Лабораторные работы по курсу
Базы данных

Лабораторная работа 8
«Разработка программы для работы с базой данных»

Москва, 2023

Оглавление

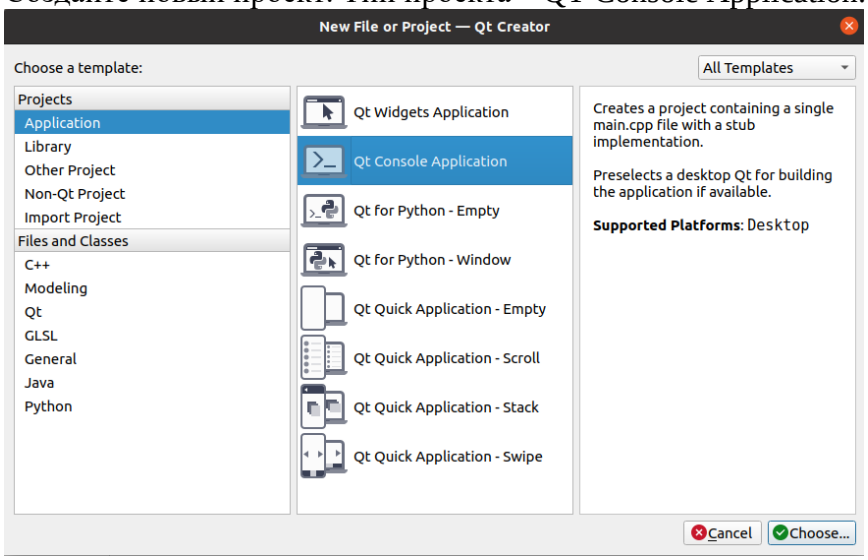
Теоретическая часть

В данной лабораторной работе будет рассмотрено создание программного интерфейса для работы с базой данных. Программа будет разработана на языке C++ в среде QT Creator.

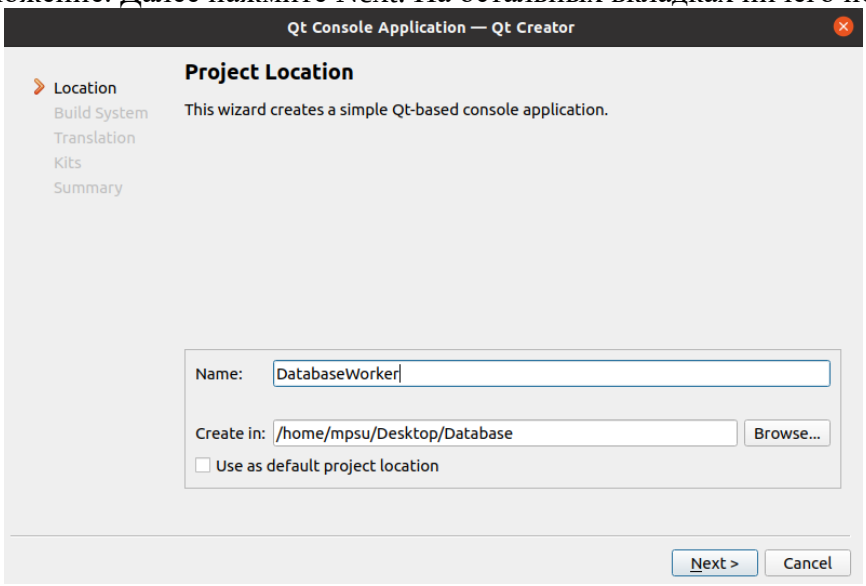
1.1. Создание проекта

Запустите программу QT Creator.

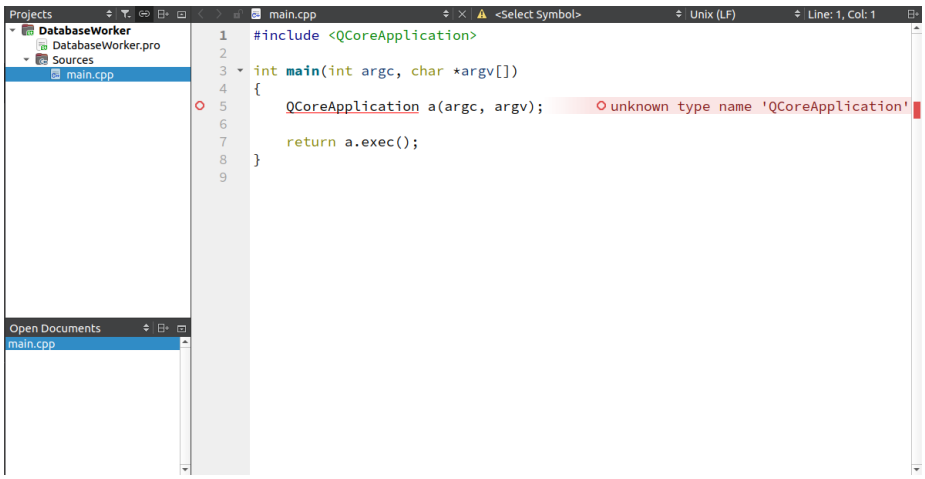
Создайте новый проект. Тип проекта – QT Console Application.



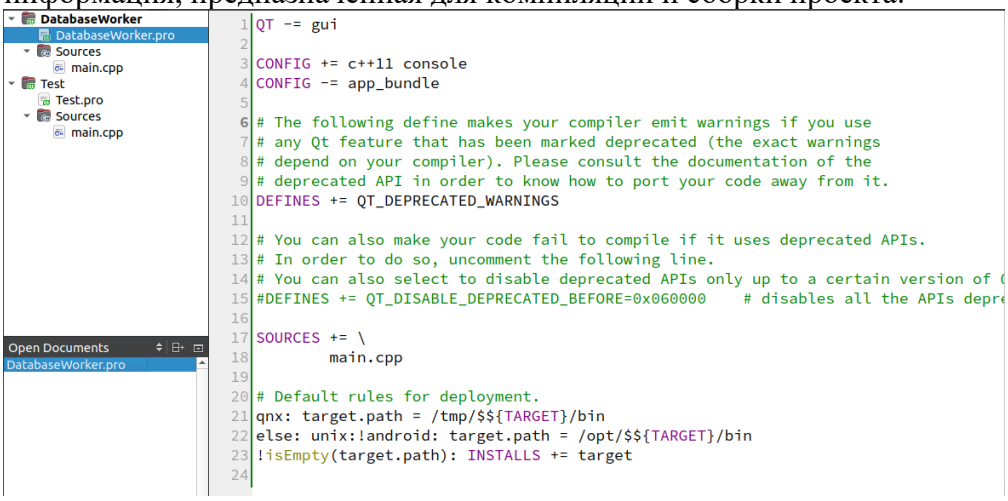
Заполните основную информацию о создаваемом проекте – название, расположение. Далее нажмите Next. На остальных вкладках ничего не изменяйте.



Созданный проект будет выглядеть следующим образом:



Откройте файл проекта с расширением .pro. В нем содержится основная информация, предназначенная для компиляции и сборки проекта.



Измените его содержимое на следующее:

```
QT -= gui

CONFIG += c++11 console
CONFIG -= app_bundle

DEFINES += QT_DEPRECATED_WARNINGS

QT += sql
QT += core
LIBS += -L/usr/lib/x86_64-linux-gnu/qt5/plugins/sqldrivers -lqsqlpsql
CONFIG += c++20
SOURCES += \
        main.cpp

qnx: target.path = /tmp/$${TARGET}/bin
else: unix:!android: target.path = /opt/$${TARGET}/bin
!isEmpty(target.path): INSTALLS += target
```

Измените код программы main.cpp на следующий:

```
#include <QCoreApplication>
#include <QSqlDatabase>
```

```
#include <QSqlError>
#include <QSqlQuery>
#include <iostream>
#include <QDebug>

int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);
    QSqlDatabase db = QSqlDatabase::addDatabase("QPSQL");
    db.setHostName("localhost"); // IP-адрес вашей виртуальной машины Ubuntu
    db.setPort(5432); // порт по умолчанию для PostgreSQL
    db.setDatabaseName("students"); // имя вашей базы данных
    db.setUserName("postgres"); // ваше имя пользователя для базы данных (см.
на виртуалке)
    db.setPassword("123456"); // ваш пароль для базы данных
    if (!db.open())
        qDebug() <<  db.lastError().text();
    else
        qDebug() << "All is good!\n";

    QSqlQuery query;
    query.exec("select * from Студент");
    while (query.next()) {
         int ID = query.value(0).toInt();
         QString surname = query.value(1).toString();
         QString name = query.value(2).toString();
         qDebug() << surname << name << ID;
       }

    db.close();

    return a.exec();
}
```

Рассмотрим программный код более подробно.

```
QSqlDatabase db = QSqlDatabase::addDatabase("QPSQL");
    db.setHostName("localhost"); // IP-адрес вашей виртуальной машины Ubuntu
    db.setPort(5432); // порт по умолчанию для PostgreSQL
    db.setDatabaseName("students"); // имя вашей базы данных
    db.setUserName("postgres"); // ваше имя пользователя для базы данных
    db.setPassword("123456"); // ваш пароль для базы данных
```

В данном участке кода происходит подключение к запущенному серверу PostgreSQL. Подключение происходит по локальному IP адресу и порту 5432. База данных – students, пользователь – postgres, пароль для подключения – 123456.

В случае успешного подключения происходит вывод сообщения «All is good!».

Создание запросов к базе данных происходит с помощью создания очередей запросов (query). Метод query.exec отправляет на сервер и выполняет SQL запрос. Результат запроса возвращается в объект query.value.

```
QSqlQuery query;
query.exec("select * from Студент");
while (query.next()) {
    int ID = query.value(0).toInt();
    QString surname = query.value(1).toString();
    QString name = query.value(2).toString();
    qDebug() << surname << name << ID;
}
```

В случае успешного подключения и выполнения запроса в консоль будет выведена информация о студентах из учебной базы данных.

1.2.SQL-инъекции

Добавим в программу функцию, позволяющую выводить на экран ФИ студента, по его номеру студенческого билета.

```
void FindStudentsByID(std::string id)
{
    QSqlQuery query;
    std::string QueryText;
    QueryText = "SELECT * FROM Студент WHERE \"Номер студенческого билета\"=";
    QueryText.append(id);
    query.exec(QString::fromStdString(QueryText));
    while (query.next()) {
         int ID = query.value(0).toInt();
         QString surname = query.value(1).toString();
         QString name = query.value(2).toString();
         qDebug() << surname << name << ID;
       }
}
```

При корректном вызове функции программа выберет из базы данных студента и выведет его фамилию и имя на экран.

```
FindStudents("882599");
```

Однако, злоумышленники могут вместо требуемых значений могут заведомо вводить ложные, чтобы получить доступ к закрытым данным.

Вызовите функцию со следующими параметрами. Объясните полученный результат.

FindStudents("1 OR 1=1");

Помимо несанкционированного доступа к данным, возможна и порча данных внутри БД.

Вызовите функцию со следующими параметрами. Объясните полученный результат.

FindStudents("1 OR 1=1; DROP TABLE IF EXISTS \"Студент\" ");

Аналогично возможно действовать в случае, когда вводимое значение – строка. Рассмотрим пример функции, которая производит поиск студентов по заданному имени.

```
void FindStudentsByName(std::string id)
{
    QSqlQuery query;
    std::string QueryText;
    QueryText = "SELECT * FROM Студент WHERE \"Имя\"='";
    QueryText.append(id);
    QueryText.append("'");
    query.exec(QString::fromStdString(QueryText));
    while (query.next()) {
         int ID = query.value(0).toInt();
         QString surname = query.value(1).toString();
         QString name = query.value(2).toString();
         qDebug() << surname << name << ID;
       }
}
```

В данном случае, чтобы запрос корректно сработал, к введенной строке добавляются кавычки.

Вызовите функцию со следующими параметрами. Объясните полученный результат.

FindStudentsByName("Сергей' OR '1'='1");

1.3. Защита от SQL-инъекций

Защититься от SQL-инъекции из предыдущего примера очень просто. Функция *FindStudentsByID* предполагает поиск студента по номеру его студенческого билета. Данное значение является числовым. Поэтому добавим простейшую проверку на то, является ли введенное значение числом.

```
bool isNumeric(std::string const &str)
{
    return !str.empty() && std::all_of(str.begin(), str.end(), ::isdigit);
}

void FindStudentsByID(std::string id)
{
    if(isNumeric(id) == 0)
        return;
    QSqlQuery query;
    ///
}
```

В случае, если введенное ID является строкой, функция *isNumeric* вернет значение 0 и не *FindStudentsByID* завершит свою работу.

Более интересным является случай защиты при вводе строки. Легко заметить, что предыдущий способ в данном случае не сработает. Чтобы совершить атаку, вводимая строка должна содержать одинарные кавычки. Первая кавычка закроет вводимую строку, далее произойдет ввод вредоносного кода, который также должен завершиться выражением, требующим закрывающую кавычку. В примере ниже синим цветом выделен код функции, а зеленым – код, вводимый злоумышленником.

```
SELECT * FROM Студент WHERE "Имя"='Сергей' OR '1'='1'
```

Защититься от такой инъекции можно двумя способами:

1. Изменение кавычек на апострофы.
   Если для пользователя не так важен вводимый символ, то возможно изменить символ ' на `. Таким образом вредоносный код будет принят как одна длинная строка и добавлен в базу данных.
   Приведем функцию, изменяющую символ апострофа.

```
void CorrectApostrof(std::string &query)
{
    size_t pos;
    while ((pos = query.find('\'')) != std::string::npos) {
            query.replace(pos, 1, "`");
        }
}
```

2. Экранирование кавычек
   Если для пользователя важен именно символ кавычки, то его для ввода в базу данных необходимо экранировать. Напомним, что кавычки в PostgreSQL экранируются повторением данного символа. Например, если мы хотим внести в таблицу строку имя John O'Brien, то запрос на добавление будет выглядеть следующим образом:

```
INSERT INTO ApostropheTest(Name, Surname)
VALUES   ('John', 'O''Brien')
```

	name character varying (20)	surname character varying (20)
1	John	O'Brien

Самостоятельно разработайте функцию, экранирующую во введенной строке все кавычки.

Практическая часть

Задание 1.

Разработать функцию для реализации любого из ваших запросов к учебной базе данных. Необходимо добавить защиту от SQL-инъекций.

Задание 2.

Разработайте приложение для работы с вашей базой данных.
Требуемые функции:

1. Поле ввода логина/пароля для подключения к базе данных от имени различных пользователей. Типы пользователей – администратор, пользователь.
2. От имени администратора сделать возможность добавлять значения в любую из таблиц.
3. От имени пользователя сделать возможность выполнять любые 5 запросов, из созданных вами ранее.

Список литературы

[1] «Исходный код СУБД postgres,» [В Интернете]. Available: https://github.com/postgres/postgres. [Дата обращения: 30 01 2023].

[2] Документация к PostgreSQL 15.1, 2022.

[3] Е. Рогов, PostgreSQL изнутри, 1-е ред., Москва: ДМК Пресс, 2023, p. 662 .

[4] Б. А. Новиков, Е. А. Горшкова и Н. Г. Графеева, Основы технологии баз данных, 2-е ред., Москва: ДМК пресс, 2020, p. 582.

[5] Е. П. Моргунов, PostgreSQL. Основы языка SQL, 1-е ред., Санкт-Петербург: БХВ-Петербург, 2018, p. 336.