об’єкт класу **QSqlError**.

Наприклад:

```
// ...
if (!db.open())
{
    qDebug() << "Error opening database: " << db.lastError();
    // ...
}
// ...
```

3. Виконання SQL-запитів

Виконання SQL-запитів до БД після встановлення з нею з’єднання можна здійснювати за допомогою класу **QSqlQuery**.

Команди SQL виконуються шляхом передачі строки-параметру з SQL-командою в конструктор класу QSqlQuery або його метод **exec()**. В разі застосування конструктора, запуск команди буде здійснюватись автоматично при створенні об'єкта.

```
QSqlQuery query;
QString sql = "CREATE TABLE phones "
                  "(id INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,”
                  "owner TEXT, phone TEXT)";

if (!query.exec(sql))
{
    qDebug() << "Error creating table!";
    db.close();
    return;
}
```

3. Виконання SQL-запитів

Ще один приклад показує, як додавати дані до таблиць БД із застосуванням класу **QSqlQuery**.

```
// ...
QSqlQuery query1(QString("INSERT INTO phones (owner, phone) "
                         "VALUES('%1', '%2')")
                         .arg("J. Kennedy").arg("22-22-22"));
QSqlQuery query2(QString("INSERT INTO phones (owner, phone) "
                         "VALUES('%1', '%2')")
                         .arg("D. Trump").arg("33-22-44"));

// ...
```

3. Виконання SQL-запитів

Якщо SQL-запит повертає дані у вигляді таблиці (наприклад, команда **SELECT**), то клас QSqlQuery дозволяє виконувати навігацію по ній. Для цього в ньому передбачені наступні методи:

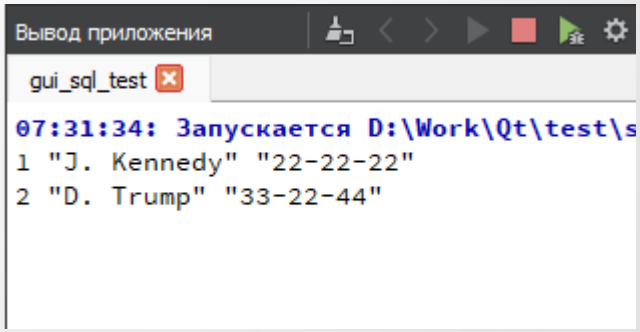
- **next()** – переміщення на наступний рядок даних;
- **previous()** – переміщення на попередній рядок даних;
- **first()**, **last()** – переміщення до першого (останнього) рядка даних;
- **seek()** – робить поточним рядок даних за вказаним цілочисельним індексом.
- **size()** – повертає кількість рядків даних в таблиці.

3. Виконання SQL-запитів

Наприклад, для виведення на екран змісту таблиці phones у алфавітному порядку за іменами абонентів слід написати наступний код:

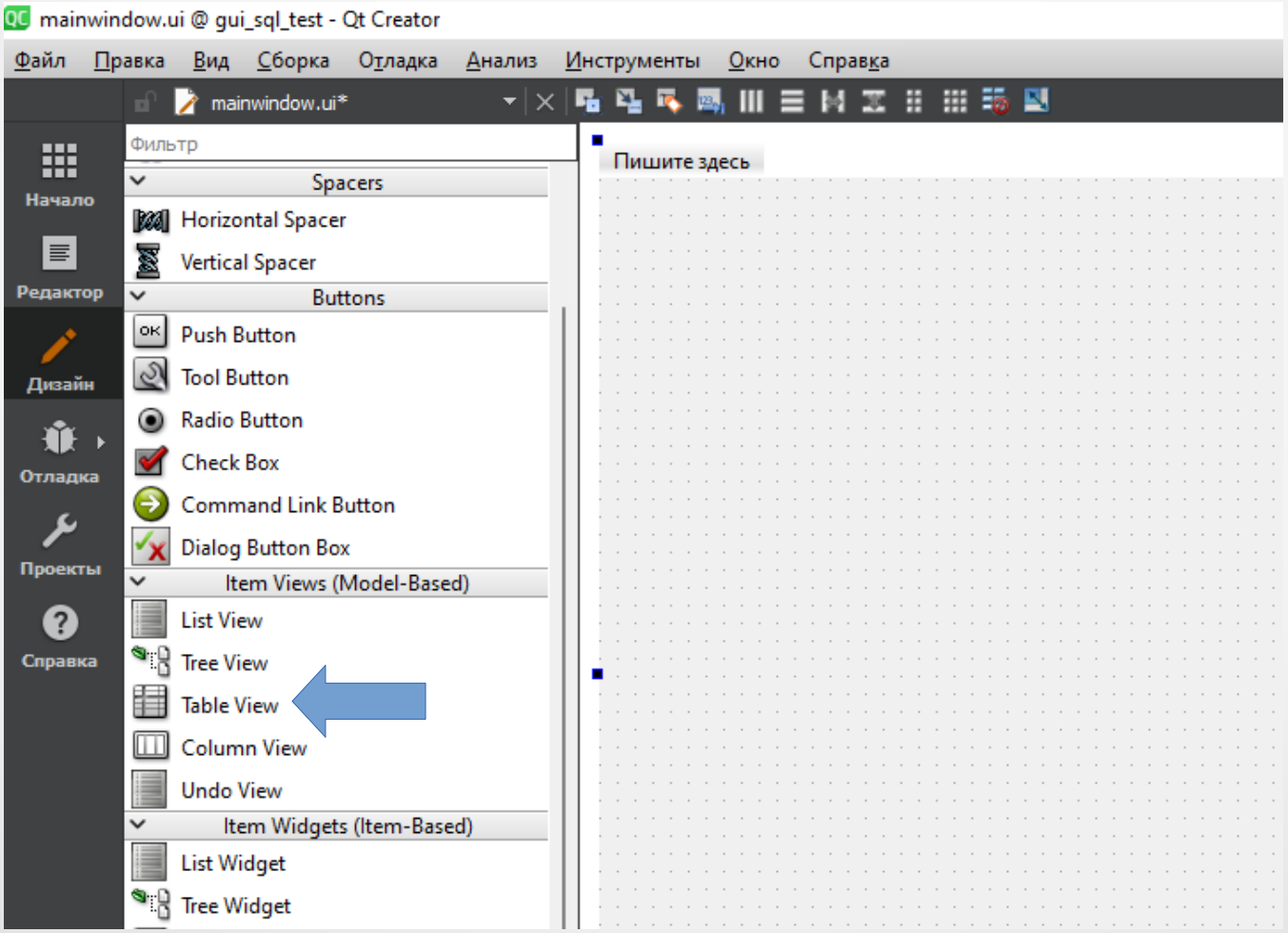
```
QSqlQuery query;
if (!query.exec(QString("SELECT * FROM phones")))
{
    qDebug() << "Error accessing to table! " << db.lastError();
    db.close();
    return;
}
while (query.next())
{
    int id = query.value(0).toInt();
    QString owner = query.value(1).toString(),
            phone = query.value(2).toString();

    qDebug() << id << owner << phone;
}
```



4. GUI для роботи з базами даних

Для реалізації графічного інтерфейсу користувача (GUI) при роботи з БД Qt надає декілька віджетів. Найбільш простим у використанні є **TableView** (реалізується класом **QTableView**).



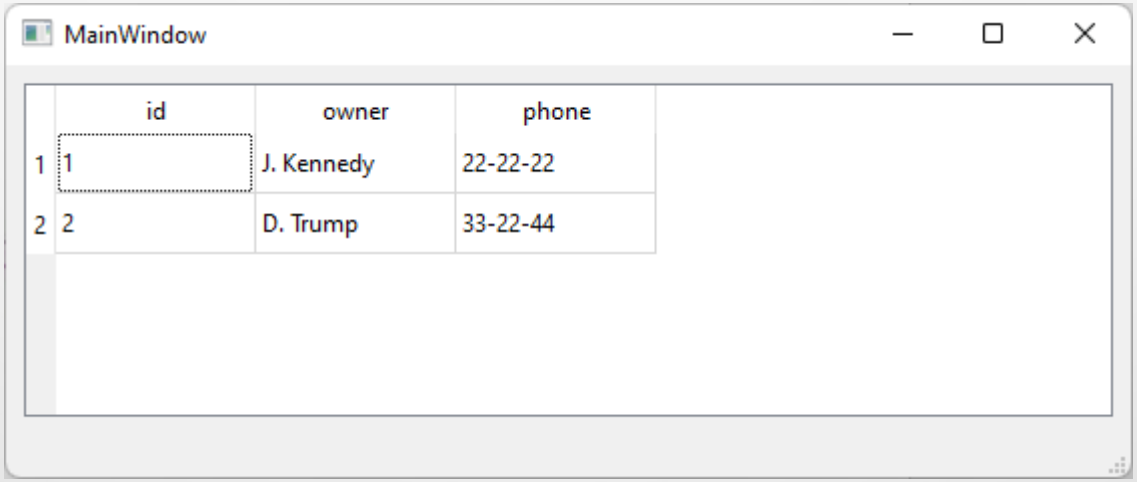
4. GUI для роботи з базами даних

Для його використання слід додати до форми відповідний компонент і налаштувати його, наприклад, наступним чином:

```
// ...
QSqlTableModel *model = new QSqlTableModel(this);

model->setTable("phones");
model->select();

ui->tableView->setModel(model);
// ...
```



4. GUI для роботи з базами даних

Клас **QSqlTableModel** реалізує інтерфейс для взаємодії (читання, запису, ...) з даними, що розташовані в окремій таблиці БД.

Він підтримує три стандартні стратегії редагування, які встановлюються за допомогою методу **setEditStrategy()**:

- **onRowChange** – виконує запис даних, як тільки користувач перейде до іншого рядку таблиці;
- **onFieldChange** – записує дані після того, як користувач перейде до іншої комірки таблиці;
- **onManualSubmit** – записує дані тільки при викликанні слоту **submitAl()** (в разі, якщо викликається слот **revertAll()**, всі зміни в поточній таблиці відкидаються).

4. GUI для роботи з базами даних

Для відображення даних – результату виконання запиту SELECT застосовується клас **QSqlQueryModel**. Наприклад, для виведення на екран лише ПІБ абонентів у зворотному порядку можна написати наступний код:

```
// ...
QSqlQueryModel *model = new QSqlQueryModel(this);

model->setQuery("SELECT owner FROM phones ORDER BY owner DESC");
ui->tableView->setModel(model);
// ...
```