CASE-SWDEP

РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

v2.0

https://github.com/cantario/CASE-SWDEP

Оглавление

Глава 1 Начало работы с CASE-SWDEP

1.1 Введение в CASE-SWDEP

CASE-SWDEP – это система проектирования базы данных, разработанная на основе фреймворка Qt. CASE-SWDEP позволяет представить сущности и атрибуты предметной области в виде концептуальной модели данных и на основе этой модели создать саму базу данных.

CASE-SWDEP позволяет сгенерировать SQL запрос на создание базы данных при поддержке языка описания различных СУБД, посредством подключения соответствующих плагинов по мере необходимости. CASE-SWDEP предоставляет возможность сохранения концептуальной модели в формат CDMOD. Файлы данного формата доступны на чтение и редактирование в любом текстовом редакторе.

1.2 Требования к аппаратному обеспечению

Для CASE-SWDEP требуется 64-битный или 32-битный процессор и не менее 30 мб свободного места на жестком диске. Рекомендуемый объем оперативной памяти – 2гб.

1.3 Требования к программному обеспечению

Для сборки CASE-SWDEP требуется фреймворк Qt версией не ниже 4.11 с основой Qt версии не ниже 5.14 с компилятором GCC разрядности 32 или 64 бит.

Текущая версия CASE-SWDEP поддерживает наиболее распространенные операционные системы:

- Windows 7/8/10 32bit/64bit
- Ubuntu Linux 32bit/64bit

1.4 Доступ к последней версии

Последняя версия программного средства CASE-SWDEP доступна по адресу https://github.com/cantario/CASE-SWDEP/releases

1.5 Доступ к исходным текстам

Последняя версия исходного кода CASE-SWDEP доступна по адресу https://github.com/cantario/CASE-SWDEP

Исходный код доступен для скачивания и компиляции, но он может содержать новые, неизданные функции и ошибки, поскольку находится на стадии разработки. Вся информация, необходимая для успешной сборки проекта находится в следующем пункте данного руководства.

1.6 Сборка программы из исходных текстов

1.6.1 *Требования*

Qt 5.14+

- QtCreator 4.11+
- gcc 7.4+

1.6.2 *Получение исходного кода*

Клонировать исходный код из репозитория:

```
git clone https://github.com/cantario/CASE-SWDEP.git
```

или скачать архив. Архив распаковать в папку CASE-SWDEP.

Рекомендация: полный путь к папке CASE-SWDEP не должен содержать русские символы и пробелы.

1.6.3 *Сборка при помощи QtCreator*

Открыть файл «CASE_SWDEP.pro» в корневой папке проекта при помощи программы QtCreator. Далее нажать кнопку «Собрать» (ctrl + B) для сборки всего проекта, включая все подпроекты плагинов.

При необходимости сборки определенного подпроекта нужно перейти в папку данного подпроекта в окне навигации «Проекты» и выбрать пункт меню «Сборка – Пересобрать подпроект»

1.6.4 *Сборка при помощи Qmake*

Запустить терминал в корневой папке проекта. Набрать команды:

```
qmake [TODO]
```

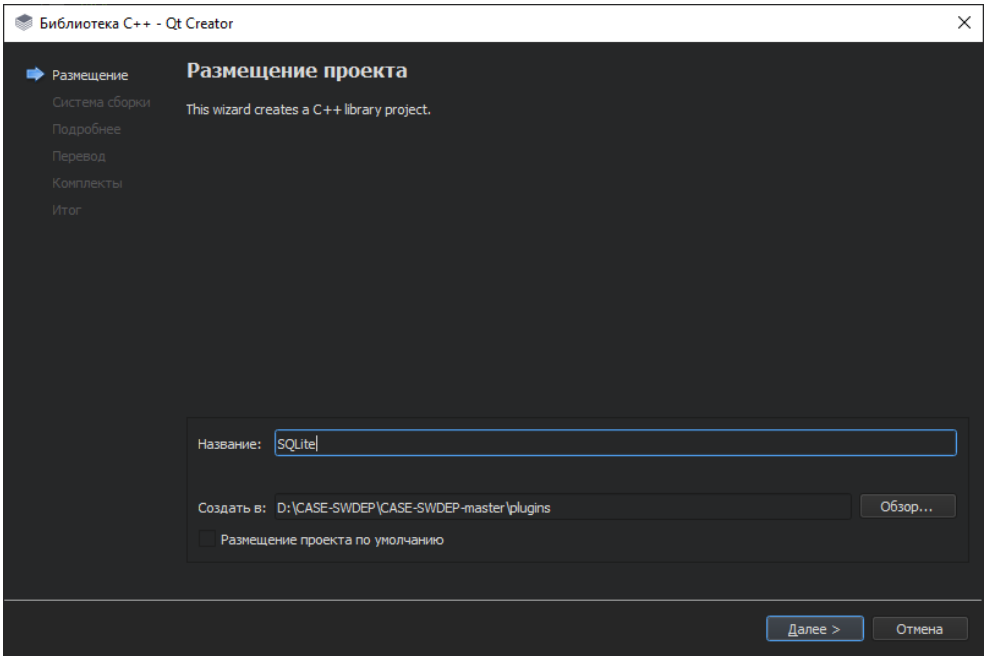
```
make
```

1.7 Запуск программы из архива

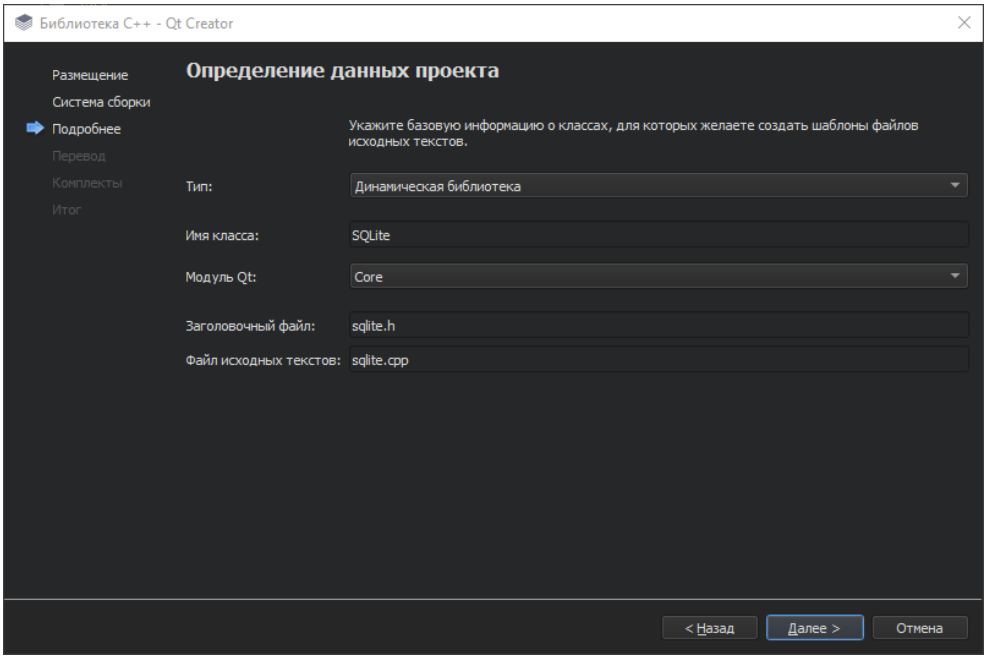
[TODO]

1.8 Процедура создания плагинов

Создание плагинов рассматривается в Qt версии 4.12.4. Чтобы создать плагин, необходимо создать проект «Библиотека C++». Пишем название плагина и выбираем его расположение. На рисунке представлено окно размещения проекта.



Выбираем систему сборки qmake -> тип библиотеки – динамическая библиотека -> модуль Qt – Core. Далее пишем название класса, для удобства назвать так же, как и проект. Информация о классе (см. рисунок ниже).



Выбираем комплект сбоки Desktop Qt 5.15.0 MinGW 64-bit.

После нажатия кнопки «Завершить» появляются три файла: sqlite.h, sqlite_global.h, sqlite.cpp и файл проекта SQLite.pro. Файл sqlite_global.h содержит внешний интерфейс плагина. Удаляем его, так как в ПС был реализован общий интерфейс для всех модулей. Сначала требуется изменить файл проекта. Для того чтобы библиотека могла конвертироваться в файл DLL для ОС Windows и PLUGIN для ОС Linux, необходимо добавить флаг DLLDESTDIR и DESTDIR соответственно и указать путь хранения динамических библиотек. После компиляции проекта готовая библиотека будет создана в указанной папке. Так же следует добавить флаг INCLUDEPATH и указать расположение интерфейса для плагинов.

```
#-------------------------------------------------
#
# Project created by QtCreator 2015-04-20T16:15:39
#
#-------------------------------------------------

QT       -= gui

TARGET = SQLite3_35_5
TEMPLATE = lib

INCLUDEPATH += ../../CASE_SW

SOURCES += \
    sqlite3_35_5.cpp

HEADERS += \
    sqlite3_35_5.h

unix {
    CONFIG += plugin
    DESTDIR = $$PWD/../../build/case/lib/
}

win32 {
    CONFIG += dll
    DLLDESTDIR = $$PWD/../../build/case/lib/
}
```

Далее следует изменить заголовочный файл sqlite.h. Для начала в нём следует подключить файл интерфейса cplugininterface.h, чтобы плагин мог взаимодействовать с интерфейсом. Перед названием класса стоит опция «название класса» +SHARED_EXPORT. Эту опцию нужно убрать, так как она не используется. После этого наследуем класс плагина от класса интерфейса CPluginInterface. Далее внутри класса необходимо установить следующие макросы: Q_OBJECT, Q_PLUGIN_METADATA, Q_INTERFACES. Стоит уточнить параметры двух последних макросов. Полная запись макроса Q_PLUGIN_METADATA выглядит так: Q_PLUGIN_METADATA(IID " CPluginInterface/Plugin/1.0"), где мы указываем внешний интерфейс в котором содержатся метаданные для плагина. В макросе Q_INTERFACES указывается класс внешнего интерфейса в котором присутствует макрос

Q_DECLARE_INTERFACE сообщающий метаобъектной системе Qt об интерфейсе. Запись выглядит следующим образом: Q_INTERFACES(DBPluginInterface).

После объявления макросов следует объявить основные функции для взаимодействия с плагином. В основной набор функций входят такие функции как: QString getVersion() – получение версии плагина; QString getName() – получение имени плагина; QString query(IDataModel *dataModel) override – генерация скрипта. Далее на рисунке приведён листинг заголовочного файла плагина.

```
#ifndef SQLITE3_35_5_H
#define SQLITE3_35_5_H

#include <cplugininterface.h>

class SQLite3_35_5 : public CPluginInterface
{
    Q_OBJECT
    Q_PLUGIN_METADATA(IID "CPluginInterface/Plugin/1.0")
    Q_INTERFACES(CPluginInterface)

public:
    QString name() override;
    QString version() override;
    QString query(IDataModel *dataModel) override;
};


#endif // SQLITE3_35_5_H
```

Реализация вышеперечисленных функций. Функция getVersion() возвращает строчку типа QString с версией плагина.

```
QString SQLite3_35_5::version()
{
    return "3.35.5";
}
```

Функция getName возвращает значение QString с названием плагина.

```
QString SQLite3_35_5::name()
{
    return "SQLite";
}
```

Функция query(IDataModel *dataModel) override принимает аргументом таблицы, хранимые во внутреннем представлении. На выходе функция возвращает строку, содержащую скриптдля данной СУБД.

```
QString SQLite3_35_5::query(IDataModel *dataModel)
{
    _dataModel = dataModel;
    QString script = "";

    foreach (CTable *table, _dataModel->tablesSortedByReference()) {
        _table = (ITable *)table;
        if(_table->foreingRows().size() != 0)
        {
            script += "PRAGMA foreign_keys = ON;\n\n";
            break;
        }
    }

    foreach (CTable *table, _dataModel->tablesSortedByReference()) {
        _table = (ITable *)table;
        QString tableBody = "";
        QString primaryKey = "";

        foreach (CRow *row, _table->rows()) {
            _row = (IRow *)row;
            QString constraint = "";
            if(_row->unique() && !(_row->primaryKey()))
                constraint += " UNIQUE";
            if(_row->notNull() && !(_row->primaryKey()))
                constraint += " NOT NULL";
            if(_row->primaryKey())
                primaryKey += QString("%1, ")
                              .arg(_row->name());
            tableBody += QString("%1 %2%3, ")
                    .arg(_row->name())
                    .arg(_row->typeAsString())
                    .arg(constraint);
        }
```

```
// TABLE | (FOREIGN ROW, ROW)
QMap<QString, QList<QPair<QString, QString>>> fRowsToTables;
foreach (CForeignRow *foreignRow, _table->foreingRows()) {
    _foreignRow = (IForeingRow *)foreignRow;
    tableBody += QString("%1 %2, ")
            .arg(_foreignRow->name())
            .arg(_foreignRow->typeAsString());
    if(_foreignRow->primaryKey())
        primaryKey += QString("%1, ")
                .arg(_foreignRow->name());
    if(fRowsToTables.contains(_foreignRow->tableName()))
    {
        QPair<QString, QString> pair(_foreignRow->name()
                                     ,_foreignRow->tableRowName());
        QList<QPair<QString, QString>> list;
        list = fRowsToTables.value(_foreignRow->tableName());
        list.append(pair);
        fRowsToTables.insert(_foreignRow->tableName(), list);
    }
    else
    {
        QPair<QString, QString> pair(_foreignRow->name()
                                     ,_foreignRow->tableRowName());
        QList<QPair<QString, QString>> list;
        list.append(pair);
        fRowsToTables.insert(_foreignRow->tableName(), list);
    }
}
tableBody.remove(tableBody.size() - 2, 2);
if(primaryKey != "")
{
    primaryKey.remove(primaryKey.size() - 2, 2);
    tableBody += QString(", PRIMARY KEY (%1)")
            .arg(primaryKey);
}
```

```
        if(!fRowsToTables.isEmpty())
        {
            tableBody += ", ";
            for(int i = 0; i < fRowsToTables.size(); i++)
            {
                QString fRowList, tRowList;
                for(int j = 0; j < fRowsToTables.values().at(i).size(); j++)
                {
                    fRowList += fRowsToTables.values().at(i).at(j).first + ", ";
                    tRowList += fRowsToTables.values().at(i).at(j).second + ", ";
                }
                fRowList.remove(fRowList.size() - 2, 2);
                tRowList.remove(tRowList.size() - 2, 2);
                tableBody += QString(" FOREIGN KEY (%3)"
                                     " REFERENCES %1 (%2)"
                                     " ON DELETE RESTRICT"
                                     " ON UPDATE RESTRICT, ")
                        .arg(fRowsToTables.keys().at(i))
                        .arg(tRowList)
                        .arg(fRowList);
            }
            tableBody.remove(tableBody.size() - 2, 2);
        }
        if(tableBody != "")
        {
            script += QString("CREATE TABLE IF NOT EXISTS %1 ("
                      "%2"
                      ");\n\n")
                    .arg(_table->name())
                    .arg(tableBody);
        }
    }
    return script;
}
```

После проведённых действий в папке, указанной в файле проекта в поле DLLDESTDIR и DESTDIR, будет создана динамическая библиотека с именем проекта и расширением “.DLL” для ОС семейства Windows и “.SO” для семейства Linux.

Глава 2 Создание простейшего SQL CREATE запроса

2.1 Пользовательский интерфейс

[TODO]

Главное окно

Окно физической модели

Окно создания SQL запроса

Окно редактирования таблицы

Окно редактирования связи

Окно редактирования уникальных групп

Окно смены таблицы

Окно изменения размера рабочей области

2.2 Создание концептуальной модели базы данных

[TODO]

2.3 Создание физической модели базы данных

[TODO]

2.4 Создание SQL CREATE запроса

[TODO]

2.5 Структура концептуальной модели базы данных в формате CDMOD

[TODO]