МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение
высшего профессионального образования

«Национальный исследовательский ядерный университет «МИФИ»

Факультет кибернетики и информационной безопасности

Кафедра №36

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту на тему:

Разработка учебно-тренировочных средств по системам управления базами данных

Группа К7-361

Студент _____ (подпись) (Воронин Д.Л.) (ФИО)

Руководитель проекта _____ (подпись) (Муравьев С.К.) (ФИО)

Оценка _____

Члены комиссии _____ (подпись) (_____) (ФИО)

_____ (подпись) (_____) (ФИО)

_____ (подпись) (_____) (ФИО)

_____ (подпись) (_____) (ФИО)

Москва 2012г.

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение
высшего профессионального образования

**«Национальный исследовательский ядерный
университет «МИФИ»**

ФАКУЛЬТЕТ КИБЕРНЕТИКИ И ИНФОРМАЦИОННОЙ БЕЗОПАСНОСТИ

КАФЕДРА ИНФОРМАЦИОННЫЕ СИСТЕМЫ И ТЕХНОЛОГИИ

Задание на УИР и КП

Студенту гр. К7-361 Воронину Дмитрию Леонидовичу

Тема УИР и КП

Разработка учебно-тренировочных средств по системам управления базами данных

ЗАДАНИЕ

1. Изучить документацию библиотеки Qt, необходимой для создания GUI-приложения.

2. Разработать структуру базы данных обучающих уроков.

3. Разработать дизайн GUI-приложения.

4. Изучить язык запросов XPath.

5. Изучить модуль библиотеки Qt QXmlPatterners,необходимого для реализации запросов XPath и XQuery.

6. Разработать и реализовать логику работы программы.

Место выполнения УИР и КП ФГУП «ЦНИИ ЭИСУ»

Руководитель Муравьев С.К. ( )

Дата выдачи задания «1» сентября 2012г.

ОТЗЫВ О РАБОТЕ СТУДЕНТА

Руководитель _____ « » 2012г.

Содержание

Введение

Современные программные продукты обладают широкими возможностями и характеризуются высокой сложностью освоения, что делает актуальной задачу построения интерактивного учебника.

Многие современные учебники по программным продуктам обычно представляют собой справку в виде документации пользователя, поэтому требуется достаточно много времени на изучение лишь документации, а не самого программного продукта. Проблема создания интерактивных учебников решается путем создания специализированных учебников для каждого программного продукта, что не является эффективным.

Поэтому актуальна задача создания такого интерактивного учебника, который предусматривал возможность обучения сразу нескольким программным продуктам в соответствии со специализированным подгружаемым алгоритмом. Таким образом, требуется создать определенную структуру уроков, единую для всех программных продуктов, чтобы иметь возможность одинаково обрабатывать ее. Кроме этого желательно, чтобы программа интерактивных уроков была кроссплатформенной, то есть разработчику интерактивного учебника не требовалось дорабатывать исходный код программы в зависимости от особенностей платформы.

Системы управления базами данных являются многофункциональными, сложными программными продуктами, что требует значительных усилий освоения принципов работы с ними и их настройки. Основными проблемами пользователей, которые не знакомы ранее с системами управления базами данных, являются специфика администрирования системы управления базы данных и язык запросов SQL.

Одной из наиболее развитых систем управления базами данных является PostgreSQL. PostgreSQL — это свободно распространяемая объектно-реляционная система управления базами данных (ORDBMS) и альтернатива современным коммерческим системам управления базами данных. PostgreSQL поддерживается большинством современных платформ, обладает широкими возможностями для хранения информации и манипулирования ей. Несмотря на наличие обширной документации по PostgreSQL, PostgreSQL остается сложным программным продуктом для новых пользователей.

В учебно-исследовательской работе предлагается разработать программу интерактивных уроков по системам управления базами данных на примере курса по PostgreSQL, описывающий установку и настройку сервера базы данных на операционной системе Linux и обучающий основным операторам языка SQL, а также продемонстрировать работу этого интерактивного учебника.

1 Постановка задачи

В настоящее время существует большое количество интерактивных учебников, которые обеспечивают обучение основам работы разных программных продуктов.

Одним из наиболее удобных интерактивных учебников является ресурс `http://try.github.com` [1]— интерактивный учебник по обучению работе с распределенной системой управления версиями файлов **git**, скриншот которого приведен *на рисунке 1.1*. Он содержит один урок по работе с git. Урок разделен на несколько шагов. На каждом шаге приводится описание команды, которую требуется выполнить пользователю. В случае правильно введенной команды осуществляется переход на следующий шаг. В случае неверных действий пользователя учебник выводит предупреждение.

Основные преимущества данного учебника:

- содержит описание команд, которые предлагается выполнить;
- простой интерфейс;
- кроссплатформенный (запускается в браузере).

Основные недостатки:

- хранение уроков осуществляется на сервере, таким образом требует обслуживания сервера базы данных разработчиками;
- отсутствие возможности расширения интерактивного учебника для других программных продуктов.

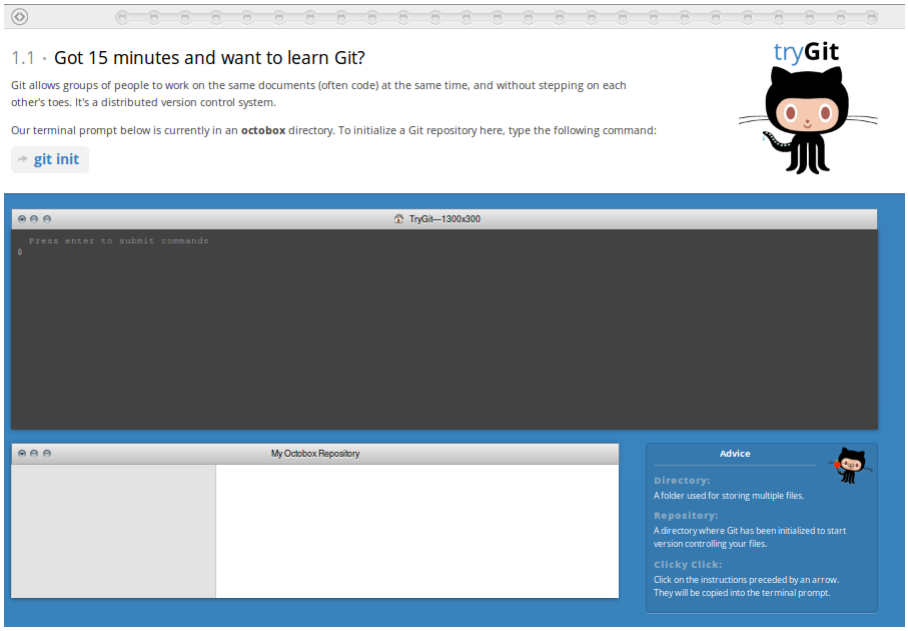

Рис. 1.1: Интерактивный учебник try.github

Для систем управления базами данных также существуют специализированные интерактивные учебники, например, интерактивный учебник по языку запросов SQL, представленный *на рисунке 1.2* и доступный по адресу `http://www.sql-tutorial.ru` [2]. Он содержит уроки по языку запросов SQL. Каждый урок посвящен только одному оператору языка SQL. Урок делится на шаги по

уровням сложности. Пользователю предлагается самому, основываясь на своих знаниях, решить определенную задачу — написать SQL-запрос, в результате которого получится указанный в задании результат. В качестве задания представлена структура таблиц базы данных и что требуется из нее получить. В этом интерактивном учебнике реализована возможность выполнения запроса на нескольких диалектах языка SQL.

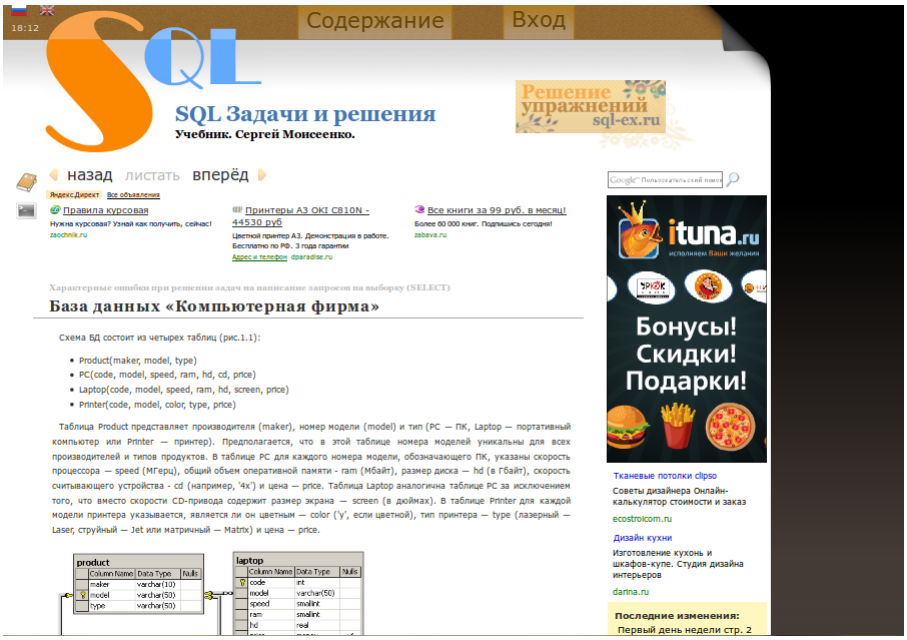

Рис. 1.2: Интерактивный учебник sql-tutorial

Основные преимущества данного учебника:

- возможность обучения диалекту языка SQL (возможность выбрать базу данных, для которой пишется запрос);
- существует несколько путей к решению каждой задачи;
- кроссплатформенный (запускается в браузере).

Основные недостатки:

- нет описания оператора SQL во время выполнения шага;
- реализация и поддержка эмулятора сервера каждой из поддерживаемой базы данных;
- хранение уроков осуществляется на сервере, таким образом требуется обслуживание сервера базы данных разработчиками;
- неизвестно ни одно решение задачи на конкретном шаге;
- отсутствие возможности расширения интерактивного учебника для других программных продуктов.

Рассмотрев все достоинства и недостатки представленных выше интерактивных учебников, предлагается создать приложение интерактивных уроков, удовлетворяющее следующим требованиям:

- приложение должно иметь единую структуру уроков;
- приложение должно иметь единый алгоритм обработки уроков;
- приложение должно быть кроссплатформенным на уровне исходных кодов.

Для создания единой структуры уроков предлагается использовать язык разметки XML, обладающий возможностью простого пользовательского редактирования. Предлагается использовать языки запросов XPath и XQuery — специальные языки, разработанные специально для извлечения информации из XML, а также Qt — кроссплатформенную библиотеку C++, имеющую большое число инструментов по созданию интерфейсов и средства по обработке XML.

Для решения задачи реализации интерактивного учебника требуется выполнить следующие шаги:

- Разработать единую структуру обучающих уроков в формате XML и XSD-схему, соответствующей этой структуре;
- Изучить языки запросов XPath и XQuery;
- Изучить методику разработки GUI-приложений в библиотеки Qt;
- Реализовать в Qt механизмы проверки корректности XML-файла уроков через XSD-схему;
- Разработать дизайн GUI-приложения. Изучить механизмы работы библиотеки Qt с XPath и XQuery;
- Разработать и реализовать механизм обработки уроков;
- Осуществить обзор СУБД PostgreSQL.

Таким образом, целью данной учебно-исследовательской работы является создание программы интерактивных уроков по системам управления базами данных.

2 Разработка архитектуры интерактивного учебника

2.1 Обзор языка разметки XML

В качестве формата хранения данных обучающих уроков был выбран формат **XML**, имеющий важное преимущество с данными, хранящимися в таблицах базы данных. Это преимущество заключается в возможности пользовательского редактирования данных, хранящихся в XML.

XML [3] — это описанная в текстовом формате иерархическая структура, предназначенная для хранения структурированных данных. Визуально она может быть представлена как дерево элементов. Любой XML-документ начинается с заголовка — это строка, указывающая версию XML-документа. Здесь может быть указана кодировка документа и иные зависимости. Например:

```
<?xml version="1.0" encoding="UTF-8"?>
```

Существует понятие правильно оформленного (*англ. well-formed*) XML-документа. В правильно оформленном XML-документе содержится только один корневой узел (*англ. root element*). Это означает, что все остальные элементы должны помещаться между открывающимся корневым тегом и соответствующим ему закрывающимся корневым тегом. Пример правильно построенного XML-документа:

```
<?xml version="1.0" encoding="UTF-8"?>
 <root>
   <element> text </element>
 </root>
```

Остальная часть XML-документа состоит из вложенных элементов, некоторые из которых имеют атрибуты и значения. Имена элементов, как и имена атрибутов, не могут содержать пробелы, но могут быть на любом языке, поддерживаемом кодировкой XML-документа. Имя может начинаться с буквы, подчеркивания, двоеточия. Остальными символами имени могут быть те же символы, а также цифры, дефис, точка.

Различают несколько типов тегов:

1. Пустой тег
   `<tag/>`
2. Тег с содержимым
   `<tag> text </tag>`
3. Тег с атрибутами
   `<tag> attribute_1="value_1" attribute_2="value_2" attribute_N="value_N"</tag>`

Так как в XML зарезервированными символами являются символы «<», «>» и «&», «’», «’’», то для их обозначения используют ссылки на сущности.

Сущностью (англ. entity) в XML называются именованные данные, обычно текстовые, в частности, спецсимволы.

Ссылка на сущность (англ. entity references) указывается в том месте, где должна быть сущность и состоит из амперсанда (`&`), имени сущности и точки с запятой (`;`). В *таблице 1* приведены соответствия между литерами и кодами символа в формате XML.

Например, чтобы описать выражение `if (a > b)` в XML нужно написать так:

```
<code> if (a &gt; b) </code>
```

Литера	Код литеры в XML
&	&amp;
<	&lt;
>	&gt;
’	&apos;
"	&quot;

Таблица 1: Таблица соответствия литеры и ее кода в XML

2.2 Обзор языков запросов XPath и XQuery

2.2.1 Язык запросов XPath

Любой XML-документ представляет собой древовидную структуру данных. У любого из элементов XML-документа могут быть предки и потомки (у корневого элемента предков быть не может). Обрабатывать данную структуру весьма сложно, поэтому был разработан язык запросов **XPath** [4], позволяющий получать данные из XML. Путь делится на шаги, каждый из которых отделяется знаком ‘/’. Каждый шаг содержит:

- Критерий отбора узлов. В качестве критерия отбора используют или имена узлов (элементов, атрибутов) или специальные обозначения множеств узлов, например `text()`, которое символизирует все текстовые узлы. Обозначение `node()` или `*` — любой узел из подмножества узлов текущего шага, а используют их тогда, когда не требуются никакие ограничения. Критерий отбора узлов является обязательной частью каждого шага.

- Предикат шага. Выражение предиката описывается в квадратных скобках и может содержать ограничения на значения атрибутов, ограничения, связанные с вложенными узлами или просто индекс элемента, например `Folder[5]`. Ограничения на значения атрибутов и вложенных элементов могут представлять собой сложные условия, объединенные логическими операциями (`and`, `or`) и содержащие математические выражения (`+`, , `*`, `div`, `mod`). Эта часть шага не является обязательной.

- Ось шага. Под осью шага понимается подмножество узлов документа, определяемое контекстом, на котором будет осуществлен поиск.

Ниже приведен XML, описывающий структуру каталога `/usr` файловой системы операционной системы Linux:

```
<Folder name="usr">
 <Folder name="share">
  <Folder name="doc">
   <Folder name="adduser"/>
   <Folder name="adobe-flashplugin"/>
  </Folder>
  <Folder name="bin"/>
</Folder>
```

и XPath запрос, получающий все имена каталогов в каталоге `/usr/share/doc`:

```
Folder[@name="usr"]/Folder[@name="share"]/Folder[@name="doc"]/*
```

Чтение запроса осуществляется слева направо.

Таким образом, XPath позволяет осуществить запросы, понятные человеку, любого XML-файла. Ниже приведем некоторые комментарии к примеру запроса, иллюстрированному выше:

- Ось атрибутов `::attribute` можно в запросах заменить на символ `@` (например `@name=`).
- Оси предков элемента можно опустить, как это и сделано (полный запрос для каталога `/usr/` будет записан как `child::usr`).
- Для вывода множества всех узлов используют функцию `*`;
- Для получения конкретного узла из множества одинаковых узлов используют индексацию через оператор `[ ]` (как это сделано выше в примере).

2.2.2 Язык запросов XQuery

Первое, на что следует обратить внимание, это то, что в **XQuery [5]** любая конструкция — это выражение, результатом вычисления которого является некоторое значение. Программа XQuery или скрипт — это просто выражение вместе с некоторыми необязательными функциями и другими определениями. Поэтому `3+4` — это завершенная, допустимая программа XQuery, которая при вычислении равняется `7`. XQuery использует **path expression XPath** (т.е. использует адресную индексацию элементов XPath).

Примитивные типы данных XQuery:

- числа, включая целые и числа с плавающей запятой;
- булевы числа: `true` (истина) и `false` (ложь);
- строки символов, например, `"Hello world!"`. Строки являются неизменными, то есть символ в строке не может быть изменен;
- различные типы для представления дат, времени и продолжительности.
- несколько типов, связанных с XML.

Производные типы являются вариациями или ограничениями других типов. Примитивные типы и типы, полученные из них, известны как атомарные типы, поскольку атомарные величины не содержат других величин. Таким образом, строка считается атомарной, так как в XQuery нет значений символов. Величины узлов и выражения

В XQuery также есть типы данных, необходимые для представления величин XML. Для этого используются величины узлов следующих семи типов: элемент, атрибут, пространство имен, текст, комментарий, инструкция обработки и документ (корень).

Для создания и возврата узлов используются различные стандартные функции XQuery. Так, функция `document` читает XML-файл, указанный аргументом URL, и возвращает корневой узел документа. (Корневой элемент - это потомок корневого узла).

Величины узлов XQuery являются неизменными (после создания узла его нельзя изменить).

Последовательности

Рассмотренные атомарные величины (числа, строки и т.п.) и величины узлов (элементы, атрибуты и т.д.) известны как простые величины. Результатом вычисления выражения XQuery на самом

деле является последовательность простых величин. Например, `3,4,5` — это последовательность, состоящая из 3 целых. Заметим, что если последовательность содержит только одно значение, то она совпадает с самой величиной. Не допускается использовать вложения для последовательностей.

Функции

Без функций, определяемых пользователем, XQuery был бы далек от языка от программирования. Определения таких функций располагаются в прологе запроса (query prologue) программы XQuery. Стоит заметить, что параметрами функции и результатами ее вычисления могут быть примитивные типы, узлы или последовательности из тех или других.

Ниже приведена рекурсивная служебная функция. Она возвращает все узлы-наследники аргумента, включая сам узел аргумента. Она проходит вглубь по аргументу и возвращает аргумент, а затем выполняет цикл по потомкам узла аргумента, рекурсивно вызывая саму себя для каждого потомка.

```
define function descendant-or-self ($x)
{
$x,
for $y in children($x)
        return descendant-or-self($y)
}
```

В результате вызова функции

```
descendant-or-self(<a>X<b>Y</b></a>)
```

получается последовательность глубины 4:

```
<a>X<b>Y</b></a>; "X"; <b>Y</b>; "Y"
```

XQuery имеет большую библиотеку функций. *В таблице 2* представлены некоторые функции из библиотеки

Название функции	В чем состоит работа функции
`doc($variable)`	Возвращает содержимое XML-документа для запросов
`concat($string1,[..], $string)`	Выполняет конкатенацию строк, переданных в аргументах
`text()`	Возвращает текстовое значение элемента
`string(element)`	Преобразует содержимое элемента `element` в строку
`count(element)`	Возвращает число элементов с названием тега `element`

Таблица 2: Некоторые функции XQuery

Определение типов

XQuery - строго типизированный язык программирования. Как Java и C#, XQuery - это смесь статического (проверка совместимости типов во время компиляции) и динамического контроля типов (тестирование типов во время выполнения). Однако, типы в XQuery отличаются от классов, присущих объектно-ориентированному программированию. Взамен XQuery включает типы, которые соответствуют модели данных XQuery, и позволяет импортировать типы из XML Schema.

2.3 Обзор библиотеки Qt

Библиотека **Qt** предоставляет обширные средства для создания пользовательских графических интерфейсов, обработки различной информации. Можно выделить основные достоинства библиотеки Qt по сравнению с остальными библиотеками, предоставляющими возможности по созданию графического интерфейса:

- кроссплатформенность на уровне исходных кодов;
- все классы библиотеки разделяются на модули, каждый из которых может быть подключен;
- наличие интерактивной документации по каждому из модулей.

2.3.1 Методика разработки GUI-приложения на Qt

В языке C++ нет удобных механизмов, позволяющих динамически взаимодействовать объектам нескольких классов, отвечающих за графический интерфейс, ничего друг о друге не зная при компиляции. Другими словами, как сообщить объекту одного класса, что в работе другого что-то произошло? Взаимодействие объектов Qt между собой и с внешним окружением происходит немного по-разному. Информацию о том, что происходит во внешнем окружении, программа получает, в основном в виде **событий**. **Обработчики событий** — это обычные методы классов Qt, которым в качестве аргумента передается указатель на объект, описывающий событие. [6]

На рисунке 2.1 представлена схема цикла обработчика событий в Qt.

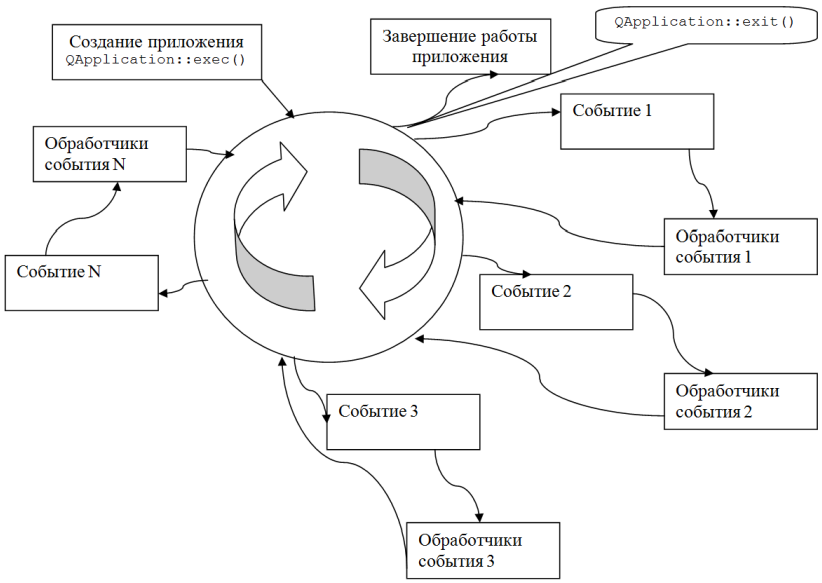

Рис. 2.1: цикл жизни приложения в Qt

Любое графическое приложение в Qt начинают свою жизнь с создания окна приложения `QMainWindow` методом `QApplication::exec()`. Далее запускается обработчик событий. Любая реакция пользователя (щелчок мыши, перетаскивание окна, нажатие клавиши на клавиатуре) может быть обработана так, как этого хочет программист. При возникновении события, описанного программистом, жизненный цикл приложения приостанавливается, событие обрабатывается и снова возвращается в цикл обработки события. В случае если вызывается метод `QApplication::exit()`, приложение прекращает свою работу.

Взаимодействие между графическими объектами приложения в Qt реализовано с помощью *механизма сигналов и слотов*.

Сигналы — это функции, не имеющие определения, возвращающие тип `void` и определенныес помощью ключевого слова Qt `signal` в заголовочном классе.

Слоты — это функции, которые обрабатывают сигналы, другими присылаемые объектами, также возвращают тип `void` и объявляются с помощью ключевого слова `slots` в заголовочном файле. Данный механизм работает только при определении в каждом классе макроса `Q_OBJECT`. Чтобы связать два объекта с помощью этого механизма, в классе-отправителе сигнала должна определена функция-сигнал [7]:

```
1  class MySignal {
2    Q_OBJECT
3    signals:
4          void signal();
5  }
```

Чтобы выслать сигнал, используется ключевое слово `emit`: `emit signal();`

В классе-приемнике создадим функцию-обработчик сигнала из класса `MySignal`:

```
1  class Reciever {
2    Q_OBJECT
3    public slots:
4    void recieverSignal() { qDebug() << "Получен сигнал"; }
5  }
```

Для связи двух объектов sender и receiver с помощью сигналов и слотов используется функция `connect`:

```
connect(sender, SIGNAL(signal()), receiver, SLOT(slot()));
```

В нашем случае для связи объекта класса `MySignal` с объектом класса `MyReciever` достаточно написать так:

```
connect(my_signal_obj,
                SIGNAL(signal()),
                my_receiver_obj,
                SLOT(recieverSignal()));
```

Для разрыва связи между двумя объектами достаточно написать функцию `disconnect`, в качестве параметров указываются имя объекта-отправителя, сигнал, имя объекта-приемника и слот обработки сигнала-приемника:

```
disconnect(my_signal_obj,
                    SIGNAL(signal()),
                    my_receiver_obj,
                    SLOT(recieverSignal()));
```

Кроме того, существует перегруженная функция `disconnect`, которая разъединяет все связи объекта:

```
disconnect(my_obj, 0, 0, 0);
```

что эквивалентно вызову функции

```
my_obj -> disconnect();
```

Компиляция приложения в Qt происходит несколько иначе, чем компиляция в чистом C++. В начале компиляции вызывается MOC (Meta Object Compiler, метаобъектный компилятор). Он анализирует классы на наличие специального макроса `Q_Object` в их определении и внедряет в отдельный файл всю необходимую дополнительную информацию. После вызова MOC уже обработанные файлы компилируются компилятором C++.

2.3.2 Обзор компонентов библиотеки Qt

В Qt все классы для удобства разработки приложений разделены на модули, в каждом из которых содержатся классы для работы с разными видами объектов (графикой, аудио, рисунками, XML и др.) В *таблице 3* приведено краткое описание некоторых модулей библиотеки Qt.

Модуль	Описание
`QtCore`	Содержит классы, обеспечивающие ядро библиотеки Qt
`QtGui`	Дополняет модуль QtCore функциональностью GUI
`QtXmlPatterns`	Обеспечивает поддержку XML, XPath, XQuery и валидацию XML схемой XSD

Таблица 3: Описание некоторых модулей Qt.

Модуль `QtCore` является базовым модулем для всей библиотеки Qt. В него входят реализации основных типов данных, контейнерные классы, классы ввода и вывода, классы работы с датой, временем, базовый класс событий, класс `QObject`, являющийся краеугольным камнем объектной модели Qt.

Модуль `QtGui` содержит важный для нас класс `QWidget` — класс-родитель, от которого наследуются все классы виджетов в Qt. Содержит виджеты компановки, кнопки, текстовые поля и так далее. **Виджеты** — это стандартные графические примитивы, выполняющие определенные действия.

Модуль `QtXmlPatterners` содержит классы для работы с XQuery и XPath. Класс `QXmlQuery` позволяет загружать XQuery-запросы, класс `QXmlQuery` необходим для загрузки шаблона XQuery-запроса и связывании переменных в шаблоне, класс `QXmlFormatter` предназначен для вывода результатов запроса, а также классы `QXmlSchema` и `QXmlValidator` позволяют загружать XSD-схему и реализовать механизм соответствия XML-файла XSD-схеме.

Далее рассмотрим подробнее некоторые классы библиотеки Qt.

2.3.3 Обзор класса QTreeWidget

Класс `QTreeWidget` обеспечивает создание виджета дерева.
Основные методы класса `QTreeWidget`:

- конструктор `QTreeWidget(QWidget *parent=0)`. Создает виджет дерева;
- `void addTopLevelItem (QTreeWidgetItem *item)` — добавляет в дерево элемент дерева `QTreeWidgetItem` как корневой элемент дерева;
- `QTreeWidgetItem* currentItem() const` — возвращает текущий элемент в дереве;
- `int indexOfTopLevelItem (QTreeWidgetItem *item) const` — возвращает индекс корневого элемента в дереве;
- `QTreeWidgetItem* itemAbove(const QTreeWidgetItem *item) const` — возвращает элемент дерева, находящийся выше элемента `item` в дереве;
- `QTreeWidgetItem* itemBelow(const QTreeWidgetItem *item) const` — возвращает элемент дерева, находящийся ниже элемента `item` в дереве;
- `void setHeaderLabel (const QString &label)` — устанавливает метку дерева;
- `QTreeWidgetItem* topLevelItem (int index) const` — возвращает элемент дерева по индексу `index`.

2.3.4 Обзор класса QTreeWidgetItem

Класс `QTreeWidgetItem` обеспечивает работу с элементами дерева `QTreeWidgetItem`.
Основные методы класса:

- конструктор `QTreeWidgetItem (QTreeWidgetItem* parent, int type=Type)` — создает новый вложенный элемент дерева. Вложенность достигается путем указания элемента-родителя `parent`;
- `int childCount() const` — возвращает число потомков элемента;
- `bool isExpanded() const` — возвращает истину, если дерево раскрыто и ложь, если нет;
- `QTreeWidgetItem* parent() const` — возвращает родителя элемента и 0, если родителей не имеет;
- `void setSelected(bool select)` — установить элемент дерева выбранным, при передаче параметра `select=true` и снять выделение при передаче параметра `select=false`;
- `void setText(int column, const QString &text)` — установить текст `text` элемента в колонку `column`;
- `QString text (int column) const` — получить текст элемента из колонки `column`;

2.4 Обзор языка запросов SQL

SQL [8](обычно произносимый как "СИКВЭЛ" или "ЭСКЮЭЛЬ") символизирует собой Структурированный Язык Запросов. Это — язык, который дает нам возможность создавать и работать в реляционных базах данных, являющихся наборами связанной информации, сохраняемой в таблицах.

Информационное пространство становится более унифицированным. Это привело к необходимости создания стандартного языка, который мог бы использоваться в большом количестве различных видов компьютерных сред. Стандартный язык позволит пользователям, знающим один набор команд, использовать их для создания, нахождения, изменения и передачи информации - независимо от того, работают ли они на персональном компьютере, сетевой рабочей станции, или на универсальной ЭВМ.

В нашем все более и более взаимосвязанном компьютерном мире, пользователь снабженный таким языком, имеет огромное преимущество в использовании и обобщении информации из ряда источников с помощью большого количества способов.

Элегантность и независимость от специфики компьютерных технологий, а также его поддержка лидерами промышленности в области технологии реляционных баз данных, сделало SQL (и, вероятно, в течение обозримого будущего оставит его) основным стандартным языком. По этой причине, любой, кто хочет работать с базами данных 90-х годов, должен знать SQL.

Стандарт SQL определяется ANSI (Американским Национальным Институтом Стандартов) и в данное время также принимается ISO (Международной Организацией по Стандартизации). Однако, большинство коммерческих программ баз данных расширяют SQL без уведомления ANSI, добавляя различные особенности в этот язык, которые, как они считают, будут весьма полезны. Иногда они несколько нарушают стандарт языка, хотя хорошие идеи имеют тенденцию развиваться и вскоре становиться стандартами "рынка" сами по себе в силу полезности своих качеств.

Состав языка SQL

Язык SQL предназначен для манипулирования данными в реляционных базах данных, определения структуры баз данных и для управления правами доступа к данным в многопользовательской среде. Поэтому, в язык SQL в качестве составных частей входят:

- язык манипулирования данными (Data Manipulation Language, DML)
- язык определения данных (Data Definition Language, DDL)
- язык управления данными (Data Control Language, DCL).

Подчеркнем, что это не отдельные языки, а различные команды одного языка. Такое деление проведено только лишь с точки зрения различного функционального назначения этих команд.

Язык манипулирования данными используется, как это следует из его названия, для манипулирования данными в таблицах баз данных. Он состоит из 4 основных команд: SELECT (выбрать), INSERT (вставить), UPDATE (обновить), DELETE (удалить)

Язык определения данных используется для создания и изменения структуры базы данных и ее составных частей - таблиц, индексов, представлений (виртуальных таблиц), а также триггеров и сохраненных процедур.

Язык управления данными используется для управления правами доступа к данным и выполнением процедур в многопользовательской среде. Более точно его можно назвать "язык управления доступом". Он состоит из двух основных команд: GRANT (дать права), REVOKE (забрать права).

3 Разработка учебно-тренировочных средств по СУБД

3.1 Алгоритм построения учебно-тренировочных средств по СУБД

Программа учебно-тренировочных средств по СУБД PostgreSQL должна выполнять следующий алгоритм:

1. Запуск программы;
2. Проверка соответствия XML-файла XSD-схеме;
3. Проверка, была ли запущена программа учебно-тренировочных средств по СУБД;
4. Если программа была запущена ранее, то восстанавливаем состояние программы;
5. Если программа не была запущена ранее, то создаем папку для сохранения состояния программы в домашнем каталоге пользователя и создаем начальное состояние программы;
6. Обработка команды, введенной пользователем. Если пользователь не выполнил все инструкции программы, то повторяем шаг 6.
7. Переход на новый шаг, обновляем информацию о шаге;

Данный алгоритм иллюстрирует следующая блок-схема приложения (*см. рисунок 3.1*):

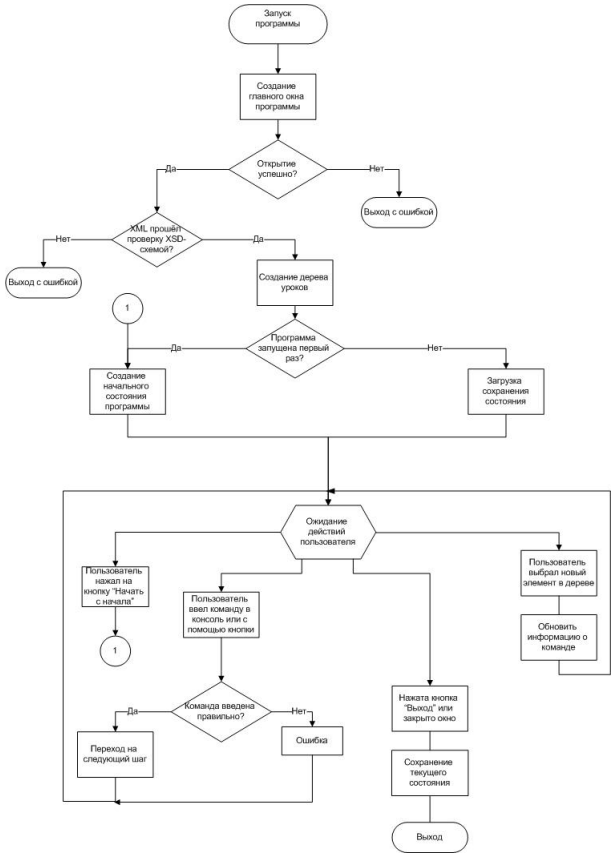

Рис. 3.1: Блок-схема программы

Работу программы можно условно разделить на два этапа: подготовительный и этап обработки действий пользователя.

На первом этапе работы программы происходят создание окна приложения, создание элементов графического интерфейса и их компоновка, открытие XML и проверка его на соответствие XSD-схеме, создание дерева уроков. Проверка соответствия загружаемого XML XSD-схеме позволяет избежать сбоев в работе программы и намеренной порчи файлов, поставляемых вместе с программой. Поэтому требуется реализовать корректный механизм проверки соответствия XML XSD-схемы непосредственно перед запуском цикла обработки событий. Сохранение непосредственно перед выходом из программы позволит пользователю не допустить сброса состояния программы и тем самым предоставляет возможность восстановить состояние программы при следующем запуске.

На втором этапе работы программы выполняется запуск цикла обработки событий.

3.2 Разработка структуры обучающих уроков

Для обучающих уроков предлагается использовать следующую иерархическую структуру (*представлена на рисунке 3.2*):

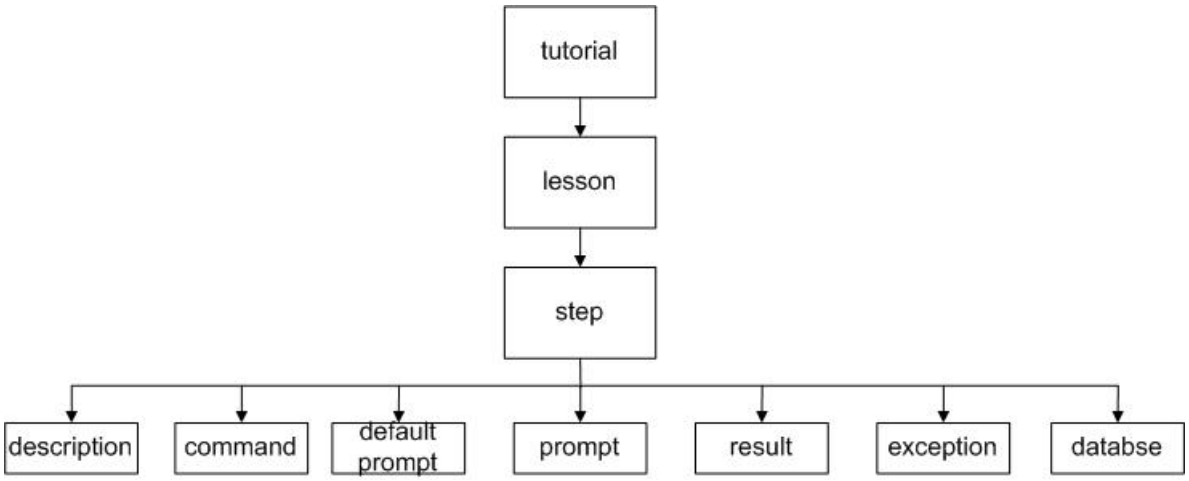

Рис. 3.2: Предлагаемая иерархическая с структура

В данной структуре корневой узел `tutorial`, имеющий несколько дочерних элементов `lesson`.

Каждый узел `lesson` содержит информацию о названии урока, содержит вложенные элементы `step`.

Узел `step` содержит информацию о названии шага, а также о том, что требуется от пользователя сделать на текущем шаге.

Узел `description` содержит краткую справку о команде и о том, что нужно выполнить для перехода на следующий шаг.

Узел `command` содержит команду для выполнения пользователем.

В узле `default_prompt` содержится приглашение ввода команды.

Узел `prompt` содержит приглашение для ввода команды в случае диалога между программой и пользователем.

Узел `result` содержит информацию, выводимую в консоль при успешном выполнении команды или при частичном ее выполнении.

Узел `exception` содержит информацию, выводимую в консоль при ошибочных действиях пользователя (например, неверная установка пароля).

Узел `database` содержит текущую информацию о базе данных или о структуре конкретной таблицы (присутствует во втором уроке).

Далее приведен XML-документ, соответствующий данной структуре:

```
<tutorial>
  <lesson name="название_урока_1">
    <step name="название_шага_1">
      <description> описание_шага_1 </description>
      <command> команда_для_выполнения </command>
      <default_prompt> приглашение_для_ввода_команды </default_prompt>
      <prompt> приглашение_для_ввода_команды_в_оболочках_1 </prompt>
       ...
      <prompt> приглашение_для_ввода_команды_в_оболочках_N </prompt>
      <result> результат_вывода_команды_1 </result>
       ...
      <result> результат_вывода_команды_M </result>
       ...
      <exception> Вывод_команды_в_случае_ошибки_K </exception>
      <database> Текущая_структура_базы_данных </database>
    </step>
  </lesson>

  <lesson name="название_урока_2">
   ...
  </lesson>
   ...
  <lesson name="название_урока_P">
   ...
  </lesson>

</tutorial>
```

Ниже представлена XSD-схема, позволяющая проверить XML-файл уроков на соответствие ей.

```
1  <?xml version="1.0"?>
2  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
3    <xs:element name="tutorial">
4      <xs:complexType>
5        <xs:sequence>
6          <xs:element name="lesson" type="lesson" minOccurs="1" maxOccurs="unbounded"/>
7        </xs:sequence>
8      </xs:complexType>
9    </xs:element>
10
11   <xs:complexType name="lesson">
12     <xs:sequence>
13       <xs:element name="step" type="step" minOccurs="1" maxOccurs="unbounded"/>
14     </xs:sequence>
15     <xs:attribute name="name" type="xs:string"/>
16   </xs:complexType>
17
18   <xs:complexType name="step">
19     <xs:sequence>
20       <xs:element name="description" type="xs:string"/>
21       <xs:element name="command" type="xs:string"/>
22       <xs:element name="default_prompt" type="xs:string"/>
```

```
23        <xs:element name="prompt" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
24        <xs:element name="result" type="xs:string" minOccurs="1" maxOccurs="unbounded"/>
25        <xs:element name="exception" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
26        <xs:element name="database" type="xs:string" minOccurs="0" maxOccurs="1"/>
27      </xs:sequence>
28      <xs:attribute name="name" type="xs:string"/>
29    </xs:complexType>
30
31  </xs:schema>
```

Данная схема позволяет не только указать вложение одного элемента в другой, но и явно указать, какой тип данных может иметь конкретный атрибут, количество этих атрибутов и так далее.

Приведем некоторые комментарии к созданию XSD-схемы:

Можно заметить, что у каждого элемента имеется префикс `xs:`. Он указывает на пространство имен через объявление в атрибуте тега `schema` пространства имен `xmlns:xs="http://w3.org/2001/XMLSchema"`. Тот же самый префикс должен использоваться у каждого элемента схемы. Целью определения пространства имен является желание связать элементы и простые типы со словарем языка XML-схем, а не со словарем автора схемы.

Обязательным корневым элементом для языка XML-схемы является элемент `schema`.

С помощью тега `complexType` объявляется сложный тип, который описывает вложенность элементов в XML-документе. Каждый `complexType` в нашей схеме для простоты понимания описывается в схеме отдельно.

С помощью тега `sequence` достигается требование последовательности появления тегов. Например, в `complexType step` объявлено, что в XML-документе, удовлетворяющем данной схеме, должны идти теги в следующем порядке: `description`, `command`, `default_prompt` и другие. *(см. строки 19-27)*

С помощью тега `element` объявляется элемент XML-документа, первый атрибут которого должен является `name`. Значение атрибута `name` является название тега в XML-документе. Второй атрибут — `type` определяет допустимый тип элемента в XML. Например, *в строке 20* указано, что тег `description` в XML должен являться строкой (`type = "xs:string"`).

Количество тегов с одинаковыми именами в XML-документе регулируются с помощью конструкций `minOccurs= maxOccurs=`, являющиеся в XML Schema атрибутами. Минимальное и максимальное количество должны быть указаны целыми неотрицательными числами, например *в строке 25* описано, что элемент `exception` может присутствовать не более одного раза (`maxOccurs="1"`), а может и не присутствовать (`minOccurs="0"`). Кроме того, для описания неограниченного количества вхождений элементов используется константа `"unbounded"`, как это сделано *в строке 23* `maxOccurs="unbounded"` (количество элементов `result` может быть от 1 и неограничено).

3.3 Реализация механизма проверки XML XSD-схемой

Как было сказано выше, возможность валидации XML XSD-схемой реализуется с помощью модуля `QtXmlPatterners` библиотеки Qt, а именно с помощью методов классов `QXmlSchema` и `QXmlSchemaValidator`. На листинге представлена реализация этого механизма.

```
1  QFile source_document("test.xml");
2  QByteArray testData;
3  testData = source_document.readAll();
4  source_document.close();
5
6  QFile file_xml_schema("schema.xsd");
7  file_xml_schema.open(QIODevice::ReadOnly);
```

```
 8  const QByteArray schemaData = file_xml_schema.readAll();
 9  file_xml_schema.close();
10  QXmlSchema schema;
11  schema.load(schemaData);
12  QXmlSchemaValidator schema_validator(schema);
13  if (false == schema_validator.validate(testData,
14                  QUrl::fromLocalFile(source_document.fileName()))){
15      qDebug() << "XML не прошел валидацию";
16  } else {
17      qDebug() << "XML прошел валидацию";
18  }
```

Поясним механизм проверки XML схемой XSD в Qt.

В *строках 1-4* происходит открытие XML-файла для проверки.

В строках 6-9 происходит открытие файла XSD-схемы и загрузки ее в программу. Далее, *в строках 10-11* происходит создание объекта `QXmlSchema`, в который загружается содержимое файла схемы.

С помощью конструктора `QXmlSchemaValidator(const QXmlSchema &schema)` создается объект `schema_validator`, который можно использовать для проверки содержимого XML-файла. Метод `validate()`, принимающий первым параметром XML-данные `testData`, а вторым — адрес их получения производит сканирование XML-документа и выполняет проверку на соответствие XML схеме.

3.4 Реализация механизма получения результатов XQuery-запросов

Для реализации механизма получения результатов XQuery и XPath запросов используется модуль Qt `QtXmlPatternes`. Класс `QXmlQuery` отвечает за загрузку **шаблона** запроса, подстановку переменных в шаблон и выполнение запроса. Класс `QXmlFormatter` преобразует вывод результата запроса к XML с помощью соответствующих методов. Далее на листинге приведена реализация механизма получения результатов запроса XQuery.

```
 1  QFile file_request(file_name);
 2  if (false == file_request.open(QIODevice::ReadOnly)) {
 3     qDebug() << "Ошибка при открытии файла запросов";
 4     return;
 5  }
 6  const QString request_string(QString::fromUtf8(file_request.readAll()));
 7  file_request.close();
 8
 9  QString step_name = "Установка и настройка СУБД";
10  QString lesson_name = "Установка пакетов базы данных";
11
12  QFile source_document("lessons.xml");
13  source_document.open(QIODevice::ReadOnly);
14
15  QBuffer buffer;
16  buffer.open(QIODevice::ReadWrite);
17
18  QXmlQuery query;
19  query.bindVariable("input_document", &source_document);
20  query.bindVariable("step_name", QVariant(step_name));
21  query.bindVariable("lesson_name", QVariant(lesson_name));
22  query.setQuery(request_string);
23
24  QXmlFormatter formatter(query, &buffer);
```

```
25  if (false == query.evaluateTo(&formatter))
26      qDebug() << "Ошибка при выводе результата запроса";
27
28  buffer.close();
29  source_document.close();
30
31  QString result=QString::fromUtf8(buffer.data());
32  result.remove(QRegExp("</?p/?>"));
```

В *строках 1-7* происходит открытие файла запроса, считывание запроса из объекта `QFile` в переменную `QString`, а потом закрытие файла. В случае, если файл не может быть открыт в режиме чтения (*строка 2*), выводится предупредительная строки об ошибке и осуществляется выход из функции с помощью ключевого слова `return`.

В *строках 9-10* создаются объекты `QString lesson_name step_name`, являющиеся значениями атрибута `name` узлов `lesson` и `step` соответственно.

Далее в *строках 12-13* происходит создание и открытие на чтение/запись объекта `QBuffer buffer`.

В *строке 18* происходит создание объекта класса `QXmlQuery`, обеспечивающий загрузку, связывание переменных и выполнение XQuery-запроса. Метод класса `QXmlQuery bindVariable` позволяет связать внешние переменные XQuery-запроса с конкретным значением. В классе `QXmlQuery` реализовано множество полиморфных функций `bindVariable()`. Мы используем следующий вариант функции

```
void QXmlQuery::bindVariable(const QString &localName,
                             QIODevice *device)
```

для связывания внешней переменной `$input_document` XQuery-запроса с объектом `QFile` XML-документа и полиморфную функцию

```
void QXmlQuery::bindVariable(const QString &localName,
                     const QXmlItem &value)
```

для связывания значений атрибутов `name` узлов `lesson` и `step` с значениями переменных `QString lesson_name` и `step_name`.[1] После вызова метода `setQuery()` в объекте `query` будет сгенерирован запрос (*см. строку 22*).

Созданный *в строке 24* объект `QXmlFormatter` осуществляет вывод результата запроса в виде XML. Выполнение запроса происходит с помощью метода `evaluateTo(const &QXmlFormatter)`, возвращающий `true` в случае успешного выполнения и `false` иначе. *В строках 28-29* закрывается объект `QFile source_document` и объект `QBuffer buffer`.

В строке 31 происходит получение результата запроса, а *в строке 32* реализуется обработка результата. Обработка результата заключается в удалении тегов `<p>` `</p>` `<p/>` по регулярному выражению `</?p/?>`.

3.5 Реализация записи данных в XML

Механизм записи данных в XML позволяет сохранить конкретные состояния программы в XML, что удобно для их обработки. Возможность записи данных в XML реализована в модуле Qt `QtCore`. Это подчеркивает значимость XML как способа хранения данных.

[1] Для преобразования объекта `QString` в `QXmlItem` используется класс `QVariant`, который создает необходимый нам объект `QXmlItem` с помощью вызова конструктора `QVariant(string)`, как это сделано в *строках 19-20*.

В работе используются классы `QXmlStreamWriter` для создания узлов XML, их атрибутов и значений для генерирования XML. Рассмотрим основные методы этого класса на примере реализации сохранения текущего состояния программы в файл:

```
1  QString file_content;
2  QXmlStreamWriter streamWriter(&file_content);
3  streamWriter.setAutoFormatting(true);
4  streamWriter.writeStartDocument();
5  streamWriter.writeStartElement("tutorial");
6  {
7  //получаем значение количества уроков и
8  //осуществляем запись их номеров в XML
9
10   streamWriter.writeStartElement("lesson");
11   streamWriter.writeAttribute("number", QString::number(i + 1));
12
13   //получаем значение количества пройденных пользователем шагов и
14   //записываем их в XML
15
16   streamWriter.writeAttribute("enabled_steps", QString::number(j));
17   streamWriter.writeEndElement();
18 }
19 streamWriter.writeEndElement();
20 streamWriter.writeEndDocument();
21 QTextStream out(&file);
22 out << file_content;
23 file.close();
```

В строках 1-2 происходит создание объекта класса `QXmlStreamWriter`. В *строке 3* устанавливаем ступенчатое форматирование и информацию о версии XML в нашей строке методом `setAutoFormatting()`, в которую будет выводить результат записи объект `streamWriter`. Метод `writeStartDocument()` указывает на начало записи XML. Метод `writeStartElement()`, вызываемый *в строке 5* записывает в строку открывающийся тег `tutorial` (строка будет следующая: `<tutorial`).

Вложенность элементов достигается с помощью очередного вызова метода `writeStartElement()`, как это и сделано *в строке 10*. В следующей строке происходит запись атрибута `name` со значением номера урока ближайшего открытого тега (в нашем случае `lesson`) а потом еще одного атрибута `enabled_steps` с указанием доступных уроков в дереве (*строка 1*6).

Для окончания записи последнего открытого элемента вызывается метод объекта `streamWriter` `writeEndElement()`, как это сделано в *строке 17* (т.о. оканчивается запись элемента `lesson`).

В строке 19 завершается запись элемента `tutorial`, а для окончания записи XML вызывается метод `writeEndDocument()` (*см. строку 20*).

В последних трех строках происходит открытие потока `QTextStream` и запись в файл строки с XML. В результате будет записан XML со следующим содержанием:

```
<?xml version="1.0"?>
<tutorial>
    <lesson number="1" enabled_steps="21"/>
    <lesson number="2" enabled_steps="4"/>
</tutorial>
```

3.6 Разработка графического интерфейса приложения

Для разработки графического интерфейса приложения предлагается взять за основу вариант компоновки элементов графического интерфейса интерактивного учебника **try.github**. Предлагается следующая модель расположения элементов графического интерфейса в приложении:

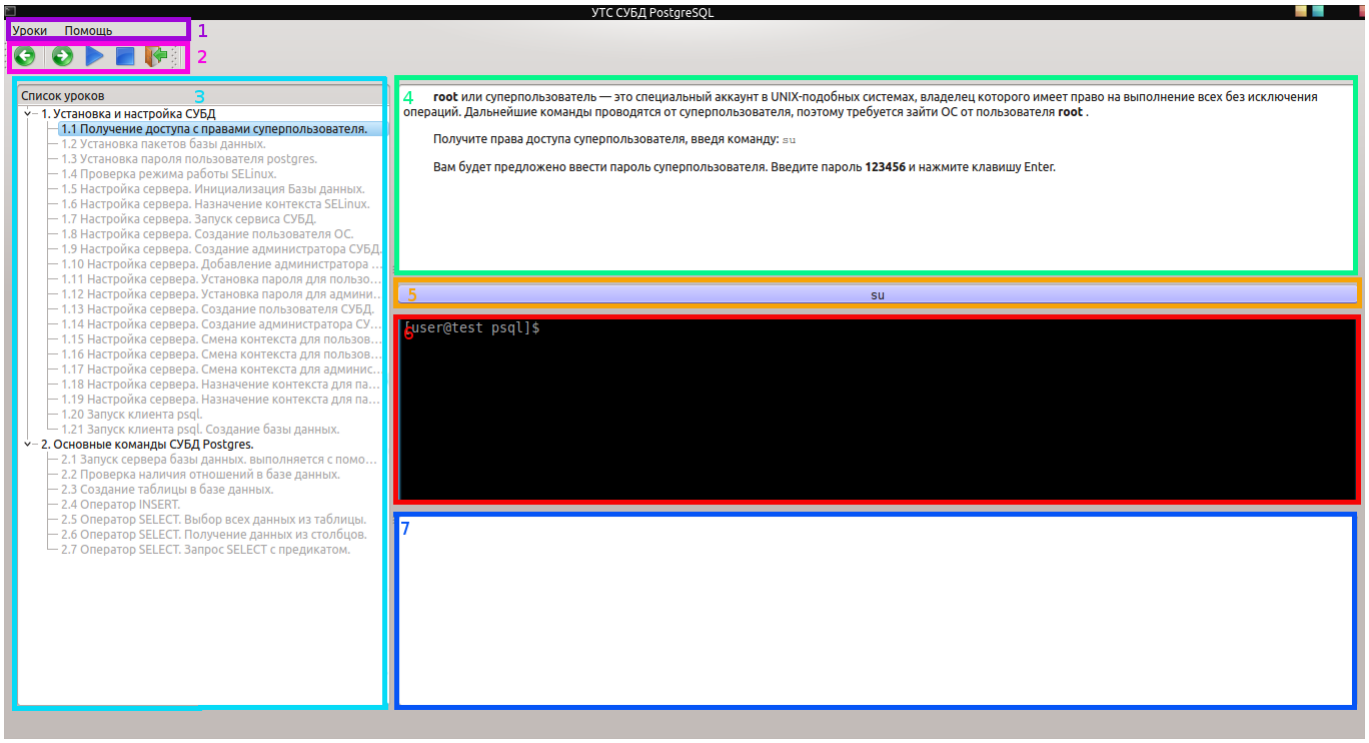

Рис. 3.3: Графический интерфейс программы в kubuntu 12.04

Интерфейс программы интерактивных уроков состоит из главного меню (*1*), панели инструментов (*2*) и сгруппированных по сетке элементов графического интерфейса (виджетов) (*3-7*). Вся текстовая информация на каждом шаге загружается в виджеты через XQuery-запросы, механизм вывода результатов которых реализован в программе.

Главное меню разделено на две секции: секция "*Уроки*" – содержит функции переключения между уроками, функцию начала обучения, при выборе которой происходит сброс состояния обучения пользователя. Вторая секция – "*Помощь*" представляет помощь по программе и предоставляет информацию о разработчике. Для удобства все функции секции "*Уроки*" вынесены на панель инструментов (*2*), а также добавлена кнопка "*Прекратить выполнения команды*". Она может быть использована в случае, когда выполняется команда, требующая диалога с пользователем.

Основное окно программы содержит сгруппированные по сетке виджеты. Виджет дерева, расположенный в левой части окна программы (*1*), создается для предоставления информации пользователю о количествах уроков в XML-файле, количества шагов в уроках, а также графически отображает состояние пройденных шагов (неактивные – пройденные, активные – пройденные). Кроме того, пользователь может выбрать любой из пройденных шагов и повторить задание.

Вывод информации о команде, предоставление описания о ней осуществляется в текстовом поле в правой верхней части программы (4).

Эмулятор консоли (6) служит для ввода команд пользователя и осуществляет проверку вводимых команд. Для облегчения работы пользователя в эмуляторе предусмотрены возможности перемеще-

ния по вводимой строке с помощью стрелок ← и →, а также выбрать уже вводимую команду с помощью ↑ и ↓.

Для удобства над виджетом консоли располагается кнопка с текущей требуемой командой от пользователя (*5*). Пользователь может воспользоваться ей, не набирая команды в консоли. При нажатии пользователя на кнопку происходит считывание команды с кнопки в консоль, пользователю требуется ввести лишь клавишу `Enter` для перехода к следующему шагу.

Нижнее текстовое поле (*7*) необходимо для вывода структуры таблицы базы данных на конкретном шаге.

3.7 Описание реализованных классов

В ходе учебно-исследовательской работы были созданы два класса, `Console` и `MainWindow`, реализующие всю логику работы программы.

3.7.1 Описание класса Console

Класс `Console` был разработан для эмуляции терминала linux на базе класса `QTextEdit` и обеспечивает ввод символов в консоль, загрузки приглашений для ввода команд и реализует полезные функции терминала. Рассмотрим некоторые методы этого класса:

- конструктор класса `Console(QWidget *parent = 0)`. Здесь происходит создание виджета, установка ширины текстового курсора, установка шрифтов (используется шрифт `Monospace 12`), инициализируются значение переключателей блокировки и ввода пароля;

- метод `keyPressEvent(QPressEvent *event)`. Метод реализует ввод символов с клавиатуры в виджет консоли, а также реагирует на нажатие пользователя клавиши `Enter` высыланием сигнала
  `signalCommandEntered(QString)`. Этот сигнал обрабатывается в классе `MainWindow`. Здесь же реализованы возможности перемещать курсор в эмуляторе с помощью клавиш ← и →, а также выбрать уже введенную команду с помощью ↑ и ↓;

- метод `Output(QString stringToOut)` выводит результат выполнения команды в консоль, тем самым эмулируя работу самой консоли;

- методы `InsertPrompt(bool insert_new_block)` и
  `InsertPrompt(QString new_prompt,bool insert_new_block)` обеспечивают вставку в эмулятор консоли приглашений для ввода. Переменная типа `bool` указывает на необходимость делать переносы на новую строку при вставке приглашения.

3.7.2 Описание класса MainWindow

Класс `MainWindow` реализовывает методы создания основного окна приложения, открытие XML-файла уроков, проверку его на соответствие XSD-схемой, обработки введенных пользователем команд, сохранение и загрузка состояний. Рассмотрим основные методы этого класса:

- Конструктор класса `MainWindow(QWidget *parent = 0)` создает окно приложения, панель инструментов и главное меню;

- Метод `CreateWindow()` создает виджеты и их компанует их. Кроме того, в методе связываются сигналы и их обработчики;

- Метод `OpenXmlDocument()` обеспечивает загрузку XML-документа в программу, выполняет проверку на соответствие XSD-схемой и в случае соответствия XML схеме производится построение дерева обучающих уроков методом `CreateXmlTree()`;
- Метод `PrepareProgramForUser()` проверяет, было ли уже запущена данная программа пользователем поиском папки `~/.config/UTS_PostrgreSQL`, в случае отсутствия эта папка создается. В указанную папку сохраняется состояние программы после закрытия окна пользователем;
- Метод `GetCountOfItem(QString file_name, QString attribute_name = QString())` позволяет получить количество элементов с помощью выполнения XQuery-запроса;
- Метод `GetRequestResult(QTreeWidgetItem *item, QString file_name, const int index = 0)` реализует механизм получения результатов XQueey-запросов для каждого шага;
- Метод `slotItemOnXmlChanged(QTreeWidgetItem *item, int)` является обработчиком сигнала смены текущего элемента в виджете дерева. Функция `ItemOnXmlChanged(QTreeWidgetItem *item)` обновляет информацию о текущем уроке с помощью вызова выполнения XQuery-запросов;
- Метод `slotItemOnXmlTreeClicked(QTreeWidgetItem *item, int)` обрабатывает нажатие пользователем элемента в дереве и по необходимости (если выбранный элемент является шагом) обновляет информацию о шаге;
- Метод `slotEnterPressedOnConsole(QString entered_command)` является слотом обработки введенной в консоль команды. В функции `ValidateEnteredCommand(QString entered_command), вызываемой в данном методе,` происходит проверка на правильность ввода команды, а в случае правильного ввода команда ищется в списке команд, требующих диалога с пользователем. Если команда не найдена в этом списке, то делается XQuery-запрос результата выполнения команды и выполняется переход на следующий шаг вызовом метода `MoveToNextStep()`. Если же команда совпадает с одной из команд из этого списка, происходит переключение слотов-обработчиков команд. Каждый из подобных слотов выполняет различные сценарии, требующие ввода дополнительных данных пользователя. Например, команда `passwd` при смене пароля некоторого пользователя попросит ввести новый пароль, подтвердить его и вывести сообщение об успешном обновлении пароля. В случае ошибок `passwd` также выводит предупреждения об ошибках. Эти слоты полностью эмитируют поведения подобных команд;
- Метод `slotCommandButtonPressed()` является обработчиком сигнала нажатия кнопки команды. При нажатии на кнопку происходит считывание текста кнопки в эмулятор консоли, затем требуется нажатие пользователя на кнопку `Enter` для начала ее обработки;
- Слоты обработки нажатий на кнопки в меню реализуют вспомогательные функции для работы пользователя в приложении, например, кнопка *«Начать обучение»* сбрасывает текущее состояние до начального;
- Метод `LoadSaveFile(QString file_name)` производит загрузку сохранения файла (сохранение проводится в формате XML), проверку на корректность файла сохранение и на загрузку сохранения в программу;
- Метод `SaveFile(QString file_name)` сохраняет текущее состояние в XML.

4 Разработка интерактивных уроков по СУБД PostgreSQL

4.1 Обзор СУБД PostgreSQL

PostgreSQL [9] - это свободно распространяемая объектно-реляционная система управления базами данных (ORDBMS), наиболее развитая из открытых СУБД в мире и являющаяся реальной альтернативой коммерческим базам данных.

PostgreSQL обладает рядом преимуществ по сравнению с остальными СУБД:

- Надежность. PostgreSQL является проверенным и доказанным фактом и обеспечивается следующими возможностями:
  - полное соответствие принципам ACID - атомарность, непротиворечивость, изолированность, сохранность данных.
  - многоверсионность (Multiversion Concurrency Control, MVCC) используется для поддержания согласованности данных в конкурентных условиях, в то время как в традиционных базах данных используются блокировки.
  - наличие Write Ahead Logging (WAL) - общепринятый механизм протоколирования всех транзакций, что позволяет восстановить систему после возможных сбоев Point in Time Recovery (PITR) - возможность восстановления базы данных (используя WAL) на любой момент в прошлом, что позволяет осуществлять непрерывное резервное копирование кластера PostgreSQL.
  - Репликация.
  - Целостность данных является сердцем PostgreSQL. Помимо MVCC, PostgreSQL поддерживает целостность данных на уровне схемы - это внешние ключи (foreign keys), ограничения (constraints).
  - Модель развития PostgreSQL, которая абсолютно прозрачна для любого, так как все планы, проблемы и приоритеты открыто обсуждаются. Пользователи и разработчики находятся в постоянном диалоге через мэйлинг листы. Все предложения, патчи проходят тщательное тестирование до принятия их в программное дерево. Большое количество бета-тестеров способствует тестированию версии до релиза и вычищению мелких ошибок.
  - Открытость кодов PostgreSQL означает их абсолютную доступность для любого, а либеральная BSD лицензия не накладывает никаких ограничений на использование кода.
- Производительность. PostgreSQL основывается на использовании индексов, интеллектуальном планировщике запросов, тонкой системы блокировок, системе управления буферами памяти и кэширования, превосходной масштабируемости при конкурентной работе.
- Расширяемость PostgreSQL означает, что пользователь может настраивать систему путем определения новых функций, агрегатов, типов, языков, индексов и операторов. Объектно-ориентированность PostgreSQL позволяет перенести логику приложения на уровень базы данных, что сильно упрощает разработку клиентов, так как вся бизнес логика находится в базе данных. Функции в PostgreSQL однозначно определяются названием, количеством и типами аргументов.
- Поддержка SQL, кроме основных возможностей, присущих любой SQL базе данных, PostgreSQL поддерживает:
  - Очень высокий уровень соответствия ANSI SQL 92, ANSI SQL 99 и ANSI SQL 2003.

- Схемы, которые обеспечивают пространство имен на уровне SQL. Схемы содержат таблицы, в них можно определять типы данных, функции и операторы.
- Subqueries - подзапросы (subselects), полная поддержка SQL92. Подзапросы делают язык SQL более гибким и зачастую более эффективным.
- Outer Joins - внешние связки (LEFT,RIGHT, FULL).
- Rules - правила, согласно которым модифицируется исходный запрос.
- Views - представления, виртуальные таблицы.
- Cursors - курсоры, позволяют уменьшить трафик между клиентом и сервером, а также память на клиенте, если требуется получить не весь результат запроса, а только его часть.
- Table Inheritance - наследование таблиц, позволяющее создавать объекты, которые наследуют структуру родительского объекта и добавлять свои специфические атрибуты.
- Prepared Statements (преподготовленные запросы) - это объекты, живущие на стороне сервера, которые представляют собой оригинальный запрос после команды PREPARE, который уже прошел стадии разбора запроса (parser), модификации запроса (rewriting rules) и создания плана выполнения запроса (planner), в результате чего, можно использовать команду EXECUTE, которая уже не требует прохождения этих стадий. Для сложных запросов это может быть большим выигрышем.
- Stored Procedures - серверные (хранимые) процедуры позволяют реализовывать бизнес логику приложения на стороне сервера. Кроме того, они позволяют сильно уменьшить трафик между клиентом и сервером.
- Savepoints (nested transactions) - в отличие от "плоских транзакций", которые не имеют промежуточных точек фиксации, использование savepoints позволяет отменять работу части транзакции, например вследствии ошибочно введенной команды, без влияния на оставшуюся часть транзакции. Это бывает очень полезно для транзакций, которые работают с большими объемами данных.
- Права доступа к объектам системы на основе системы привилегий. Владелец объекта или суперюзер может как разрешать доступ (GRANT), так и отменять (REVOKE).
- Система обмена сообщениями между процессами - LISTEN и NOTIFY позволяют организовывать событийную модель взаимодействия между клиентом и сервером (клиенту передается название события, назначенное командой notify и PID процесса).
- Триггеры позволяют управлять реакцией системы на изменение данных (INSERT, UPDATE, DELETE), как перед самой операцией (BEFORE), так и после (AFTER). Во время выполнения триггера доступны специальные переменные NEW (запись, которая будет вставлена или обновлена) и OLD (запись перед обновлением).
- Cluster table - упорядочивание записей таблицы на диске согласно индексу, что иногда за счет уменьшения доступа к диску ускоряет выполнение запроса.

- Богатый набор типов данных PostgreSQL включает:
  - Символьные типы (character(n)) как определено в стандарте SQL и тип text с практически неограниченной длиной.
  - Numeric тип поддерживает произвольную точность, очень востребованную в научных и финансовых приложениях.
  - Массивы согласно стандарту SQL:2003.

  - Большие объекты (Large Objects) позволяют хранить в базе данных бинарные данные размером до 2Gb.
  - Геометрические типы (point, line, circle,polygon, box,...) позволяют работать с пространственными данными на плоскости.
  - ГИС (GIS) типы в PostgreSQL являются доказательством расширяемости PostgreSQL и позволяют эффективно работать с трехмерными данными.
  - Сетевые типы (Network types) поддерживают типы данных inet для IPV4, IPV6, а также cidr (Classless Internet Domain Routing) блоки и macaddr
  - Композитные типы (composite types) объединяют один или несколько элементарных типов и позволяют пользователям манипулировать со сложными объектами.
  - Временные типы (timestamp, interval, date, time) реализованы с очень большой точностью.
  - Псевдотипы serial и bigserial позволяют организовать упорядоченную последовательность целых чисел (AUTO_INCREMENT у некоторых СУБД).

- PostgreSQL имеет очень богатый набор встроенных функций и операторов для работы с данными.

- Поддержка 25 различных наборов символов (charsets), включая ASCII, LATIN, WIN, KOI8 и UNICODE, а также поддержка locale, что позволяет корректно работать с данными на разных языках.

- Поддержка NLS(Native Language Support) - документация, сообщения об ошибках доступны на различных языках, включая японский, немецкий, итальянский, французский, русский, испанский, португальский, словенский, словацкий и несколько диалектов китайского языков.

- Интерфейсы в PostgreSQL реализованы для доступа к базе данных из ряда языков (C, C++, C# ,python, perl, ruby ,php, Lisp и другие) и методов доступа к данным (JDBC, ODBC).

- Процедурные языки позволяют пользователям разрабатывать свои функции на стороне сервера, тем самым переносить логику приложения на сторону базы данных, используя языки программирования, отличные от встроенных SQL и C. К настоящему времени поддерживаются (включены в стандартный дистрибутив) PL/pgSQL, pl/Tcl, Pl/Perl и pl/Python. Кроме них, существует поддержка PHP, Java, Ruby, R, shell.

- Простота использования всегда являлась важным фактором для разработчиков. Предоставляются следующие средства взаимодействия пользователя с базой данных:

  - Утилита psql (входит в дистрибутив) предоставляет удобный интерфейс для работы с базой данных, содержит краткий справочник по SQL, облегчает ввод команд (используя стрелки для повтора и табулятор для расширения), поддерживает историю и буфер запросов, а также позволяет работать как в интерактивном режиме, так и потоковом режиме.
  - phpPgAdmin (лицензия GPL) представляет возможность с помощью веб браузера администрировать PostgreSQL кластер.
  - pgAdmin III (GNU Artistic license) предоставляет удобный интерфейс для работы с базами данных PostgreSQL и работает под Linux, FreeBSD и Windows 2000/XP.
  - PgEdit - программная среда для разработки приложений и SQL-редактор, доступна для Windows и Mac.

- Безопасность данных также является важнейшим аспектом любой СУБД. В PostgreSQL она обеспечивается 4-мя уровнями безопасности:
  - PostgreSQL нельзя запустить под привилегированным пользователем - системный контекст.
  - SSL,SSH шифрование трафика между клиентом и сервером - сетевой контекст.
  - сложная система аутентификации на уровне хоста или IP адреса/подсети. Система аутентификации поддерживает пароли, шифрованные пароли, Kerberos, IDENT и прочие системы, которые могут подключаться используя механизм подключаемых аутентификационных модулей.
  - Детализированная система прав доступа ко всем объектам базы данных, которая совместно со схемой, обеспечивающая изоляцию названий объектов для каждого пользователя, PostgreSQL предоставляет богатую и гибкую инфраструктуру.

4.2 Реализация уроков по СУБД PostgreSQL

4.2.1 Пример выполнения шага 1.3

На примере шага по установке пароля пользователю `postgres` продемонстрируем работу программы интерактивных уроков. В листинге ниже представлено описание шага по установке пароля пользователю `postgres`.

```
<step name="Установка пароля пользователя postgres.">
  <description>
   &lt; style type="text/css"&gt;
   p { text-indent: 30px;}
   &lt;/style&gt;
    &lt;p&gt;При установке СУБД автоматически создается учетная запись postgres,
    от имени которого будут выполняться административные работы. &lt;/p&gt;
    &lt;p&gt; &lt;b&gt;passwd &lt;/b&gt; - утилита UNIX-систем управления
    пользовательскими паролями.&lt;/p&gt;

    &lt;p&gt;Общий синаксис следующий: &lt;br&gt; &lt;norb&gt; &lt;code&gt;
    passwd [параметры][имя пользователя] &lt;/code&gt; &lt;/norb&gt; &lt;/p&gt;

    &lt;p&gt;С помощью команды &lt;code&gt;passwd postgres &lt;/code&gt;
    установите пароль для пользователя postgres. &lt;/p&gt;
  </description>

  <command>passwd postgres</command>
  <default_prompt>[root@test psql]# </default_prompt>

  <prompt>Новый пароль : </prompt>
  <prompt>Повторите ввод нового пароля : </prompt>

  <result>Смена пароля для пользователя postgres. </result>
  <result>passwd: все токены проверки подлинности успешно обновлены.
  </result>

  <exception>Пароль не указан
  passwd: Ошибка при операциях с маркером проверки подлинности
  </exception>
  <exception>Извините, пароли не совпадают </exception>
  <exception>passwd: Использовано максимальное число попыток, заданное для службы
```

```
33     </exception>
34
35   </step>
```

Вся информация в поля программы устанавливается с помощью соответствующих XQuery-запросов в XML-файл уроков.

Можно заметить, что текст тега `description` описан с помощью тегов HTML. Из-за того, что в XML зарезервированными являются символы «`<`» и «`>`» использованы ссылки на сущность, например `<p>`, что означает `<p>` или новый абзац. Текст в поле `description` устанавливается с помощью метода класса `QTextEdit::setHtml(QString text)`, таким образом в этом поле достигается равнение и выделение текста. Ниже *на рисунке. 4.1* можно увидеть, как преобразуется текст тега `description` при переходе на шаг 1.3.

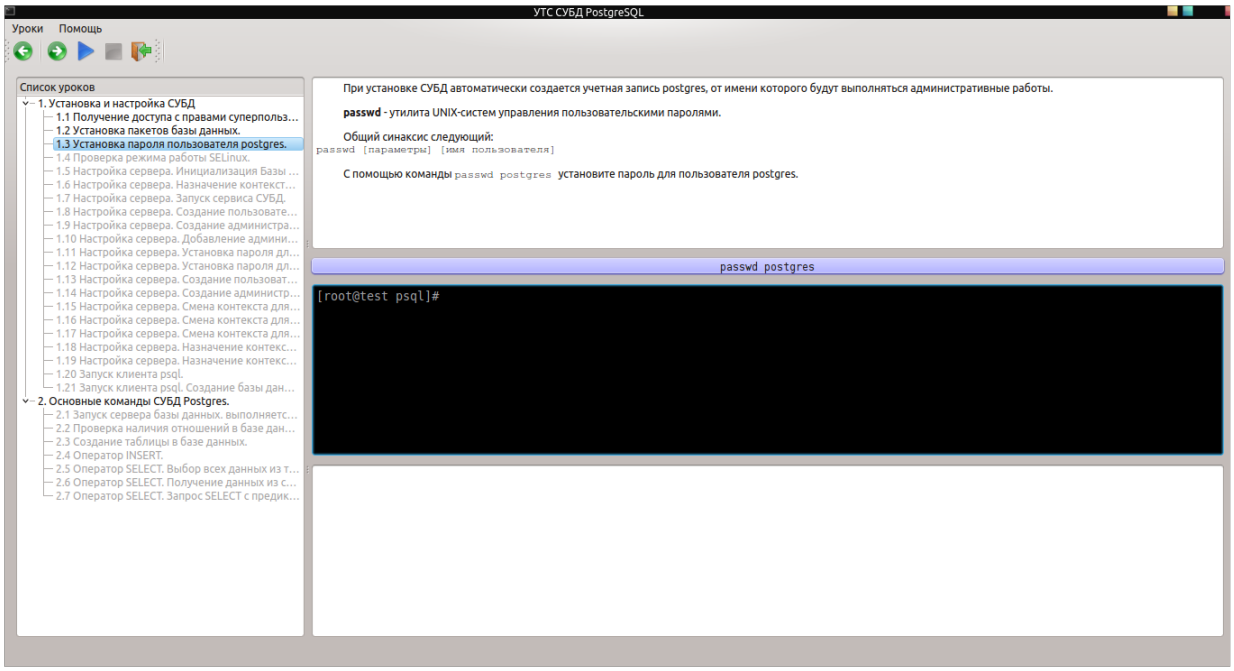

Рис. 4.1: Окно программы при начале выполнения шага 1.3

Введенная команда проверяется сначала в стандартном обработчике команды на правильность ввода. Далее она ищется в списке команд, требующих специфической обработки. Если команда найдена в списке, то происходит переключение слотов обработки команд и команда обрабатывается в специализированном слоте. Иначе выполняется запрос XQuery для тега `result` и выполняется переход на следующий шаг. В случае, если пользователь неправильно вводит команду, программа интерактивных уроков выводит предупреждение об ошибке в эмулятор консоли, что можно увидеть *на рисунке 4.2.* [2]

[2]Здесь была специально введена неверная команда `password postgres` вместо эталонной `passwd postgres`

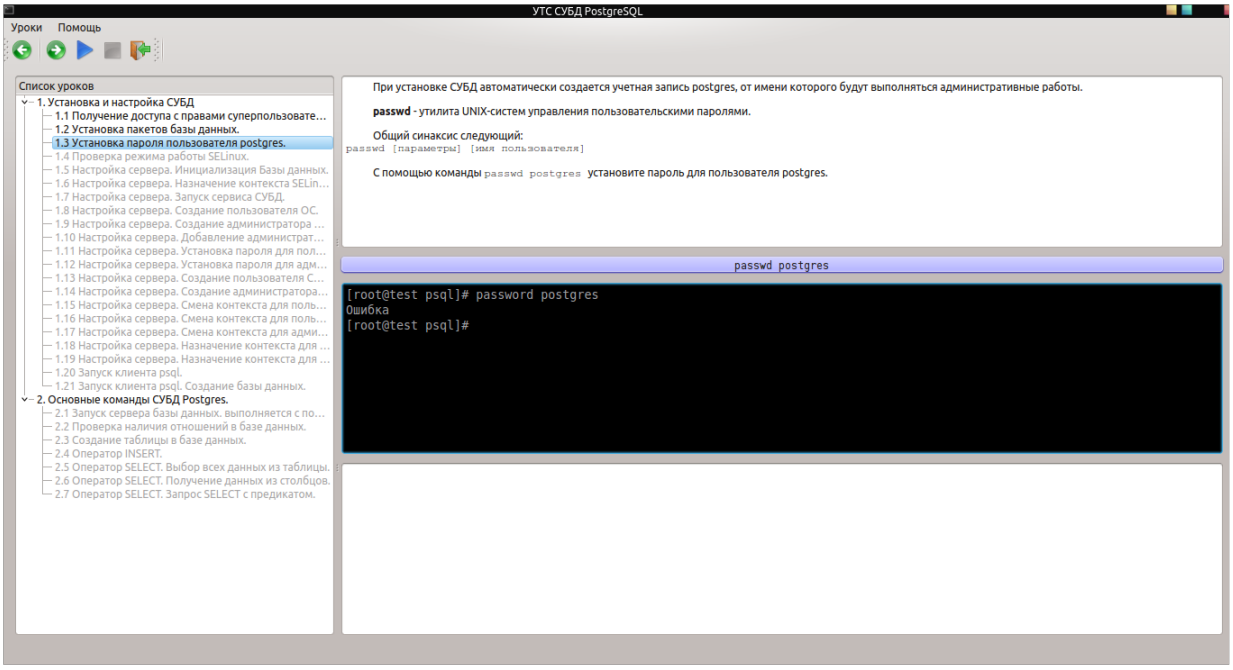

Рис. 4.2: Неправильный ввод команды

Если же пользователь введет правильно команду, начинается диалог с пользователем (программа полностью эмитирует работу команды `passwd` операционной системы linux), как это показано на *рисунке 4.3*. В это время программа получает данные с помощью XQuery-запросов тегов `prompt[1]` и `result[1]` из XML-файла урока, часть которого приведена на листинге выше. Результаты выполнения запросов следующие:

```
Смена пароля для пользователя postgres.
```

для тега `result[1]` и

```
Новый пароль :
```

для тега `prompt[1]`. Эти результаты запроса выводятся в эмулятор консоли. Так пользователю предлагается установить пароль для пользователя СУБД `postgres`.

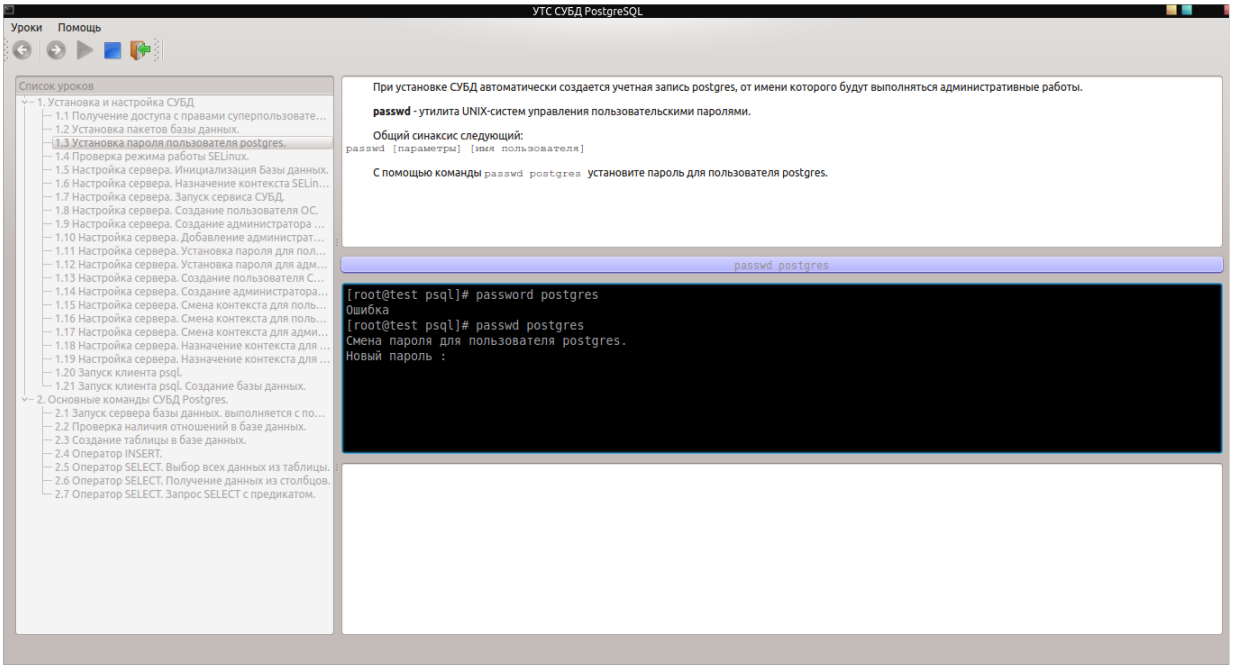

Рис. 4.3: Начало выполнения шага 1.3

По окончании ввода нового пароля требуется его повторить (*см. рисунок 4.4*). При этом выполняется XQuery-запрос тега `prompt[2]` и результат запроса выводится в эмулятор консоли.

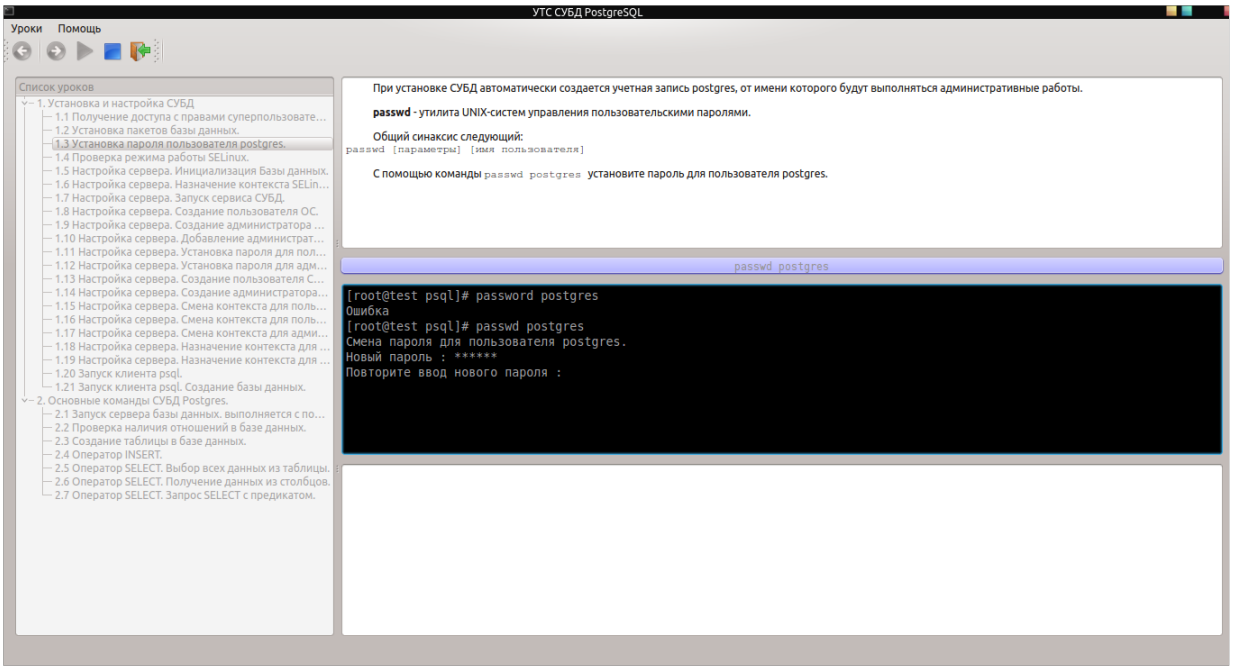

Рис. 4.4: Подтверждение пароля

В случае неверно введенного повторения пароля программа выводит предупреждение, выполнив XQuery-запрос для тега `exeption[2]` и в эмуляторе консоли будут следующие строки

(*см. рисунок 4.5*):

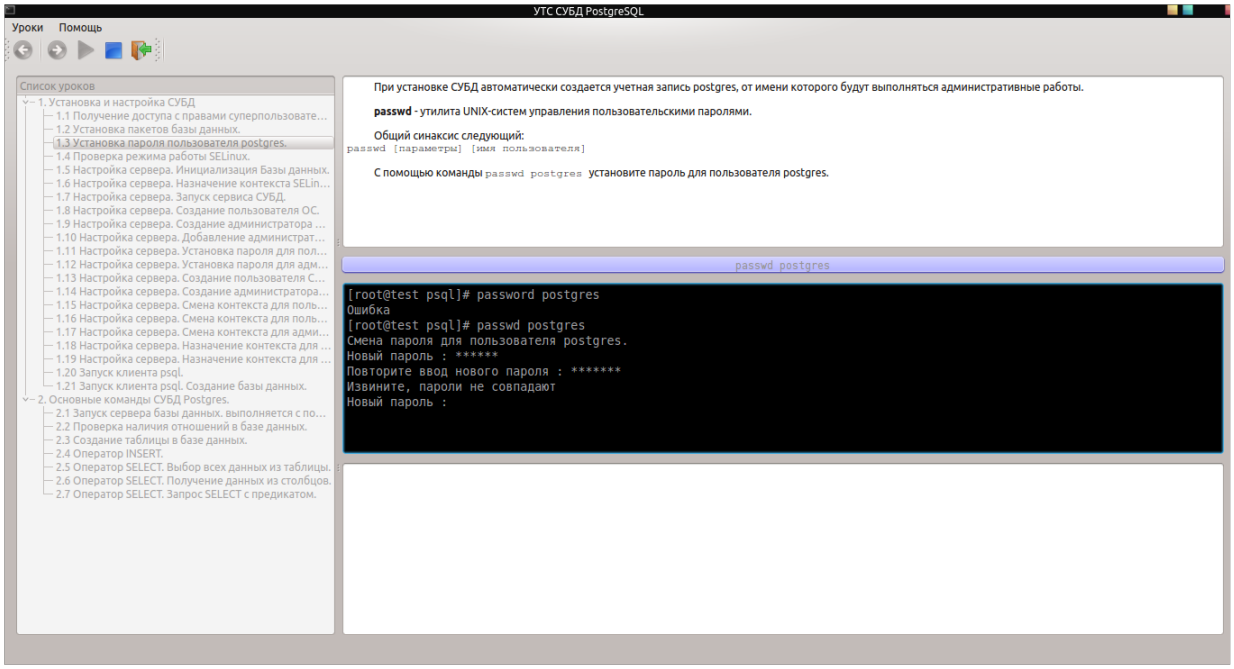

Рис. 4.5: Вывод ошибки при несовпадении паролей

Аналогично, программа предупреждает пользователя об отсутствии пароля. В этом случае выполняется XQuery-запрос для тега `exeption[1]` и выполняется выход из слота обработки команды `passwd` согласно работе программы `passwd` в операционной системе linux (*см. рисунок 4.6*).

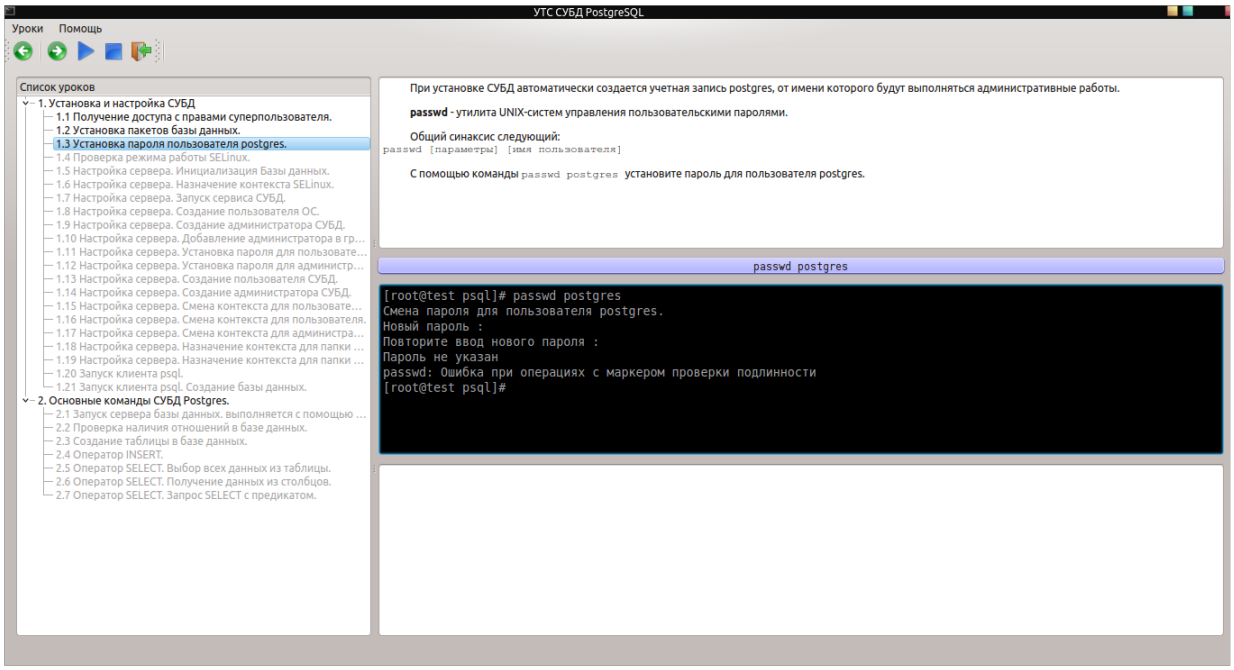

Рис. 4.6: Ошибка при отсутствии пароля

В случае правильных выполнения действий пользователя осуществляется переход на следующий шаг (в нашем случае на 1.4), результат перехода можно посмотреть *на рисунке 4.7*.

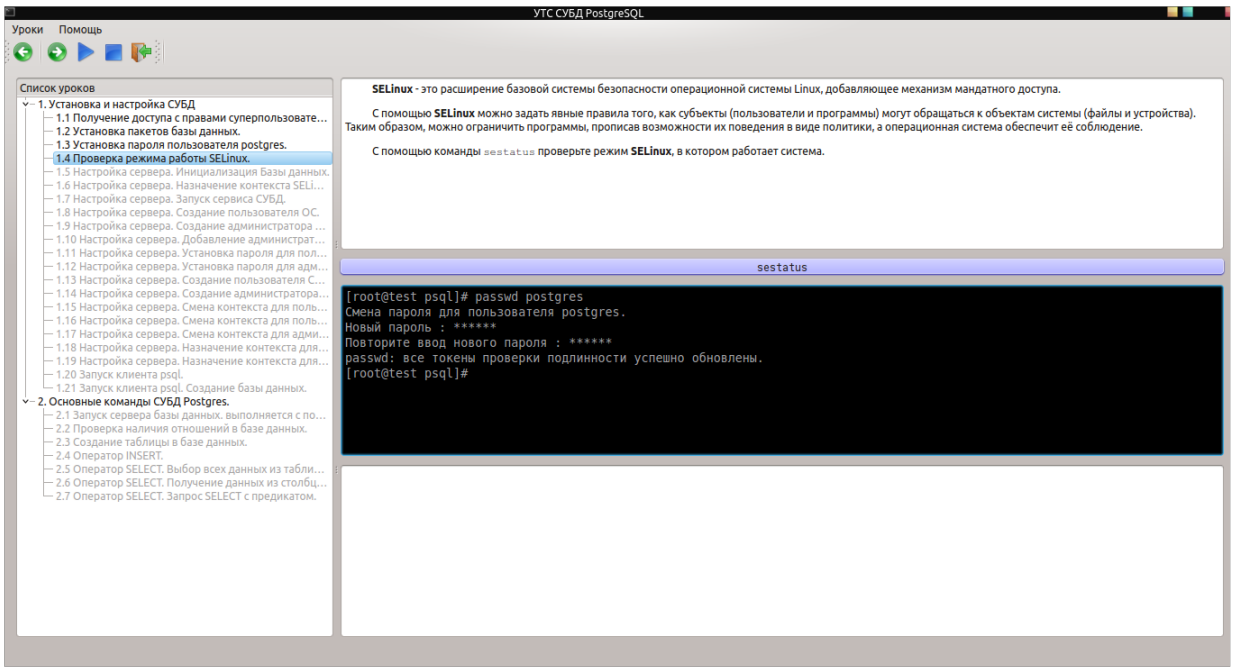

Рис. 4.7: Успешное выполнение шага 1.3 и переход на шаг 1.4

В данном случае пользователь уведомляется с помощью вывода в консоли строки о том, что пароль успешно обновлен и выполняется переход на следующий шаг. При переходе меняется положение текущего элемента в дереве уроков, а выполняются XQuery-запросы для обновления полей программы.

4.2.2 Пример выполнения шага 2.3

Шаг 2.3 иллюстрирует команду языка SQL `CREATE TABLE` по созданию таблицы в базе данных. Ниже в листинге приведено описание этого шага в XML.

```
<step name="Создание таблицы в базе данных.">
 <description>
  &lt; style type="text/css"&gt;
  p { text-indent: 30px;}
  &lt;/style&gt;
  &lt;p&gt;Для создания таблицы используется SQL-оператор &lt;b&gt;
  CREATE TABLE &lt;/b&gt;,наиболее простейший синтаксис которого следующий:&lt;/p&gt;

  &lt;code&gt; CREATE TABLE имя_таблицы (имя_колонки_1 тип_данных_колонки_2,
  имя_колонки_2 тип_данных_колонки_2, ..., имя_колонки_N тип_данных_колонки_N);
  &lt;/code&gt;

  &lt;p&gt;Вы можете создать таблицу, указав ее название, а затем имена полей и их типы.
  Создайте таблицу &lt;code&gt; students &lt;/code&gt; с полями: &lt;br&gt;
    &lt;code&gt;student_id     : integer &lt;br&gt;
    student_name   : varchar  &lt;br&gt;
    group_id       : integer &lt;/code&gt; &lt;/p&gt;
```

```
</description>

<command>CREATE TABLE students (student_id integer, student_name varchar,
 group_id integer);</command>

<default_prompt>test-# </default_prompt>

<result>CREATE TABLE </result>

<database> No relations found.</database>

</step>
```

Как и в первом примере, рассмотренном в пункте 4.3.1, тег `description` описан с помощью тегов языка HTML. Но в отличие от первого примера, в описании данного шага присутствует тег `database`, отвечающий за вывод структуры базы данных в соответствующее поле программы. *На рисунке 4.8* представлено окно программы при начале выполнения шага 2.3.

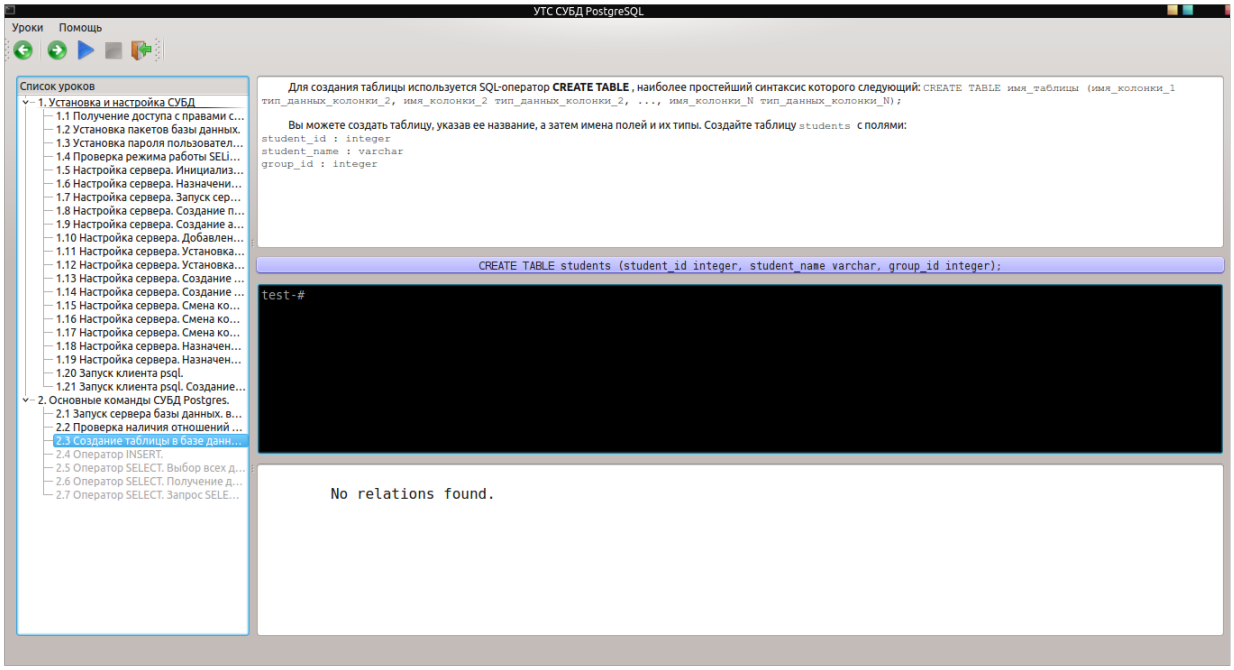

Рис. 4.8:

При выполнении шага поле, отвечающее за отражение структуры базы данных, изменяется. Это можно увидеть *на рисунке 4.9.*

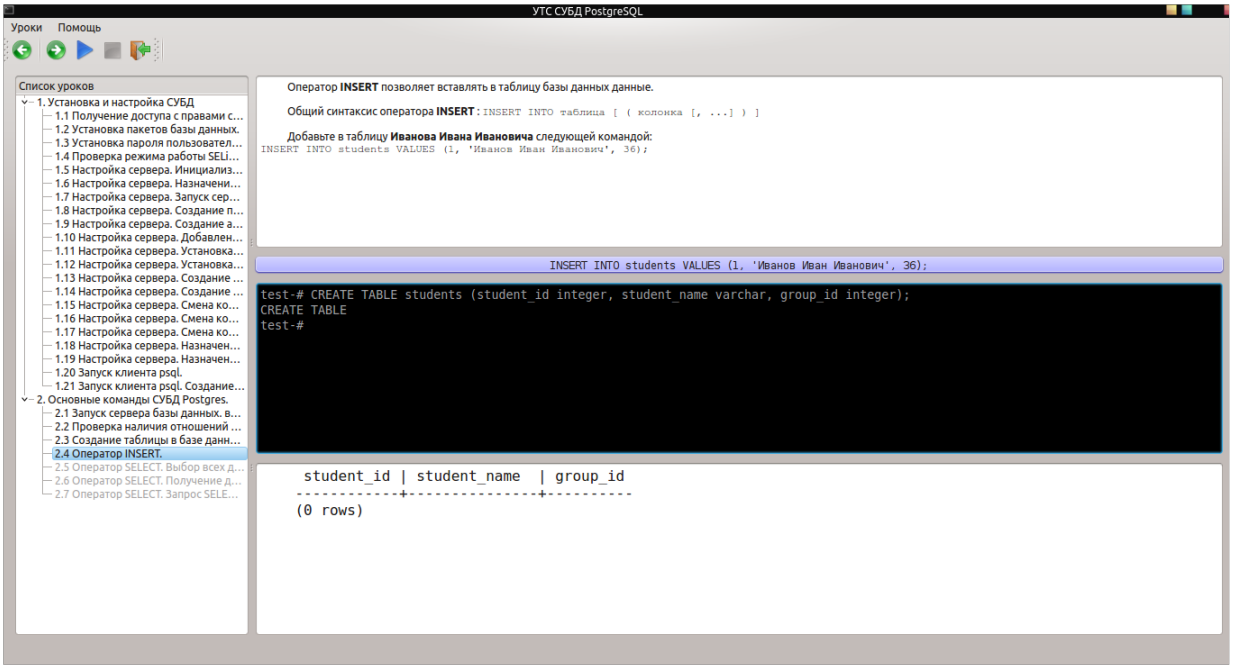

Рис. 4.9: Успешное выполнение шага 2.3

Таким образом, данная программа позволяет обучить пользователя порядку настройки сервера СУБД, а также обучить синтаксису команд языка SQL. Благодаря наличию краткой справке по каждой выполняемой пользователем команды пользователю не требуется прибегать к чтению документации или прочей литературы, что тем самым, упрощает процесс понимания принципов работы с СУБД.

Заключение

Данная учебно-исследовательская работа посвящена созданию кроссплатформенного интерактивного учебника по СУБД.

Были достигнуты основные результаты:

1. Разработана иерархическая структура данных обучающих уроков на языке разметки XML;
2. Разработана кроссплатформенная интерактивная программа обучающих уроков, использующую данную структуру и обрабатывающую ее;
3. Возможности программы интерактивных уроков продемонстрированы на примере курса по свободно распространяемой СУБД PostgreSQL для операционной системы Linux.

В целом можно сделать вывод о достижении поставленной цели работы. В продолжение темы учебно-исследовательской работы можно реализовать возможность использования полноценной консоли в программе интерактивных уроков по СУБД с помощью виртуальных машин.

Список литературы

[1] Интерактивный урок по git. [электронный ресурс] — http://try.github.com 7

[2] Интерактивный учебник по SQL. [электронный ресурс] — http://www.sql-tutorial.com 7

[3] Язык разметки XML. [электронный ресурс] — http://ru.wikipedia.org/wiki/XML 10

[4] Язык запросов XPath. [электронный ресурс]. — http://codingcraft.ru/xpath.php 11

[5] XQuery: что это такое. [электронный ресурс]. — http://www.iso.ru/rus/document5992.phtml 12

[6] Шлее М. — Qt 4.5. Профессиональное программирование на C++. — СПб.: БВХ-Петербург, 2010. — 896с.: ил. + DVD — (В подлиннике), ISBN: 978-5-9775-0398-3 14

[7] Qt 4.8 — документация. [электронный ресурс]. — http://qt-project.org/doc/qt-4.8/ 15

[8] Основы языка SQL. [электронный ресурс]. — http://citforum.ru/programming/32less/les44.shtml 18

[9] Что такое PostgreSQL? [электронный ресурс]. — http://postgresql.ru.net/manual/intro-whatis.html

29