Министерство образования Республики Беларусь
Учреждение образования
«Белорусский государственный университет
информатики и радиоэлектроники»

Факультет компьютерного проектирования

Кафедра электронной техники и технологии

Б. А. Тонконогов, О. П. Высоцкий

ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ЭЛЕКТРОННЫХ СРЕДСТВ. ЛАБОРАТОРНЫЙ ПРАКТИКУМ

Рекомендовано УМО по образованию в области информатики и радиоэлектроники в качестве пособия для специальности 1-39 02 02 «Проектирование и производство программно-управляемых электронных средств»

Минск БГУИР 2017

УДК 004.42(076.5)
ББК 32.973.26-018.2я73
Т57

Р е ц е н з е н т ы:

кафедра интеллектуальных систем
Белорусского национального технического университета
(протокол №8 от 08.02.2017);

доцент кафедры энергоэффективных технологий учреждения образования
«Международный государственный экологический институт имени
А. Д. Сахарова» Белорусского государственного университета,
кандидат технических наук, доцент В. П. Яновский

Тонконогов, Б. А.

Т57 Программное обеспечение электронных средств. Лабораторный практикум : пособие / Б. А. Тонконогов, О. П. Высоцкий. – Минск : БГУИР, 2017. – 67 с. : ил.

ISBN 978-985-543-375-1.

Содержит теоретические и практические аспекты, касающиеся технологии и средств проектирования, реализации программного обеспечения для мобильных электронных устройств и встраиваемых систем, формирования у студентов представлений о типах, функциональности и назначении специализированных программных модулей и средств, а также навыков работы с системами управления базами данных и интегрированными средами разработки для дальнейшего квалифицированного использования в учебном процессе, научных исследованиях и практической работе.

Предназначено для студентов специальности «Проектирование и производство программно-управляемых электронных средств» дневной и заочной (дистанционной) форм получения высшего образования. Может быть использовано студентами родственных специальностей и специалистами, занимающимися проектированием и программной реализацией приложений, а также созданием широко- и узкоспециализированных программных комплексов.

УДК 004.42(076.5)
ББК 32.973.26-018.2я73

ISBN 978-985-543-375-1

СОДЕРЖАНИЕ

Лабораторная работа №1

ИЗУЧЕНИЕ АППАРАТНОЙ И ПРОГРАММНОЙ ЧАСТЕЙ БАЗОВОГО КОНСТРУКТИВА ПЛАТФОРМЫ ARDUINO

Цель работы: изучить средство разработки программируемых электронных устройств Arduino (свойства и режимы работы портов ввода/вывода) и ознакомиться с возможностями языка программирования C/C++ при реализации программных модулей для указанных устройств.

1.1. Теоретические сведения

Arduino – это инструмент для проектирования электронных устройств (электронный конструктор), более плотно взаимодействующих с окружающей физической средой, чем стандартные персональные компьютеры, которые фактически не выходят за рамки виртуальности (рис. 1.1).

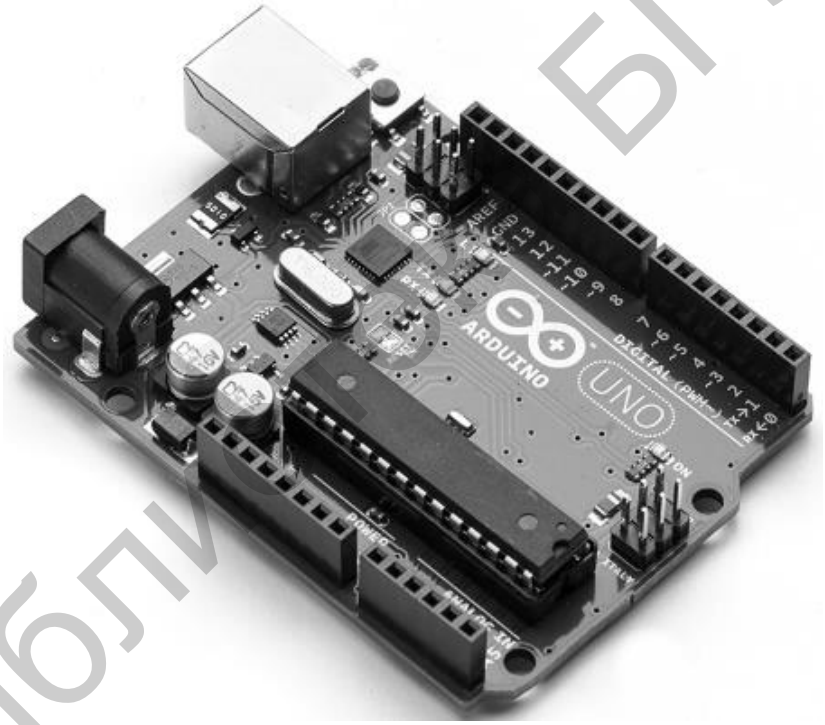


Рис. 1.1. Внешний вид модификации Arduino Uno

Arduino применяется для создания электронных устройств с возможностью приема сигналов от различных цифровых и аналоговых датчиков, которые могут быть подключены к нему, и управления различными исполнительными устройствами. Проекты устройств, основанные на Arduino, могут работать самостоятельно или взаимодействовать с программным обеспечением на компьютере (например, Flash, Processing и MaxMSP).

Все платы Arduino содержат основные компоненты, необходимые для программирования и совместной работы с другими схемами (рис. 1.2).

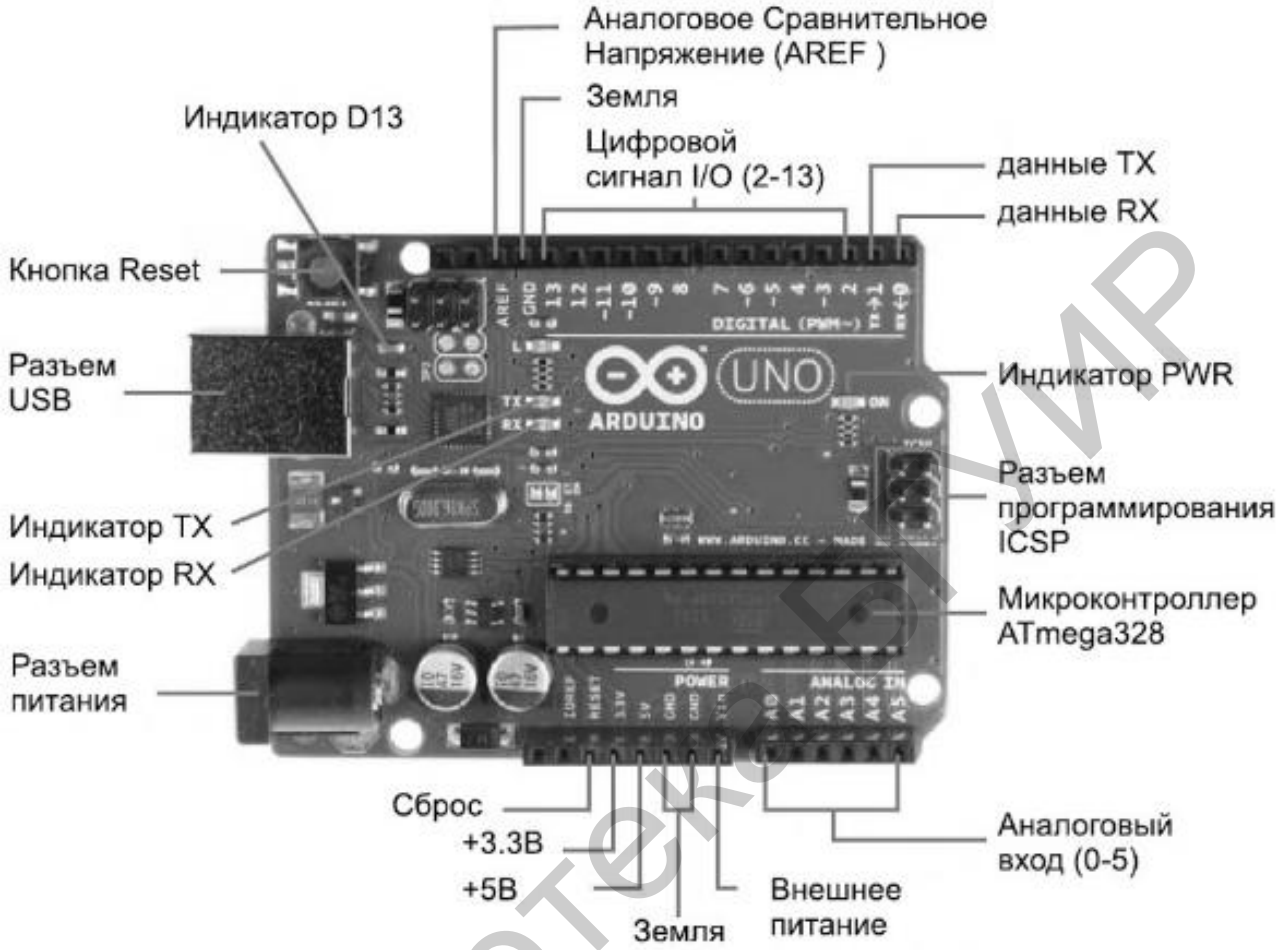


Рис. 1.2. Компоненты платы Arduino Uno

Большинство плат Arduino оснащено светодиодом отладки (Debug), подсоединенным к контакту 13, который позволит реализовать программу с мигающим светодиодом без дополнительных компонентов.

Краткая характеристика данной платы представлена в табл. 1.1. Особенностью микроконтроллерных плат семейства Arduino является возможность загружать в микроконтроллер программу, написанную на языке высокого уровня С непосредственно по интерфейсу USB, не используя специальные программаторы.

Платформа Arduino пользуется популярностью во всем мире благодаря удобству и простоте языка программирования, а также открытой архитектуре и программному коду.

Цоколевка микросхемы ATmega328 представлена на рис. 1.3.

Таблица 1.1

Характеристики микроконтроллерной платы Arduino Uno

Характеристика	Значение
Микроконтроллер	ATmega328
Рабочее напряжение	5 В
Входное напряжение (рекомендуемое)	7–12 В
Входное напряжение (предельное)	6–20 В
Цифровые входы/выходы	14 (6 из которых могут использоваться как выходы ШИМ)
Аналоговые входы	6
Постоянный ток через вход/выход	40 мА
Постоянный ток для вывода 3,3 В	50 мА
Флеш-память	32 Кбайт (ATmega328), из которых 0,5 Кбайт используются для загрузчика
ОЗУ	2 Кбайт (ATmega328)
EEPROM	1 Кбайт (ATmega328)
Тактовая частота	16 МГц

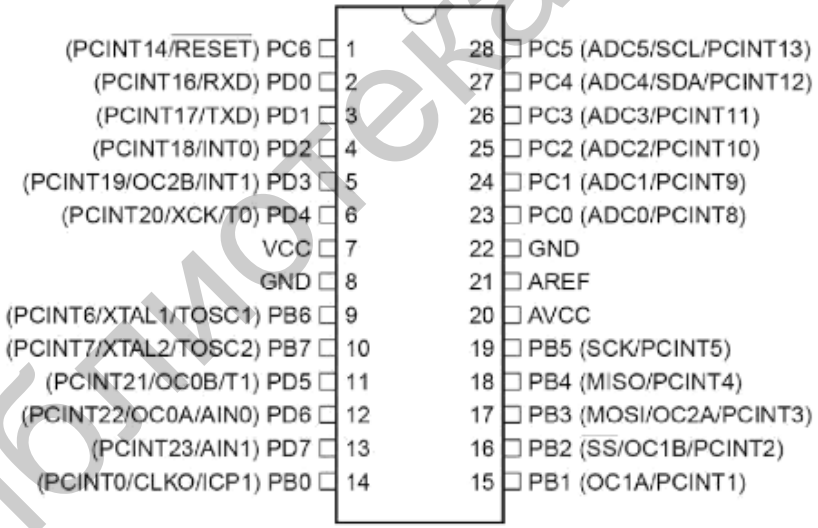


Рис. 1.3. Цоколевка микросхемы ATmega328

На приведенном рисунке AREF – вывод для подключения опорного напряжения для аналого-цифрового преобразователя (<5 В) и GND – заземление.

1.1.1. Цифровые входы

Схема подключения выводов ATmega368 к пронумерованным контактам Arduino представлена на рис. 1.4.

Выводы платформы Arduino могут работать как входы или как выходы. Также необходимо обратить внимание на то, что большинство аналоговых входов Arduino (ATmega) могут конфигурироваться и работать так же, как и цифровые порты ввода/вывода.

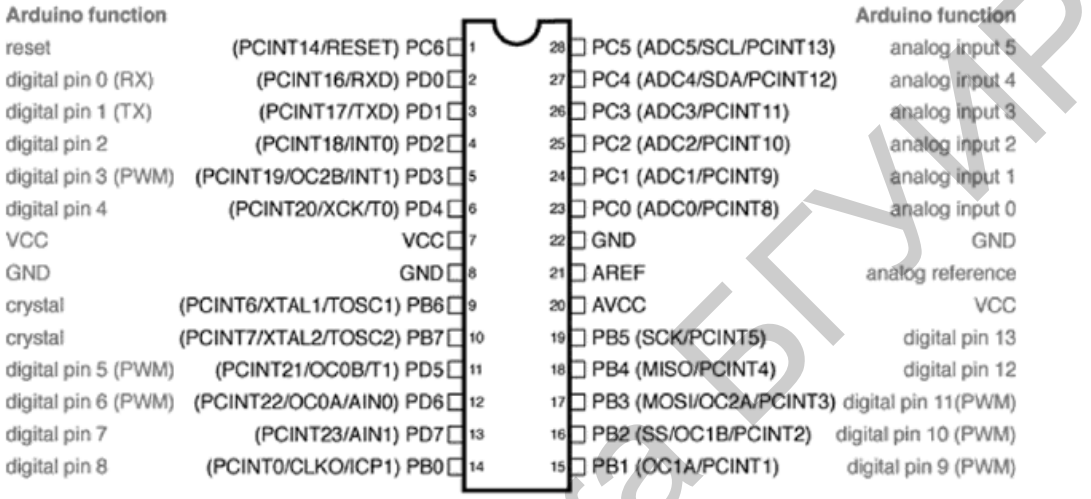


Рис. 1.4. Схема подключения выводов ATmega328 к пронумерованным контактам Arduino

Выводы Arduino (ATmega) стандартно настроены как порты ввода, таким образом, не требуется явной декларации в функции pinMode(). Сконфигурированные порты ввода находятся в высокоимпедансном состоянии. Это означает, что порт ввода дает слишком малую нагрузку на схему, в которую он включен. Эквивалентом внутреннему сопротивлению будет резистор 100 МОм, подключенный к выводу микросхемы. Таким образом, для перевода порта ввода из одного состояния в другое требуется низкое значение тока. Это позволяет применять выводы микросхемы для подключения емкостного датчика касания, фотодиода, аналогового датчика со схемой, похожей на RC-цепь и так далее.

С другой стороны, если к данному выводу ничего не подключено, то значения на нем будут принимать случайные величины, наводимые электрическими помехами или емкостной взаимосвязью с соседним выводом.

Если на порт ввода не поступает сигнал, то в данном случае рекомендуется задать порту известное состояние. Это делается добавлением согласующих резисторов 10 кОм, подключающих вход либо к +5 В, либо к «земле».

Микроконтроллер ATmega имеет программируемые встроенные подтягивающие к питанию резисторы 20 кОм. Программирование данных резисторов осуществляется следующим образом:

```
pinMode(pin, INPUT); // назначить выводу порт ввода
digitalWrite(pin, HIGH); // включить согласующий резистор
```

Согласующий резистор пропускает ток, достаточный для того, чтобы слегка светился светодиод, подключенный к выводу, работающему как порт ввода. Также легкое свечение светодиодов означает то, что при программировании вывод не был настроен как порт вывода в функции pinMode().

Согласующие резисторы управляются теми же регистрами (внутренние адреса памяти микроконтроллера), что управляют состояниями вывода: HIGH или LOW. Следовательно, если вывод работает как порт ввода со значением HIGH, это означает включение подтягивающего к питанию резистора и конфигурация функцией pinMode() порта вывода на данном выводе микросхемы передаст значение HIGH. Данная процедура работает и в обратном направлении, то есть если вывод имеет значение HIGH, то конфигурация вывода микросхемы как порта ввода функцией pinMode() включит подтягивающий к питанию резистор.

Следует отметить, что затруднительно использовать вывод 13 микросхемы в качестве порта ввода из-за подключенных к нему светодиода и резистора. При подключении подтягивающего к питанию резистора в 20 кОм на вводе будет 1,7 В вместо 5 В, так как происходит падение напряжения на светодиоде и включенном последовательно резисторе. При необходимости использовать вывод 13 микросхемы как цифровой порт ввода требуется подключить между выводом и «землей» внешний подтягивающий резистор.

Выводы, сконфигурированные как порты вывода, находятся в низкоимпедансном состоянии. Данные выводы могут пропускать через себя достаточно высокий ток. Выводы микросхемы ATmega могут быть источником (положительным) или приемником (отрицательным) тока до 40 мА для других устройств. Такого значения тока достаточно, чтобы подключить светодиод (обязателен последовательно включенный резистор) и датчики, но недостаточно для большинства реле, соленоидов и двигателей.

Короткие замыкания выводов Arduino или попытки подключить энергоемкие устройства могут повредить выходные транзисторы вывода или весь микроконтроллер ATmega. В большинстве случаев данные действия приведут к отключению вывода на микроконтроллере, но остальная часть схемы будет работать согласно программе. Рекомендуется к выходам платформы подключать устройства через резисторы 470 Ом или 1 кОм, если устройству не требуется больший ток для работы.

1.1.2. Аналоговые входы

Микроконтроллеры ATmega, используемые в Arduino, содержат 6-канальный аналого-цифровой преобразователь. Разрешение преобразователя составляет 10 бит, что позволяет на выходе получать значения от 0 до 1 023. Основным применением аналоговых входов большинства платформ Arduino является чтение данных с аналоговых датчиков, но в то же время они имеют функциональность вводов/выводов широкого применения (GPIO) (то же, что и цифровые порты ввода/вывода 0–13).

Таким образом, при необходимости применения дополнительных портов ввода/вывода имеется возможность сконфигурировать неиспользуемые аналоговые входы.

Выводы Arduino, соответствующие аналоговым входам, имеют номера от 14 до 19. Это относится только к выводам Arduino, а не к физическим номерам выводов микроконтроллера ATmega. Аналоговые входы могут использоваться как цифровые выводы портов ввода/вывода. Например, код программы для установки вывода 0 аналогового входа на порт вывода со значением HIGH следующий:

```
pinMode(14, OUTPUT); // назначить выводу порт вывода
digitalWrite(14, HIGH); // включить резистор на выводе
аналогового входа 0
```

Выводы аналоговых входов имеют согласующие резисторы, работающие, как на цифровых выводах. Включение резисторов производится указанной выше командой, пока вывод работает как порт ввода.

Подключение резистора повлияет на величину, сообщаемую функцией analogRead() при использовании некоторых датчиков. Большинство пользователей использует согласующий резистор при применении вывода аналогового входа в его цифровом режиме.

Для вывода, работавшего ранее как цифровой порт вывода, команда analogRead будет работать некорректно. В этом случае рекомендуется сконфигурировать его как аналоговый вход. Аналогично, если вывод работал как цифровой порт вывода со значением HIGH, то обратная установка на ввод подключит согласующий резистор.

Руководство по микроконтроллеру ATmega не рекомендует производить быстрое переключение между аналоговыми входами для их чтения. Это может вызвать наложение сигналов и внести искажения в аналоговую схему. Однако после работы аналогового входа в цифровом режиме может потребоваться настроить паузу между чтением функцией analogRead() других входов.

1.1.3. Источники питания

Для большинства проектов достаточно 5-вольтового питания, получаемого по кабелю USB. Однако при необходимости разработки автономного устройства схема способна работать от внешнего источника от 6 до 20 В (рекомендуется напряжение 7–12 В). Внешнее питание может подаваться через разъем DC или на контакт V_{in}.

У Arduino есть встроенные стабилизаторы напряжения на 3,3 и 5 В:

1. Напряжение 3,3 В выведено на отдельный контакт для подключения внешних устройств.
2. Напряжение 5 В используется для всех логических элементов на плате, при этом уровень на цифровых контактах ввода/вывода находится в пределах 0–5 В.

1.1.4. Среда разработки Arduino

Написание программ для платформы Arduino и их загрузка в микроконтроллер осуществляется в среде разработки IDE Arduino. Кроме меню в верхней части окна имеется панель инструментов, которая позволяет быстро выполнять наиболее часто используемые команды (рис. 1.5).

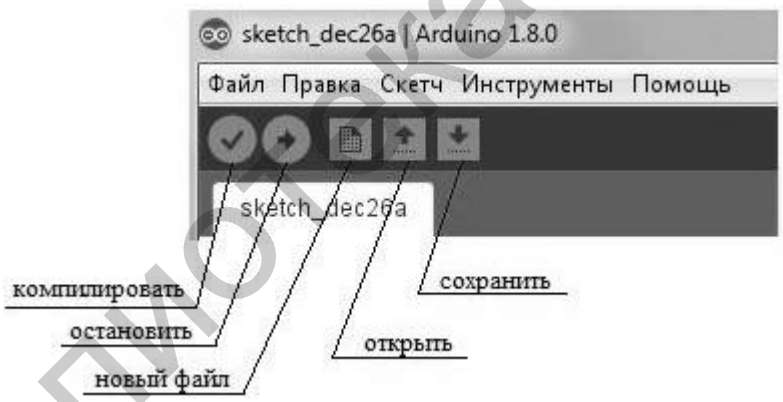


Рис. 1.5. Панель инструментов среды программирования IDE Arduino

Программа, написанная в среде Arduino, называется *скетч* (sketch). Скетч пишется в текстовом редакторе, имеющем инструменты вырезки/вставки и поиска/замены текста. Во время сохранения и экспорта проекта в области сообщений появляются пояснения, также могут отображаться возникшие ошибки. Окно вывода текста (консоль) показывает сообщения Arduino, включающие полные отчеты об ошибках и другую информацию. Кнопки панели инструментов позволяют проверить и записать программу, создать, открыть и сохранить скетч, открыть мониторинг последовательной шины:

 – проверка программного кода на ошибки, компиляция

– загрузка кода на контроллер
– создание нового скетча
– открытие меню доступа ко всем скетчам в блокноте
– сохранение скетча

Дополнительные команды сгруппированы в главном меню: *Файл*, *Правка*, *Скетч*, *Инструменты* и *Помощь*. Доступность меню определяется работой, выполняемой в определенный момент (рис. 1.6–1.8).

Средой Arduino используется принцип блокнота – стандартное место для хранения программ (скетчей). Скетчи из блокнота открываются через меню *Файл* или кнопкой открытия на панели инструментов. При первом запуске программы Arduino автоматически создается директория для блокнота. Расположение блокнота меняется через соответствующее диалоговое окно.

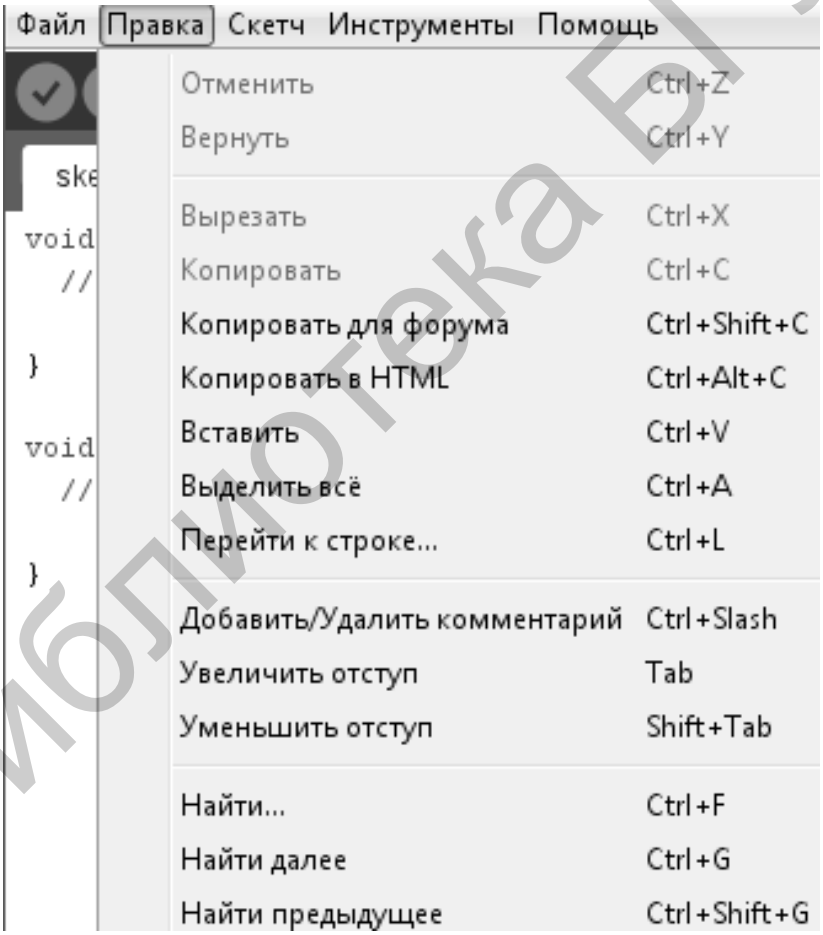


Рис. 1.6. Команды меню *Правка*

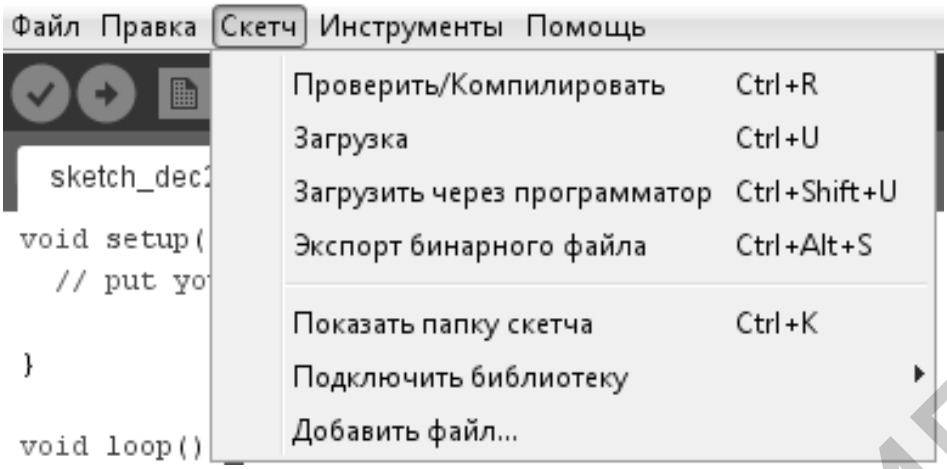


Рис. 1.7. Команды меню *Скетч*

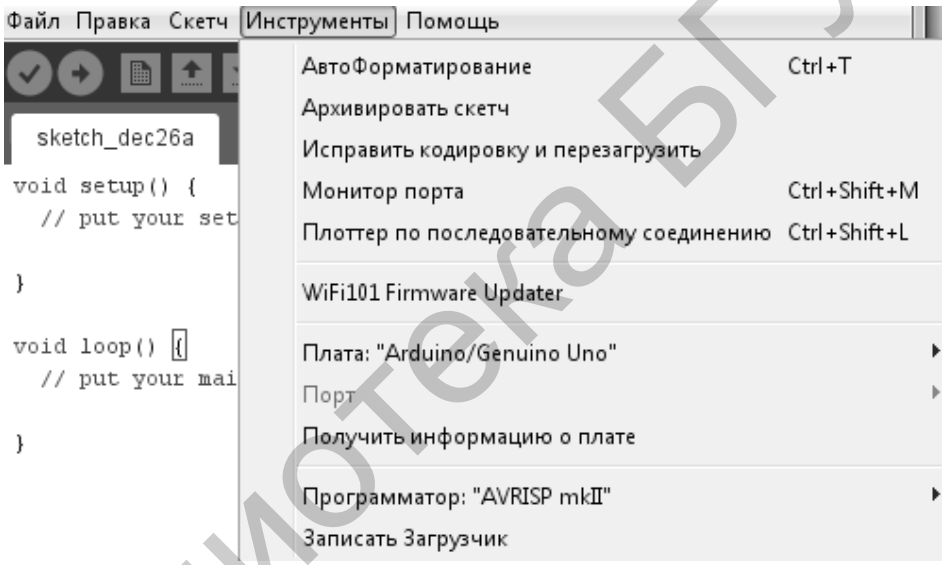


Рис. 1.8. Команды меню *Инструменты*

1.1.5. Загрузка скетча в Arduino

Перед загрузкой скетча требуется задать необходимые параметры в меню *Инструменты* для соответствующего порта. В операционной системе Mac OS последовательный порт может обозначаться как dev/tty.usbserial-1B1 (для платы USB) или /dev/tty.USA19QW1b1P1.1 (для платы последовательной шины, подключенной через адаптер Keyspan USB-to-Serial). В операционной системе Microsoft Windows порты могут обозначаться как COM1, COM2 (для платы последовательной шины) или COM4, COM5, COM7 и выше (для платы USB). Определение порта USB производится в поле последовательной шины USB-диспетчера устройств Windows. В операционной системе Linux порты могут обозначаться как /dev/ttyUSB0, /dev/ttyUSB1.

После выбора порта и платформы необходимо нажать кнопку загрузки на панели инструментов или выбрать соответствующий пункт меню *Файл*. Современные платформы Arduino перезагружаются автоматически перед загрузкой. На предыдущих платформах необходимо нажать кнопку перезагрузки. На большинстве плат во время процесса будут мигать светодиоды RX и TX. Среда разработки Arduino выведет сообщение об окончании загрузки или об ошибках.

При загрузке скетча используется загрузчик (bootloader) Arduino – небольшая программа, загружаемая в микроконтроллер на плате. Она позволяет загружать программный код без использования дополнительных аппаратных средств. Загрузчик активен в течение нескольких секунд при перезагрузке платформы и при загрузке любого из скетчей в микроконтроллер. Его работа распознается по миганию светодиода (вывод 13), например, при перезагрузке платы.

1.1.6. Библиотеки

Библиотеки добавляют дополнительную функциональность скетчам, например, при работе с аппаратной частью или при обработке данных. Для использования библиотеки необходимо выбрать соответствующее меню *Скетч*. Одна или несколько директив #include будут размещены в начале кода скетча с последующей компиляцией библиотек вместе со скетчем. Загрузка библиотек требует дополнительное место в памяти Arduino. Неиспользуемые библиотеки можно удалить из скетча, убрав директиву #include.

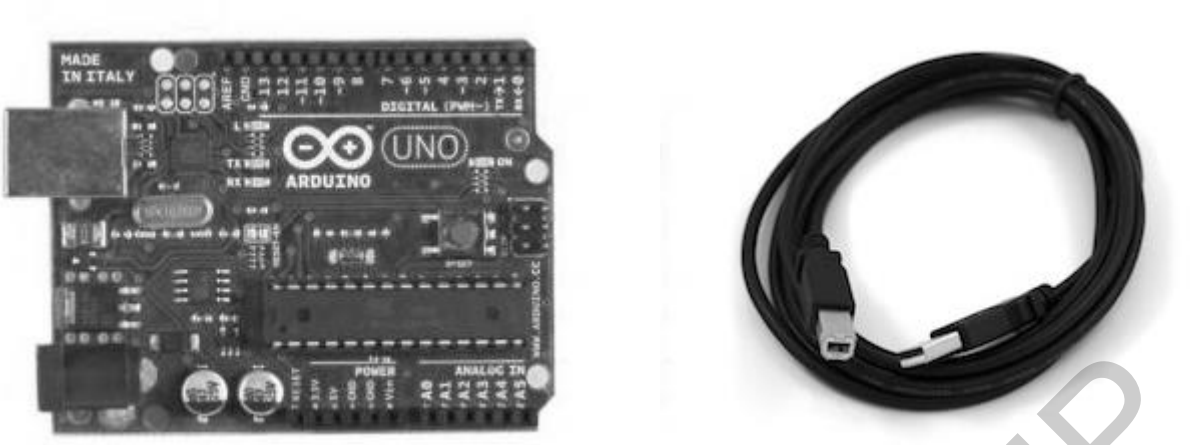
На Arduino.cc имеется список библиотек. Некоторые библиотеки включены в среду разработки Arduino. Другие могут быть загружены с различных ресурсов. Для установки новых библиотек необходимо создать подкаталог «libraries» в каталоге блокнота и затем распаковать архив. Например, для установки библиотеки DateTime ее файлы должны находиться в подкаталоге /libraries/DateTime каталога блокнота.

1.2. Практическая часть

Необходимые компоненты: плата Arduino Uno и USB-кабель (рис. 1.9).

Необходимые действия и рекомендации:

1. Подключение платы Arduino Uno с помощью USB-кабеля к компьютеру. Должен загореться красный светодиод питания, помеченный PWR.
2. Запуск среды разработки Arduino.
3. Выбор платы из пункта меню *Инструменты* → *Плата*, соответствующей используемой плате Arduino (рис. 1.10).



а *б*

Рис. 1.9. Необходимые компоненты:

а – Arduino Uno; *б* – USB-кабель

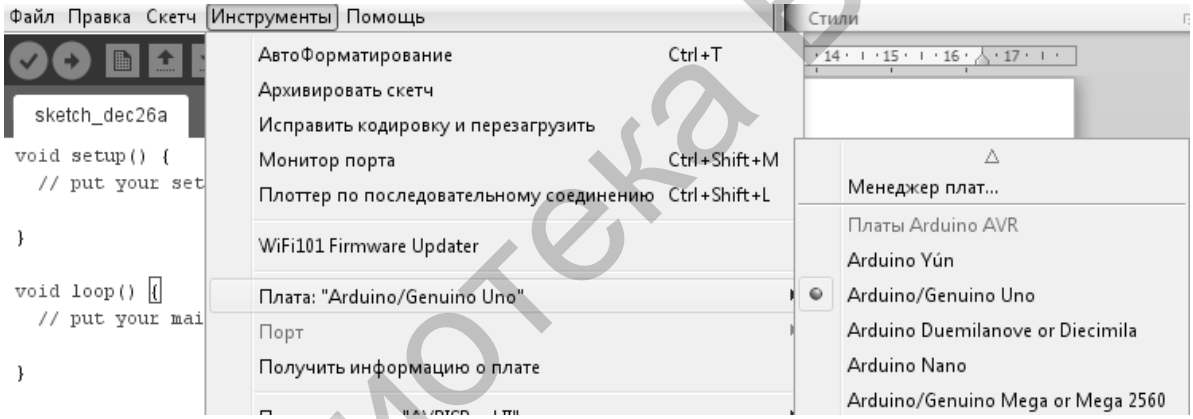


Рис. 1.10. Выбор необходимой платы Arduino

4. Выбор последовательного порта – устройства последовательной передачи данных для платы Arduino – из соответствующего меню *Инструменты*. Вероятно, это будет COM3 или выше (COM1 и COM2 обычно резервируются для аппаратных COM-портов). Чтобы найти нужный порт, можно отсоединить плату Arduino и повторно открыть меню: порт, который исчез и будет портом платы Arduino. Затем нужно вновь подсоединить плату и выбирать последовательный порт.

5. Загрузка скетча в Arduino нажатием соответствующей кнопки в среде разработки. Необходимо подождать несколько секунд – можно увидеть мигание светодиодов RX и TX на плате. В случае успешной загрузки в строке состояния появится сообщение «Загрузка выполнена».

Несколько секунд спустя после окончания загрузки можно увидеть, как светодиод вывода 13 (L) на плате начнет мигать красным цветом. Таким образом, получен готовый к работе комплект Arduino.

1.2.1. Индивидуальные задания

Написать программу для управления работой светодиода, находящегося на pin 13.

Вариант 1. Плавное увеличение яркости светодиода до максимального значения. Горение на максимальной яркости в течение 5 с, после чего плавное затухание.

Вариант 2. Старт происходит при светодиоде, включенном на максимальную яркость. Плавное уменьшение яркости светодиода до затухания, 3 с бездействия, после чего плавное нарастание яркости.

Вариант 3. Плавное увеличение яркости до половины максимального значения, закрепление состояния на 4 с, снижение яркости до четверти максимального значения, 2 с бездействия и постепенное увеличение до максимального значения.

Вариант 4. При изначально включенном светодиоде снизить яркость на 75 % от максимального значения, бездействие в течение 3 с, увеличение мощности до половины максимального значения, простой в течение 1 с, плавное уменьшение яркости до минимального значения.

1.3. Содержание отчета

1. Цель работы.
2. Краткие теоретические сведения.
3. Индивидуальные задания.
4. Код выполненной программы с комментариями.
5. Выводы.

1.4. Контрольные вопросы

1. Что такое Arduino?
2. В каких областях может быть использована технология Arduino?
3. Каковы основные компоненты платы Arduino Uno?
4. Какие существуют выводы на плате Arduino? В чем их различия?
5. Как назначить выводу порт ввода?
6. Когда выводы могут пропускать через себя достаточно высокий ток?
7. Какой контроллер заложен в основу Arduino Uno?

Рекомендуемые источники

1. Arduino – ArduinoBoardUno [Электронный ресурс]. – Режим доступа : https://www.Arduino.cc/en/Main/ArduinoBoardUno.

2. FEZ Cerbuino Net – GHI Electronics [Электронный ресурс]. – Режим доступа : https://www.ghielectronics.com/catalog/product/473.

3. STM32F405RG High-performance foundation line, ARM Cortex-M4 core with DSP and FPU, 1 Mbyte Flash, 168 MHz CPU, ART Accelerator – STMicroelectronics [Электронный ресурс]. – Режим доступа : http://www.st.com/web/catalog/mmc/FM141/SC1169/SS1577/LN1035/PF252144.

4. STM32F4DISCOVERY Discovery kit with STM32F407VG MCU – STMicroelectronics [Электронный ресурс]. – Режим доступа : http://www.st.com/web/catalog/tools/FM116/SC959/SS1532/PF252419.

Лабораторная работа №2

СНЯТИЕ И ОБРАБОТКА ПОКАЗАНИЙ ДАТЧИКОВ И ИСПОЛНИТЕЛЬНЫХ УСТРОЙСТВ, СОВМЕСТИМЫХ С ПЛАТФОРМОЙ ARDUINO

Цель работы: изучить особенности широтно-импульсной модуляции и процесс получения данных на платформе Arduino посредством реализации програмного модуля.

2.1. Теоретические сведения

2.1.1. Широтно-импульсная модуляция

Пример использования аналогового выхода для управления светодиодом доступен из меню *Файл* среды разработки Arduino.

Широтно-импульсная модуляция (ШИМ) (рис. 2.1) – это операция получения изменяющегося аналогового значения посредством цифровых устройств. Устройства используются для получения прямоугольных импульсов – сигналов, которые постоянно переключаются между максимальным и минимальным значениями. Данный сигнал моделирует напряжение между максимальным (5 В) и минимальным (0 В) значениями, изменяя при этом длительность времени включения 5 В относительно включения 0 В. Длительность включения максимального значения называется шириной импульса. Для получения различных аналоговых величин изменяется ширина импульса. При достаточно быстрой смене периодов включения-выключения можно подавать постоянный сигнал между 0 и 5 В на светодиод, тем самым управляя яркостью его свечения.

На диаграмме линии деления отмечают постоянные временные периоды. Длительность периода обратно пропорциональна частоте ШИМ. То есть если частота ШИМ составляет 500 Гц, то указанные линии будут отмечать интервалы длительностью в 2 мс каждый. Вызов функции analogWrite() с масштабом 0–255 означает, что значение analogWrite(255) будет соответствовать 100%-му рабочему циклу (постоянное включение 5 В), а значение analogWrite(127) – 50%-му рабочему циклу.

Для примера можно взять платформу и аккуратно начать двигать ее влево-вправо, что «превращает» мигание светодиода в светящиеся линии. Наращивание или уменьшение ширины импульса будет увеличивать или уменьшать светящиеся линии светодиода.

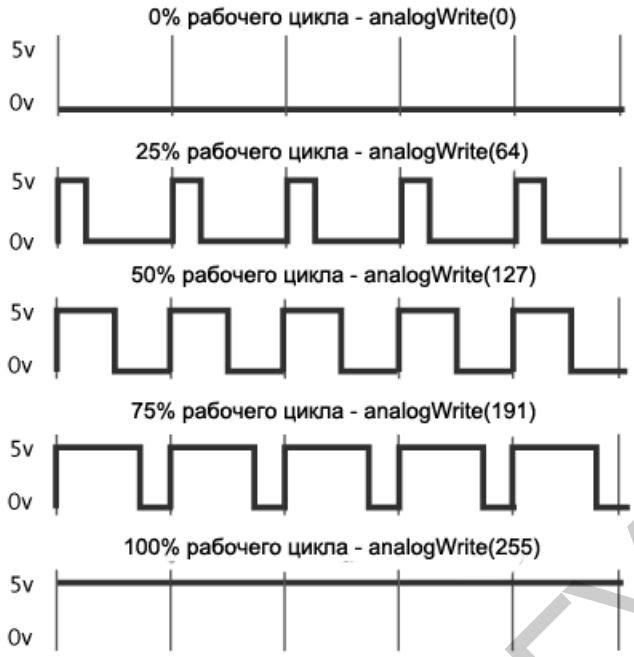


Рис. 2.1. Широтно-импульсная модуляция

2.1.2. Функции библиотеки Serial

Основные функции следующие:

1. *Serial.read()* – функция читает 1 байт данных из поступившей на Arduino информации. Каждый вызов этого метода будет возвращать следующий байт данных из тех, что поступили на Arduino. Если возвращать нечего, то данная функция вернет –1 (минус один).

Допустим, был отправлен 1 байт данных на Arduino и использован нижеприведенный участок кода:

```
int incomingByte;
void loop()
{
    if (Serial.available() > 0)
    {
      incomingByte = Serial.read();
    }
}
```

После того как считается этот байт данных, он будет перемещен в используюмую переменную, а функция Serial.available() снова будет возвращать пустое значение (Null), пока не поступят новые данные. То есть, когда считывается байт, показания счетчика принятых байтов уменьшаются и Serial.available() будет показывать на 1 байт меньше.

Следует отметить, что данная функция возвращает только 1 байт данных. Если нужно передать 4 символа каждый по 1 байту, потребуется 4 раза вызвать

данную функцию, чтобы прочитать эти символы и самостоятельно разместить их в массив символов либо воспользоваться функцией Serial.readBytes().

2. *Serial.readBytes()* – функция практически идентична Serial.read().

В данную функцию нужно передать переменную массива типа char[] или byte[] и числовое значение, соответствующие тому, какое количество байтов из поступивших нужно прочитать. Прочитанные байты будут размещены в переданном массиве и убраны из счетчика функции Serial.available().

Следует отметить, что массив должен быть достаточно большим, чтобы вместить то количество байтов, которое требуется. Для примера можно получить на Arduino строку приветствия «Hi max» и вернуть ее на персональный компьютер:

```
char incomingBytes[7];
void loop()
{
    if (Serial.available() > 0)
    {
       Serial.readBytes(incomingBytes, 6);
       Serial.pritln(incomingBytes);
    }
}
```

В указанном блоке кода проверяется, что данные поступили, затем вызывается функция, передается в нее массив, в который требуется записать полученные байты данных, и указывается количество байтов, которое требуется прочитать. Функция осуществит чтение 6 байт и последовательно разместит их в ячейки массива, после чего полученный массив символов передается обратно на персональный компьютер.

Следует отметить, что на работу данной функции влияют настройки Serial.setTimeout(). Функция прекращает свою работу досрочно, как только считает указанное количество байтов, либо будет ждать установленный функцией Serial.SetTimeout() промежуток времени.

Функция не очищает тот массив, в который размещает переданные байты, отсюда возникают ситуации наподобие следующей. Если передана строка «Hi max», она состоит ровно из 6 байт и каждый байт последовательно будет размещен в массиве с самого начала (на мониторе получится «Hi max»). Далее, если передать строку «Hello», она займет лишь 5 байт и будет записана в сформированный массив с самого начала. В результате на экране получится «Hellox». Лишняя буква «x» – это остаток от предыдущей строки, то есть если не записать все 6 байт или в программе не отчистить массив, то при передаче блока меньшего размера получатся остатки от прежнего более длинного блока, который хранится в массиве в результате предыдущего вызова функции.

3. *Serial.parseInt()* – функция просматривает данные, поступившие на Arduino, и ищет среди них набор кодов (чисел) от 48 до 57, которые соответствуют символам чисел от 0 до 9, а затем преобразует все это в правильное целочисленное значение.

Таким образом, если из порта передать число 72, данная функция увидит 2 последовательных байта – 55 и 51, корректно преобразует их в число 72 и вернет как правильное целочисленное значение.

Эта функция может превратить строку числовых символов, разделенных нечисловыми символами, например запятой или символом новой строки, к набору правильных целочисленных значений, которые можно использовать в арифметических операциях.

4. *Serial.parseFloat()* – это аналог функции Serial.parseInt(). Разница в том, что данная функция пытается получить корректное число с плавающей точкой.

2.2. Порядок выполнения

Необходимые компоненты: плата Arduino Uno, термодатчик (1 вариант), фоторезистор (2 вариант), датчик Холла (3 вариант), ультразвуковой датчик (4 вариант) и USB-кабель.

Необходимые действия и рекомендации – корректное подключение соответствующих компонентов.

2.2.1. Индивидуальные задания

Написать программу для снятия и обработки данных, получаемых с датчика, выданного преподавателем. Порядок выбора варианта индивидуального задания соответствует номеру варианта по списку.

Вариант 1. Снять и обработать показания с термодатчика. Вывести значения в последовательный порт. Значения должны быть с точностью до 2 знаков после запятой.

Вариант 2. Снять и обработать показания с фоторезистора. Вывести значения в последовательный порт. Значения должны быть в процентном соотношении к минимальному/максимальному возможному значению.

Вариант 3. Снять и обработать показания с датчика Холла. Вывести значения в последовательный порт. Значения должны показывать расстояние в сантиметрах, на которых находится источник магнитных волн, относительно датчика.

Вариант 4. Снять и обработать показания с ультразвукового датчика. Вывести значения в последовательный порт. Значения должны показывать расстояние в метрах, на которых находится препятствие, с точностью до 3 знаков после запятой.

2.3. Содержание отчета

1. Цель работы.

2. Краткие теоретические сведения.
3. Индивидуальные задания.
4. Код программы и копии экрана с результатами вывода данных в последовательный порт.
5. Выводы.

2.4. Контрольные вопросы

1. Что такое ШИМ-сигнал?
2. Для чего изменяется ширина импульса?
3. С помощью какой команды можно регулировать уровень ШИМ-сигнала?
4. Для чего может быть использован ШИМ-сигнал?
5. В чем суть функции Serial.read()?
6. В чем суть функции Serial.readBytes()?
7. В чем суть функции Serial.parseInt()?
8. В чем суть функции Serial.parseFloat()?

Рекомендуемые источники

1. Встраиваемые системы управления [Электронный ресурс]. – Режим доступа : http://www.controlengrussia.com/programmnye-sredstva/vstraivaemy-e-sistemy-upravleniya/.
2. Хоровиц, П. Искусство схемотехники / П. Хоровиц, У. Хилл ; пер. с англ. – 2-е изд. – М. : БИНОМ. – 2014. – 704 с.

Лабораторная работа №3

УПРАВЛЕНИЕ НИЗКОВОЛЬТНЫМ ШАГОВЫМ ДВИГАТЕЛЕМ С ИСПОЛЬЗОВАНИЕМ ПЛАТФОРМЫ ARDUINO

Цель работы: изучить устройство и принцип работы шагового двигателя, разработать и программно реализовать алгоритм управления им.

3.1. Теоретические сведения

Шаговый электродвигатель (ШД) – это синхронный бесщеточный электродвигатель с несколькими расположенными в статоре обмотками, в котором ток, подаваемый в одну из обмоток, вызывает фиксацию ротора. Последовательная активация обмоток ШД вызывает дискретные угловые перемещения (шаги) ротора (рис. 3.1).

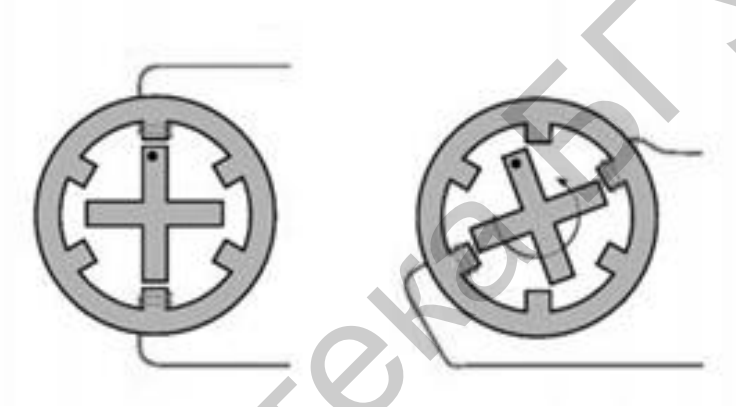


Рис. 3.1. Принцип работы ШД

Мощность шаговых двигателей лежит в диапазоне от единиц ватт до нескольких киловатт. Конструктивно ШД имеют различные исполнения и в целом их можно разделить на 3 типа:

1. Реактивные ШД, в которых ротор выполняется из магнитомягкого (ферромагнитного) материала.
2. ШД с постоянными магнитами, в которых ротор выполняется из магнитотвердого (магнитного) материала.
3. Гибридные ШД, сочетающие в себе лучшие черты реактивных ШД и ШД с постоянными магнитами.

Преимущества. Главное преимущество шаговых приводов – точность. При подаче потенциалов на обмотки шаговый двигатель повернется строго на определенный угол. К положительным моментам можно отнести стоимость шаговых приводов: в среднем в 1,5–2 раза дешевле сервоприводов. Шаговый привод как недорогая альтернатива сервоприводу наилучшим образом подходит для автоматизации отдельных узлов и систем, где не требуется высокая динамика.

Недостатки. Возможность «проскальзывания» ротора – наиболее известная проблема этих двигателей. Это может произойти при превышении нагрузки на валу, неверной настройке управляющей программы (например, ускорение старта или торможения не адекватно перемещаемой массе) и приближении скорости вращения к резонансной. Наличие датчика позволяет обнаружить проблему, но автоматически скомпенсировать ее без остановки производственной программы возможно только в очень редких случаях. Чтобы избежать проскальзывания ротора, можно увеличить мощность двигателя (один из способов).

3.2. Порядок выполнения

Необходимые компоненты: плата Arduino Uno, ШД и USB-кабель (рис. 3.2).

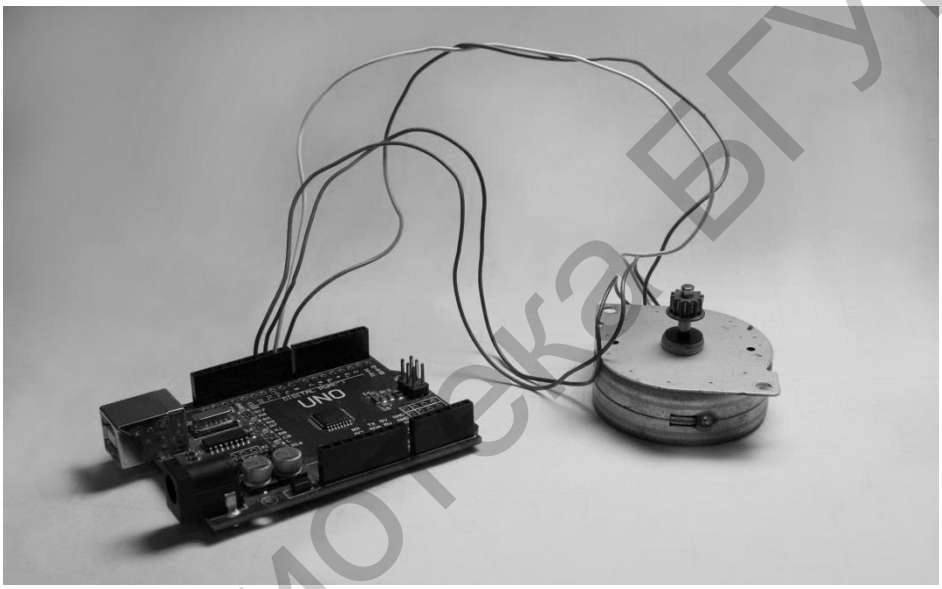


Рис. 3.2. Соединенные плата Arduino и ШД

Необходимые действия и рекомендации:

1. Соединение ШД и платы Arduino и подключение полученной системы с помощью USB-кабеля к компьютеру.
2. Для управления шаговым двигателем требуется подавать импульсы на выводы обмоток согласно типу и количеству обмоток ШД.
3. Для подачи одного импульса требуется подать напряжение на порт, после чего отключить его.
4. Длительность импульса может быть подобрана на практике либо взята из паспорта ШД.
5. Для фиксации положения ШД требуется подача сигнала на определенные выводы обмоток. Выводы обмоток берутся из паспорта ШД.

3.2.1. Индивидуальные задания

Вариант 1. Разогнать шаговый двигатель из состояния покоя до максимальной скорости, после чего сделать по два полных оборота в разные стороны и зафиксировать положение ШД.

Вариант 2. Постепенно увеличивать скорость ШД до половины максимального значения, после чего сменить направления и продолжать увеличивать скорость до максимального значения.

Вариант 3. Разработать и реализовать алгоритм, при котором двигатель делает один оборот по часовой стрелке, два – против часовой стрелки, три – по часовой и так далее до пяти оборотов.

Вариант 4. Разработать и реализовать алгоритм, при котором ШД будет постепенно увеличивать скорость до максимального значения, после чего снижать до минимального.

3.3. Содержание отчета

1. Цель работы.
2. Краткие теоретические сведения.
3. Индивидуальные задания.
4. Код программы.
5. Выводы.

3.4. Контрольные вопросы

1. Что такое шаговый двигатель?
2. Где могут быть применены шаговые двигатели?
3. С помощью чего регулируется скорость вращения ШД?

Рекомендуемые источники

Соммер, У. Программирование микроконтроллерных плат Arduino/Freeduino / У. Соммер. – СПб. : БХВ-Петербург, 2016. – 256 с.

Лабораторная работа №4

ВИЗУАЛИЗАЦИЯ ИНФОРМАЦИИ ПОСРЕДСТВОМ ЖИДКОКРИСТАЛЛИЧЕСКОГО ЭКРАНА, РЕАЛИЗОВАННОЙ НА ПЛАТФОРМЕ ARDUINO ПРИ ПОМОЩИ I2C ШИНЫ

Цель работы: изучить процесс подключения LCD-дисплея и произвести вывод данных с датчика.

4.1. Теоретические сведения

Жидкокристаллический дисплей (LCD-дисплей) – плоский дисплей на основе жидких кристаллов, а также устройство на основе такого дисплея. Простые приборы с дисплеем могут иметь монохромный или 2–5-цветный дисплей. Многоцветное изображение в более сложных устройствах формируется с помощью RGB-триад. Дисплей на жидких кристаллах используется для отображения графической или текстовой информации в компьютерных мониторах (также и в ноутбуках), телевизорах, телефонах, цифровых фотоаппаратах, электронных книгах, навигаторах, планшетах, электронных переводчиках, плеерах, термометрах, калькуляторах, часах, а также во многих других электронных устройствах.

Ранее в большинстве настольных мониторов, а также во всех дисплеях ноутбуков использовались матрицы с 18-битным цветом (6 бит на каждый RGB-канал), 24-битность эмулируется мерцанием с дизерингом. Жидкокристаллический дисплей с активной матрицей – разновидность жидкокристаллического дисплея, в котором используется активная матрица, управляемая тонкопленочными транзисторами.

В данной лабораторной работе используется дисплей LCD 1602 (рис. 4.1).

Рис. 4.1. Дисплей LCD 1602

Дисплеи LCD 1602 на базе контроллера HD44780 являются одними из самых простых, доступных и востребованных дисплеев для разработки различных электронных устройств. Его можно встретить как в устройствах, собранных вручную, так и в промышленных устройствах, таких как, например,

автоматы для приготовления кофе. На базе данного дисплея собраны самые популярные модули в тематике Arduino, такие как LCD I2C-модуль и LCD Keypad Shield.

Подключать данный экран возможно с помощью 2 вариантов:

1. Прямое подключение к Arduino (рис. 4.2).
2. Использование дополнительных модулей (рис. 4.3).

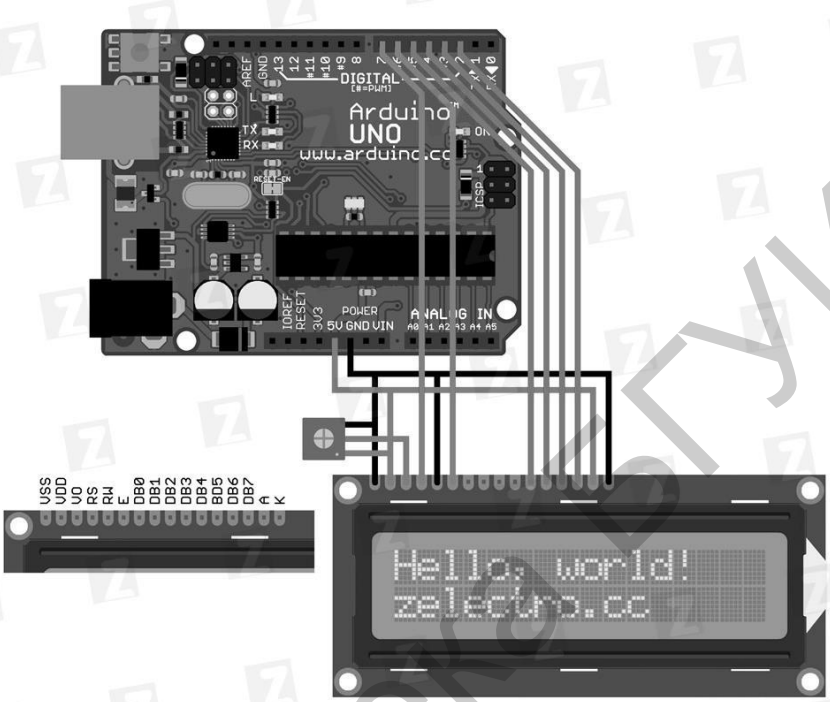


Рис. 4.2. Прямое подключение к Arduino

В данной работе используется второй вариант, так как это значительно упрощает работу и настройку данного экрана. В качестве дополнительного модуля потребуется конвертор в IIC/I2C.

Этот способ подключения работает только со специальной библиотекой. Загрузить библиотеку можно по названию LiquidCrystal_I2C1602V1.

Пример кода, реализующего инициализацию, подсветку и вывод информации на дисплей:

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
LiquidCrystal_I2C lcd(0x27,16,2); // Устанавливаем дисплей
void setup()
{
          lcd.init();
          lcd.backlight();// Включаем подсветку дисплея
          lcd.print("MyFirstText");
          lcd.setCursor(8, 1);
```

```
                lcd.print("LCD 1602");
    }
    void loop()
    {
                // Устанавливаем курсор на вторую строку и на
нулевой символ
                lcd.setCursor(0, 1);
                 // Выводим на экран количество секунд с момента
запуска Arduino
                lcd.print(millis()/1000);
    }
```

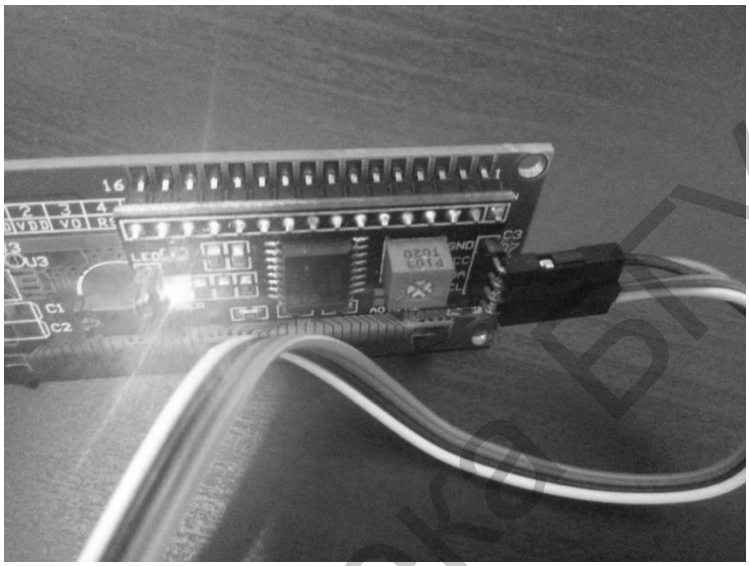


Рис. 4.3. Использование дополнительных модулей

4.2. Порядок выполнения

Необходимые компоненты: плата Arduino Uno, LCD-дисплей и USB-кабель.

Необходимые действия и рекомендации:

1. Соединение LCD-дисплея и платы Arduino.
2. Подключение полученной системы с помощью USB-кабеля к компьютеру.
3. Написание согласно примеру скетча для LCD-дисплея.
4. Вывод результата на экран.

4.2.1. Индивидуальные задания

Требуется вывести на экран в первой строке дисплея данные из лабораторной работы №2, а во второй – фамилию и инициалы студента.

4.3. Содержание отчета

1. Цель работы.
2. Краткие теоретические сведения.
3. Индивидуальные задания.
4. Код программы.
5. Выводы.

4.4. Контрольные вопросы

1. Что такое жидкокристаллический дисплей?
2. Что формируется с помощью RGB-триад?
3. Где используются жидкокристаллические дисплеи?
4. Назовите способы подключения жидкокристаллических дисплеев к отладочным платам.
5. Каковы преимущества I2C-шины?

Рекомендуемые источники

1. Соммер У. Программирование микроконтроллерных плат Arduino/Freeduino / У. Соммер. – СПб. : БХВ-Петербург, 2016. – 256 с.
2. Arduino – Home [Электронный ресурс]. – Режим доступа : https://www.arduino.cc/.
3. Banzi, M. Getting Started with Arduino / M. Banzi. – USA, 2011. – 72 с.
4. Блум, Дж. Изучаем Ардуино / Дж. Блум. – СПб. : БХВ-Петербург, 2015. – 336 с.
5. iArduino.ru – Купить микроконтроллеры Arduino в Москве [Электронный ресурс]. – Режим доступа : http://iarduino.ru/.

Лабораторная работа №5

РЕАЛИЗАЦИЯ РЕЛЯЦИОННОЙ СТРУКТУРЫ БАЗЫ ДАННЫХ И РАЗМЕЩЕНИЕ В ЕЕ ОБЪЕКТАХ ИНФОРМАЦИИ, СЧИТАННОЙ С ДАТЧИКОВ ВСТРАИВАЕМОЙ СИСТЕМЫ ИЗМЕРЕНИЙ И МОНИТОРИНГА

Цель работы: изучить средство реализации реляционной структуры базы данных и размещение в ее объектах информации, считанной с датчиков встраиваемой системы измерений и мониторинга.

5.1. Теоретические сведения

5.1.1. Структура системы сбора и обработки результатов измерений и мониторинга

Основными технологическими платформенными составляющими системы сбора и обработки результатов измерений и мониторинга являются:

– *аппаратная* (оборудование (солнечный коллектор) и модуль сбора *данных* (устройство сбора данных (контроллеры и интерфейсы) и блок датчиков, реализующий сбор, преобразование, передачу и форматирование данных));

– *программная* (серверное программное обеспечение для мониторинга параметров (клиентская часть) и база данных, в частности, для получения, обработки, манипулирования, хранения и визуализации данных);

– *информационная* (интерактивные электронные информационные ресурсы с возможностью аутентификации через Web-интерфейс и справочная документация) (рис. 5.1).

Блок датчиков содержит в себе как аналоговые, так и цифровые измерительные устройства, регистрирующие текущие параметры работы коллектора. Устройство сбора данных предназначено для контроля работы датчиков, их опроса и обеспечения передачи данных между датчиками и сервером. Физически оно представляет собой прибор, управляемый микроконтроллером. Сервер обеспечивает сбор и хранение полученной информации, а также формирует данные для статистических отчетов. Компьютеры пользователей выступают в роли клиентов, формирующих при помощи Web-браузера различные запросы и получающих ответы от сервера посредством локальной компьютерной сети или глобальной сети Internet.

Передача данных осуществляется по сетевому протоколу HTTP. Сервер сохраняет информацию в специальной базе данных. Такой подход имеет ряд преимуществ, поскольку исключает сильную зависимость одних компонентов системы от других. Передача и накопление информации происходят под управлением серверного программного обеспечения.

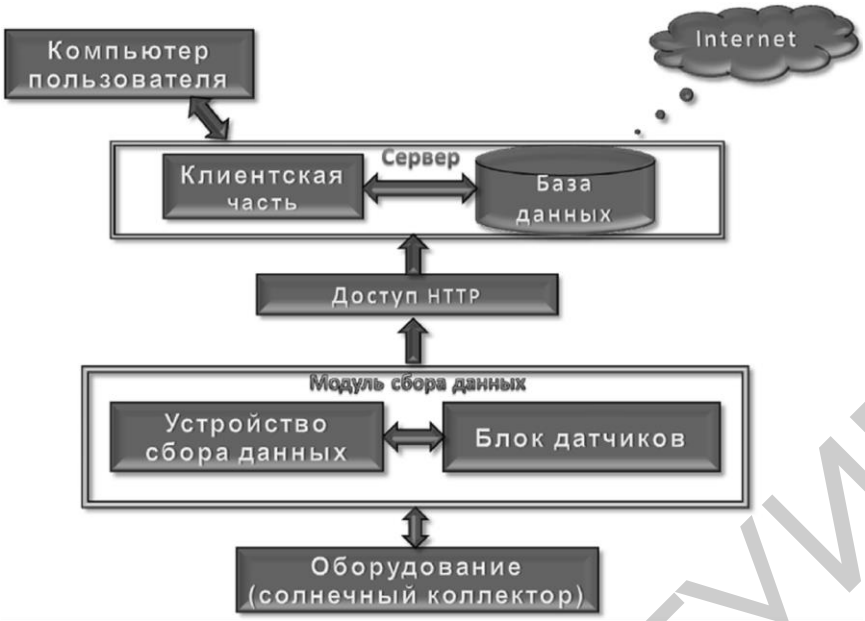


Рис. 5.1. Структурная схема системы сбора и обработки результатов измерений и мониторинга

Хранение данных, поступающих от автономных датчиков температуры теплоносителя на входе и выходе коллектора, на входе, выходе и внутри бака (бойлера) и датчиков расхода (перемещаемого объема) теплоносителя, может быть реализовано в стандартном текстовом формате представления табличных данных CSV или MDF или структурированном XML. Измерив указанные показатели в совокупности с интенсивностью света, можно рассчитать (оценить) мощность технической установки, а также потери при транспортировке тепла с отображением полученных результатов в соответствующем виде. Для хранения информации используется сервер с системой управления базами данных, доступ к которому организован не напрямую, а через Web-сервер. Модуль сбора данных периодически опрашивает датчики, формирует пакет данных и отправляет POST-запрос Web-серверу. На стороне сервера реализовано программное обеспечение для работы с данными, разработанное на языке программирования C#. Такой подход повышает надежность всей системы в целом и делает ее более гибкой. Например, при обновлении SQL-сервера, переходе на другую систему управления базами данных или изменении структуры самой базы данных не требуется вносить изменения в программное обеспечение. К тому же данный подход подразумевает достаточную простоту представления измеренных данных в сети Internet.

5.1.2. Система управления базами данных

Среда *Microsoft SQL Server Management Studio 2012* (SSMS) – это графический набор средств для разработки сценариев на структурированном языке запросов T-SQL и управления всеми компонентами сервера баз данных Microsoft SQL Server. SSMS является основным инструментом любого разработчика или администратора указанного сервера баз данных.

В SSMS объединены возможности таких программ, как Enterprise Manager, Query Analyzer и Analysis Manager.

С помощью SSMS можно подключаться к любым компонентам SQL Server, таким как Database Engine, службы Integration Services, службы Analysis Services и службы Reporting Services (рис. 5.2).

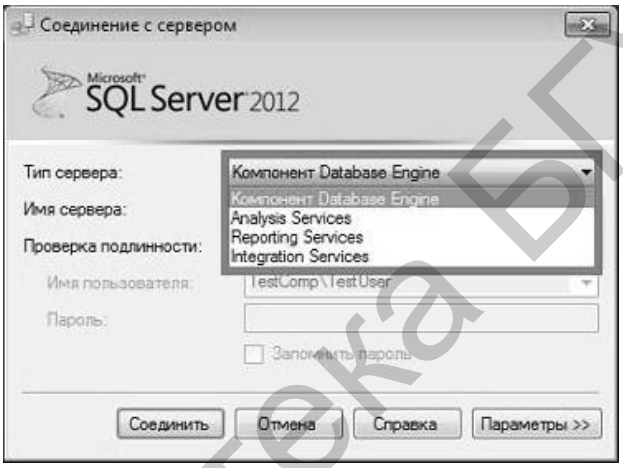


Рис. 5.2. Подключение к компонентам Microsoft SQL Server

Другими словами, с помощью SSMS можно управлять не только основным компонентом Database Engine, но и другими немаловажными компонентами. Таким образом, SSMS – это единая полнофункциональная программа по управлению SQL-сервером.

5.1.3. Обозреватель объектов

В среде SSMS встроен обозреватель объектов, который позволяет просматривать все объекты сервера и предоставляет графический интерфейс для управления этими объектами. Для запуска можно использовать меню *Вид* → *Обозреватель объектов*, хотя последний, как правило, отображен по умолчанию (рис. 5.3).

То есть с помощью обозревателя объектов можно посмотреть, какие базы данных, таблицы, функции и так далее существуют в проекте, какие пользователи созданы и какие связанные серверы настроены, в общем, получить доступ ко всем объектам сервера.

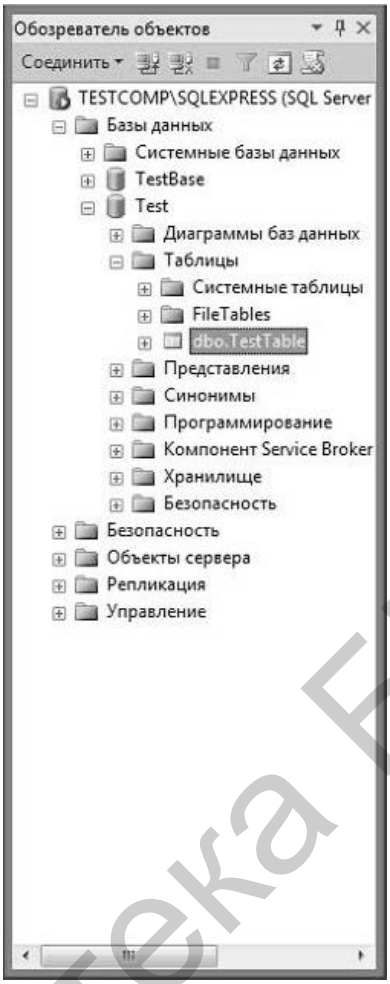


Рис. 5.3. Обозреватель объектов

5.1.4. Создание и редактирование сценариев

Данная возможность позволяет писать на T-SQL запросы или скрипты, то есть именно здесь пишутся все SQL-инструкции. Написать запрос к базе или SQL-инструкцию можно с помощью редактора кода SSMS. Чтобы открыть окно редактора кода, нужно нажать *Создать запрос* (рис. 5.4).

5.1.5. Просмотр предлагаемого плана выполнения запроса

Для упрощения оптимизации запросов в SSMS в редактор кода встроен функционал, который позволяет посмотреть план выполнения запросов, и в случае если он не оптимален, он предложит, например, создать тот или иной индекс (рис. 5.5).

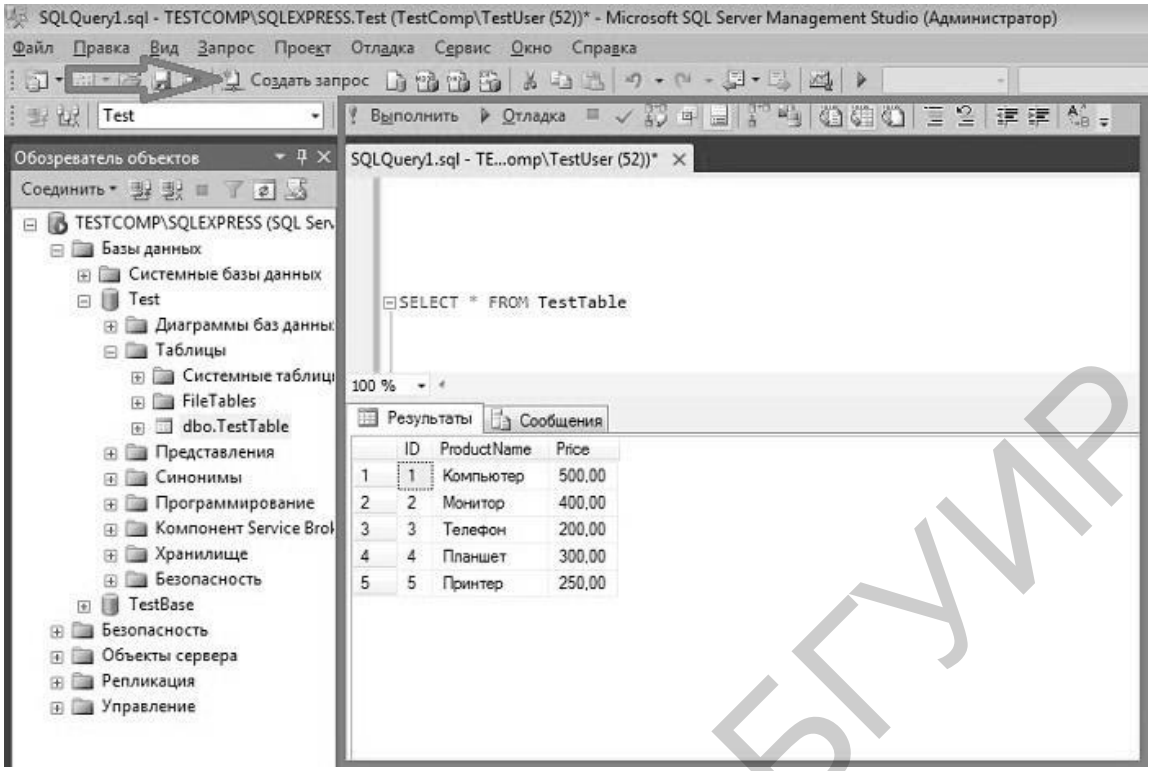


Рис. 5.4. Создание и редактирование сценариев

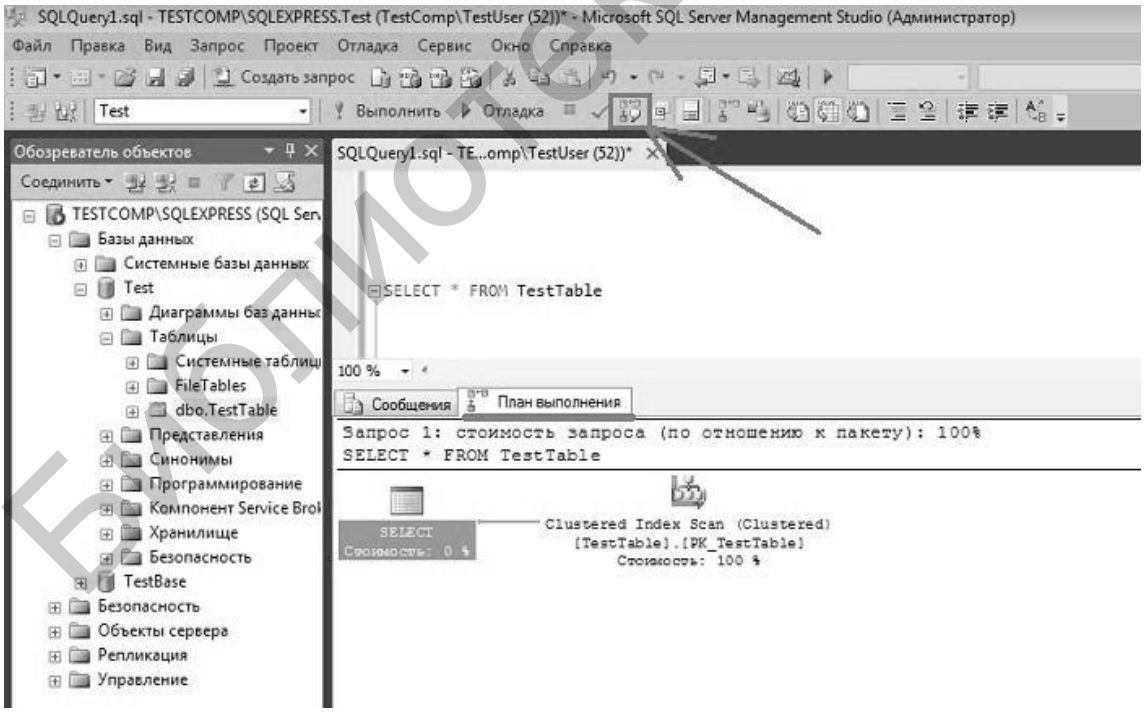


Рис. 5.5. Просмотр предлагаемого плана выполнения запроса

5.1.6. Обозреватель решений

В SSMS есть такой функционал, с помощью которого можно сгруппировать все сценарии, или скрипты, в один проект с целью их систематизации, удобного хранения и доступа к ним. Для отображения данного обозревателя следует нажать *Вид* → *Обозреватель решений* (рис. 5.6).

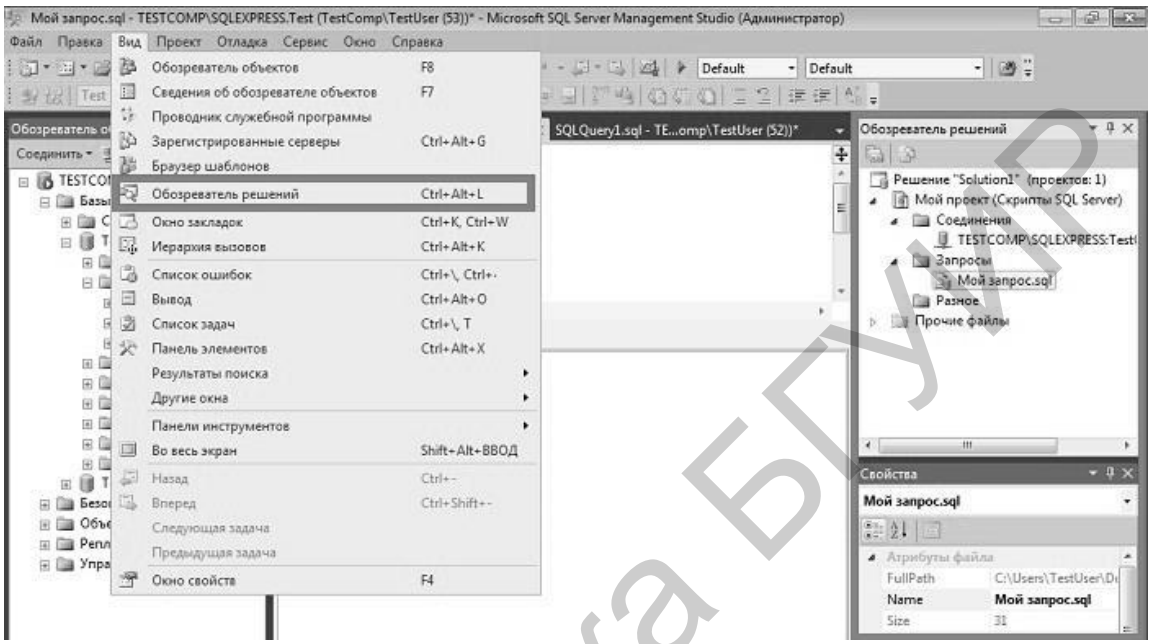


Рис. 5.6. Обозреватель решений

5.1.7. Браузер шаблонов

Для облегчения написания типичных сценариев создания или модификации объектов базы данных в среде SSMS есть возможность использовать встроенные шаблоны SQL-инструкций, то есть своего рода заголовки этих сценариев. Другими словами, если понадобилось, например, создать или изменить таблицу, но забыт синтаксис, можно открыть обозреватель шаблонов *Вид* → *Браузер шаблонов*, выбрать подходящий и всего лишь подставить нужные названия объектов (рис. 5.7).

Также существует возможность создавать собственные шаблоны для часто выполняемых задач или для случаев, когда нет подходящего встроенного шаблона.

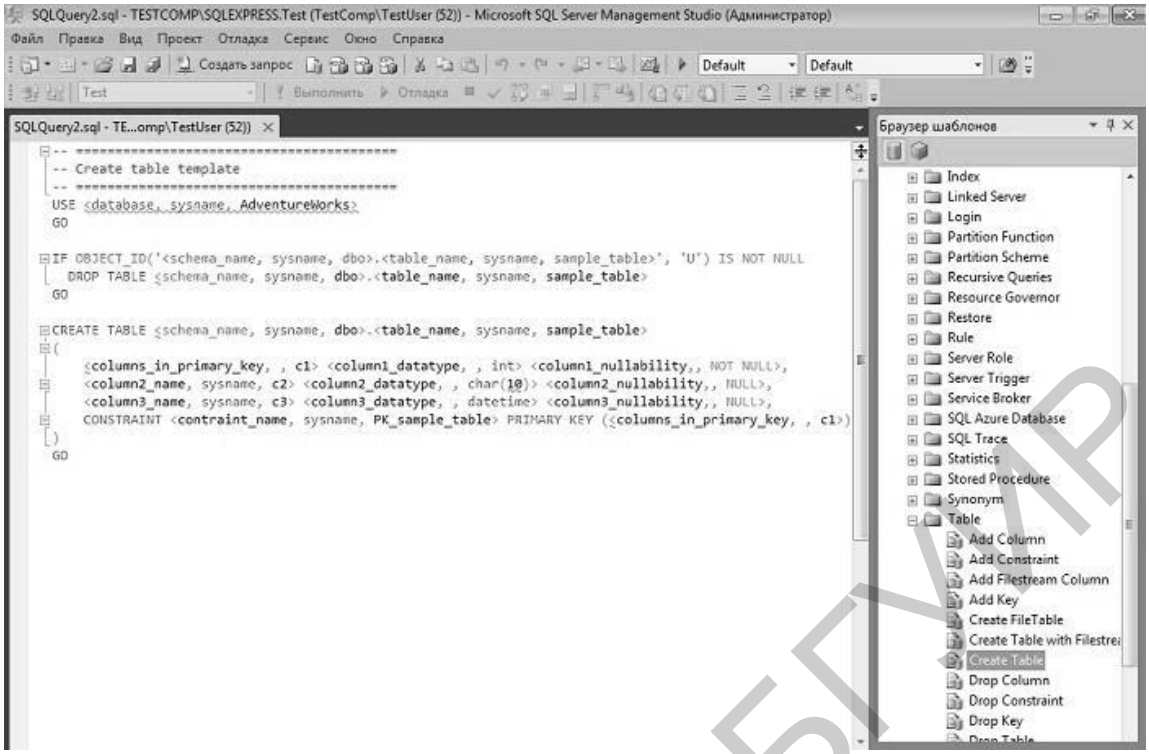


Рис. 5.7. Браузер шаблонов

5.1.8. Монитор активности

В среде SSMS есть средство, позволяющее контролировать текущую активность на сервере, например: какие запросы в данный момент выполняются, какие пользователи подключены и так далее. Для запуска можно щелкнуть правой кнопкой мыши на сервере в обозревателе объектов и выбрать *Монитор активности* или щелкнуть на значке на панели инструментов (рис. 5.8).

5.1.9. Создание резервных копий баз данных и восстановление из backup

С помощью SSMS можно легко в графическом интерактивном режиме создавать резервные копии баз данных, а также восстанавливать базы из backup. Чтобы это сделать, нужно щелкнуть правой кнопкой мыши на необходимой базе данных: *Задачи* → *Создать резервную копию/Восстановить* (рис. 5.9).

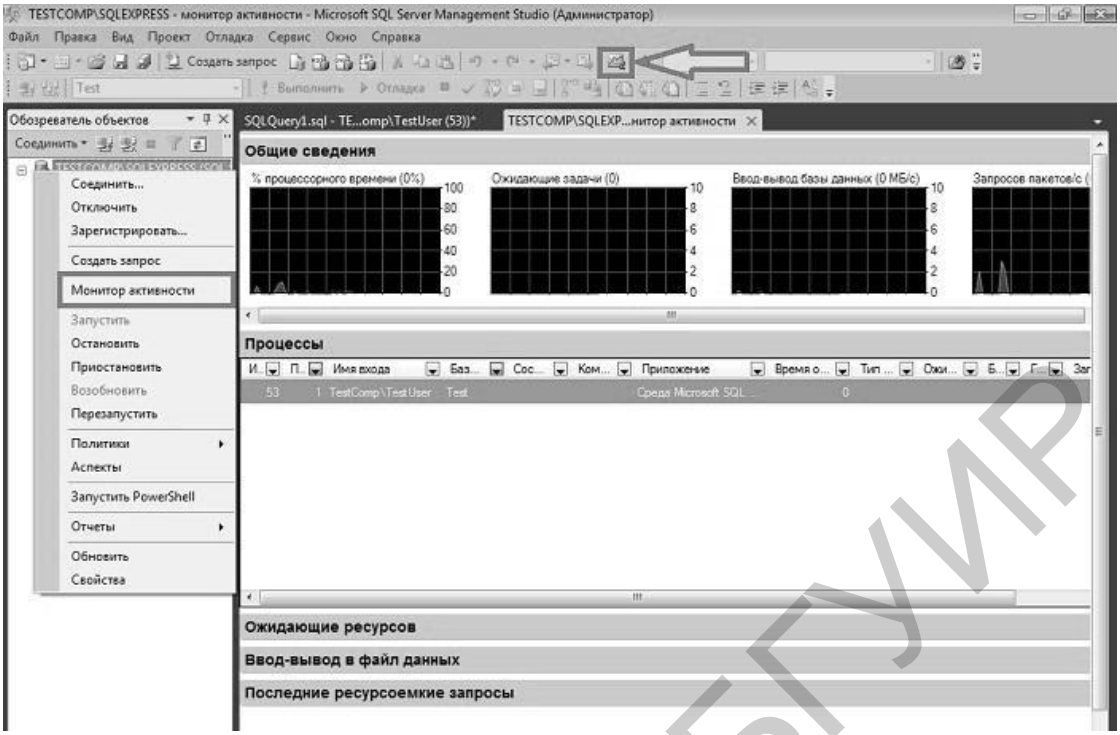


Рис. 5.8. Монитор активности

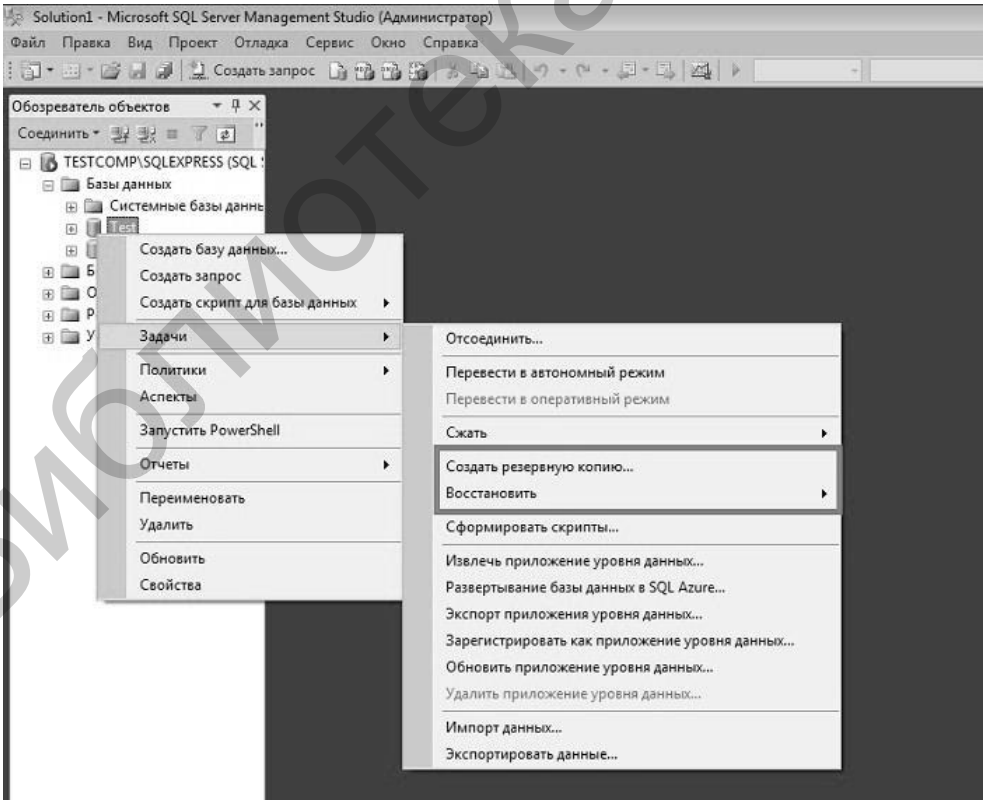


Рис. 5.9. Создание резервных копий баз данных.
Восстановление баз данных из резервных копий

5.1.10. Настройка свойств сервера, баз данных и других объектов

Среда SSMS позволяет изменять свойства сервера и объектов этого сервера. Практически у каждого объекта на SQL-сервере есть свойства, которые как раз и можно изменить с помощью графических инструментов SSMS или просто посмотреть. Например для редактирования свойств базы данных необходимо выбрать базу, щелкнуть на ней правой кнопкой мыши и выбрать *Свойства*. Некоторые свойства доступны только для чтения, а некоторые можно изменить, например в данном случае можно перейти в раздел *Параметры* и изменить необходимые параметры (рис. 5.10).

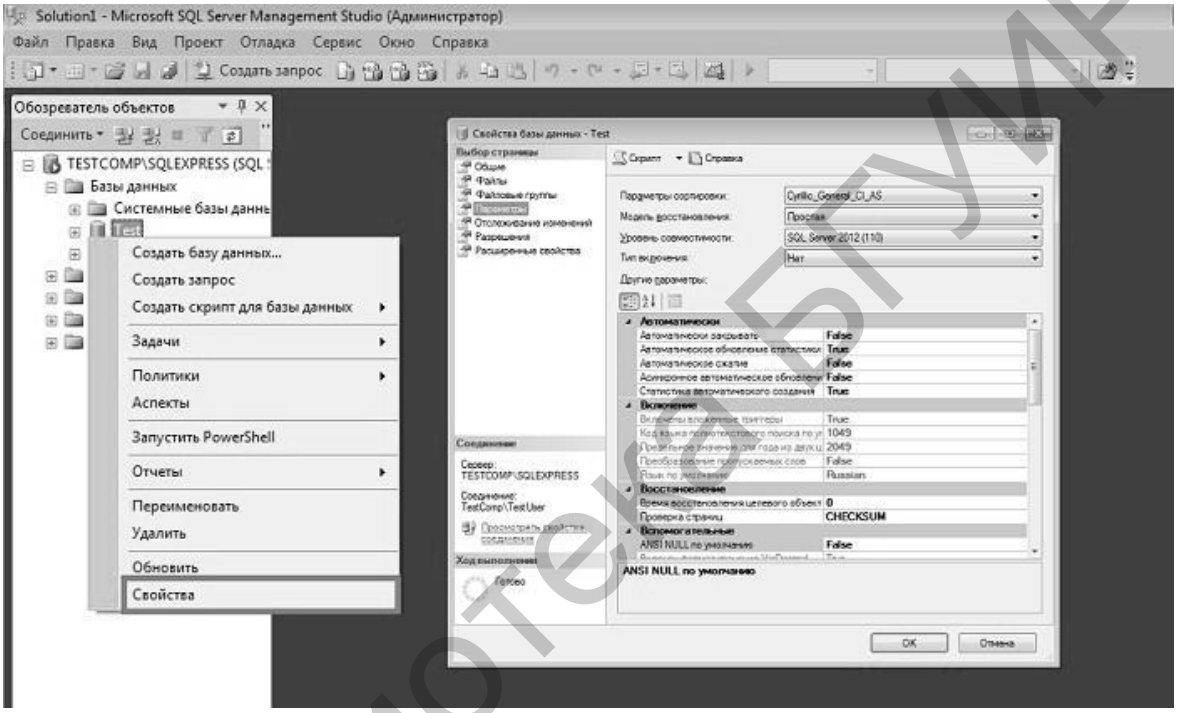


Рис. 5.10. Настройка свойств сервера, баз данных и других объектов

5.1.11. Присоединение и отсоединение баз данных

В SSMS есть возможность присоединять и отсоединять базы данных. Например, если возникла необходимость перенести базу данных с одного сервера на другой, это можно сделать с помощью графических инструментов SSMS. Для этого на одном сервере нужно правой кнопкой мыши отсоединить базу данных: *Задачи* → *Отсоединить*… (рис. 5.11).

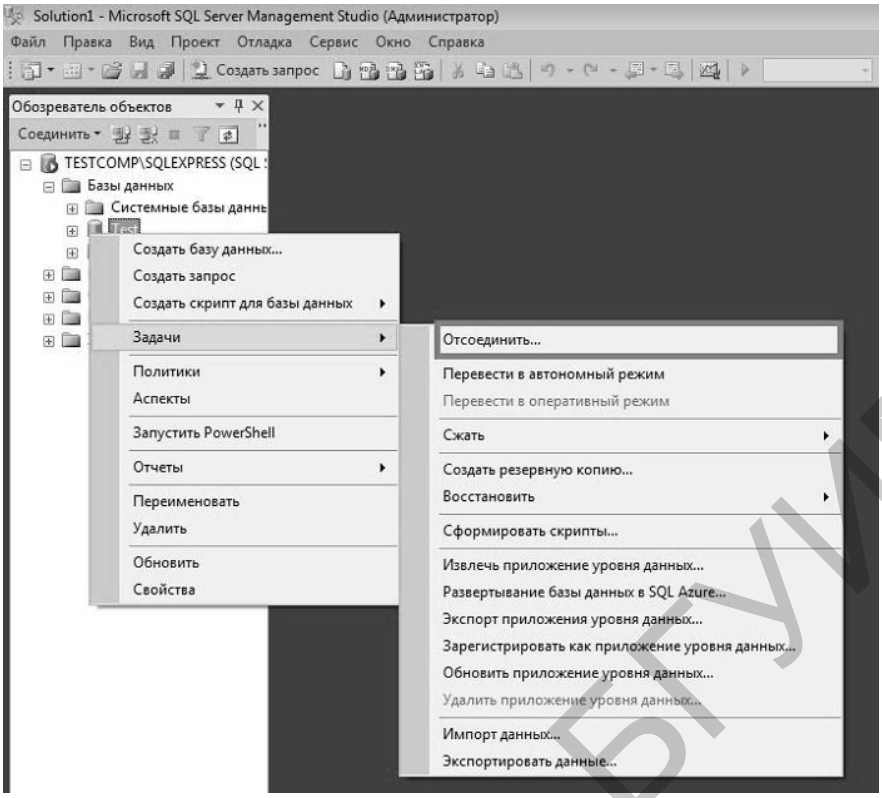


Рис. 5.11. Отсоединение баз данных

Затем можно скопировать файлы базы данных на новый сервер и в SSMS щелкнуть правой кнопкой мыши на объекте *Базы данных*, далее – *Присоединить*… (рис. 5.12).

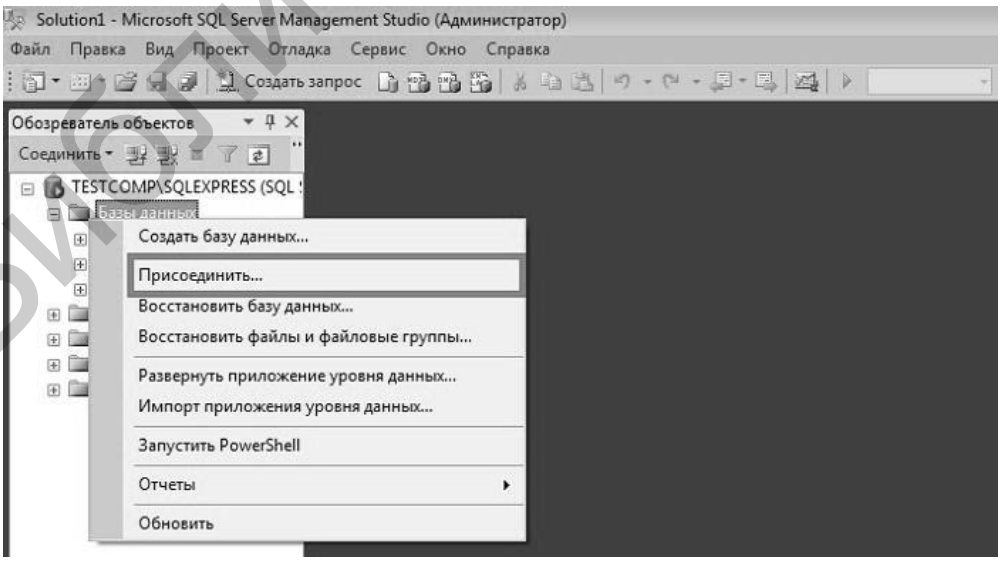


Рис. 5.12. Присоединение баз данных

5.1.12. Управление безопасностью сервера

Среда SSMS предназначена и для управления безопасностью SQL-сервера, то есть с помощью нее можно создать имя входа на сервер, пользователя базы данных или, например, настраивать доступ к объектам сервера. Для того чтобы, например, создать пользователя базы данных, необходимо выбрать *Базы данных → Нужная база данных → Безопасность → Пользователи (щелчок правой кнопки мыши) → Создать пользователя* (рис. 5.13).

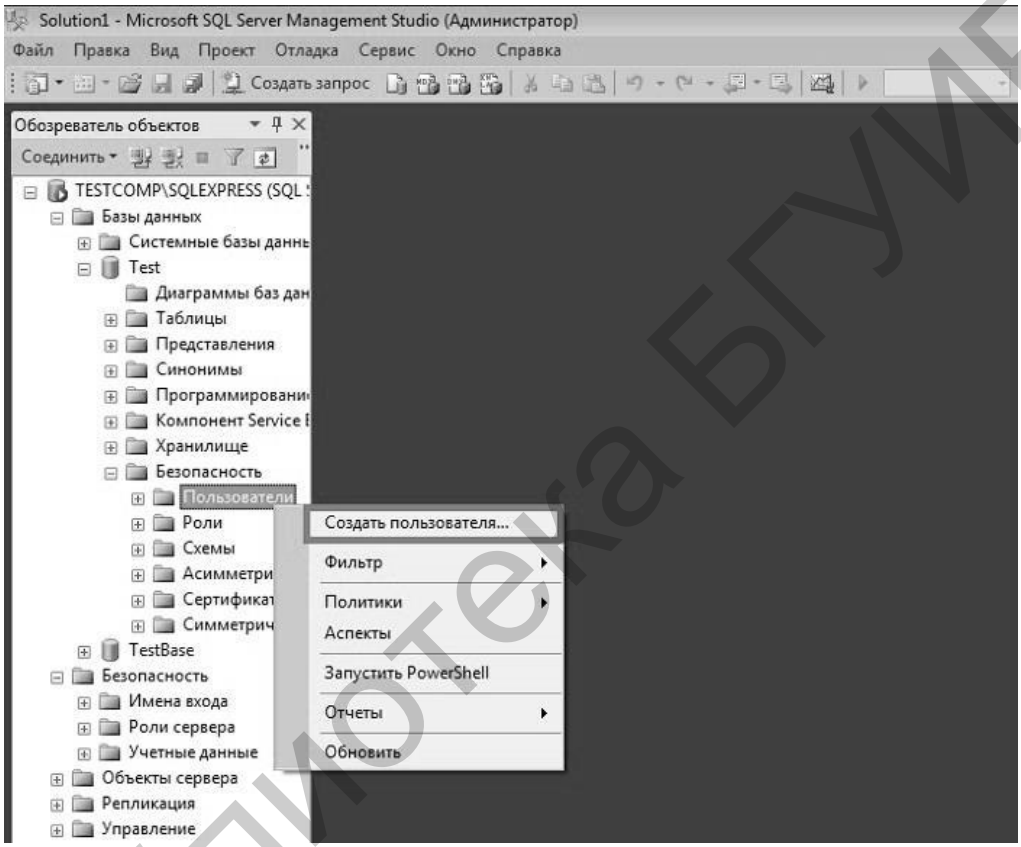


Рис. 5.13. Создание пользователя

В SSMS есть графические инструменты для создания связанных серверов. Функционал доступен в контейнере *Объекты сервера → Связанные серверы* (рис. 5.14).

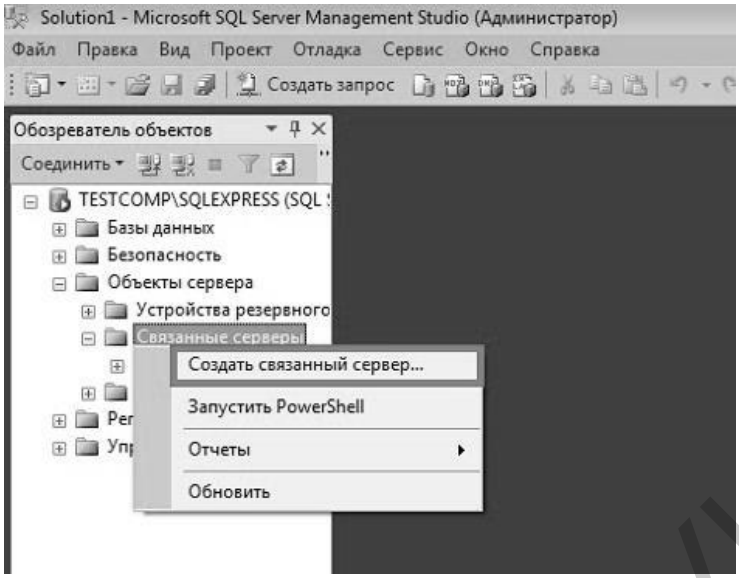


Рис. 5.14. Создание пользователя

5.1.13. Настройка репликации баз данных

Для настройки репликации баз данных есть отдельный контейнер *Репликация* (рис. 5.15).

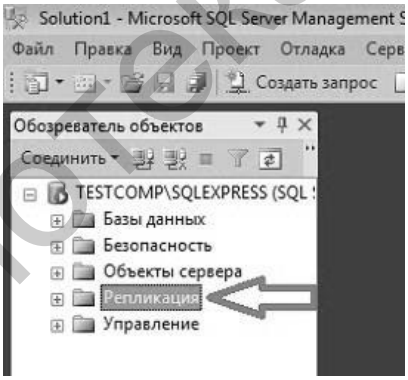


Рис. 5.15. Настройка репликации баз данных

5.1.14. Выполнение административных задач

SSMS выступает и средством администрирования SQL-сервера. С помощью данной среды можно, например, создавать планы обслуживания баз данных или просматривать системные журналы. Функционал доступен в контейнере *Управление* (рис. 5.16).

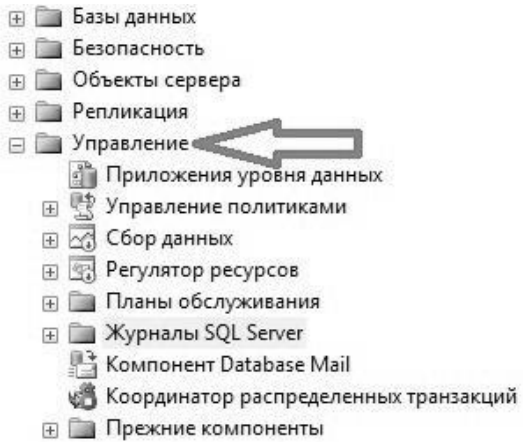


Рис. 5.16. Выполнение административных задач

Более подробную справку можно найти на официальном Web-сайте корпорации Microsoft.

5.1.15. Процедуры, хранимые в среде Microsoft SQL Server

При работе в Microsoft SQL Server пользователи могут создавать собственные *хранимые процедуры*, реализующие те или иные действия. Эти процедуры являются полноценными объектами базы данных, а потому каждая из них хранится в конкретной базе данных. Непосредственный вызов хранимой процедуры возможен, только если он осуществляется в контексте той базы данных, где находится процедура.

В Microsoft SQL Server имеется несколько типов хранимых процедур.

1. *Системные хранимые процедуры* предназначены для выполнения различных административных действий. Практически все действия по администрированию сервера выполняются с их помощью. Можно сказать, что системные хранимые процедуры являются интерфейсом, обеспечивающим работу с системными таблицами, которая, в конечном счете, сводится к изменению, добавлению, удалению и выборке данных из системных таблиц как пользовательских, так и системных баз данных. Системные хранимые процедуры имеют префикс *sp_*, хранятся в системной базе данных и могут быть вызваны в контексте любой другой базы данных.

2. *Пользовательские хранимые процедуры* реализуют те или иные пользовательские действия. Хранимые процедуры – полноценный объект базы данных. Вследствие этого каждая хранимая процедура располагается в конкретной базе данных, где и выполняется.

3. *Временные хранимые процедуры* существуют лишь некоторое время, после чего автоматически удаляются сервером. Они делятся на локальные и глобальные. Локальные временные хранимые процедуры могут быть вызваны только из того соединения, в котором созданы. При создании такой процедуры

ей необходимо дать имя, начинающееся с одного символа #. Как и все временные объекты, хранимые процедуры этого типа автоматически удаляются при отключении пользователя и перезапуске или остановке сервера. Глобальные временные хранимые процедуры доступны для любых соединений сервера, на котором имеется такая же процедура. Для ее определения достаточно дать ей имя, начинающееся с символов ##. Удаляются эти процедуры при перезапуске или остановке сервера, а также при закрытии соединения, в контексте которого они были созданы.

Создание хранимой процедуры предполагает решение следующих задач:

1. Определение типа создаваемой хранимой процедуры: временная или пользовательская. Кроме этого, можно создать свою собственную системную хранимую процедуру, назначив ей имя с префиксом *sp_* и поместив ее в системную базу данных. Такая процедура будет доступна в контексте любой базы данных локального сервера.

2. Планирование прав доступа. При создании хранимой процедуры следует учитывать, что она будет иметь те же права доступа к объектам базы данных, что и создавший ее пользователь.

3. Определение параметров хранимой процедуры. Подобно процедурам, входящим в состав большинства языков программирования, хранимые процедуры могут обладать входными и выходными параметрами.

4. Разработка кода хранимой процедуры. Код процедуры может содержать последовательность любых команд SQL, включая вызов других хранимых процедур.

Создание новой и изменение имеющейся хранимой процедуры осуществляется с помощью следующей команды:

```
<определение_процедуры>::=
{CREATE | ALTER } [PROCEDURE] имя_процедуры
 [;номер]
[{@имя_параметра тип_данных } [VARYING ]
[=default][OUTPUT] ][,...n]
[WITH { RECOMPILE | ENCRYPTION | RECOMPILE,
ENCRYPTION }]
[FOR REPLICATION]
AS
sql_оператор [...n]
```

Используя префиксы *sp_*, *#*, *##*, создаваемую процедуру можно определить в качестве системной или временной. Как видно из синтаксиса команды, не допускается указывать имя владельца, которому будет принадлежать создаваемая процедура, а также имя базы данных, где она должна быть размещена. Таким образом, чтобы разместить создаваемую хранимую процедуру в конкретной базе данных, необходимо выполнить команду *CREATE PROCEDURE* в контексте этой базы данных. При обращении из тела хранимой процедуры к объектам той же базы данных можно

использовать укороченные имена, то есть без указания имени базы данных. Когда же требуется обратиться к объектам, расположенным в других базах данных, указание имени базы данных обязательно.

Номер в имени – это идентификационный номер хранимой процедуры, однозначно определяющий ее в группе процедур. Для удобства управления процедурами логически однотипные хранимые процедуры можно группировать, присваивая им одинаковые имена, но разные идентификационные номера.

Для передачи входных и выходных данных в создаваемой хранимой процедуре могут использоваться параметры, имена которых, как и имена локальных переменных, должны начинаться с символа @. В одной хранимой процедуре можно задать множество параметров, разделенных запятыми. В теле процедуры не должны применяться локальные переменные, чьи имена совпадают с именами параметров этой процедуры.

Для определения типа данных, который будет иметь соответствующий параметр хранимой процедуры, годятся любые типы данных SQL, включая определенные пользователем. Однако тип данных *CURSOR* может быть использован только как выходной параметр хранимой процедуры, то есть с указанием ключевого слова *OUTPUT*.

Наличие ключевого слова *OUTPUT* означает, что соответствующий параметр предназначен для возвращения данных из хранимой процедуры. Однако это вовсе не означает, что параметр не подходит для передачи значений в хранимую процедуру. Указание ключевого слова *OUTPUT* предписывает серверу при выходе из хранимой процедуры присвоить текущее значение параметра локальной переменной, которая была указана при вызове процедуры в качестве значения параметра. Отметим, что при указании ключевого слова *OUTPUT* значение соответствующего параметра при вызове процедуры может быть задано только с помощью локальной переменной. Не разрешается использование любых выражений или констант, допустимое для обычных параметров.

Ключевое слово *VARYING* применяется совместно с параметром *OUTPUT*, имеющим тип *CURSOR*. Оно определяет, что выходным параметром будет результирующее множество.

Ключевое слово *DEFAULT* представляет собой значение, которое будет принимать соответствующий параметр по умолчанию. Таким образом, при вызове процедуры можно не указывать явно значение соответствующего параметра.

Так как сервер кэширует план исполнения запроса и компилированный код, при последующем вызове процедуры будут использоваться уже готовые значения. Однако в некоторых случаях все же требуется выполнять перекомпиляцию кода процедуры. Указание ключевого слова *RECOMPILE* предписывает системе создавать план выполнения хранимой процедуры при каждом ее вызове.

Параметр *FOR REPLICATION* востребован при репликации данных и включении создаваемой хранимой процедуры в качестве статьи в публикацию.

Ключевое слово *ENCRYPTION* предписывает серверу выполнить шифрование кода хранимой процедуры, что может обеспечить защиту от использования авторских алгоритмов, реализующих работу хранимой процедуры.

Ключевое слово *AS* размещается в начале собственно тела хранимой процедуры, то есть набора команд SQL, с помощью которых и будет реализовываться то или иное действие. В теле процедуры могут применяться практически все команды SQL, объявляться транзакции, устанавливаться блокировки и вызываться другие хранимые процедуры. Выход из хранимой процедуры можно осуществить посредством команды *RETURN*.

Удаление хранимой процедуры осуществляется командой *DROP PROCEDURE {имя_процедуры} [,...n]*.

5.2. Порядок выполнения

Необходимое программное обеспечение и исходные данные: Microsoft SQL Server Management Studio 2012 и файл с данными измерений и мониторинга.

Необходимые действия и рекомендации:

1. Формирование объектов базы данных и ее реляционной структуры, состоящей из 2 таблиц с полями:
   - «номер датчика»;
   - «пространственная координата датчика (долгота)»;
   - «пространственная координата датчика (широта)»;
   - «время и дата считывания показаний»;
   - «значение температуры».
2. Заполнение данными таблиц базы данных.
3. Создание хранимых процедур для обращения к данным таблиц:
   - выборка диапазона в соответствии с временем и датой считывания показаний;
   - определение минимума и максимума значений в выбранном временном интервале;
   - определение отклонения значений от среднего показателя;
   - определение территориального размещения датчиков в соответствии с выбранным регионом.
4. Формирование скриптов для автоматизации создания базы данных на удаленном сервере.

Для создания новой базы данных в SSMS нужно выбрать контейнер *Базы данных*, щелкнуть правой кнопкой мыши и выбрать в контекстном меню *Создать базу данных…* (рис. 5.17).

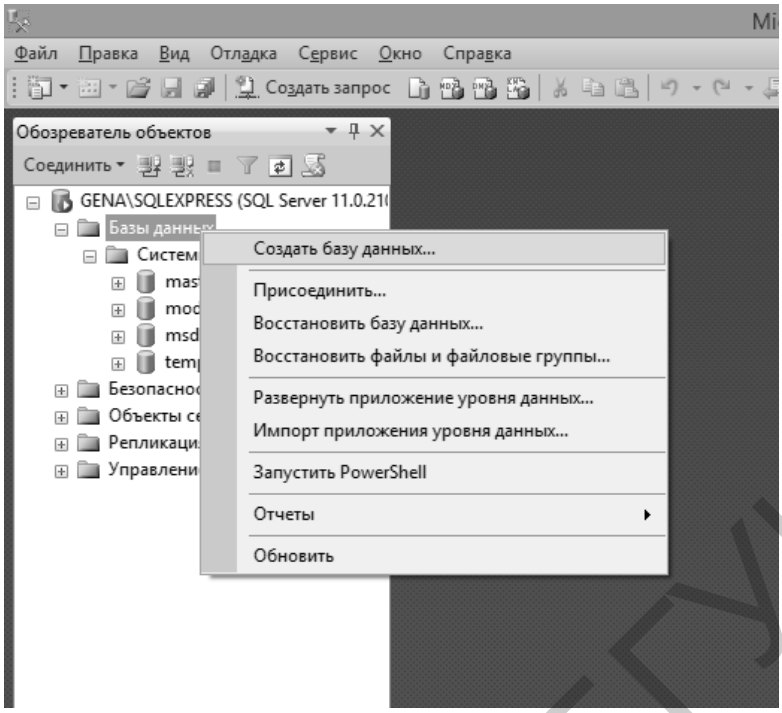


Рис. 5.17. Создание новой базы данных

Появляется форма, в которой нужно указать имя базы данных, например *Datchiki_temperatur* (рис. 5.18).

Создать новую базу данных можно и программным путем, используя конструкцию *CREATE DATABASE* на языке T-SQL (рис. 5.19).

Выполнив созданный запрос, можно прийти к результату, аналогичному тому, что получается при использовании графического пользовательского интерфейса.

Информация в базе данных хранится в таблицах, которые являются отображением некоторых логических сущностей. В базе данных *Datchiki_temperatur*, которая будет предназначена для доступа к хранящейся в ней информации (данных определенного формата) о значениях температур, снятых со специальных датчиков, будут необходимы 2 таблицы:

1. *Datchiki* – таблица будет содержать информацию о датчиках и включать в себя следующие поля:
   - «номер датчика»;
   - «пространственная координата датчика (долгота)»;
   - «пространственная координата датчика (широта)».
2. *Temperatura* – таблица будет хранить данные, поступающие с датчиков:
   - «номер датчика»;
   - «время и дата считывания показаний»;
   - «значение температуры».

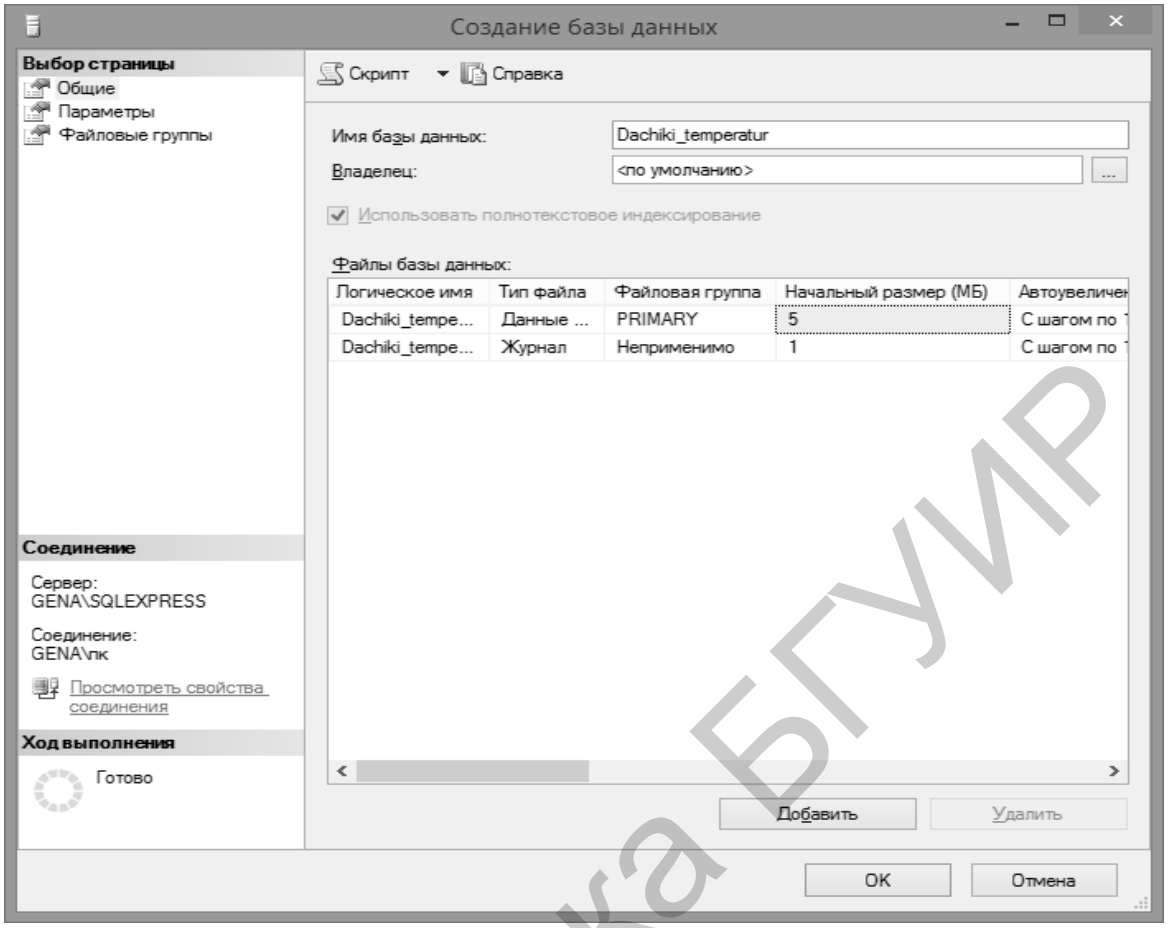


Рис. 5.18. Указание имени базы данных

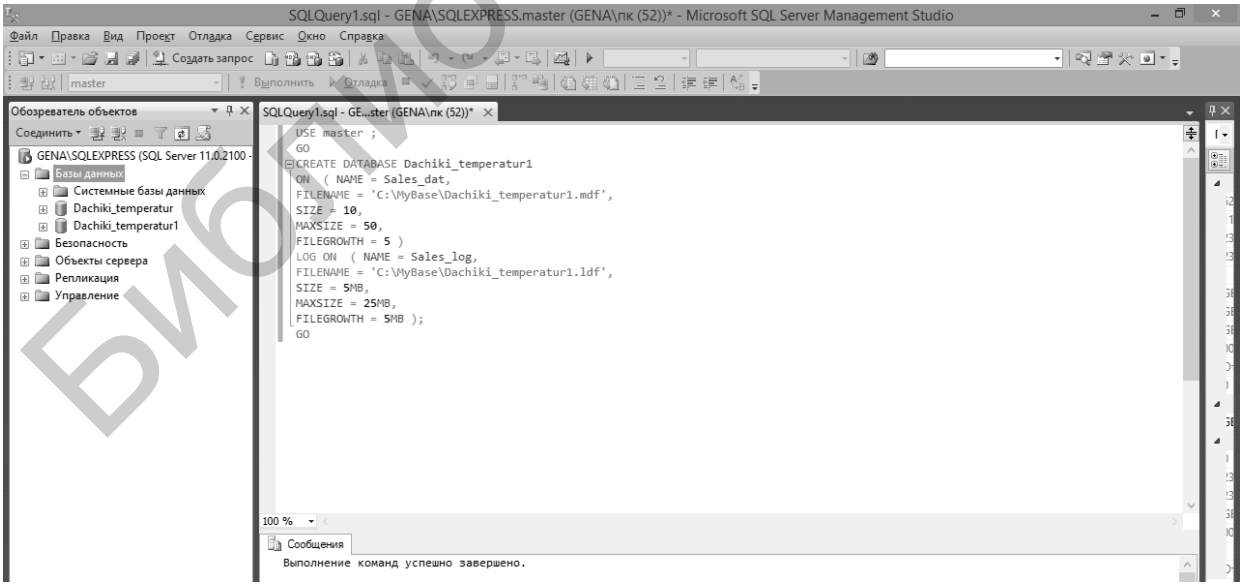


Рис. 5.19. Создание новой базы программным путем

Создание таблиц и их полей происходит через соответствующие контекстные меню SSMS: *База данных* → *Таблицы* → *Создать таблицу*…. В колонки *Имя столбца* и *Тип данных* следует внести соответствующие реквизиты полей таблицы (рис. 5.20).

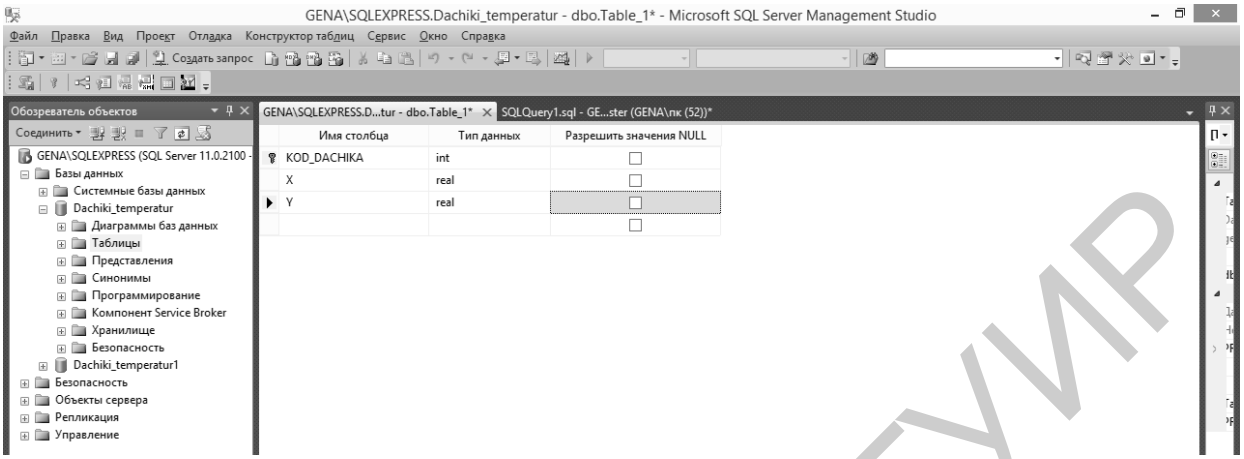


Рис. 5.20. Внесение соответствующих реквизитов полей таблицы

После обновления в *Обозревателе объектов* можно увидеть в списке таблиц созданную таблицу *Datchiki* (рис. 5.21). Префикс *dbo* в имени таблицы означает имя владельца таблицы – *database owner*.

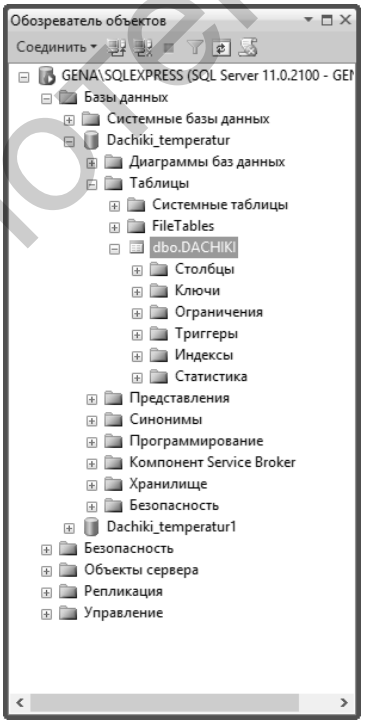


Рис. 5.21. Созданная таблица

Аналогично создается вторая таблица и связывается с первой по *первичному ключу* – полю, однозначно определяющему запись в таблице.

Далее необходимо создать соответствующие хранимые процедуры. Для этого нужно щелкнуть правой кнопкой мыши на элементе *Хранимые процедуры* и выбрать пункт *Создать хранимую процедуру…*.

Запрос SQL 1. Пример запроса SQL, в котором используется оператор *SELECT* для осуществления выборки из таблиц базы данных с целью получения необходимых полей в нужной последовательности:

```
SELECT X, Y, ID_DATCHIKA
FROM [Datchiki_temperatur].[dbo].[ DATCHIKI];
```

Запрос SQL 2. В этом примере запроса SQL символ звездочка (*) использован для вывода всех столбцов (получения всех полей отношения) таблицы *Datchiki*:

```
SELECT *
FROM Datchiki_temperatur.dbo.DATCHIKI
```

Запрос SQL 3. Инструкция *DISTINCT* используется для исключения повторяющихся записей и получения множества уникальных записей:

```
SELECT DISTINCT ID_DATCHIKA
FROM Datchiki_temperatur.dbo.DATCHIKI;
```

Запрос SQL 4. Инструкция *ORDER BY* используется для сортировки (упорядочивания) записей по значениям определенного поля. Имя поля указывается за этой инструкцией:

```
SELECT *
FROM Datchiki_temperatur.dbo.DATCHIKI
ORDER BY X;
```

Запрос SQL 5. Инструкция *ASC* используется как дополнение к инструкции *ORDER BY* и служит для определения сортировки по возрастанию. Инструкция *DESC* используется как дополнение к инструкции *ORDER BY* и служит для определения сортировки по убыванию. В случае когда ни *ASC*, ни *DESC* не указаны, подразумевается наличие *ASC*:

```
SELECT *
FROM Datchiki_temperatur.dbo.DATCHIKI
ORDER BY x DESC, y;
```

Запрос SQL 6. Для отбора необходимых записей из таблицы пользуются различными логическими выражениями, которые определяют условия отбора. Логическое выражение приводится после инструкции *WHERE*. Пример получения из таблицы всех записей, для которых значение *x* больше 40:

```
SELECT *
FROM Datchiki_temperatur.dbo.DATCHIKI
WHERE x>40;
```

Запрос SQL 7. Инструкция *BETWEEN* используется для проверки принадлежности некоторому диапазону значений. Пример запроса SQL, выводящий информацию о значениях, снятых с датчиков между 23 и 24 декабря 2016 года:

```
SELECT *
FROM Datchiki_temperatur.dbo.temperatur INNER JOIN
Datchiki_temperatur.dbo.temperatur
Datchiki_temperatur.dbo.DATCHIKI
ON Datchiki_temperatur.ID_DATCHIKA= DATCHIKI.ID_DATCHIKA
WHERE data BETWEEN '2016-12-23' And '2016-12-24'
```

5.2.1. Индивидуальные задания

Вариант 1. Создать базу данных и запрос, сортирующий по возрастанию значения температур и показывающий координаты датчиков.

Вариант 2. Создать базу данных и запрос, показывающий координаты и номера датчиков, с которых были сняты значения температур с 24.12.2016 по 25.01.2017.

Вариант 3. Создать базу данных и запрос, показывающий дату и координаты снятия температуры в диапазоне от –7 до –10 С°.

Вариант 4. Создать базу данных и запрос, показывающий минимальное и максимальное значение температуры, а также дату снятия значений температур и координаты датчиков.

5.3. Содержание отчета

1. Цель работы.
2. Краткие теоретические сведения.
3. Индивидуальные задания.
4. Код хранимой процедуры.
5. Выводы.

5.4. Контрольные вопросы

1. Каким образом можно получить доступ к базам данных SSMS?
2. С помощью каких средств можно создать таблицу в SSMS?

3. Что такое и каково назначение первичного ключа?
4. Что из себя представляют хранимые процедуры?
5. Что означает Not Null?

Рекомендуемые источники

Троелсен, Э. Язык программирования C# 6.0 и платформа .NET 4.6 / Э. Троелсен, Ф. Джепикс ; пер. с англ. – М. : Вильямс, 2016. – 1 400 с.

Лабораторная работа №6

ДОСТУП И ВИЗУАЛИЗАЦИЯ ИНФОРМАЦИИ, ХРАНЯЩЕЙСЯ В БАЗЕ ДАННЫХ РЕЗУЛЬТАТОВ ИЗМЕРЕНИЙ И МОНИТОРИНГА, С ИСПОЛЬЗОВАНИЕМ WEB-ИНТЕРФЕЙСА

Цель работы: изучить методы доступа и визуализации информации, хранящейся в базе данных результатов измерений и мониторинга, с использованием Web-форм и ознакомиться с возможностями языка программирования C# при реализации соответствующих алгоритмов.

6.1. Теоретические сведения

6.1.1. Графический пользовательский интерфейс

Исходя из представленной ранее функциональности информационной системы измерений и мониторинга должен быть разработан ее Web-ориентированный графический пользовательский интерфейс, основанный на входящих в состав универсальных элементах, компонентах и объектах, предназначенных для удаленного доступа и взаимодействия как с аппаратурой, так и с серверным программным обеспечением для работы с данными в соответствии с учетными записями пользователей. Клиентская часть включает в себя программные модули для формирования графиков и диаграмм, работы с SQL-запросами и обмена данными с базой данных. Для отображения собранных данных в удобной форме должно быть создано клиентское кроссплатформенное программное обеспечение, написанное на языке программирования C#, с учетом возможности использования под управлением многих операционных систем. Его основные возможности:

– построение различных графических зависимостей температур в соответствии с запросами пользователей, а также энергетических показателей за выбранный период;

– формирование диаграмм (гистограмм), характеризующих энергоэффективность солнечного коллектора;

– отображение текущих значений параметров установки;

– экспорт полученных данных в определенном формате и так далее.

Internet-ресурс должен предоставлять пользователям удобный Web-интерфейс для формирования и просмотра графиков и данных о работе солнечного коллектора.

Клиент посылает запрос серверу, в котором содержится информация о том, что пользователь хочет получить. Сервер обрабатывает запрос, создает запрос на выборку базе данных, в которой находится вся необходимая информация о результатах работы оборудования, и отсылает эти данные обратно пользователю в табличном или графическом виде. Пользователь может выбрать, за какой период времени необходимо посмотреть информацию о

работе солнечного коллектора. На графике можно увидеть информацию о температуре теплоносителя на выходе коллектора, а также мощность, вырабатываемую самим коллектором. Также на Web-сайте должна располагаться интерактивная схема солнечного коллектора, при наведении указателя на виртуальные датчики которой с помощью SQL-запросов выводится информация о тех характеристиках, которые снимаются в текущий момент времени. Должна также быть возможность вывести в виде гистограммы информацию о массе сбереженного условного топлива по дням (рис. 6.1).

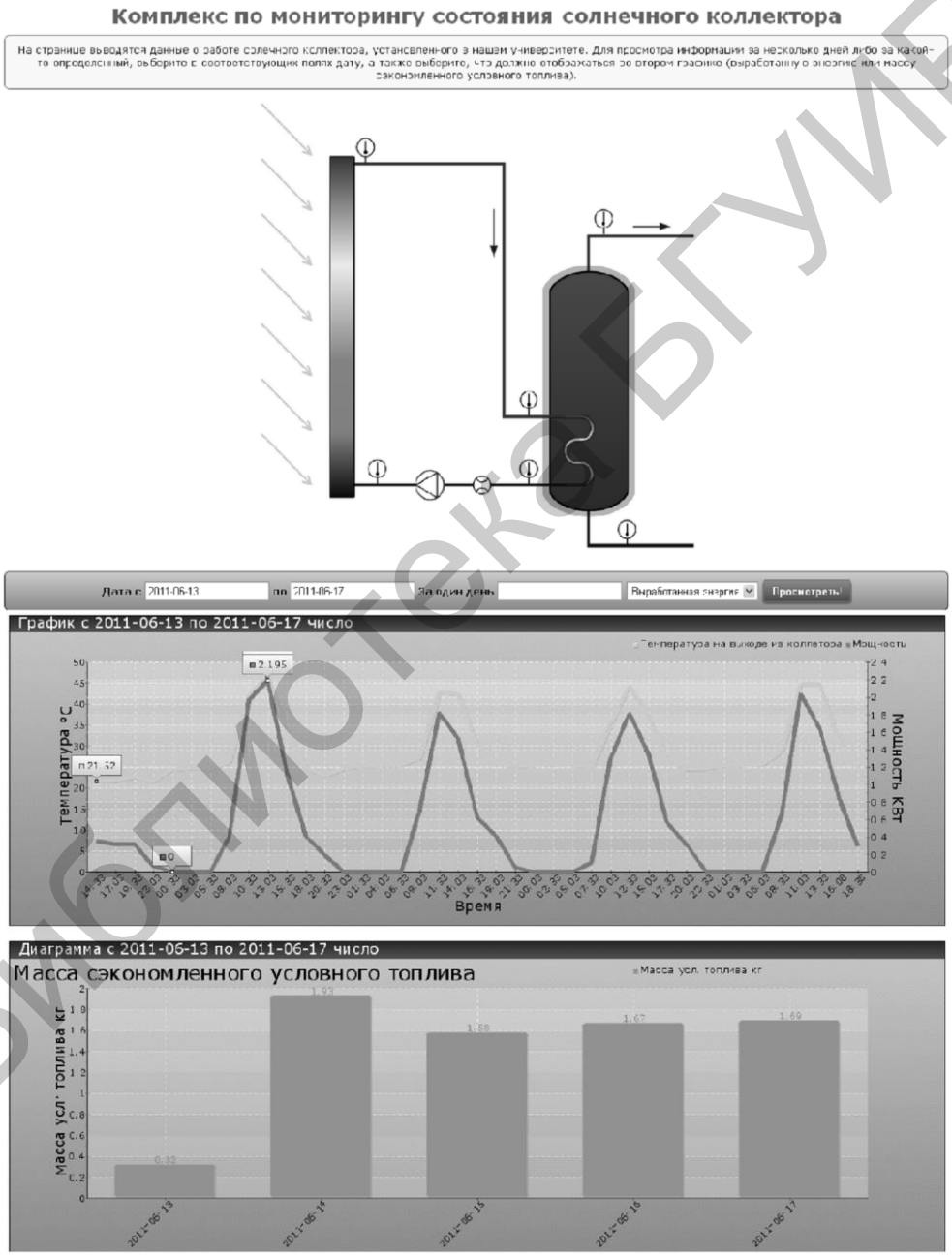


Рис. 6.1. Графический пользовательский интерфейс системы

6.1.2. Технология ASP.NET для создания Web-приложения и Web-сервера

Microsoft ASP.NET – это технология создания Web-приложений и Web-серверов. Она является составной частью платформы Microsoft .NET и развитием прежней технологии ASP.

ASP.NET основывается на единой среде выполнения *Common Language Runtime* (CLR), которая является основой всех приложений .NET. Программисты при создании приложений на ASP.NET могут писать код, используя различные языки программирования, поддерживаемые в .NET Framework, как коммерческие (Visual Basic, Visual C#, Visual C++, Visual J# и другие), так и «открытые» (Python, Perl и другие), а не только интерпретируемые языки скриптов. ASP.NET предлагает достаточно надежную модель создания Web-приложений. Например, можно разделить логику представления на HTML и бизнес-логику при помощи техники, называемой *Codebehind* (фоновый код). Для доступа к Web-приложениям клиентам нужен лишь Web-браузер.

Web-приложение – это набор взаимосвязанных файлов (HTM (HTML), ASP, ASPX, файлов изображений и тому подобное), а также связанных с ними компонентов (двоичных файлов .NET или классической COM), которые размещены на Web-сервере.

Web-сервер – это программный продукт, на котором размещаются Web-приложения и который обычно обеспечивает набор связанных с Web-приложениями служб, таких как интегрированные средства обеспечения безопасности, поддержка протокола FTP, поддержка средств передачи электронной почты и тому подобное. Web-сервер (Web-сервисы) от Microsoft называется *Internet Information Services* (IIS).

Большинство современных сред разработки Web-приложений (в том числе *Microsoft Visual Studio*) позволяют создавать Web-страницы, почти не обращаясь непосредственно к самому коду HTML с помощью специальных интегрированных средств разработки интерфейса, однако тем не менее разработчик Web-приложений должен, безусловно, знать этот язык.

Документ HTML обычно начинается с набора тегов, в которых содержится общая информация о документе: заголовок, метаданные файла и тому подобное, за которыми следует само тело документа (то есть набор текста, изображений, таблиц, гиперссылок и так далее). Теги HTML не чувствительны к регистру.

Разработку документа HTML можно начать с загрузки интегрированной среды разработки Microsoft Visual Studio и создания шаблона *Пустой веб-сайт ASP.NET* (рис. 6.2).

После этого можно добавлять в созданный проект шаблоны динамически формируемых на сервере Web-страниц ASP.NET, состоящие из файлов разметки и программной логики, содержащих коды серверной стороны.

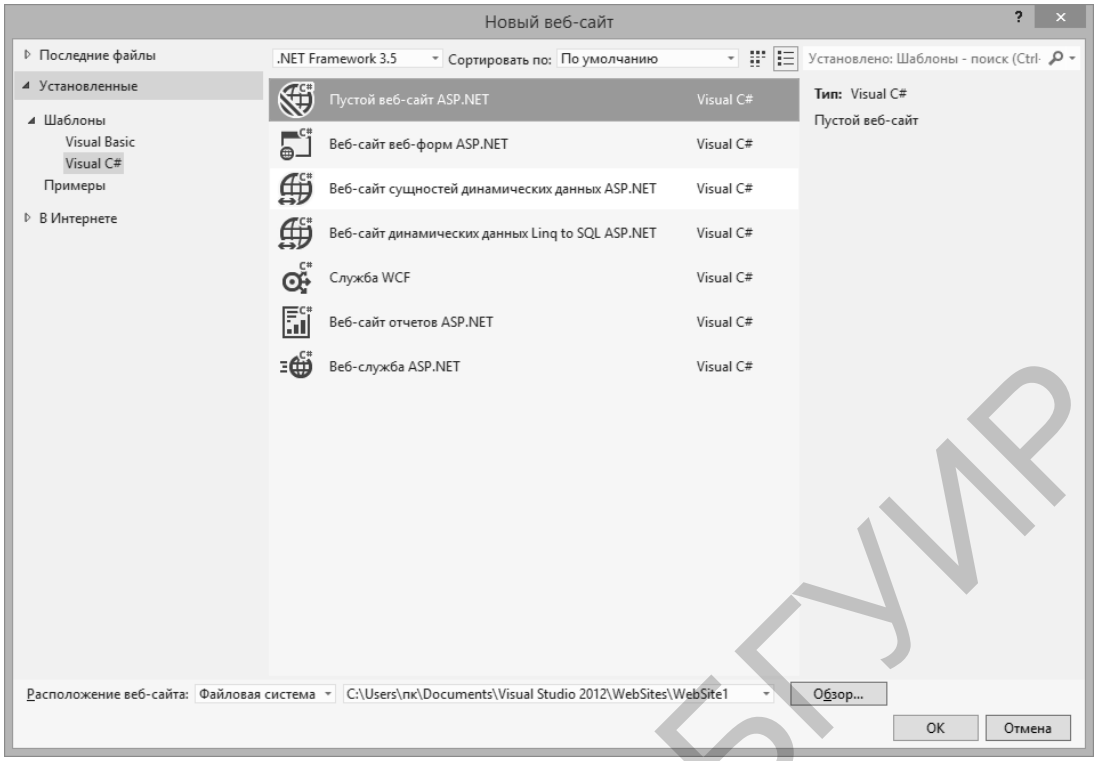


Рис. 6.2. Формирование шаблона

Графические средства, которые применяются для управления отображением всего документа, находятся в свойствах объекта Document (рис. 6.3).

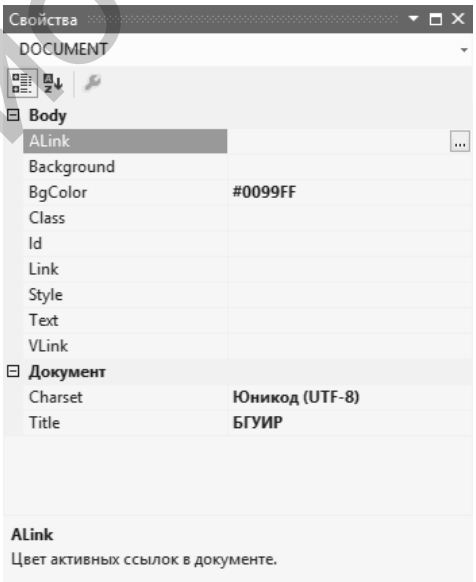


Рис. 6.3. Панель свойств объекта

В Microsoft Visual Studio также предусмотрена панель форматирования HTML. С ее помощью можно управлять представлением блоков текста, выбирая для них уровень заголовка, разметку списков, шрифт и его атрибуты и тому подобное.

Таким образом, при помощи графических средств Microsoft Visual Studio можно оформлять гипертекстовые страницы – весь необходимый для этого код HTML будет сгенерирован автоматически. Эти средства позволяют сэкономить много времени, однако иногда Web-разработчику приходится создавать код для страницы HTML вручную.

6.1.3. Разработка форм и создание пользовательского интерфейса

После внешнего оформления страницы целесообразно наделить ее новыми свойствами – возможностью принимать ввод пользователя. Для этого придется прибегнуть к помощи элементов управления HTML. Как станет очевидным далее, в среде ASP.NET предусмотрен набор элементов управления WebForm, при применении которых все необходимые теги для элементов управления HTML будут генерироваться автоматически. Однако знакомство с тегами для создания элементов управления HTML также будет нелишним. Еще раз следует подчеркнуть, что элементы управления WebForm в ASP.NET и элементы управления HTML – это разные вещи, и в процессе выполнения Web-приложения первые преобразуются во вторые.

Форма HTML – это именованная группа элементов пользовательского интерфейса HTML, используемых для ввода пользователем данных. Затем эти данные передаются на Web-сервер по протоколу HTTP. Теги для элементов пользовательского интерфейса на форме HTML помещаются между тегами <form> и </form>:

```
<form name="MainForm">
</form>
```

В этом коде создана форма и присвоено ей имя. Также можно указать идентификатор формы через свойство *Id* с передачей в него строки, например *id="MainForm"*, так как не все браузеры поддерживают свойство *Name*. С технической точки зрения использовать имя и (или) идентификатор необязательно, однако во многих ситуациях это очень удобно.

Как правило, в открывающий тег <form> помещается атрибут для действия, выполняемого этой формой. В нем содержится информация об адресе URL, на который будут передаваться данные, введенные пользователем, а также сведения о методе передачи данных. В Microsoft Visual Studio предусмотрена специальная *Панель элементов*, в которой можно выбрать различные элементы управления формой (рис. 6.4).

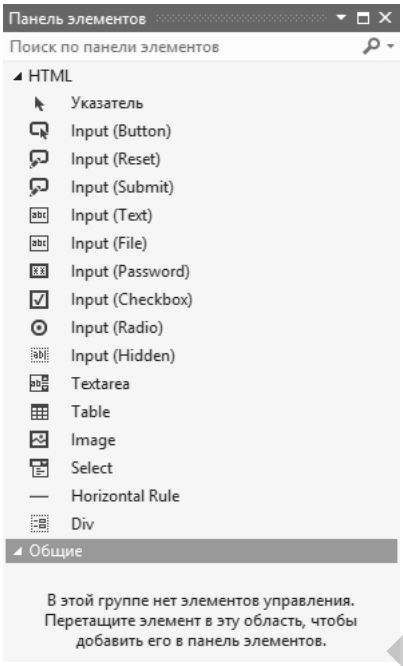


Рис. 6.4. Панель элементов

Краткий перечень наиболее часто используемых элементов представлен в табл. 6.1.

Таблица 6.1

Элементы управления HTML

Элемент управления	**Описание**
Image	Позволяет указать изображение, которое будет выведено на форме
Input (Button)	Используется для того, чтобы выполнить блок кода клиентского скрипта
Input (Checkbox) Input (Radio)	То же самое, что и аналогичные элементы управления Windows Forms, предназначенные для выбора или переключения соответственно
Input (Password)	Специальное текстовое поле, предназначенное для ввода пользователем пароля. Все введенные данные в это поле могут отображаться одинаковыми символами
Input (Reset) Input (Submit)	Специальные кнопки на форме, при нажатии которых все значения в форме принимают свой исходный вид (состояние) или производится отправка данных формы на Web-сервер соответственно
Input (Text) Textarea	Предназначены для ввода пользователем одной или нескольких строк текста
Table	Используется для формирования таблицы

В библиотеке базовых классов .NET предусмотрен набор типов .NET, которые соответствуют элементам управления HTML. Они определены в пространстве имен System.Web.UI.HtmlControls.

Если открыть автоматически сгенерированный файл ASPX, то можно найти в нем минимальный набор тегов с единственной формой:

```
<%@ Page Language="C#" AutoEventWireup="true"
CodeFile="Default.aspx.cs" Inherits="_Default" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
     <head runat="server">
          <title>Untitled Page</title>
     </head>
     <body>
          <form id="form1" runat="server">
               <div>
               </div>
          </form>
     </body>
</html>
```

В этом коде достойны внимания 2 детали:

1. Следует обратить внимание на атрибут *Runat* в открывающем теге *<form>*. Этот атрибут – один из важнейших в ASP.NET. Он означает, что данный элемент должен быть обработан средой выполнения ASP.NET, которая вернет результат Web-браузеру клиента.

2. В коде предусмотрено сразу несколько моментов, относящихся ко всей Web-странице в целом. В самом начале используется атрибут *Language*. Его значение определяет, что для создания кода HTML, возвращаемого браузеру клиента, будет использован Visual C#. Атрибут *CodeFile* определяет имя файла Visual C#, в котором содержится код этого языка и который будет использован для всех фоновых вычислений (обработок) на стороне сервера. Атрибут *Inherits* определяет имя класса, представляющего класс, определенный в *CodeFile*.

Файл *Web.config* – это текстовый файл в формате XML, который используется для определения множества параметров Web-приложения. Обычно этот файл расположен в корне виртуального каталога, соответствующего физическому, и используется для каждого подкаталога. По умолчанию в него помещается информация о компиляции, ошибках, безопасности, отладке и глобализации. Кроме того, в него могут быть помещены и другие необходимые данные. Таким образом, файл *Web.config* позволяет настраивать основные параметры Web-приложения.

Как и в классических ASP, в ASP.NET используется глобальный файл (*Global.asax*), который позволяет взаимодействовать с событиями как уровня всего приложения, так и уровня сеанса подключения. Кроме того, этот файл делает возможным совместное использование различных общих данных. Это реализуется при помощи информации, представленной классом *Global*, являющимся производным от базового класса *HttpApplicatlon*.

В некоторых отношениях класс *Global* действует в качестве промежуточного звена между внешним клиентом и элементами управления WebForm, как в классических ASP. В общем, эти события позволяют реагировать на запуск и прекращение работы как Web-приложения в целом, так и отдельных сеансов подключения.

Для более глубокого понимания механизма работы Web-приложений ASP.NET следует рассмотреть пример кода на Visual C#, внедренного в страницу ASPX. Если после создания шаблона обратиться по адресу Web-приложения, то среда выполнения ASP.NET вернет пустую страницу. Можно исправить эту ситуацию, например, изменив содержание файла Default.aspx таким образом, чтобы возвращалась информация о произведенном запросе HTTP:

```
<body>
     <span style="font-size: 14pt">
          <strong>
               Источник:
          </strong>
     </span>
     <%=Request.ServerVariables["HTTP_USER_AGENT"] %>
     <br />
     <br />
     <span style="font-size: 14pt">
          <strong>
               Приемник:
          </strong>
     </span>
     <%=this.ToString() %>
     <form id="form1" runat="server" method="post">
     </form>
</body>
```

6.2. Порядок выполнения

При создании пользовательского интерфейса первой необходимостью является обеспечение возможности страницы воспринимать ввод информации пользователя. Когда форма создана, можно приступать к добавлению в нее элементов управления. Это можно сделать при помощи графических средств Microsoft Visual Studio, а можно создать все необходимые теги вручную. Каждый элемент управления описывается атрибутом имени (имя используется при выполнении программы, чтобы определить, например, в какой элемент

управления были введены данные) и атрибутом типа (этот атрибут и определяет разновидность элемента управления). Для разных элементов управления существуют разные наборы дополнительных атрибутов, которые могут быть использованы для определения их параметров. Конечно же, эти дополнительные параметры можно также настроить при помощи окна свойств для соответствующего элемента управления в Microsoft Visual Studio.

Предполагается, что форма будет содержать, в частности, 2 текстовых поля (одно – для ввода имени пользователя, другое – специальное – для ввода пароля) и две кнопки для передачи информации на сервер и восстановления формы в исходном состоянии, если пользователь решает отменить свой ввод (рис. 6.5).

Рис. 6.5. Интерфейс главной формы

Далее необходимо реализовать подключение и настроить доступ и получение информации из базы данных с выводом результатов в соответствующие элементы управления Web-формы.

6.2.1. Настройка доступа к базам данных в SSMS

Прежде чем приступать к использованию элементов управления для работы с источниками данных, необходимо определенным образом произвести настройку доступа к базам данных в SSMS (рис. 6.6–6.8).

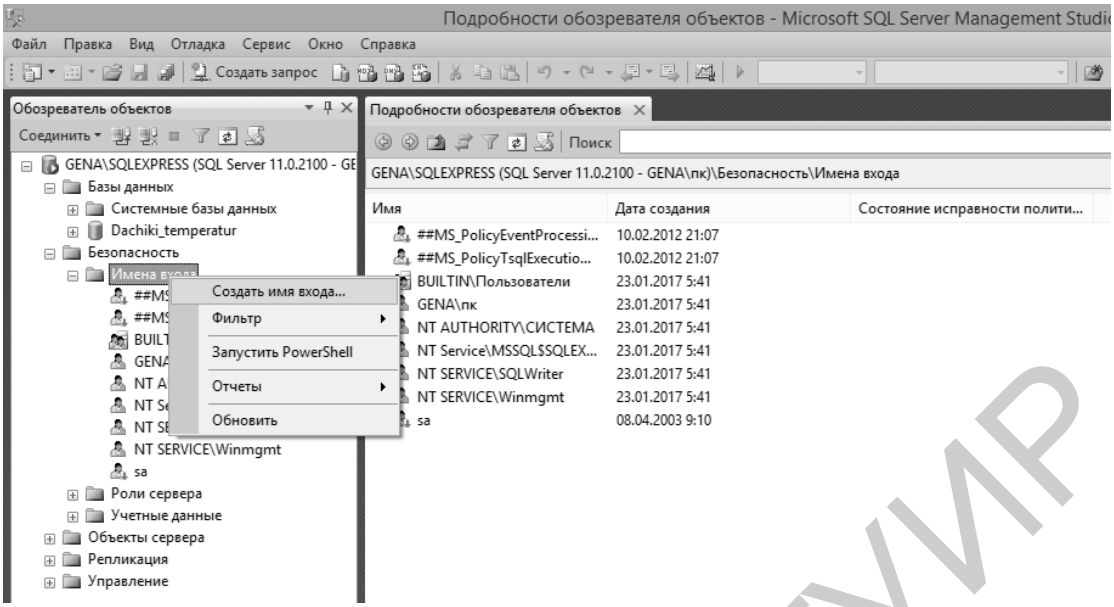


Рис. 6.6. Добавление имени новой учетной записи (сеанса)

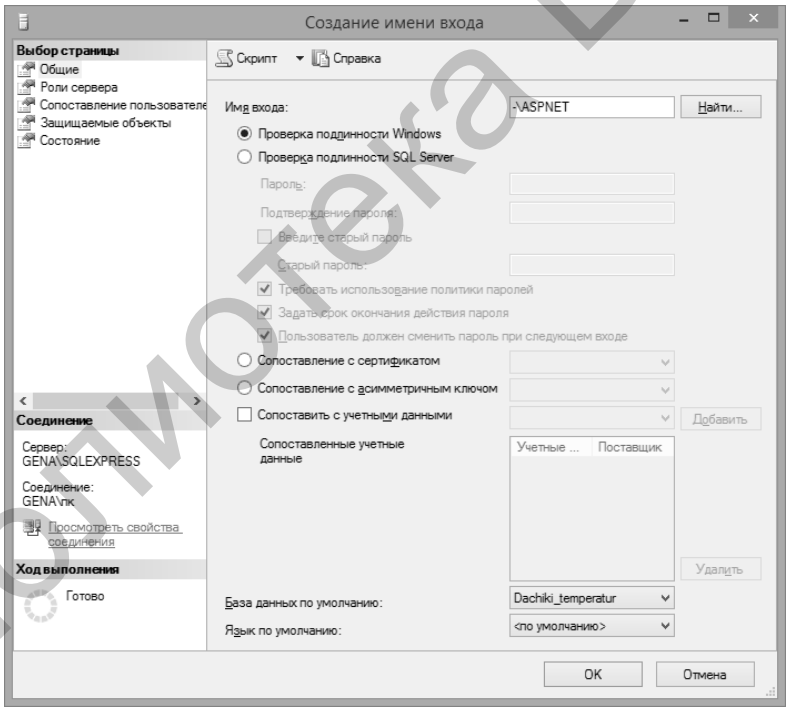


Рис. 6.7. Настройка свойств учетной записи ASPNET
(указание исходной базы данных)

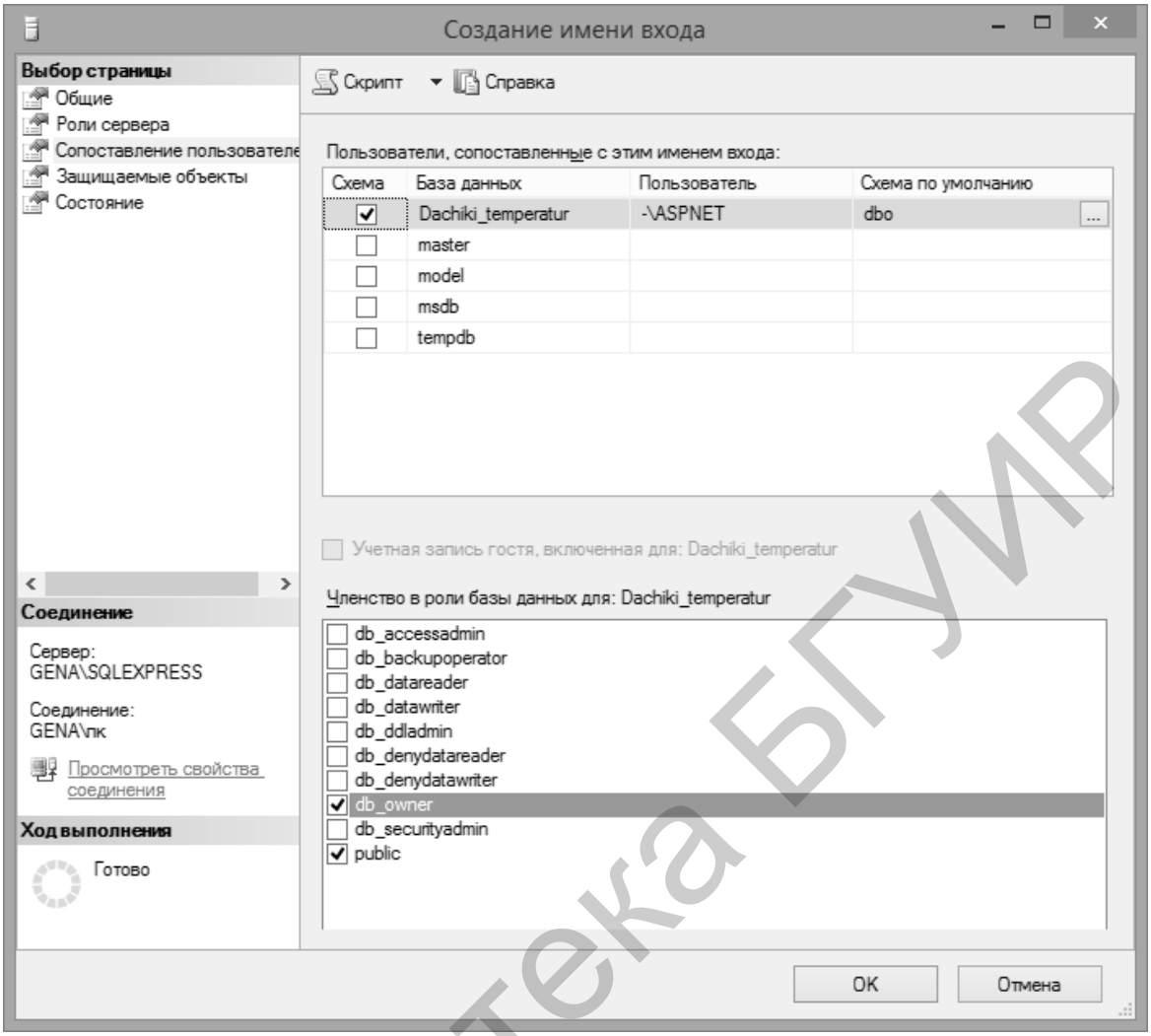


Рис. 6.8. Настройка свойств учетной записи ASPNET
(выбор ролевого членства для базы данных)

Фиксированная роль *db_owner* предоставляет все возможности манипулирования базой данных, в частности конфигурирование, сопровождение и выполнение хранимых процедур. Разрешения для этой роли включают разрешения всех остальных фиксированных ролей базы данных. В последних версиях Microsoft SQL Server члены указанной роли могут удалять базу данных.

6.2.2. Элементы управления для работы с источниками данных

В ASP.NET предусмотрено 2 элемента управления WebForm, предназначенных для отображения данных, полученных из источника (обычно в качестве источника в приложениях ASP.NET выступает объект ADO.NET *DataSet*, который, в свою очередь, может быть, например, заполнен данными с сервера баз данных). Эти элементы управления представлены в табл. 6.2.

Таблица 6.2

Элементы управления WebForm, предназначенные для работы с источниками данных

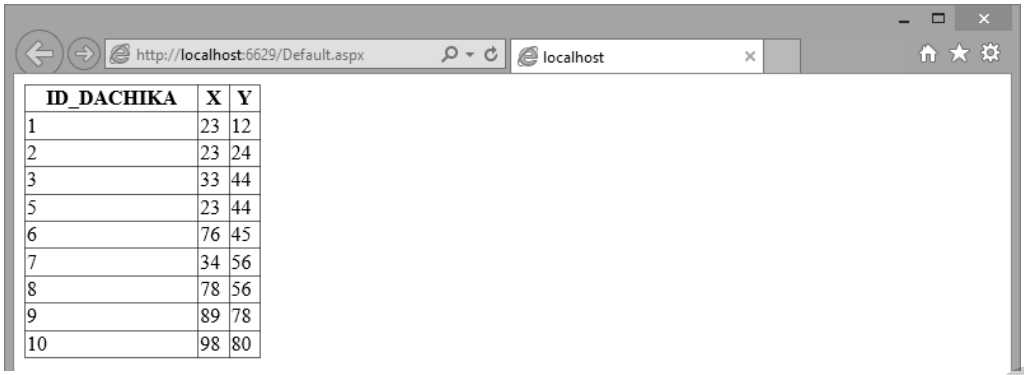
Элемент управления	**Описание**
DataGrid	Отображает содержимое объекта ADO.NET DataSet в виде таблицы
DataList	Служит для выбора значений, поступающих из источника данных

Кроме того, для работы с источниками данных можно настроить некоторые из базовых типов данных. Одна из наиболее часто встречающихся задач Web-приложений – нахождение каких-либо данных в источнике данных по запросу пользователя и возврат их в табличном формате. В классических ASP это делалось путем создания объекта ADO *Recordset* и таблицы HTML «на лету» с использованием данных из этого объекта. Тех же самых результатов гораздо проще можно достичь при помощи элемента управления WebForm – *DataGrid*.

Предполагается, что необходимо предоставить пользователю в ответ на его запрос данные из базы данных *Datchiki_temperatur*, созданной ранее. Первое, что нужно сделать, – создать обработчик для события *Load* страницы. В нем следует установить соединение с базой данных, создать и заполнить объект *DataSet* и указать его в качестве источника данных для элемента управления *DataGrid*. Соответствующий код на C# может выглядеть так:

```
protected void Page_Load(object sender, EventArgs e)
{
     if (!IsPostBack)
     {
          SqlConnection connection = new SqlConnection();
          connection.ConnectionString = @"Data Source=-
          \SQLEXPRESS; Initial Catalog=Datchiki_temperatur;
          Integrated Security=sspi";
          SqlDataAdapter dsc = new SqlDataAdapter("Select *
          from DATCHIKI", connection);
          DataSet ds = new DataSet();
          dsc.Fill(ds, "DATCHIKI");
          DataGrid1.DataSource =
          ds.Tables["DATCHIKI"].DefaultView;
          DataGrid1.DataBind();
     }
}
```

Возможный результат работы программы представлен на рис. 6.9.



ID_DACHIKA	X	Y
1	23	12
2	23	24
3	33	44
5	23	44
6	76	45
7	34	56
8	78	56
9	89	78
10	98	80

Рис. 6.9. Элемент управления DataGrid с данными, полученными из базы данных под управлением Microsoft SQL Server

Для текстовых полей Web-интерфейса может возникнуть необходимость использования элементов управления WebForm для проверки вводимых пользователем данных. Наиболее важные элементы управления этого типа представлены в табл. 6.3.

Таблица 6.3

Элементы управления для проверки данных

Элемент управления	Описание
CompareValidator	Сравнивает значение, введенное в один элемент управления, со значением, введенным во второй
CustomValidator	Позволяет определить пользовательский метод, при помощи которого будет производиться проверка
RangeValidator	Определяет, попадает ли введенное пользователем значение в определенный диапазон
RegularExpressionValidator	Проверяет введенное значение на соответствие подстановочному выражению
RequiredFieldValidator	Позволяет убедиться, что в соответствующий элемент управления действительно введено значение (он не оставлен пустым)
ValidationSummary	Отображает все ошибки, обнаруженные при проверке ввода, в виде списка, маркированного списка или обычного абзаца. Ошибки могут отображаться на Web-странице или в специальном окне оповещения Web-браузера

6.2.3. Реализация визуализации графических зависимостей

Одной из наиболее распространенных графических задач является построение диаграмм. Элемент управления *Chart* в ASP.NET предлагает широкий набор типов диаграмм и параметров конфигурации. Элемент управления *Chart* был доступен как загружаемый дополнительно в версии .NET 3.5 SP1, но теперь входит в состав .NET 4.0.

Диаграмму можно добавить, переместив элемент управления *Chart* из панели инструментов (он находится в группе *Data* (*Данные*)) на поверхность визуального конструктора или поместив дескриптор *<asp:Chart>* в код разметки. Объявление для этого примера диаграммы выглядит так:

```
<asp:Chart ID="Chart1" runat="server">
     <ChartAreas>
          <asp:ChartArea Name="ChartArea1" />
     </ChartAreas>
</asp:Chart>
```

При подключении к базе данных получится график, изображенный на рис. 6.10.

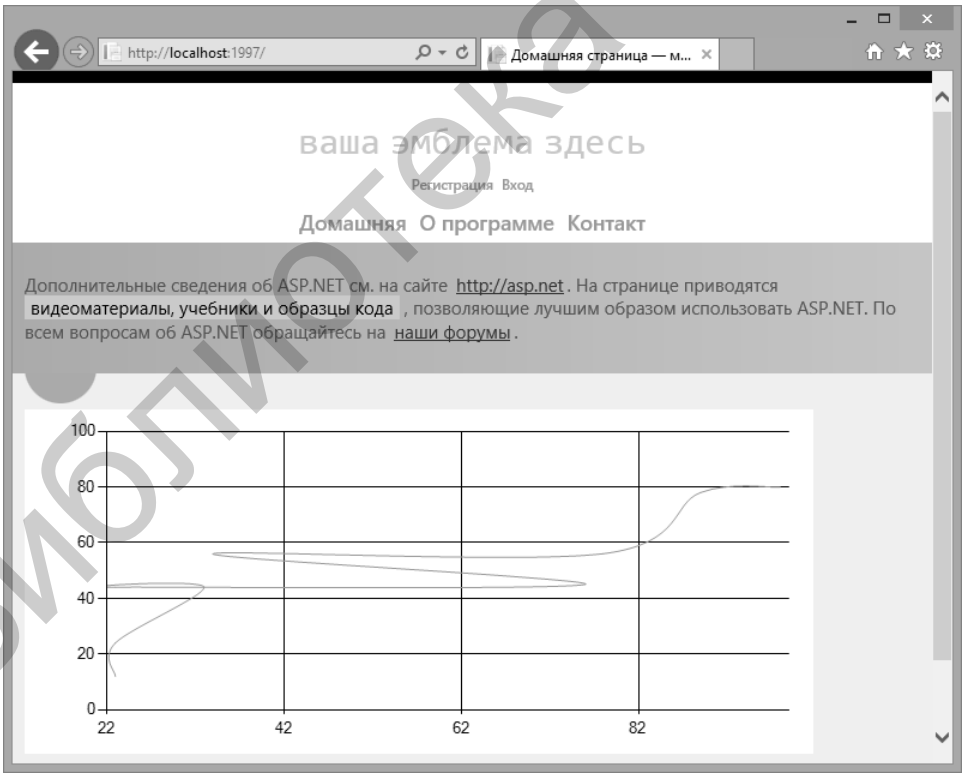


Рис. 6.10. Визуализация графической зависимости

Также для построения диаграмм можно использовать средство для формирования отчетов ASP.NET (рис. 6.11).

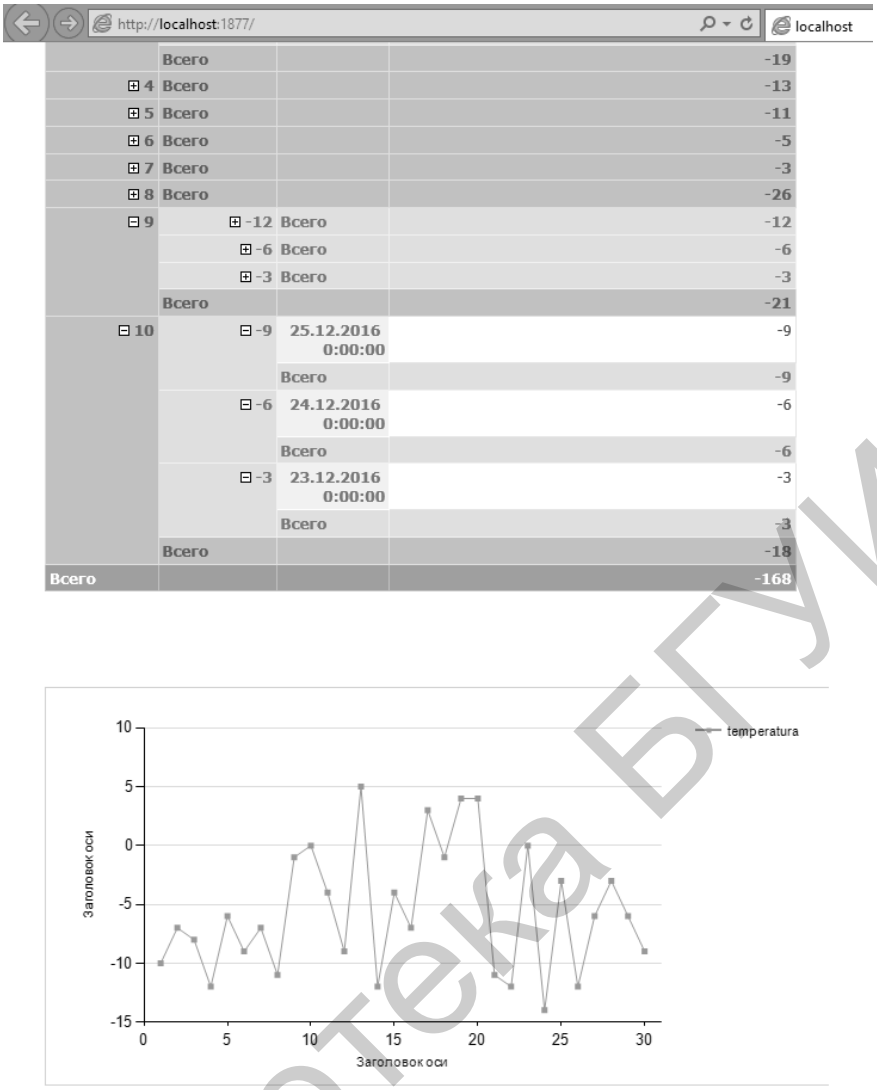


Рис. 6.11. Средство для формирования отчетов

6.2.4. Обработка событий элементов управления

События, генерируемые элементами управления WebForm, можно перехватывать и обрабатывать двумя способами:

1. Делать все непосредственно в Web-браузере клиента при помощи клиентских браузерных скриптов JavaScript. Это традиционный подход, который наиболее удобен в тех ситуациях, когда нужно выполнять форматирование на Web-странице, выводить оповещения в окне Web-браузера или осуществлять прочие взаимодействия с объектной моделью, реализованной в Web-браузерах.

2. Обрабатывать и перехватывать события элементов управления WebForm на Web-сервере. Для этого достаточно добавить обработчик события при помощи окна Properties свойств элемента управления (пиктограмма Events). Обычно такой способ наиболее удобен для выполнения операций, не связанных

с графическим интерфейсом, например для производства каких-то вычислений, редактирования таблицы с данными и тому подобное.

Таким образом, при разработке Web-приложений ASP.NET необходимо учитывать то, что каждому шаблону HTML-приложения (файлу ASPX) соответствует ASP.NET-класс, производный от System.Web.UI.Page. Работать с этим классом можно средствами привычных языков программирования .NET, например C#. Поэтому в ASP.NET можно использовать технологии объектно-ориентированного программирования, создавая код, пригодный для повторного использования. Основные свойства объекта Page (Session, Application, Request и Response) обеспечивают доступ к внутренним объектам класса, производного от Page. Элементы управления WebForm во многом аналогичны стандартным элементам управления Windows Forms, с помощью которых можно избежать трудоемкой и утомительной обязанности создавать теги HTML и клиентские скрипты вручную.

6.2.5. Индивидуальные задания

Требуется реализовать Web-сайт, состоящий из следующих Web-страниц:

1. Авторизация.
2. Доступ к данным и визуализация.
3. Выведение на Web-страницу данных из предыдущей лабораторной работы, а также фамилии и инициалов студентов.

6.3. Содержание отчета

1. Цель работы.
2. Краткие теоретические сведения.
3. Индивидуальные задания.
4. Код программы.
5. Выводы.

6.4. Контрольные вопросы

1. Каким образом происходит компиляция ASP.NET-приложения?
2. Как компилируются страницы ASPX?
3. Каким образом среда исполнения ASP.NET может отслеживать изменение страниц ASP.NET?
4. Что происходит при изменении страниц ASP.NET на Web-сервере?

Рекомендуемые источники

Троелсен, Э. Язык программирования C# 6.0 и платформа .NET 4.6 / Э. Троелсен, Ф. Джепикс ; пер. с англ. – М. : Вильямс, 2016. – 1 400 с.

Св. план 2017, поз. 5

Учебное издание

Тонконогов Борис Александрович
Высоцкий Олег Павлович

ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ЭЛЕКТРОННЫХ СРЕДСТВ. ЛАБОРАТОРНЫЙ ПРАКТИКУМ

ПОСОБИЕ

Редактор *М. А. Зайцева*
Корректор *Е. И. Герман*
Компьютерная правка, оригинал-макет *В. М. Задоля*

Подписано в печать 05.12.2017. Формат 60x84 1/16. Бумага офсетная. Гарнитура «Таймс».
Отпечатано на ризографе. Усл. печ. л. 4,07. Уч.-изд. л. 4,0. Тираж 30 экз. Заказ 338.

Издатель и полиграфическое исполнение: учреждение образования
«Белорусский государственный университет информатики и радиоэлектроники».
Свидетельство о государственной регистрации издателя, изготовителя,
распространителя печатных изданий №1/238 от 24.03.2014,
№2/113 от 07.04.2014, №3/615 от 07.04.2014.
ЛП №02330/264 от 14.04.2014.
220013, Минск, П. Бровки, 6