

1С:Программирование для начинающих
Детям и родителям, менеджерам и руководителям
Разработка в системе 1С:Предприятие 8.3

Радченко М. Г.

2017

Выходные данные

Книга составлена по материалам [электронного аналога](#) издания «1С:Программирование для начинающих. Детям и родителям, менеджерам и руководителям. Разработка в системе 1С:Предприятие 8.3» (ISBN 978-5-9677-2628-6, М.: ООО «1С-Паблишинг», 2017) опубликованного в информационной системе ИТС ПРОФ (ISBN 978-5-9677-2631-6).

Книга адресована читателям, которые совсем не знают программирования, но хотят научиться создавать собственные программы в системе 1С:Предприятие 8. Она подойдёт и школьникам 12–16 лет, и взрослым, которые хотели бы научиться «программировать в 1С».

В книге рассматривается практический пример создания простого прикладного решения. Он позволяет освоить базовые понятия и базовые приёмы программирования, научиться использовать среду разработки (конфигуратор), овладеть встроенным языком и языком запросов, познакомиться с устройством базы данных, приобрести навыки отладки прикладных решений.

Книга содержит большое количество рисунков и примеров кода на встроенном языке, снабжённых подробными комментариями. Кроме этого после многих ключевых разделов даются задания для самостоятельной работы, ответы на эти задания содержатся в конце книги.

Для создания демонстрационных примеров использована учебная версия платформы 8.3.8.1933. Для самостоятельного выполнения этих примеров требуется доступ к Интернету, чтобы скачать (бесплатно) учебную версию платформы и демонстрационные конфигурации.

Предисловие

Это необычная книга.

Это книга «про 1С» для тех, кто совсем не знает программирования.

Она научит вас создавать собственные программы в системе 1С:Предприятие 8. Она научит вас разбираться в чужих программах, исправлять ошибки, добавлять в программы что-то новое. Но это не главное.

Главное и необычное заключается в том, что эта книга научит вас думать «как программист». Мыслить не определениями и правилами, а образами и аналогиями. Именно таким способом вы будете знакомиться и с компьютером вообще, и с программами, и с языками, и с самой системой 1С:Предприятие.

Поэтому книга подойдёт и школьникам 12–16 лет, и взрослым, которые хотели бы научиться «программировать в 1С». Она даст вам основы, о которых обычно не рассказывают или рассказывают очень мало.

Обычно объясняют, что из чего состоит, какая команда что делает, какую кнопку нужно нажать. И только через некоторое время, с опытом, вы начинаете понимать, как же всё это «устроено на самом деле». В голове у вас появляется какой-то образ.

В этой книге будет ровно наоборот. Вы сразу будете пытаться искать аналогии, придумывать образы, воображать, на что из привычных вещей это похоже. Что вы хотите сделать или где вы сейчас находитесь. А какую именно команду написать или какую кнопку нажать, вам всегда подскажет компьютер и 1С:Предприятие. Главное — знать, куда обратиться за подсказкой. Этому вы тоже научитесь.

Все эти базовые знания пригодятся вам не только при создании программ «1С», но и при работе в любой другой среде разработки, при использовании любого другого языка программирования. Поэтому книга будет полезна просто для того, чтобы «войти» в мир программирования и понимать, «как всё устроено».

Благодарности

Спасибо Егору Радченко, Ивану Бойко и Егору Бойко за тестирование глав книги. Их отзывы и комментарии помогли улучшить эту книгу и сделать её более понятной.

Как работать с книгой

Уровень ваших знаний о компьютерах и программировании может быть разным. Это не зависит от возраста. Часто школьники и студенты могут знать больше, чем взрослые.

Но мне хотелось сделать эту книгу удобной и интересной независимо от того, какая начальная подготовка у вас есть. Поэтому существуют два способа читать эту книгу.

Если вы ничего не знаете, вы можете читать её по порядку. Все объяснения будут вам понятны независимо от возраста. Незнакомые действия и термины я буду вводить

постепенно. Вы не должны столкнуться с тем, что нужно сделать что-то, о чём я ещё не рассказывал.

Если вы что-то знаете, вы можете пропускать отдельные части книги. Специально для этого в начале каждой главы есть такие примечания:

Не читайте эту главу!

Если вы:

- знаете, что такое программа и что такая операционная система;
- понимаете, чем прикладное программное обеспечение отличается от системного;
- знаете, что такое среда исполнения;

вы можете смело перейти к разделу [1.3 «Как устроено 1С:Предприятие»](#) на странице [18](#).

Это избавит вас от «унылого» чтения про очевидные вещи. Вы сможете заняться только «крутыми» заданиями и примерами.

Но на этом пути вы можете столкнуться с тем, что я использую незнакомые вам термины или прошу вас выполнить незнакомое действие. Тут вам помогут два приложения: **В «Указатель понятий»** и **Г «Указатель действий»**. Они находятся в конце книги. В них вы найдёте то, что вам незнакомо, и номер страницы, где об этом написано.

По ходу всей книги вы будете выполнять один большой пример, который в конце превратится в настоящую полезную программу. **Разные части этого примера** вы можете скачать и посмотреть в любой момент, для того чтобы проверить себя, если где-то запутались или если что-то не получается. Как это сделать, написано в разделе [А.1 «Как подключить демонстрационную базу»](#) на странице [543](#).

После многих разделов есть **задания для самостоятельной работы**. А в конце книги, в приложении **Б**, есть **решения этих заданий**. Тренируйтесь, проверяйте себя. Если ваше решение не совпадает с ответом, это ещё не значит, что ваше решение неправильное. Программирование — это творческий процесс. Одну и ту же вещь можно сделать разными способами. В решении я показываю понятный и удобный способ. Но во многих случаях существуют и другие способы — может быть, менее понятные или менее удобные.

Что вы будете уметь

Вы научитесь создавать прикладные решения 1С:Предприятия. Научитесь изменять их. Научитесь работать с ними не только как программисты, но и как обычные пользователи. Но это не самое интересное.

Самое интересное, что вы научитесь многим вещам, которые пригодятся даже без системы программ 1С:Предприятие.

В процессе создания своей программы вы освоите базовые понятия, базовые приёмы программирования. Они одинаковы для всех распространённых систем и языков программирования.

Вы научитесь использовать среду разработки 1С:Предприятия, конфигуратор. Приёмы и понятия, которые в ней используются, есть и в других средах разработки. Если потом вы решите освоить другую систему разработки, многие вещи будут вам понятны и знакомы.

Вы научитесь писать программы на встроенном языке 1С:Предприятия. Он имеет много общих черт с другими популярными языками программирования. Если потом вы захо-

тите освоить другой язык, сделать это будет гораздо проще, чем если бы вы начинали с нуля.

Вы познакомитесь с тем, как и где 1С:Предприятие хранит свои данные. Вы будете понимать, как устроены базы данных. Если потом вы захотите ближе познакомиться с одной из них, начальные знания у вас уже будут.

И, наконец, вы научитесь использовать ещё один язык, который есть внутри 1С:Предприятия, — язык запросов. Язык запросов 1С:Предприятия основан на одном из самых популярных языков работы с данными — SQL. Поэтому последующее освоение языка SQL или егоialectов будет для вас совсем простым.

Что вы будете делать

Всё, что вы будете изучать, вы будете сразу же пробовать на компьютере, в 1С:Предприятии. В результате вы сделаете программу, прикладное решение, которая будет называться *Дневник*.

Вы наверняка учились в школе или ещё учитесь. Поэтому вы прекрасно знаете, что такое школьный дневник. Школьный дневник раньше был в виде тетрадки (рисунок 1) теперь во многих школах он существует в электронном виде.

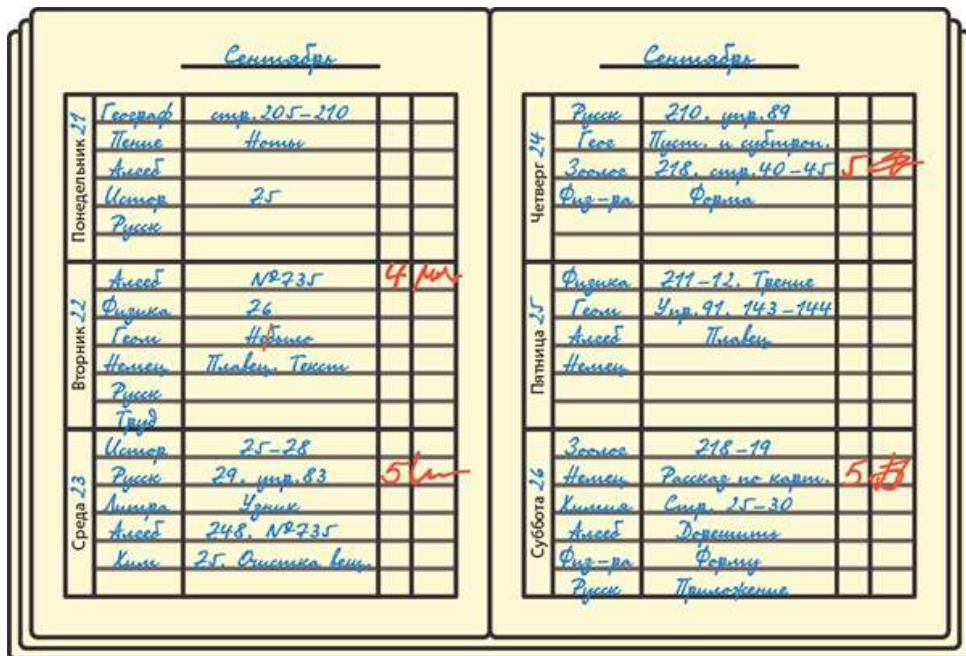


Рисунок 1. Школьный дневник

Вы сделаете свою версию электронного школьного дневника с помощью системы программ 1С:Предприятие.

Глава 1

Начало

Не читайте эту главу!

Если вы:

- знаете, что такое программа и что такое операционная система;
- понимаете, чем прикладное программное обеспечение отличается от системного;
- знаете, что такое среда исполнения;

вы можете смело перейти к разделу [1.3 «Как устроено 1С:Предприятие»](#) на странице [18](#).

Если кроме этого вы:

- знаете, что такое прикладные решения 1С:Предприятия;
- понимаете, для чего нужны платформа и конфигуратор;
- знаете, почему 1С:Предприятие является системой программ;

вы можете смело перейти к разделу [1.5 «Установка»](#) на странице [25](#).

Если у вас уже установлена платформа 1С:Предприятие и вы знаете, как её запускать, смело переходите к главе [2 «Визуальное конструирование»](#) на странице [37](#).

Если вы не прочитали предисловие, сейчас самое время это сделать. Иначе вам будет непонятно, почему следующий раздел называется [«Воображение»](#).

1.1 Воображение

Одно упражнение понадобится вам с самого начала. Вы его хорошо знаете, но вряд ли вы думали, что оно может иметь непосредственное отношение к компьютерам.

Представьте, что вы стоите на лугу и смотрите вдаль (рисунок [1.1](#)). Что вы видите?

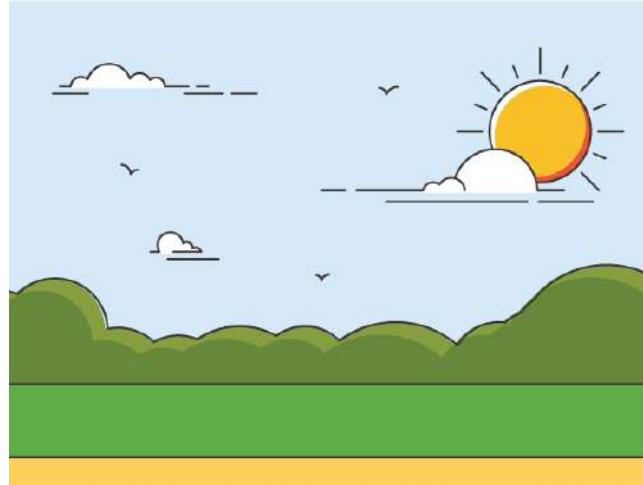


Рисунок 1.1. Лес вдалеке

Вы видите, что где-то есть лес. Где-то его нет. Какой именно там лес, густой он или нет, не видно с такого расстояния. Единственное, что вы можете точно сказать, что «в той стороне есть лес». И в другой стороне тоже есть ещё один лес.

Теперь подойдите ближе (рисунок 1.2).

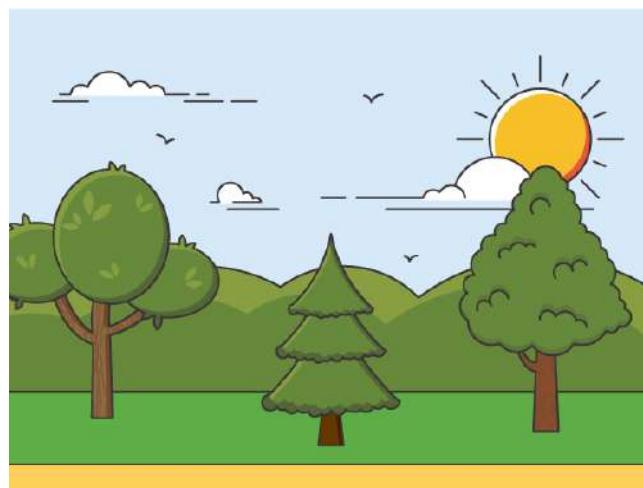


Рисунок 1.2. Лес вблизи

Всё изменилось. Вы видите, что лес — это не сплошная зелёная масса. Он состоит из разных деревьев. В нём есть высокие деревья, есть деревья пониже. Есть хвойные деревья, есть лиственные. Вы можете сказать, что справа и слева есть лиственные деревья. А в центре есть хвойные.

Однако теперь вы ничего не можете сказать про лес «вообще». Теперь вы не можете ответить на вопросы: «А в той стороне есть лес?», «А в этой?» С того места, где вы находитесь, уже не видно «другие леса».

Что же произошло? Ведь лес никуда не делся. Деревья не умеют ходить.

Произошло то, что вы изменили своё положение. Вы приблизились к лесу. И, находясь здесь, в новом месте, вы уже не можете говорить теми словами, которыми говорили на старом месте: «лес», «луг». Зато здесь у вас появились новые слова: «хвойные деревья», «лиственные деревья». А раньше для вас это была одна «зелёная масса».

Идите дальше. Зайдите в лес (рисунок 1.3).

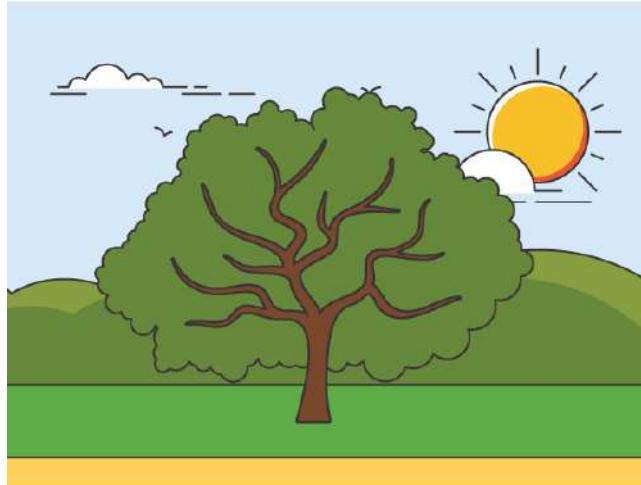


Рисунок 1.3. Дерево

Опять всё изменилось.

Теперь вы видите, что каждое дерево состоит из ствола и веток. Ствол толстый и расположен вертикально. Ветки тоньше и расположены горизонтально по разные стороны ствола. Вы опять используете новые слова: «ствол», «ветка».

Теперь, находясь в этом месте, вы можете сделать пару важных замечаний.

Примечание

Оказывается, вы не просто двигаетесь вперёд. Оказывается, не просто меняются «картинки» вокруг вас одна за другой. Нет. На самом деле вы как бы «проникаете внутрь» тех слов, понятий, которые видите. Вы даже так и говорите: «Войти в лес».

В самом начале вы видели лес, теперь вы в него вошли. По мере своего движения вы видите, что предмет, понятие, которое раньше казалось вам одним целым, на самом деле состоит из множества разных частей.

Примечание

Во время каждой остановки вы используете слова и понятия, обозначающие то, что находится вокруг вас. При этом слова и понятия, которые вы использовали раньше, для вас уже недоступны. Они находятся где-то «снаружи».

Но благодаря им вы имеете представление о том, где вы находитесь сейчас. Вы понимаете, что вы «внутри леса». Внутри какого именно леса, уже не так важно. Того, что был на пригорке, или того, что был на горизонте.

Продолжайте рассматривать деревья. Вы видите, что есть деревья с короткими стволами и множеством веток. А есть деревья, у которых длинный и высокий ствол, а веток даже и не видно. Настолько высоко они находятся. Но вы понимаете, что, несмотря на это, все деревья похожи друг на друга. У них у всех есть корни, ствол и ветки.

Из этого вы можете сделать ещё одно интересное наблюдение.

Примечание

Несмотря на то что все деревья разные, вам не нужно изучать каждое из них в отдельности. Достаточно понять, как устроено одно из деревьев, а остальные деревья, по большому счёту, будут устроены точно так же.

Конечно, у каждого из них будут свои особенности. И вообще двух совершенно одинаковых деревьев не бывает. Но сейчас, когда вы стоите рядом с ними, все они для вас выглядят похожими друг на друга. Корень, ствол, ветки. Где-то наверху «зелёная шапка». Особенностей вы не замечаете. Вам их не видно. И сейчас они вас даже и не интересуют.

В лес вы уже зашли, теперь забирайтесь на дерево. Когда вы окажетесь наверху, вы увидите, что на ветках есть листья (рисунок 1.4).



Рисунок 1.4. Листья на ветках

Здесь вы снова можете заметить, что «зелёная шапка», которая раньше казалась вам одним целым, на самом деле состоит из отдельных листьев. А все листья похожи друг на друга.

Если притянуть ветку к себе, то вы увидите, как выглядит один лист (рисунок 1.5).



Рисунок 1.5. Лист

Таким образом вы максимально глубоко проникли «внутрь» леса. «Залезть» ещё глубже, внутрь листа, вам не удастся без специальных инструментов. Да это и не нужно.

А теперь спуститесь с дерева и потом выйдите обратно на луг. Что будет происходить? Что вы увидите?

В какой момент вы снова увидите ствол, ветви? Когда вы увидите, что лес хвойный? Или, наоборот, лиственный? Когда вам станут видны другие рощи и перелески?

В каждый из этих моментов попробуйте представить, где вы находитесь. И где вы находились до этого.

Такое, казалось бы, простое упражнение очень поможет вам во всех дальнейших занятиях.

Задание 1.1

Если вы хотите потренироваться ещё, вот вам несколько картинок. Попробуйте из них составить такое же «путешествие». Попробуйте, глядя на каждую из картинок, записать, какие слова вы используете для того, чтобы описать то, что видите. Попробуйте представить, где вы находитесь. Представьте, где вы находились до этого, когда «были» на предыдущей картинке.

Чтобы не упрощать вашу задачу, я не буду давать картинкам названий. Просто «пункт 1», «пункт 2» и «пункт 3» (рисунок 1.6–1.8).



Рисунок 1.6. Пункт 1



Рисунок 1.7. Пункт 2



Рисунок 1.8. Пункт 3

Задание 1.2

Вы можете сами придумать похожий пример.

Вы летали на самолёте и смотрели в иллюминатор? Вспомните, что вы видели с того момента, когда самолёт находился высоко в воздухе, до того момента, когда он приземлился.

Задание 1.3

Вы ходили по морю на корабле? Вспомните, что вы видели с того момента, когда на горизонте показался маленький кусочек суши, до того момента, когда корабль пришвартовался в порту.

1.2 Программа

Теперь, когда вы сделали небольшую зарядку для своего воображения, вы можете заняться компьютером и программами. Отойдите от компьютера и посмотрите на него издалека.

Отсюда видно, что компьютер — это просто железный ящик (или пластмассовая коробка), в котором что-то работает. Вы совершенно точно не ошибётесь, если скажете, что внутри него *программы*. Сам по себе этот железный или пластмассовый ящик совершенно бесполезен. Но именно программы, которые внутри, делают его интересным и полезным для вас.

Совет

В процессе разработки прикладного решения я буду использовать разные новые для вас термины. Например, программа. Но только в первый раз я буду рассказывать, что это такое. Если потом вы вдруг встретите термин, но не сможете вспомнить, что он значит, в конце книги на странице 578 есть приложение В «Указатель понятий». В этом приложении вы можете найти термин и узнать, на какой странице книги о нём рассказывается.

Вспомните первую картинку, с которой вы начинали (рисунок 1.9).



Рисунок 1.9. Лес вдалеке

Вся эта зелень, которую вы видите, — это программы. Их много, они разные. Сейчас вам не важно, как выглядит отдельная травинка, кустик, дерево. Вас интересует один принципиальный вопрос.

Все эти растения не висят в воздухе. Они растут в земле, в почве. Существует почва — значит, на ней может что-то вырасти. Нет почвы — не будет травы, не будет и деревьев.

В каждом компьютере тоже есть такая почва. Она называется *операционная система* (рисунок 1.10).



Рисунок 1.10. Операционная система

Примечание

Я не рассматриваю специализированные вычислительные устройства, в которых операционной системы может и не быть.

Операционная система есть в компьютере всегда. Когда вы включаете компьютер, проходит некоторое время до того, как вы можете начать что-то делать. В это время как раз запускается операционная система. Пока компьютер работает, операционная система тоже работает.

Операционные системы бывают разные. Операционная система не одна. Их не две и не три. Их много, и они разные. Точно так же, как и почва. Бывает песок, бывают камни, бывает чернозём (рисунок 1.11).



Рисунок 1.11. Каменистая, песчаная почва и чернозём

На домашних и офисных компьютерах сейчас больше всего распространена операционная система Microsoft Windows (или просто Windows). Но встречаются и другие операционные системы, например Linux или OS X.

На мобильных устройствах чаще всего используются такие операционные системы, как Android, iOS или Windows Phone.

Большая часть операционной системы вам не видна. Если вы снова посмотрите на первую картинку (рисунок 1.12), то заметите, что почву практически не видно.

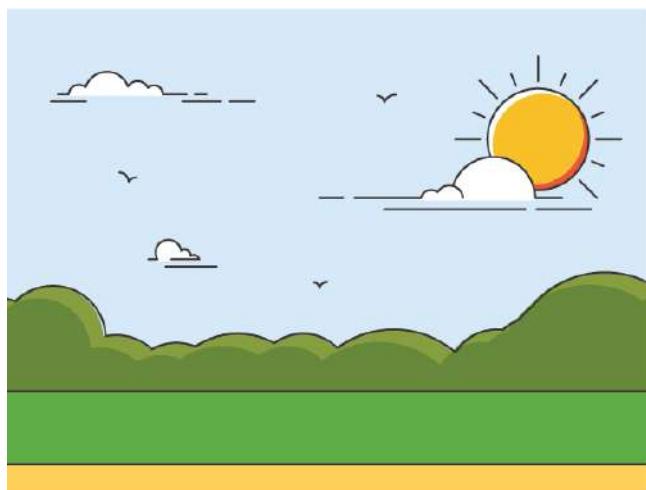


Рисунок 1.12. Лес вдалеке

Кругом одна зелень. Но вы знаете, что почва есть. Если вы посмотрите вниз, то обязательно увидите верхний слой почвы. Вы можете посадить в него какой-нибудь цветок или, наоборот, выдернуть какое-то растение.

С операционной системой то же самое. После включения компьютера или планшета вы видите какие-то значки, на которые можно нажать мышью или прикоснуться к ним пальцем. Вы можете посмотреть, какие программы есть, можете запустить какую-нибудь программу.

Все эти действия позволяет вам делать операционная система. Но это лишь небольшая, «верхняя» её часть. Основная «масса» операционной системы от вас скрыта, вам не видна и вам, что самое интересное, не нужна (рисунок 1.13).

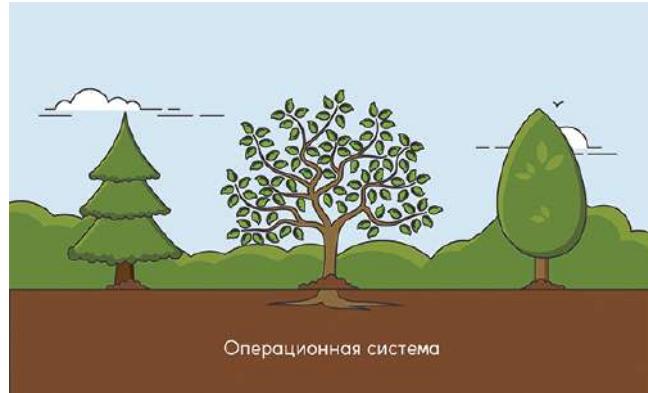


Рисунок 1.13. Операционная система

Вы постоянно пользуетесь операционной системой, но включаете компьютер совсем не ради неё. Это очень похоже на прогулки по лесу или по лугу. Да, вы постоянно «топчете» ногами верхний слой почвы. Если бы его не было, вы вообще не смогли бы никуда дойти. Но вы выходите на прогулку не ради этого. А ради того, чтобы посмотреть, какие красивые цветы растут на лугу, какие высокие деревья в лесу (рисунок 1.14).



Рисунок 1.14. Человек разглядывает деревья

Так же и с компьютером. Вы включаете его для того, чтобы запустить какую-нибудь полезную или интересную для вас программу. А совсем не ради того, чтобы посмотреть на операционную систему.

Операционная система — это тоже программа, но «другая». Из того, что я только что рассказал, самое время сделать один важный вывод. Есть программы, «ради которых вы включаете компьютер». И есть программы, «без которых ничего не будет работать». И для тех, и для других программ существуют свои общие названия.

Программы, «без которых ничего не будет работать», называют *системными программами* или *системным программным обеспечением*. Людей, которые разрабатывают такие программы, называют *системными программистами*. А само это занятие — *системным программированием*. Очень хороший пример системной программы — это операционная система (рисунок 1.15).



Рисунок 1.15. Системное и прикладное программное обеспечение

А программы, «ради которых вы включаете компьютер», называют *прикладными программами* или *прикладным программным обеспечением*. Людей, которые разрабатывают такие программы, называют *прикладными программистами*, а само это занятие — *прикладным программированием*.

Примечание

1С:Предприятие — это прикладная программа. И заниматься вы будете прикладным программированием.

Прикладная программа предназначена для определённой операционной системы. Растения не могут расти в какой угодно почве. Например, кактусы хорошо растут в каменистой почве. Но если их посадить в чернозём, то они погибнут. Потому что они не приспособлены к такой почве (рисунок 1.16).

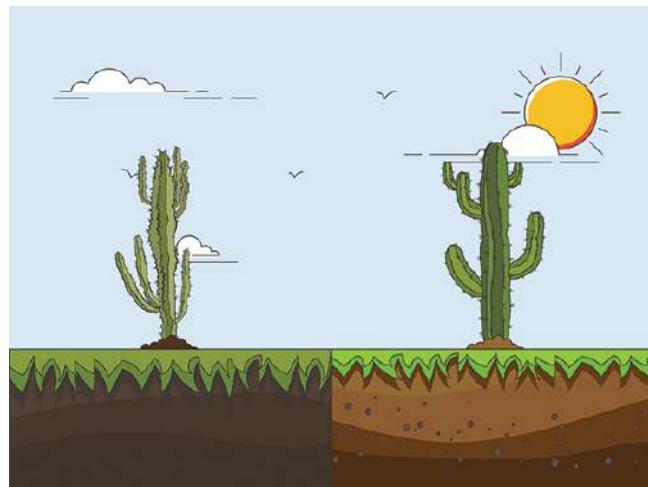


Рисунок 1.16. Кактус здоровый и кактус погибший

То же самое и с программами. Программа не может работать в какой угодно операционной системе. Она «приспособлена» только к одной определённой операционной системе. Однако бывает так, что пользуются программой хотят люди с разными операционными системами. Это очень похоже на ситуацию с фруктами. Например, с яблоками.

Яблоня — замечательное дерево с очень вкусными плодами. Яблоки любят и люди, живущие на юге, и те, кто живут на севере. Но для того, чтобы яблоня давала хорошие плоды, она должна расти в удобных и комфортных для неё условиях. Только вот почва и климат на севере совсем не такие, как на юге. Как же быть?

Специально для этого люди создают разные сорта яблонь. Есть северные сорта яблонь. Они выдерживают сильные морозы до 40–50 градусов. Им не требуется много солнца,

они любят умеренный полив. И есть южные сорта яблонь. Они любят много солнца, им требуется обильный полив, и они плохо переносят морозы (рисунок 1.17).

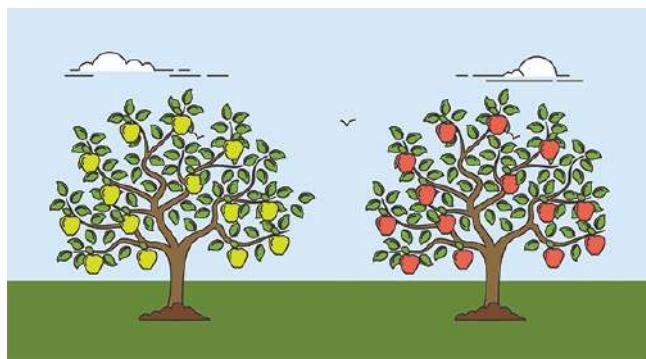


Рисунок 1.17. Северный сорт «Алые паруса» и южный сорт «Апорт»

Точно так же у одной и той же программы могут существовать разные *версии*, предназначенные для работы в определённой операционной системе.

Например, система 1С:Предприятие имеет несколько версий. Чаще всего используется версия, предназначенная для работы в операционной системе **Windows**. Просто потому, что компьютеры с этой операционной системой наиболее распространены. Вы будете использовать именно такую версию 1С:Предприятия.

Но существуют и другие версии 1С:Предприятия, которые могут работать под управлением операционной системы **Linux** или **OS X**. Кроме этого есть версии 1С:Предприятия, которые работают на мобильных устройствах под управлением операционных систем **Android**, **iOS** и **Windows Phone**.

Прикладные программы «общаются» с операционной системой. До сих пор я говорил только о программах. Но теперь вспомните и о «железе», которое есть в компьютере.

Наверное, вы знаете, что там есть какой-то процессор, который умеет складывать и вычитать. Там есть какой-то диск, на котором хранится информация. У компьютера обычно есть дисплей, на который вы смотрите. Есть клавиатура, на которую вы нажимаете. Есть принтер, который что-то печатает. Есть wi-fi, который к чему-то подключается, и так далее. Всем этим «хозяйством» умеет пользоваться операционная система. А прикладные программы могут даже и не знать о том, какие «железяки» есть в компьютере. Но как же тогда всё это работает?

А очень просто. Когда вы нажимаете что-то на клавиатуре, это «что-то» «попадает в руки» операционной системы. Операционная система тут же обращается к прикладной программе и говорит ей:

— Смотри, пользователь набрал на клавиатуре «привет».

Программа говорит:

— Отлично, спасибо, я знаю, что с этим делать.

И вставляет это слово, например, в текст письма, которое вы пишете.

Когда прикладной программе нужно что-то посчитать, она обращается к операционной системе и говорит:

— Мне нужно сложить 2 и 2.

И ждёт. А операционная система в это время уже сама обращается к процессору, выполняет нужные команды и получает от него результат — 4. После этого операционная система «дёргает за рукав» прикладную программу и говорит:

— Вот твой результат, держи.

Прикладная программа смотрит на него и говорит, например:

— Отлично, покажи теперь этот результат пользователю.
И операционная система рисует на вашем экране цифру 4.
Тут опять вспомните про деревья, растущие в почве (рисунок 1.18).



Рисунок 1.18. Дерево, растущее в почве

Дерево умеет брать из почвы воду, питательные вещества. Но ему совершенно не важно (и оно «не знает»), как это всё в почву попало. Может быть, дождь прошёл, а может, садовник полил. Не это главное. А главное то, что дерево умеет извлекать из почвы нужные ему вещества.

Операционная система управляет прикладными программами. Ещё одну важную историю про операционную систему я расскажу на примере обычной школы. Представьте, что школа — это такой большой компьютер. Ученики — это прикладные программы. Что будет, если взять пустую школу и запустить в неё учеников?

Скорее всего, ничего полезного из этого не выйдет. А может быть, даже и ничего хорошего не выйдет. Чтобы из школы и учеников получился какой-то толк, нужны учителя (рисунок 1.19).

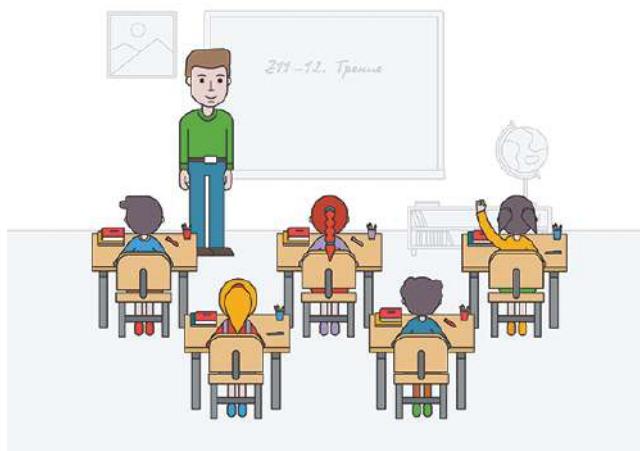


Рисунок 1.19. Учитель и ученики

Именно они скажут, когда нужно прийти в школу. Скажут, когда начинаются уроки. Покажут, в каком классе будут проходить занятия.

Именно учителя составят расписание занятий. Чтобы классы не пустовали, чтобы в одном классе не сидело полшколы. Чтобы каждый учитель занимался с тем количеством учеников, которое удобно для обучения. Чтобы на занятия физкультурой классы приходили по очереди, один за другим, а не все сразу, одновременно. В этом смысле учителя в школе являются своеобразной операционной системой.

Точно так же операционная система в компьютере управляет работой прикладных программ. Каждой программе она выделяет место, следит за тем, чтобы программы не мешали друг другу. Выстраивает их в очередь, если они хотят обратиться к одному и тому же устройству.

Операционная система исполняет прикладные программы. Наконец вы подошли к самому интересному замечанию. Из последних двух историй вы знаете, что:

- операционная система управляет прикладными программами;
- прикладные программы «общаются» с операционной системой: передают ей данные и команды, получают от неё данные и команды.

Чтобы описать всю эту конструкцию коротко, говорят, что операционная система является *средой исполнения* для прикладных программ. Или говорят, что прикладные программы *исполняются в среде* операционной системы.

Слово «среда» здесь обозначает, конечно же, не день недели, а некое пространство, в котором что-то происходит. И подразумевается, что в другом пространстве то же самое происходить не может.

Например, чтобы самолёт летал, ему нужен воздух. По земле, или под землёй, самолёт летать не может. И в космосе, где нет воздуха, самолёт тоже летать не сможет. То есть для самолёта среда — это воздух (рисунок 1.20).

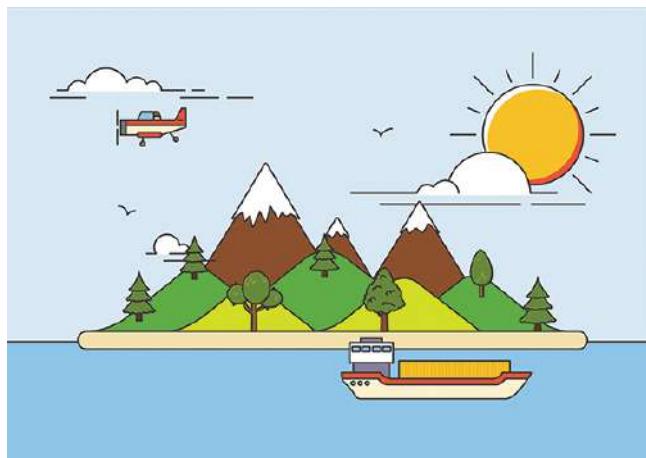


Рисунок 1.20. Самолёт в небе и корабль в море

Другой пример. Корабль. Его среда — вода. В воздухе или на земле корабль не умеет передвигаться.

Слово «исполняет» означает, что программа ничего не может сделать сама по себе. Она не может сама по себе вдруг начать выполняться. Только если «среда» запустит эту программу. Если среда выполнит её просьбы и желания, её команды. Если среда передаст ей какие-то данные. Или получит от неё какие-то данные и сохранит их до того времени, когда они понадобятся.

Это очень похоже на то, как симфонический оркестр исполняет произведение (рисунок 1.21).



Рисунок 1.21. Симфонический оркестр исполняет

Примечание

Теперь вы знаете много новых слов и можете сказать, что 1С:Предприятие — это прикладная программа, которая исполняется в среде операционной системы. Есть разные версии 1С:Предприятия, предназначенные для разных операционных систем.

1.3 Как устроено 1С:Предприятие

На протяжении всей книги я буду обращаться к примеру, который вы рассматривали в начале. Он поможет вам не потеряться внутри компьютера и внутри системы 1С: Предприятие.

Итак, сейчас вы стоите на опушке леса. Помните второй рисунок (рисунок 1.22)?

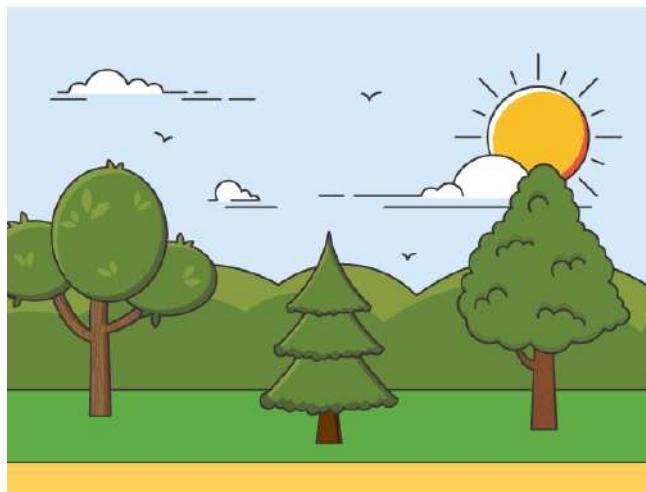


Рисунок 1.22. Лес вблизи

Здесь вам нужно разобраться с тем, что не всё перед вами одинаково зелёное. Вы совершенно точно видите, что в этом массиве зелени есть части, непохожие друг на друга.

Вы ещё не забыли, что я говорю о программах? О прикладных программах.

Так вот. Сейчас перед вами не какое-то «сборище» программ, а система 1С:Предприятие. И вам нужно разобраться, что у неё внутри. Разобраться по большому счёту, не вдаваясь в подробности.

Тут есть особенность. Деревья чаще всего растут сами по себе, но иногда их выращивает человек. А программы сами по себе не растут никогда. Их всегда «выращивает»

человек. Чтобы понять, как «выращивать» программы, вспомните, как люди выращивают растения: деревья, овощи, цветы.

Первое время, пока растения не окрепли, люди содержат их в специальной почве, в специальном помещении. Обычно такое помещение называется теплицей.

Это может быть большое помещение, как в лесопитомнике (рисунок 1.23).

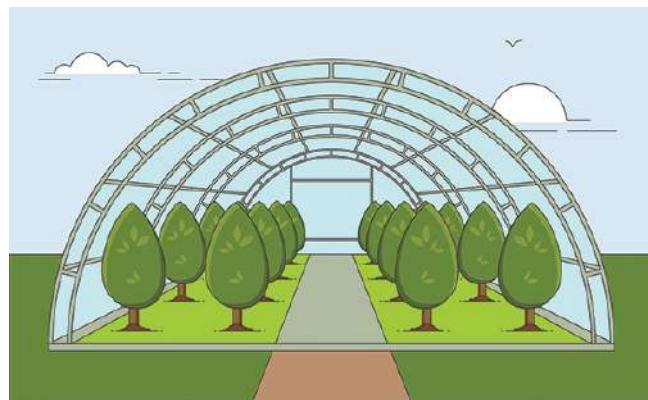


Рисунок 1.23. Теплица лесопитомника

Это может быть маленькая теплица, как у вас на даче (рисунок 1.24).

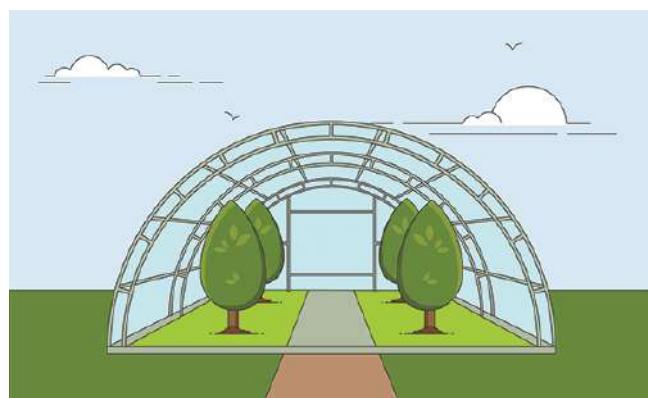


Рисунок 1.24. Теплица на даче

Размеры сейчас не имеют значения, вам важен лишь принцип. Прежде чем растения начнут приносить пользу, люди выращивают их в специальных условиях при нужной температуре, с нужным режимом полива и так далее.

У программ тоже есть такое место, где они выращиваются, создаются, разрабатываются. Это место называется *среда разработки*. Она содержит всё, что нужно для того, чтобы написать программу, проверить правильность её работы, исправить ошибки.

Среды разработки бывают разные. Бывают *универсальные среды разработки*. Это такие теплицы, в которых можно вырастить любое растение для любых условий. То есть можно создать программу любого назначения для любой операционной системы. Например, компьютерную игру для операционной системы Windows. Или редактор картинок для операционной системы Android (рисунок 1.25).



Рисунок 1.25. Универсальная среда разработки

А есть *специализированные среды разработки*. В них можно вырастить только некоторые виды растений. Например, только плодовые деревья.

Такие деревья не смогут расти где угодно. Для них нужна не обычная, а специальным образом подготовленная почва. Для этого люди делают грядки, посадочные ямы. И когда деревья готовы к самостоятельной жизни, люди высаживают их в эту специально подготовленную почву.

Так вот, 1С:Предприятие содержит внутри себя такую теплицу, такую специализированную среду разработки. С её помощью создаются разные программы 1С:Предприятия. Эта среда разработки называется *конфигуратор* (рисунок 1.26).

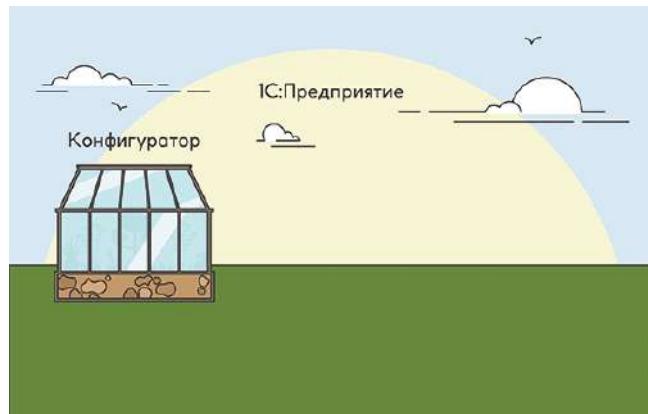


Рисунок 1.26. Конфигуратор

В конфигураторе создаются *прикладные решения 1С:Предприятия*. Или просто *прикладные решения*. То есть те программы, которыми пользуются люди (рисунок 1.27). Например, 1С:Бухгалтерия. Или 1С:Деньги. Или любая другая программа системы 1С:Предприятие.



Рисунок 1.27. Конфигуратор и прикладные решения

Прикладные решения не приспособлены для того, чтобы исполняться в среде операционной системы. Для их работы нужна специальная среда, специальная почва. И эта почва называется *технологическая платформа 1С:Предприятие*. Или просто *платформа 1С:Предприятие* (рисунок 1.28).



Рисунок 1.28. Конфигуратор, прикладные решения и платформа

Таким образом, система 1С:Предприятие — это целый сад, в котором есть теплица и в котором есть разные прикладные решения. Весь этот сад находится на одной почве, которая называется платформой.

Ну и для того, чтобы мой рисунок был полностью закончен, вспомните, что вы выяснили в конце предыдущего раздела. А выяснили вы то, что 1С:Предприятие — это прикладная программа, которая **исполняется в среде операционной системы**.

Это значит, что весь сад, называемый 1С:Предприятие, не висит в воздухе, а находится на земле, покоятся на основании, которое называется операционной системой (рисунок 1.29).



Рисунок 1.29. 1С:Предприятие и операционная система

Чем эта картинка полезна для вас? На ней хорошо видно следующее: чтобы воспользоваться прикладным решением 1С:Предприятия (например 1С:Бухгалтерией), нужно сначала запустить платформу 1С:Предприятие, а потом уже, находясь внутри платформы, запустить прикладное решение.

Таким образом, получается «бутерброд». Операционная система является средой исполнения для платформы 1С:Предприятие. А платформа 1С:Предприятие является, в свою очередь, средой исполнения для прикладных решений. Без платформы 1С:Предприятие прикладные решения работать не смогут, погибнут.

Наверняка вы уже подумали о том, что это очень сложная конструкция. Ведь как вы работаете с другими программами? Вы берёте дистрибутив программы и запускаете его. Компьютер что-то делает, устанавливает программу, и после этого вы сразу можете пользоваться программой. Другими словами, вы берёте саженец и сажаете его в землю (рисунок 1.30).



Рисунок 1.30. Установка обычной программы

А тут, получается, нужны грядки, специально подготовленная почва, теплица какая-то сбоку стоит... Где всё это взять и как всё это построить?

Оказывается, тут тоже всё очень просто. Когда вы покупаете любую программу 1С:Предприятия, вы сразу же получаете всё необходимое. И после установки этой программы у вас на компьютере автоматически получается вот такой сад (рисунок 1.31).



Рисунок 1.31. Установка прикладного решения 1С:Предприятия

Вы сразу можете использовать прикладное решение. Если оно вас чем-то не устраивает, вы можете изменить его с помощью конфигуратора. Вы даже можете создать своё, совершенно новое, прикладное решение и пользоваться им. При этом в вашем саду может расти сколько угодно прикладных решений 1С:Предприятия.

Говоря другими словами, вместе с прикладным решением вы получаете сразу все инструменты, которые нужны для его изменения. Будете вы что-то изменять или не будете — это уже ваше дело. Но главное, что если вам вдруг захочется, то всё необходимое у вас есть.

Примечание

1С:Предприятие — это не просто прикладная программа. Это не какое-то одиноко стоящее дерево.

1С:Предприятие — это система программ, которая включает в себя:

- технологическую платформу;
- созданные на её основе прикладные решения различного масштаба и различной направленности.

1.4 Зачем нужны прикладные решения 1С:Предприятия

Чтобы ответить на этот вопрос, вам понадобится «зайти» внутрь системы 1С:Предприятие. Вспомните прогулку, с которой начинается эта книга. Сейчас вы зашли в лес и видите перед собой много разных деревьев (рисунок 1.32).

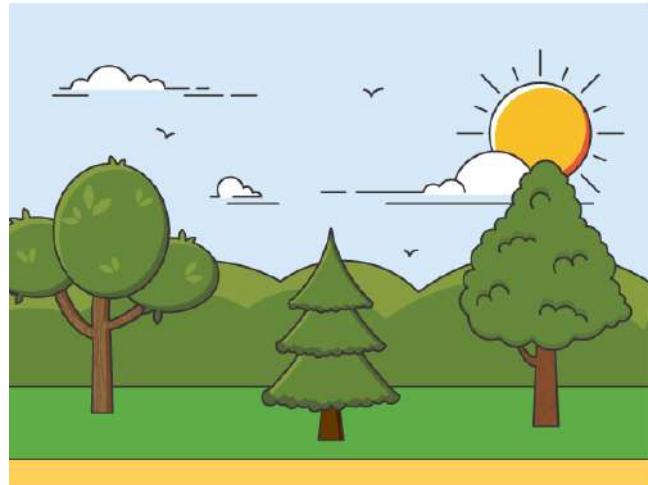


Рисунок 1.32. Деревья в лесу

Здесь вам нужно понять, что за деревья вас окружают.

А если вспомнить сад, то вам нужно понять, какие деревья растут в вашем саду (рисунок 1.33). Яблони? Груши? А может быть, кокосовые пальмы?



Рисунок 1.33. Система программ 1С:Предприятие

Одним или двумя словами трудно объяснить, для каких задач хороши программы 1С: Предприятие. Поэтому я расскажу про некоторые области деятельности, для которых они подходят.

Например, все известные вам программы вы можете совершенно точно разделить на те, которые вы запускаете ради развлечения, и те, которые помогают вам выполнять какую-нибудь работу, не связанную с развлечением.

Ради развлечения вы можете поиграть в компьютерную игру, посмотреть фильм на компьютере, почитать книгу, поговорить с друзьями, поделиться с ними фотографиями или впечатлениями. Для создания таких программ 1С:Предприятие совсем не подходит.

Зато 1С:Предприятие хорошо помогает в работе. Например, тогда, когда нужно запомнить большое количество информации. Некоторым образом систематизировать эту информацию, разложить её «по полочкам». Сделать на её основе выводы.

Например, вы печёте хлеб. Вы берёте муку, дрожжи, соль, воду, и у вас получается сколько-то буханок хлеба. Тогда 1С:Предприятие поможет вам запомнить, кто из ваших покупателей на какой день сколько хлеба заказал. Сколько хлеба вы можете ещё выпечь из тех запасов, которые у вас есть. Когда и сколько нужно закупить муки и дрожжей, чтобы ваши покупатели не остались без хлеба.

Например, вы работаете не один, а с помощниками. Кто-то работает лучше, выпекает больше. Кто-то выпекает меньше. Кто-то заболел и не приходил на работу. Как посчитать в конце месяца, кто сколько заработал? И здесь 1С:Предприятие будет очень кстати.

Другими словами, система программ 1С:Предприятие удобна и полезна там, где нужно автоматизировать деятельность большого количества людей, предприятия, фирмы. Или там, где людей, может быть, немного, даже всего один человек, но зато много информации, которую трудно запомнить или обработать вручную.

Систему программ 1С:Предприятие придумали и сделали в России. Сделала её фирма «1С», которая находится в Москве. Фирма «1С» сама практически не занимается автоматизацией предприятий. Она развивает и улучшает платформу, разрабатывает большой набор основных прикладных решений. Такие прикладные решения называют *типовыми*.

Автоматизацией предприятий занимаются её фирмы-партнёры. Их много, они находятся во многих городах России и во многих странах ближнего и дальнего зарубежья. Фирмы-партнёры непосредственно работают с заказчиками, адаптируют, дорабатывают типовые прикладные решения к их пожеланиям. Многие фирмы-партнёры создают собственные прикладные решения с помощью системы 1С:Предприятие.

Так как система 1С:Предприятие придумана в России, внутри у неё всё на русском языке. Это очень удобно для вас. В процессе обучения вам не придётся использовать иностранный язык. Если бы вы взяли любую другую систему, без знания английского языка вы бы вряд ли обошлись.

Ещё один положительный момент связан с тем, что программы 1С:Предприятия используются во многих организациях и предприятиях нашей страны. Они очень сильно распространены, поэтому многие люди либо сами работают с ними, либо слышали о них.

1.5 Установка

Совет

Если на вашем компьютере уже установлено 1С:Предприятие версии 8.3, вы можете смело перейти к разделу 1.5.3 «Как запускать 1С:Предприятие» на странице 33.

Это будет самый неинтересный и скучный раздел во всей книге. Но, к сожалению, он необходим. Без него вы не сможете выполнить ни одно упражнение.

Вам нужно будет скачать из Интернета бесплатную версию 1С:Предприятия и установить её на ваш компьютер. Для этого не потребуется никаких знаний или умений. Просто выполните по порядку все инструкции, которые написаны ниже.

Для получения ссылки на скачивание вам понадобится электронная почта. Поэтому, если у вас ещё нет своего адреса электронной почты, самое время его создать. Например, на сайте [Яндекс.Почта](#).

Может случиться так, что при покупке книги вы не обратили внимания на то, что требуется доступ в Интернет. Или ваше интернет-подключение недостаточно хорошее, чтобы скачать большой объём данных. В этом случае вам нужно будет приобрести специальный учебный программный продукт фирмы «1С», который называется «1С:Предприятие 8.3. Версия для обучения программированию» (рисунок 1.35).

Его можно приобрести у партнёров фирмы «1С», в крупных книжных магазинах или в магазинах сети «1С Интерес». Координаты в вашем городе можно узнать в фирме «1С» (например, по телефону (495) 737-92-57).

1.5.1 Скачивание дистрибутива

Первым делом зайдите в интернет-магазин фирмы «1С». Это можно сделать с помощью любого интернет-браузера: Windows Internet Explorer, Mozilla Firefox, Google Chrome, Safari и т. д.

По адресу <http://online.1c.ru/catalog/free/learning.php> находятся бесплатные версии 1С:Предприятия, предназначенные для обучения (рисунок 1.34).

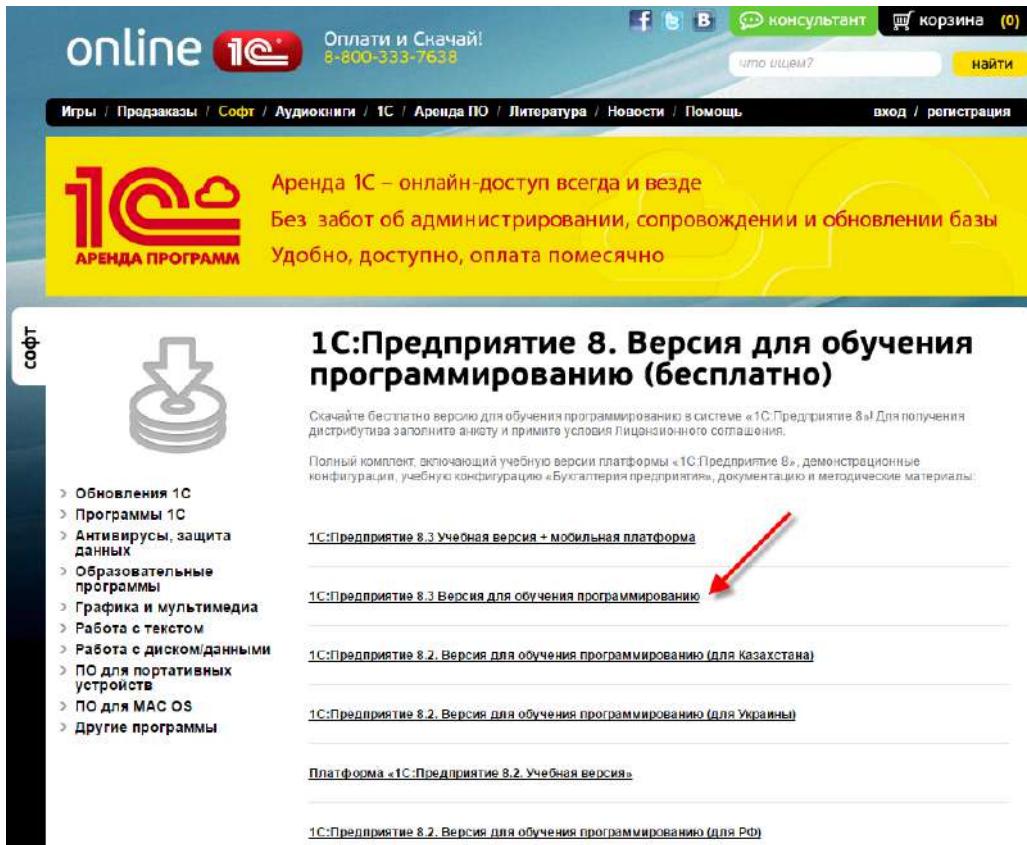


Рисунок 1.34. Интернет-магазин фирмы «1С»

Подробнее

Для того чтобы выполнять упражнения из книги, вы могли бы скачать только то, что необходимо — учебную платформу 1С:Предприятия. Этого было бы достаточно. На рисунке 1.34 это строчка «1С:Предприятие 8.3. Учебная версия + мобильная платформа».

Но вы скачаете больше. Вы возьмёте специальный продукт, который содержит не только учебную платформу, но и много полезных материалов, которые пригодятся вам в процессе обучения. Например, там есть документация, к которой вы можете обратиться, если у вас что-то не получится. Там есть другие полезные книги, если вы хотите углубить свои знания.

Вы скачаете электронную версию этого продукта, но она содержит всё то же самое, что находится внутри коробки, которую можно купить в магазине (рисунок 1.35).



Рисунок 1.35. 1С:Предприятие 8.3. Версия для обучения программированию

Итак, вам нужен продукт, который называется «1С:Предприятие 8.3. Версия для обучения программированию». Нажмите на него (рисунок 1.34).

The screenshot shows the 'online 1C' website interface. At the top, there's a navigation bar with links like 'Игры', 'Предзаказы', 'Софт', 'Аудиокниги', '1С', 'Лизинг ПО', 'Литература', 'Новости', 'Помощь', ' вход / регистрация', and a search bar. Below the header, there's a yellow banner with the '1C' logo and text: 'Аренда 1С – онлайн-доступ всегда и везде', 'Без забот об администрировании, сопровождении и обновлении базы', and 'Удобно, доступно, оплата помесячно'. On the left, there's a sidebar with a 'Софт' category and a list of software categories: 'Обновления 1С', 'Программы 1С', 'Антивирусы, защита данных', 'Образовательные программы', 'Графика и мультимедиа', 'Работа с текстом', 'Работа с диском/данными', 'ПО для персональных устройств', 'ПО для MAC OS', and 'Другие программы'. A red arrow points from the sidebar to a button labeled 'Получить продукт бесплатно' (Get product for free) on the main content page. The main content page has a title '1С:Предприятие 8.3 Версия для обучения программированию' and a sub-section '1С:Предприятие 8. Версия для обучения программированию'. It includes a brief description of the software and a note about learning through the platform.

Рисунок 1.36. Ссылка на бесплатное получение продукта

На следующем экране (рисунок 1.36) просто нажмите ссылку *Получить продукт бесплатно*.

online 1C

Оплати и Скачай!
8-800-333-7638

Игры / Предзаказы / Софт / Аудиокниги / 1С / Аренда ПО / Литература / Новости / Помощь

вход / регистрация

что ищем? найти

1C
АРЕНДА ПРОГРАММ

Аренда 1С – онлайн-доступ всегда и везде
Без забот об администрировании, сопровождении и обновлении базы
Удобно, доступно, оплата помесячно

софт

1C:Предприятие 8.3 Версия для обучения программированию

Скачайте бесплатно версию для обучения программированию в системе 1С:Предприятие 8.3! Для получения дистрибутива заполните анкету и примите условия Лицензионного соглашения.

После заполнения анкеты, на указанный Вами E-mail придет письмо с ссылкой на дистрибутив, которая действительна в течении 1 суток с момента получения.

ФИО*

E-mail*

Возраст:

Род занятий:

Место работы/учебы:

Мобильный телефон:

Сообщение:

Я принимаю Лицензионное соглашение

Отправить

Рисунок 1.37. Анкета для получения ссылки

В анкете, которая появится на следующем экране (рисунок 1.37), заполните поля ФИО и E-mail. Все поля заполнять не нужно, этих двух полей достаточно.

После этого установите флажок *Я принимаю Лицензионное соглашение* и нажмите кнопку *Отправить*.

Через некоторое время в ваш почтовый ящик придёт письмо. В этом письме будет ссылка для скачивания продукта.

Нажмите эту ссылку, и браузер начнёт скачивать файл *EducFull83.zip*.

Этот файл лучше всего сохранить в новую папку с понятным названием. Назовите её *Версия для обучения программированию*. Или можете назвать её любым другим образом. Главное, чтобы для вас это название было понятно.

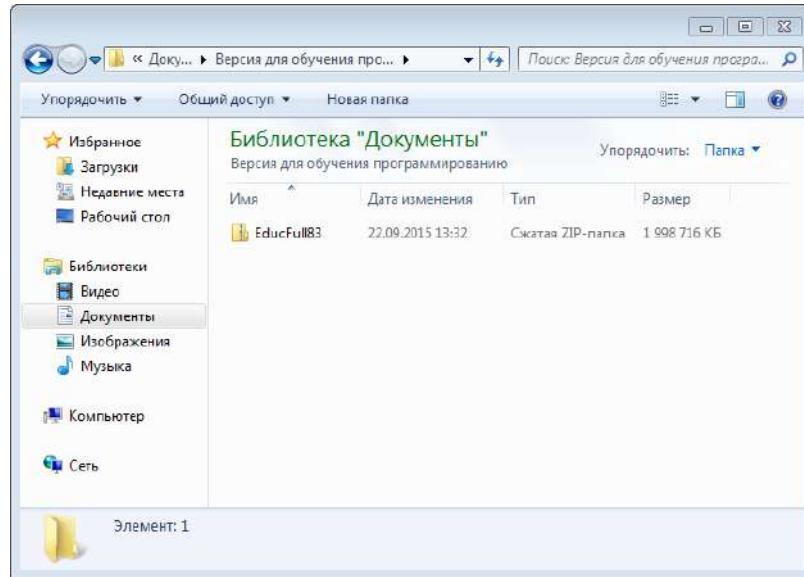


Рисунок 1.38. Архив версии для обучения программированию

Файл *EducFull83.zip* — это архив, в котором содержатся нужные вам файлы (рисунок 1.38). Современные версии операционной системы Windows умеют самостоятельно извлекать файлы из таких архивов. Возможно, ваша операционная система этого не умеет. Тогда вы можете извлечь файлы из этого архива одной из бесплатных программ для работы с zip-архивами, которые есть в Интернете.

Будем считать, что ваша операционная система всё умеет сама. Тогда, чтобы извлечь файлы из архива, дважды щёлкните на нём мышью. Операционная система покажет содержимое архива (рисунок 1.39).

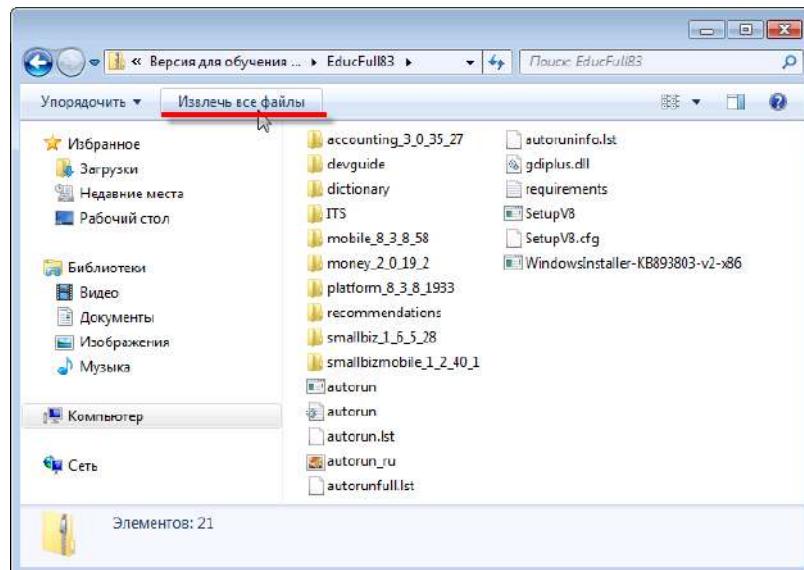


Рисунок 1.39. Команда извлечения файлов из архива

Нажмите кнопку *Извлечь все файлы*.

Операционная система спросит вас, в какую папку поместить файлы (рисунок 1.40).

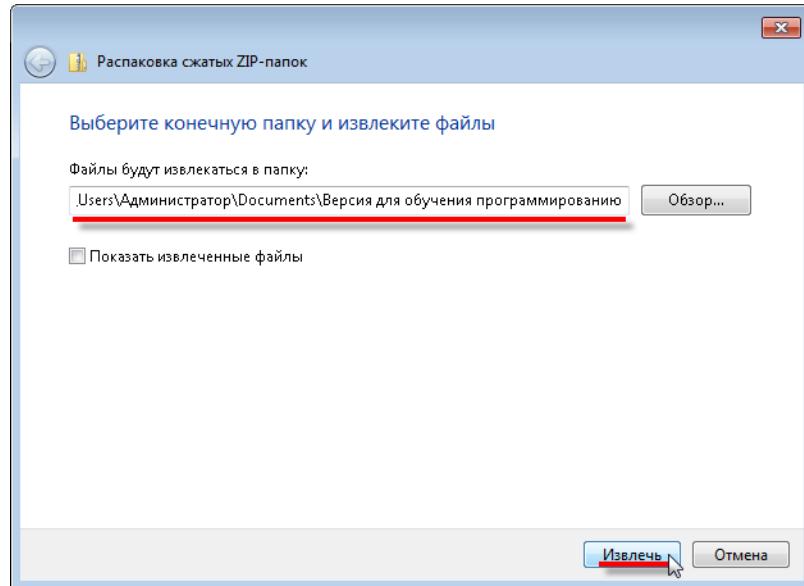


Рисунок 1.40. Выбор папки, в которую будут извлечены файлы

Удобнее всего указать ту же самую папку, в которой лежит архив. То есть папку *Версия для обучения программированию*.

После этого нажмите кнопку *Извлечь*.

Когда извлечение файлов закончится, перейдите в папку *Версия для обучения программированию* (рисунок 1.41).

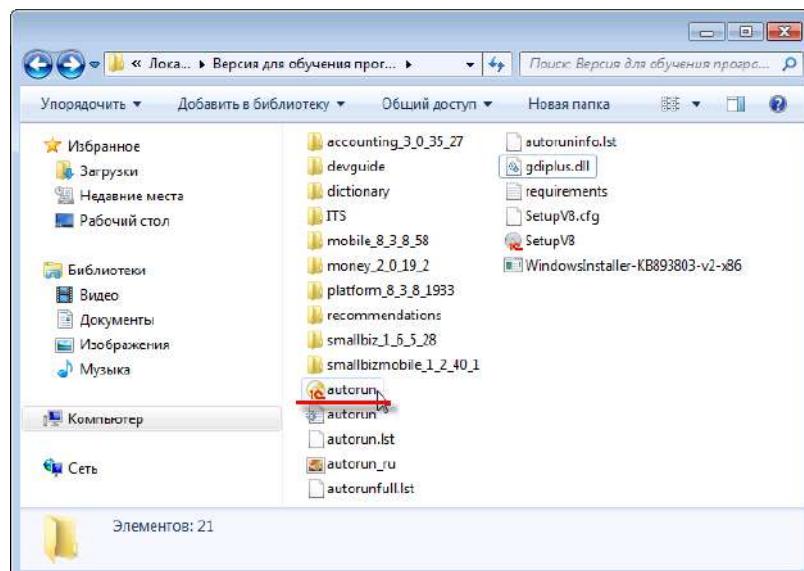


Рисунок 1.41. Файл autorun.exe

Запустите файл *autorun.exe*.

Откроется программа установки продукта «1С:Предприятие 8. Версия для обучения программированию» (рисунок 1.42).

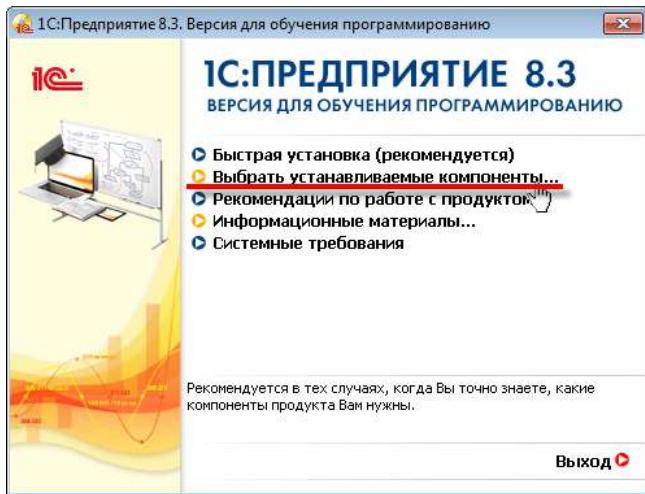


Рисунок 1.42. Меню установки продукта «1С:Предприятие 8. Версия для обучения программированию»

Она позволяет установить все программы, содержащиеся в продукте.

Но сейчас вам не нужны все программы. Возможно, вы заинтересуетесь ими позже, когда прочитаете эту книгу. Сейчас вам нужна только платформа 1С:Предприятие 8.

Поэтому откройте пункт *Выбрать устанавливаемые компоненты...* (рисунок 1.42).

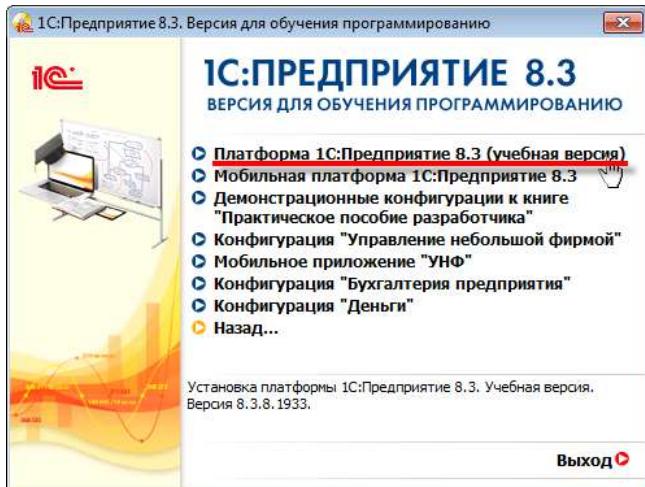


Рисунок 1.43. Команда установки платформы 1С:Предприятие 8

И выберите первую строчку — *Платформа 1С:Предприятие 8.3 (учебная версия)* (рисунок 1.43).

1.5.2 Установка платформы 1С:Предприятие 8

Подробнее

Подробнее вы можете прочитать про установку 1С:Предприятия в документации «[Руководство администратора 8.3. Глава 2. Установка системы 1С:Предприятие](#)».

Запустится программа установки платформы. Она проведёт вас через несколько шагов. На этих шагах вам нигде ничего менять не нужно. Нужно только нажимать кнопку *Далее* (рисунок 1.44).

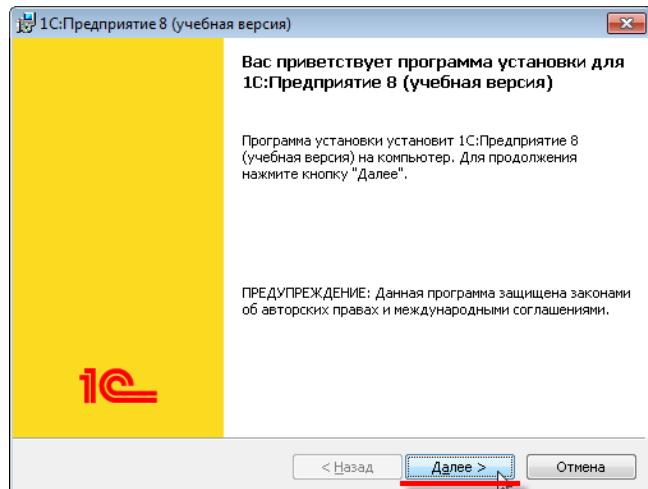


Рисунок 1.44. Установка платформы, 1-й шаг

Для наглядности я покажу все шаги на картинках (рисунок 1.45–1.47).

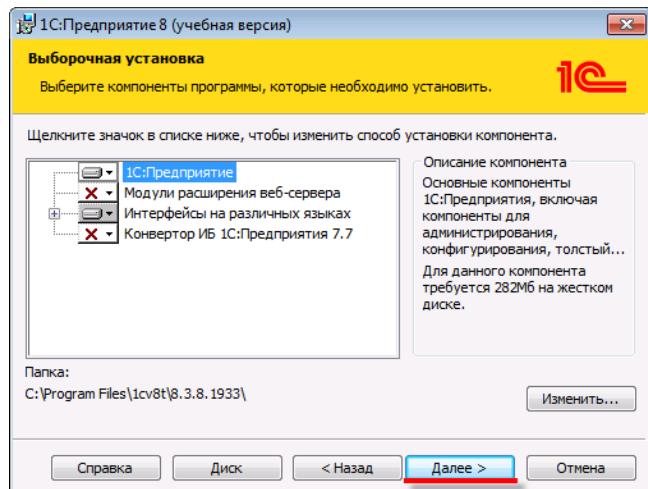


Рисунок 1.45. Установка платформы, 2-й шаг

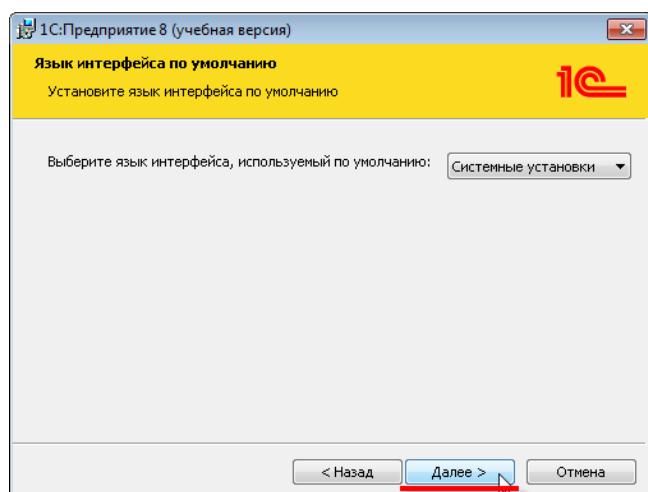


Рисунок 1.46. Установка платформы, 3-й шаг

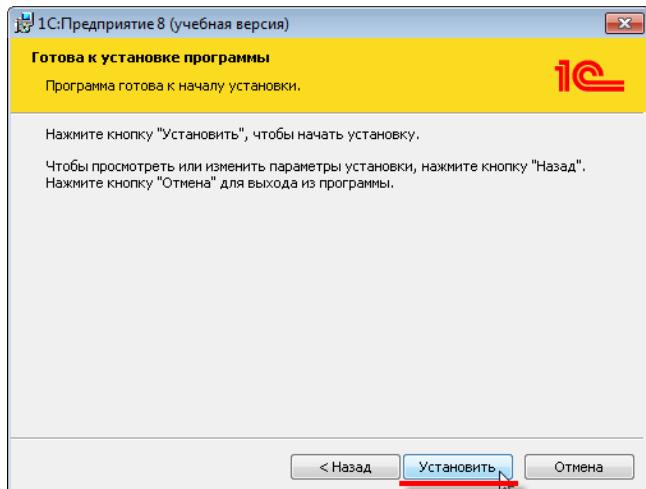


Рисунок 1.47. Установка платформы, 4-й шаг

После того как вы нажмёте кнопку *Установить*, программа скопирует новые файлы. Затем она сообщит, что установка завершена (рисунок 1.48).

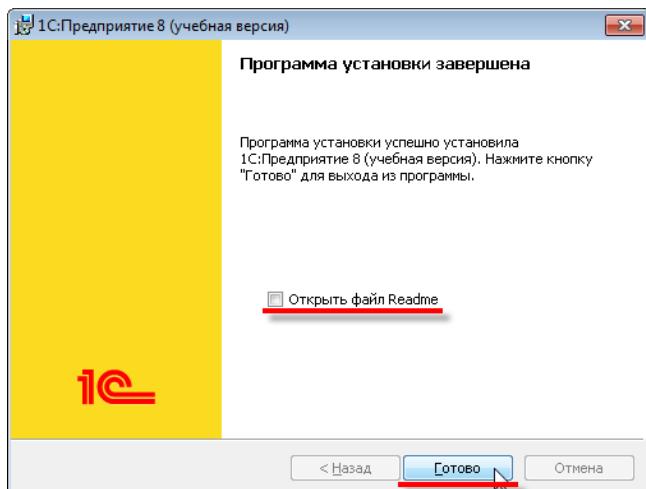


Рисунок 1.48. Установка платформы, 5-й шаг

Здесь лучше всего снять флажок *Открыть файл Readme* и нажать кнопку *Готово*.

Программу установки продукта «1С:Предприятие 8. Версия для обучения программированию» тоже можете закрыть. Она вам больше не понадобится.

Файл *Readme* содержит полезную информацию о платформе 1С:Предприятие. Но прямо сейчас вам эта информация не нужна. Позже, когда вы дойдёте до середины или до конца книги, вы можете посмотреть, что содержится в этом файле. Где его найти, я скажу чуть позже.

1.5.3 Как запускать 1С:Предприятие

Итак, установка платформы закончилась, и у вас появилось несколько команд для запуска платформы 1С:Предприятие 8. Не все эти команды подойдут для того, чтобы выполнять примеры из книги. Поэтому сейчас будьте внимательны.

Во-первых, появился ярлык *1С Предприятие* на рабочем столе. Им можно пользоваться (рисунок 1.49).



Рисунок 1.49. Команды для запуска платформы 1С:Предприятие 8

Во-вторых, появилась команда *1С Предприятие (тонкий клиент)*. Она находится в меню *Пуск*, в кратком списке программ (рисунок 1.49).

Этой командой пользоваться не нужно. Она не позволит выполнять упражнения, содержащиеся в книге.

Также в списке *Все программы* появилась папка *1С Предприятие (учебная версия)* (рисунок 1.50).

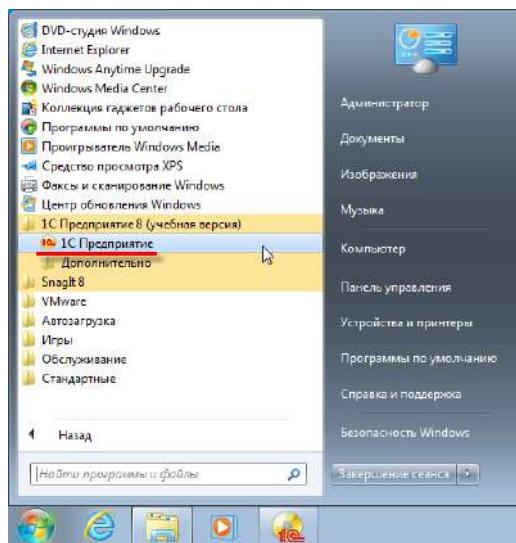


Рисунок 1.50. Команда «1С Предприятие»

А внутри этой папки появилась команда *1С Предприятие*. Это тоже «хорошая» команда. Ей вы можете пользоваться.

Команды, которые находятся в папке *Дополнительно*, сейчас вам не пригодятся (рисунок 1.51).

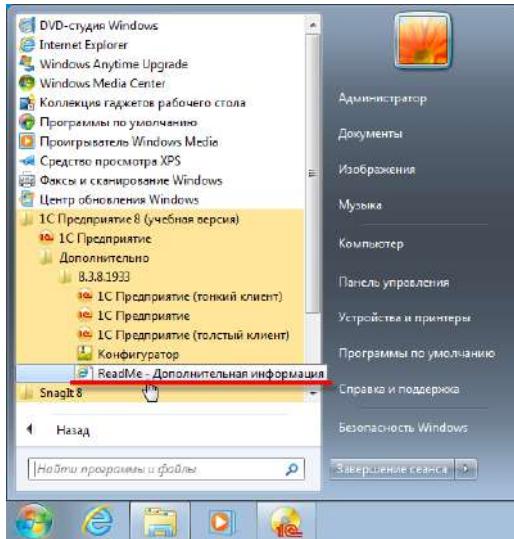


Рисунок 1.51. Файл «Readme»

Но потом вы можете заглянуть в эту папку, чтобы почитать файл *Readme*. Это тот файл, о котором я рассказывал раньше.

Примечание

Когда я буду говорить: «Запустите 1С:Предприятие», — это будет значить, что вам нужно выполнить одну из этих двух команд, на ваш выбор (рисунок 1.52).

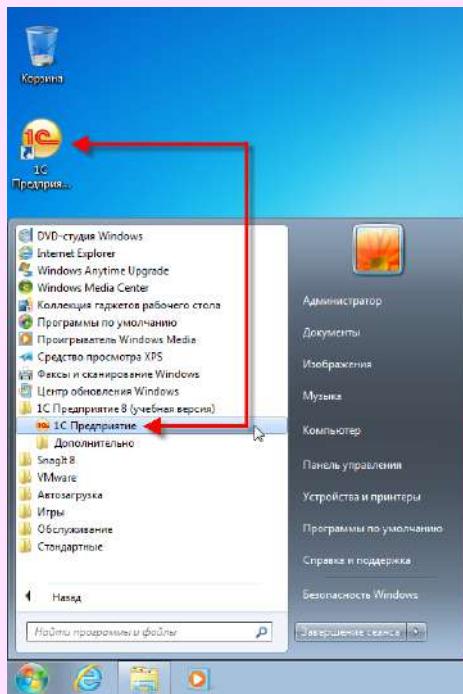


Рисунок 1.52. Запуск 1С:Предприятия

Совет

В процессе разработки прикладного решения вы будете выполнять разные действия несколько раз. Например, запускать 1С:Предприятие. Но только в первый раз я буду рассказывать, как это действие выполнить. Дальше, чтобы не повторяться, я буду говорить кратко: «Сделайте то-то». Если вдруг вы забыли, как это делается, в конце книги на странице 582 есть приложение Г «Указатель действий». В нём вы можете найти действие и узнать, на какой странице книги о нём рассказывается.

Подробнее

Подробнее вы можете прочитать про запуск 1С:Предприятия в документации «Руководство администратора 8.3. Глава 4. Запуск компонентов системы».

Глава 2

Визуальное конструирование

Не читайте эту главу!

Если вы знаете:

- что такое информационная база и как её добавить;
- как запустить платформу в режиме конфигурирования или в режиме отладки;
- что такое дерево объектов конфигурации;
- чем объект конфигурации отличается от объекта данных;
- для каких данных нужен справочник, а для каких — документ;
- как устроен прикладной интерфейс 1С:Предприятия;
- что такое формы и как их редактировать;

смело переходите к главе 3 «Встроенный язык» на странице 182.

Единственное, что вам понадобится в дальнейшем, это сделать пару настроек в конфигураторе. Они описаны в разделе 2.9.2 «Режим отладки» на странице 74.

2.1 С чего начинается прикладное решение

Теперь, не откладывая в долгий ящик, сразу начните создавать свою программу. Вы скажете: «Но ведь я ничего ещё не умею. Как я буду программировать?»

Очень просто. Многие вещи за вас сделает платформа 1С:Предприятие. В ней есть уже готовые блоки, из которых вы соберёте свою программу. И она будет работать.

А вот потом, дальше, вы займётесь тем, что узнаете, как эти «типовые» блоки можно изменять под свои желания и под свои нужды.

Итак, запустите 1С:Предприятие. На экране вы увидите такую картинку (рисунок 2.1).

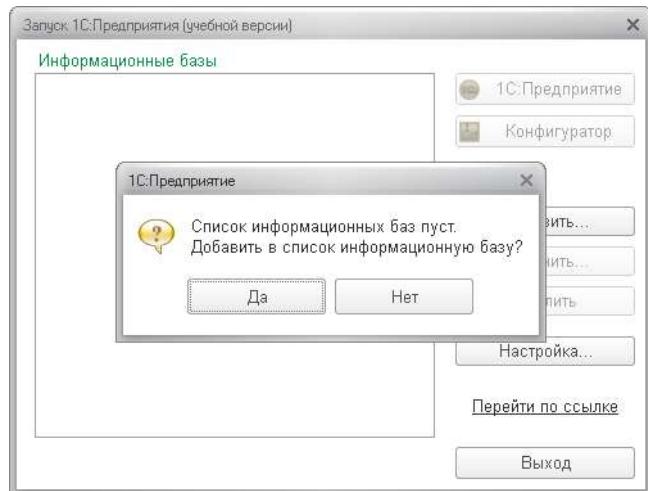


Рисунок 2.1. Запуск 1С:Предприятия

Примечание

Что вы сделали сейчас? Сейчас вы запустили платформу 1С:Предприятие.

Неожиданно вас спрашивают про какую-то «информационную базу». Дело в том, что это важное понятие в системе 1С:Предприятие. Но прямо сейчас оно вам совершенно не нужно.

С другой стороны, если это совсем никак не объяснить, вы будете чувствовать себя неуверенно. Поэтому объяснение будет коротким и не совсем точным, но потом, когда придёт время, я расскажу об этом более подробно.

Итак, вспомните ваш сад (рисунок 2.2).



Рисунок 2.2. Прикладное решение 1С:Предприятие

Сейчас вы запустили платформу 1С:Предприятие и хотите создать прикладное решение. То есть посадить в саду одно дерево.

Так вот, *информационная база* — это такая посадочная яма, в которую вы будете сажать своё дерево.

И то, что 1С:Предприятие говорит вам, что список информационных баз пуст, значит, что в вашем саду нет ни одной ямы, в которую можно сажать. И платформа предлагает вам такую посадочную яму создать.

Поэтому *добавьте новую информационную базу*. Нажмите кнопку Да (рисунок 2.1).

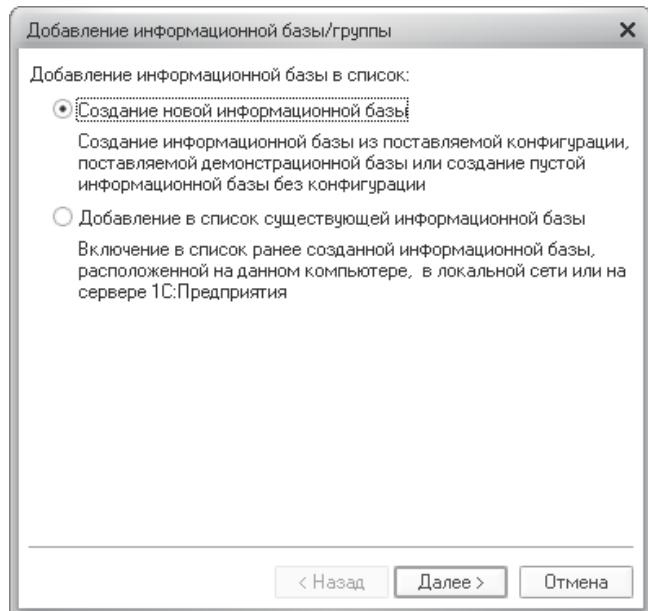


Рисунок 2.3. Добавление информационной базы, шаг 1

Тут вам нужно будет пройти через несколько шагов. Смысл этих действий для вас пока не важен, поэтому просто, без объяснений, выполните следующие инструкции.

Согласитесь с тем, что будет создана новая информационная база и, ничего не меняя, нажмите кнопку *Далее* (рисунок 2.3).

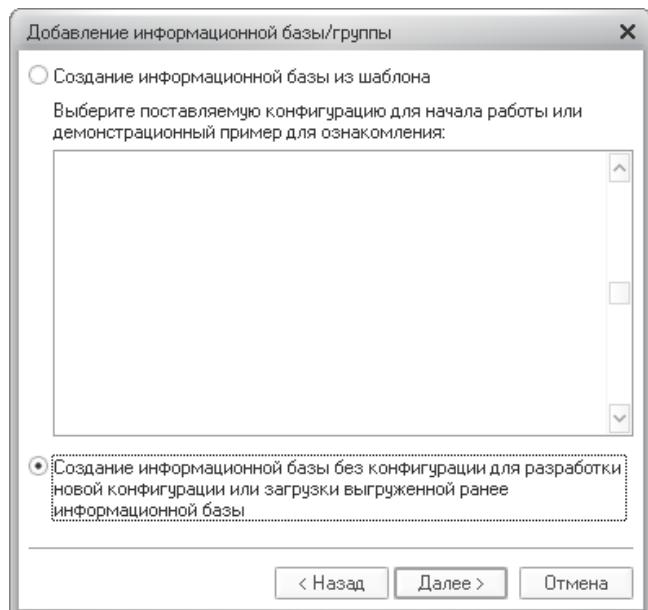


Рисунок 2.4. Добавление информационной базы, шаг 2

На следующем шаге тоже ничего не меняйте и нажмите *Далее* (рисунок 2.4).

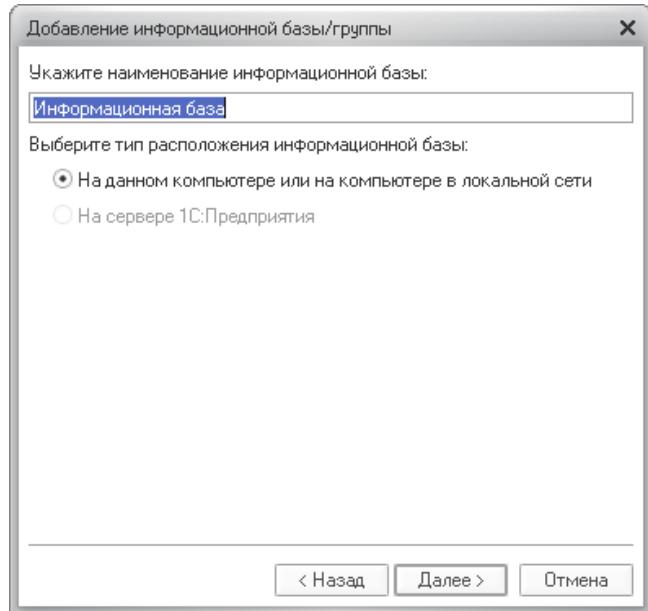


Рисунок 2.5. Добавление информационной базы, шаг 3

На этом шаге (рисунок 2.5) напишите, как будет называться ваша информационная база. Информационных баз у вас может быть много, и вам нужно отличать одну от другой. Вместо *Информационная база* напишите *Дневник*. Вы, конечно, можете придумать и другое название, но в книге я буду использовать это.

Больше ничего не меняйте и нажмите *Далее* (рисунок 2.5).

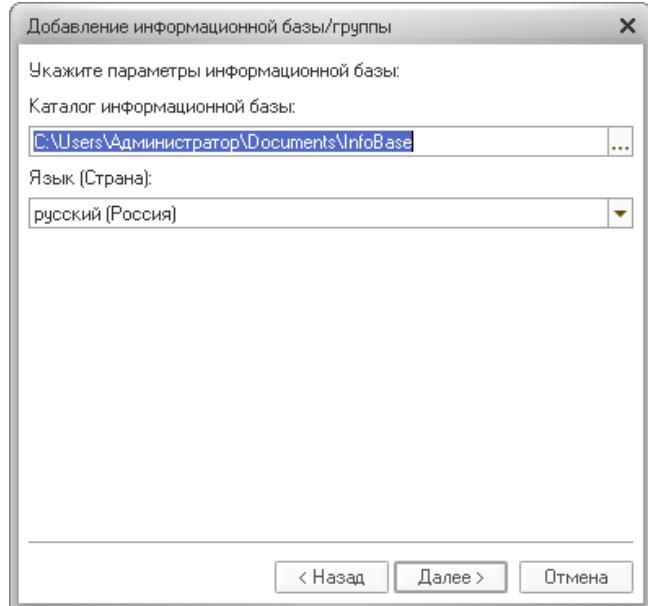


Рисунок 2.6. Добавление информационной базы, шаг 4

На этом шаге (рисунок 2.6) тоже ничего не меняйте и нажмите *Далее*.

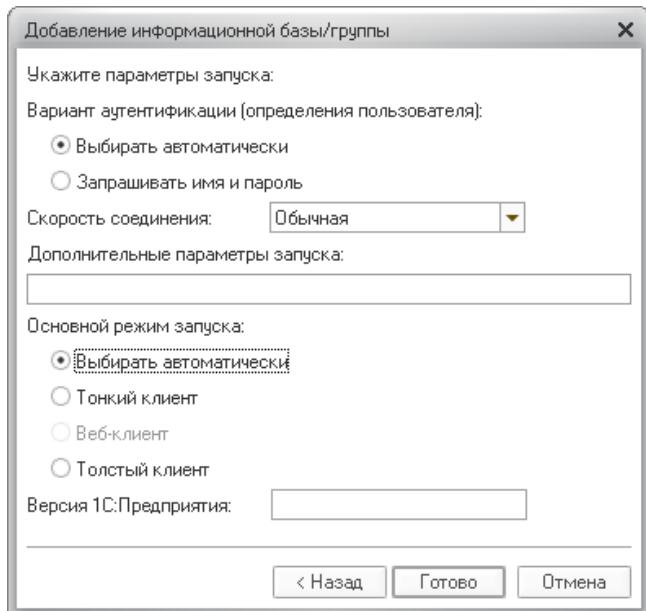


Рисунок 2.7. Добавление информационной базы, шаг 5

И, наконец, на последнем шаге (рисунок 2.7) тоже ничего не меняйте и нажмите *Готово*. 1C:Предприятие немного подумает и создаст информационную базу.

После этого вы увидите такую картинку (рисунок 2.8).

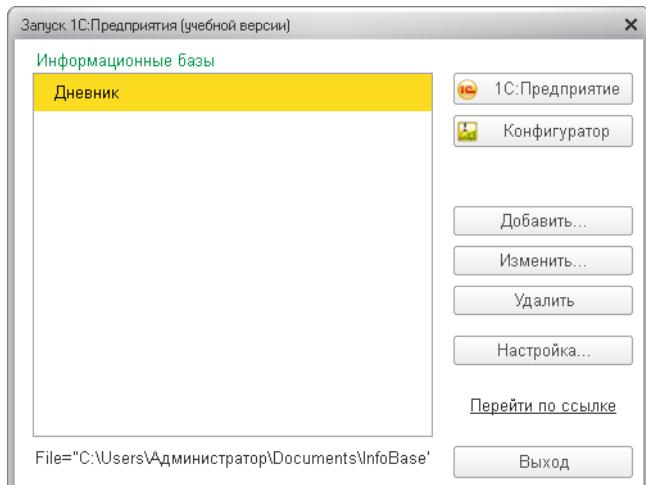


Рисунок 2.8. Список информационных баз

Это интересная и важная картинка, и сейчас я объясню почему.

Подробнее

Подробнее вы можете прочитать про список информационных баз в документации «[Руководство администратора 8.3. Глава 5. Ведение списка информационных баз](#)».

2.2 Список информационных баз

Когда вы первый раз запустили 1C:Предприятие (рисунок 2.1), вы увидели не совсем обычную картинку. Потому что компьютер был «голый». Теперь же (рисунок 2.8) вы видите ту картинку, которая обычно и бывает после запуска 1C:Предприятия.

Вы видите список информационных баз. В центральном окне у вас одна информационная база — *Дневник*. Но обычно их может быть и две, и три, и вообще много.

Вам важно понять, что означает эта картинка. Вспомните ещё раз про сад и про то, что сейчас вы запустили платформу 1С:Предприятие.

Так вот, эта картинка показывает весь ваш сад в таком виде (рисунок 2.9).

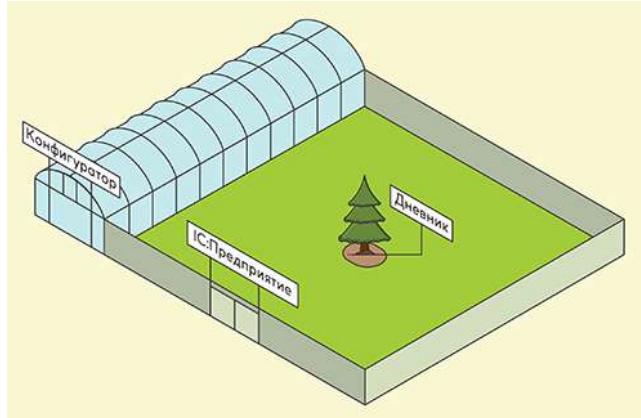


Рисунок 2.9. Ваш сад

Вы видите, какие деревья в вашем саду, и видите две двери, для того чтобы войти в сад.

Одна дверь называется *Конфигуратор*. Это уже знакомое вам слово. Через эту дверь вы попадёте в теплицу, в которой выращиваются прикладные решения. В эту дверь обычно заходят садовники, ну, или, в вашем случае, разработчики прикладных решений, программисты.

Есть и ещё одна дверь — *1С:Предприятие*. Через эту дверь вы попадёте сразу в сад. То есть туда, где прикладные решения цветут, плодоносят и приносят людям всяческую пользу. В эту дверь заходят те, кто хочет воспользоваться «плодами» прикладного решения. То есть пользователи: бухгалтеры, менеджеры, сотрудники организации, в которой работает это прикладное решение.

Конечно, 1С:Предприятие немного отличается от этой картинки. В 1С:Предприятии сначала нужно решить для себя, выбрать, с какой информационной базой вы будете работать, а потом уже нажимать Конфигуратор или *1С:Предприятие* (рисунок 2.10). Но это мелкое отличие, на которое можно и не обращать внимания.

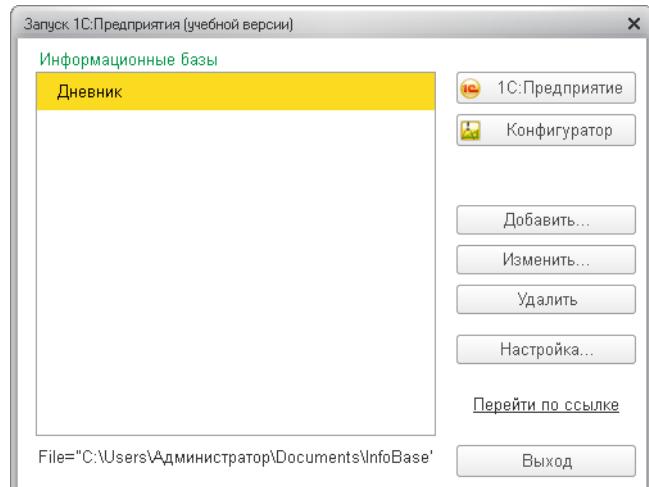


Рисунок 2.10. Список информационных баз

Итак, раз в саду есть две двери, говорят, что у системы 1С:Предприятие есть два режима работы.

Режим работы «Конфигуратор». Он для разработчиков. Для тех, кто создаёт прикладные решения.

Режим работы «1С:Предприятие». Он для пользователей. Для тех, кто выполняет свою работу с помощью этого прикладного решения.

Когда выбирают информационную базу и нажимают кнопку Конфигуратор, говорят *запустить в режиме конфигуратора, открыть в конфигураторе или зайти в конфигуратор*.

Когда выбирают информационную базу и нажимают кнопку 1С:Предприятие, говорят *запустить в режиме 1С:Предприятие или запустить в пользовательском режиме*.

Теперь, когда вы всё знаете, можно заходить внутрь. Информационная база у вас всего одна, выбирать ничего не надо.

А что вы собираетесь делать? Вы собираетесь создавать новое прикладное решение. Поэтому вы хотите нажать Конфигуратор.

И это правильно. Но не спешите. Сначала посмотрите на одну интересную деталь.

2.3 Конфигурация

Когда вы в первый раз запустили 1С:Предприятие, платформа предложила вам создать новую информационную базу. Я сказал, что информационная база — это посадочная яма, в которую вы посадите своё дерево, своё прикладное решение. И вы создали её, не вчитываясь в то, что платформа писала вам на каждом шаге.

Теперь же вам будет очень полезно вспомнить одну из картинок. Покажу её ещё раз (рисунок 2.11).

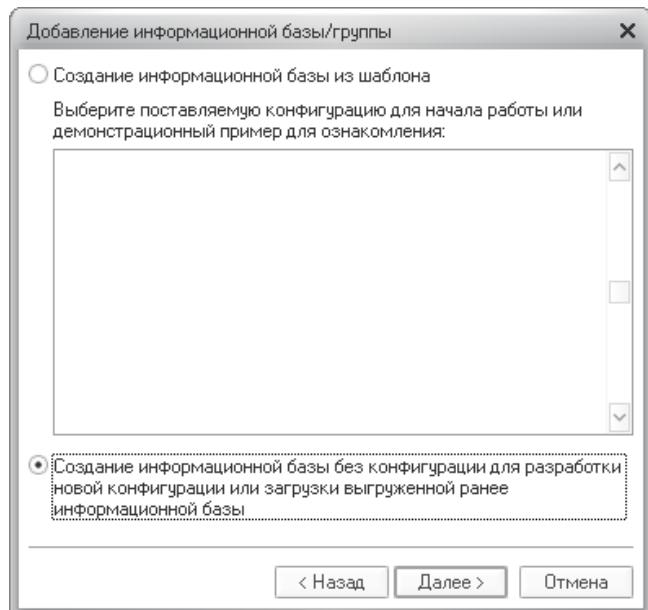


Рисунок 2.11. Добавление информационной базы, шаг 2

Здесь выбран пункт «...без конфигурации для разработки новой конфигурации...» Что такое конфигурация?

Если дерево — это прикладное решение, то *конфигурация* — это ствол (рисунок 2.12).

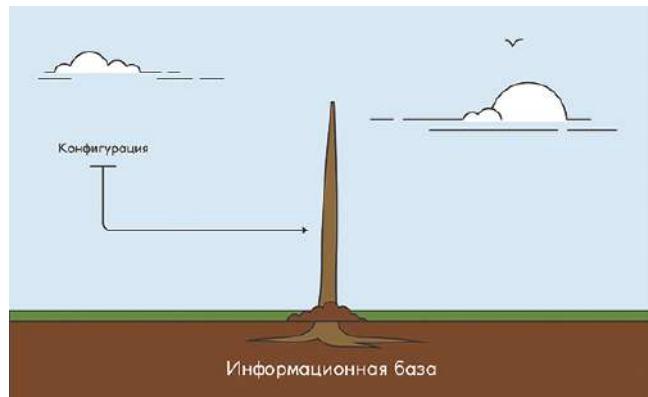


Рисунок 2.12. Конфигурация и информационная база

И когда вы прочитали, что «...информационная база без конфигурации...», вы наверняка подумали, что платформа создала вам пустую ямку, в которой ничего нет. Но это не так.

На самом деле в посадочной яме уже есть ствол. Правда, он без веток, именно такой, как на рисунке 2.12.

Когда вы будете дальше разрабатывать прикладное решение, к этому стволу вы прикрепите ветки (рисунок 2.13).



Рисунок 2.13. Конфигурация с объектами конфигурации

А когда вы запустите прикладное решение в режиме 1С:Предприятие и начнёте работать с ним как пользователи, вы добавите на это дерево свои плоды — *данные* (рисунок 2.14).



Рисунок 2.14. Конфигурация с данными

Но это будет потом. А сейчас я хочу показать вам, что новая информационная база, которую вы создали, не пустая. В ней есть ствол, есть основа прикладного решения. Основа, которую вы можете запустить и которая будет работать.

Нажмите на 1С:Предприятие (рисунок 2.10). На экране появится такая картинка (рисунок 2.15).

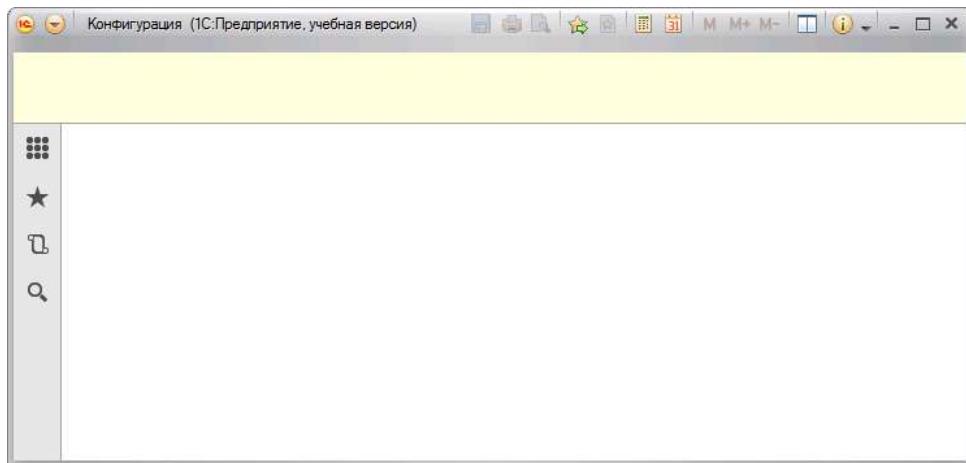


Рисунок 2.15. Пустое прикладное решение

Примечание

Что вы сделали сейчас? Сейчас вы запустили своё прикладное решение. Прикладное решение 1С:Предприятия.

На этой картинке вы видите те элементы, которые есть у любого прикладного решения 1С:Предприятия. Например, серая полоса слева — это *панель инструментов*. Круглая кнопка слева вверху открывает *главное меню*.

А вот то, что зависит от конкретного прикладного решения, появится потом в центре, на месте белого прямоугольника. И в верхней части, на жёлтой полосе.

Примечание

Итак, теперь вы знаете, что прикладные решения 1С:Предприятия содержатся каждое в своей информационной базе. И что даже новая, только что созданная, информационная база уже содержит основу, заготовку будущего прикладного решения.

2.4 Дерево объектов конфигурации

Как выглядит ваше будущее прикладное решение для пользователя в режиме 1С:Предприятие, вы уже видели. Теперь посмотрите, как оно выглядит для разработчика.

Закройте прикладное решение. Для этого просто нажмите на крестик в правом верхнем углу окна (рисунок 2.16).

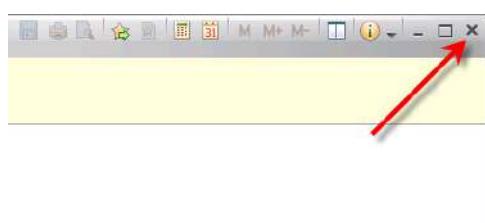


Рисунок 2.16. Закрыть прикладное решение

После этого снова запустите 1С:Предприятие и нажмите Конфигуратор. Перед вами откроется среда разработки — конфигуратор (рисунок 2.17).

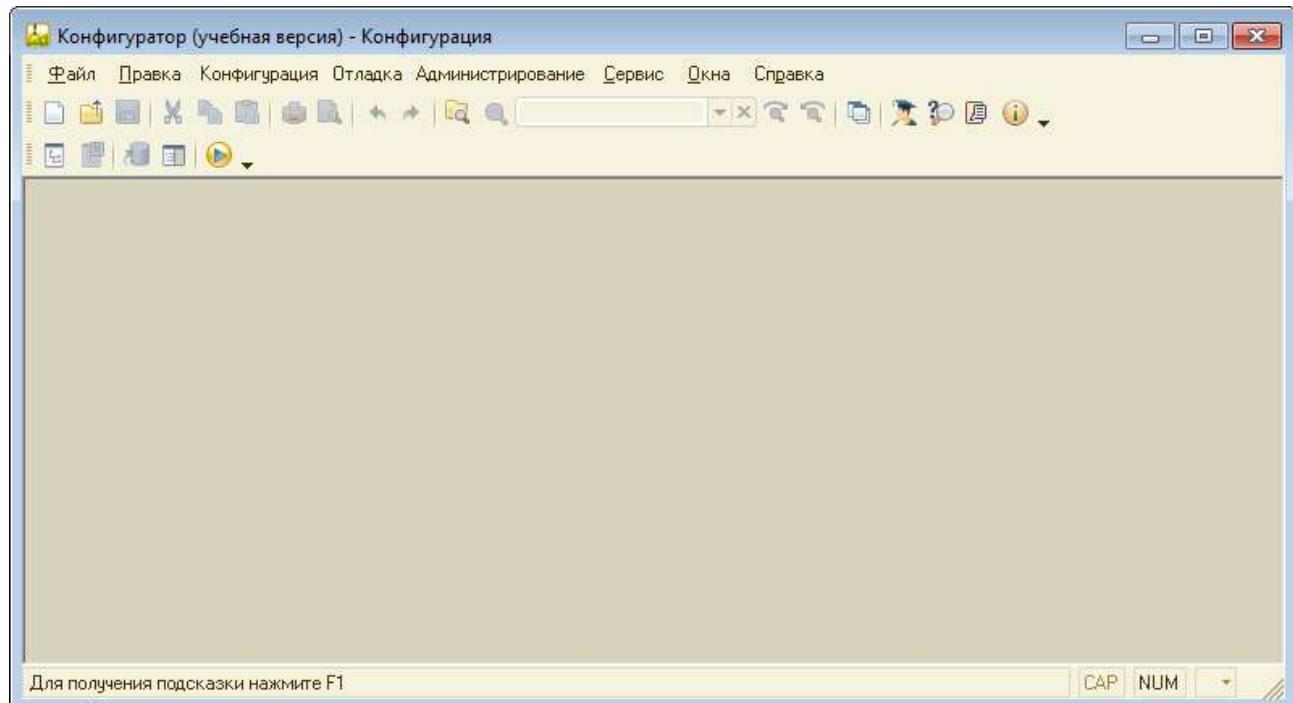


Рисунок 2.17. Конфигуратор

Самый главный элемент здесь — это дерево объектов конфигурации. Но сейчас его не видно. Чтобы его увидеть, нажмите на значок *Открыть конфигурацию* (рисунок 2.18).

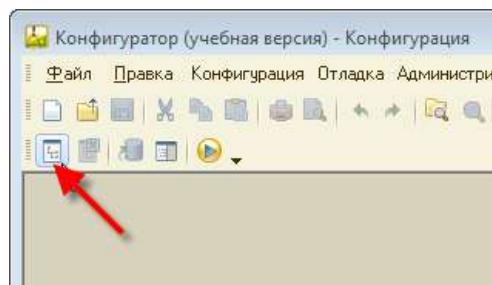


Рисунок 2.18. Открыть конфигурацию

Перед вами появится *дерево объектов конфигурации* (рисунок 2.19).

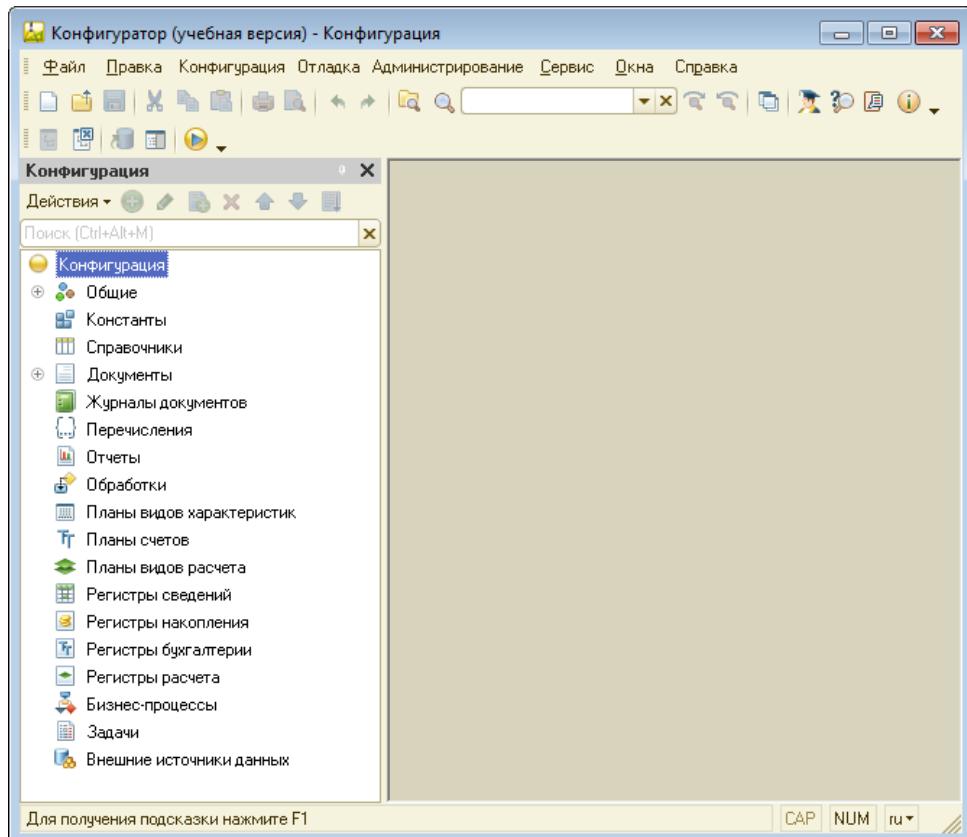


Рисунок 2.19. Дерево объектов конфигурации

Теперь вам нужно разобраться, что это такое и почему оно так называется. Для этого вам снова понадобится воображение.

Прежде всего, дерево объектов конфигурации — это способ, которым в конфигураторе, в среде разработки, представляется прикладное решение.

Вы уже видели, что ваша информационная база содержит конфигурацию. Ствол. Основу. Без веток. Поэтому и дерево объектов конфигурации пока не содержит ни одной ветки. Только ствол (рисунок 2.20).

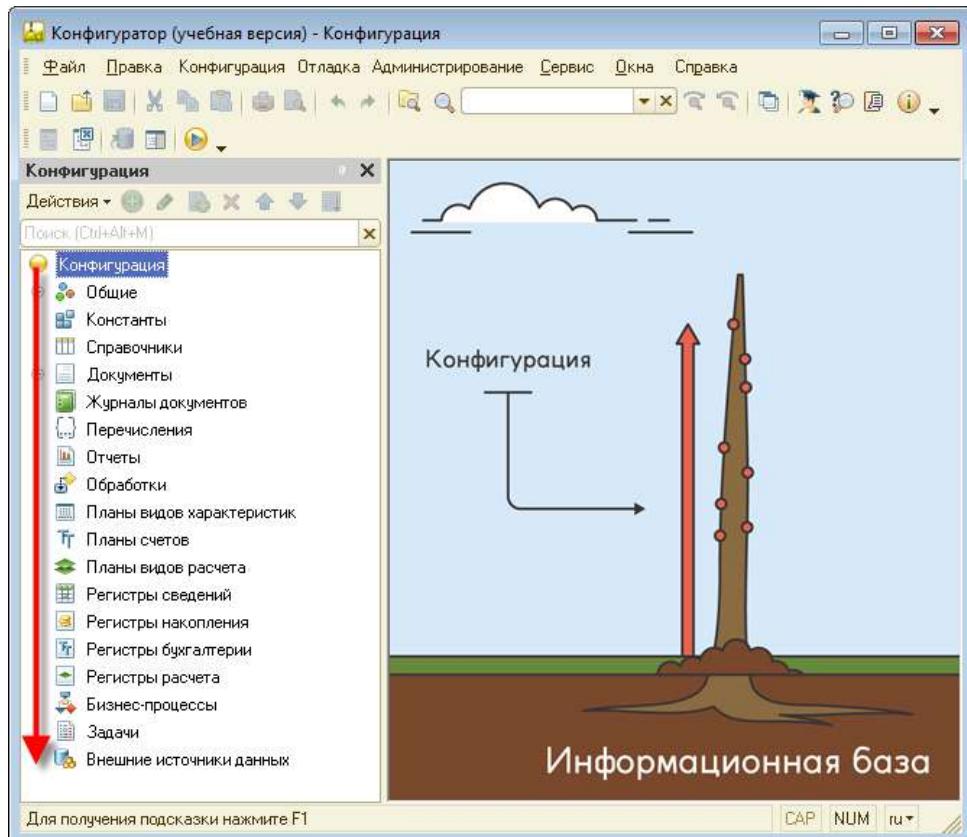


Рисунок 2.20. Дерево объектов конфигурации

Дерево перевёрнуто, корень дерева находится в самом верху — там, где жёлтый круглый юникодный символ. Программистам так удобнее, а вам это просто полезно знать. Потому что иногда я буду использовать этот термин, *корень дерева*. Так вот, в компьютере корень дерева наверху, а не внизу.

Дерево сейчас не содержит ни одной ветки, только ствол. Надписи, которые есть на дереве (*Общие*, *Константы*, *Справочники*, *Документы* и так далее) обозначают те места, в которых к этому дереву можно прикреплять ветки. На рисунке справа эти места обозначены точками.

Ваша задача как разработчика будет заключаться в том, чтобы добавить к этому дереву несколько веток, несколько *объектов конфигурации*. Таких веток, таких объектов конфигурации, которые нужны для решения вашей задачи.

Вы можете спросить: «Если конфигурация состоит из объектов конфигурации, то зачем представлять всё это в виде дерева? Почему нельзя просто один за другим перечислить те объекты, из которых она состоит?»

Дело в том, что, во-первых, каждый объект конфигурации — это не просто палка, воткнутая в ствол. У него самого есть свои ветки, на этих ветках ещё ветки, и так далее (рисунок 2.21). То есть всё это действительно похоже на дерево. Чуть позже вы это увидите.



Рисунок 2.21. Объекты конфигурации — ветки

А во-вторых, если попытаться сравнить дерево объектов конфигурации с обычными деревьями, то на дереве объектов конфигурации одновременно могут расти и яблоки, и груши, и сливы, и другие фрукты. Причём разных сортов. Каждый сорт на своей ветке.

Представьте, если бы вместо *Справочники* было написано *Яблоневые ветки*. А вместо *Документы* — *Сливовые ветки*. Тогда яблоневые ветки (разных сортов) прикрепляются в точке, которая называется *Яблоневые ветки*. Сливовые ветки (их опять же может быть много, и они могут быть разных сортов) прикрепляются в точке *Сливовые ветки*. И так далее (рисунок 2.22).

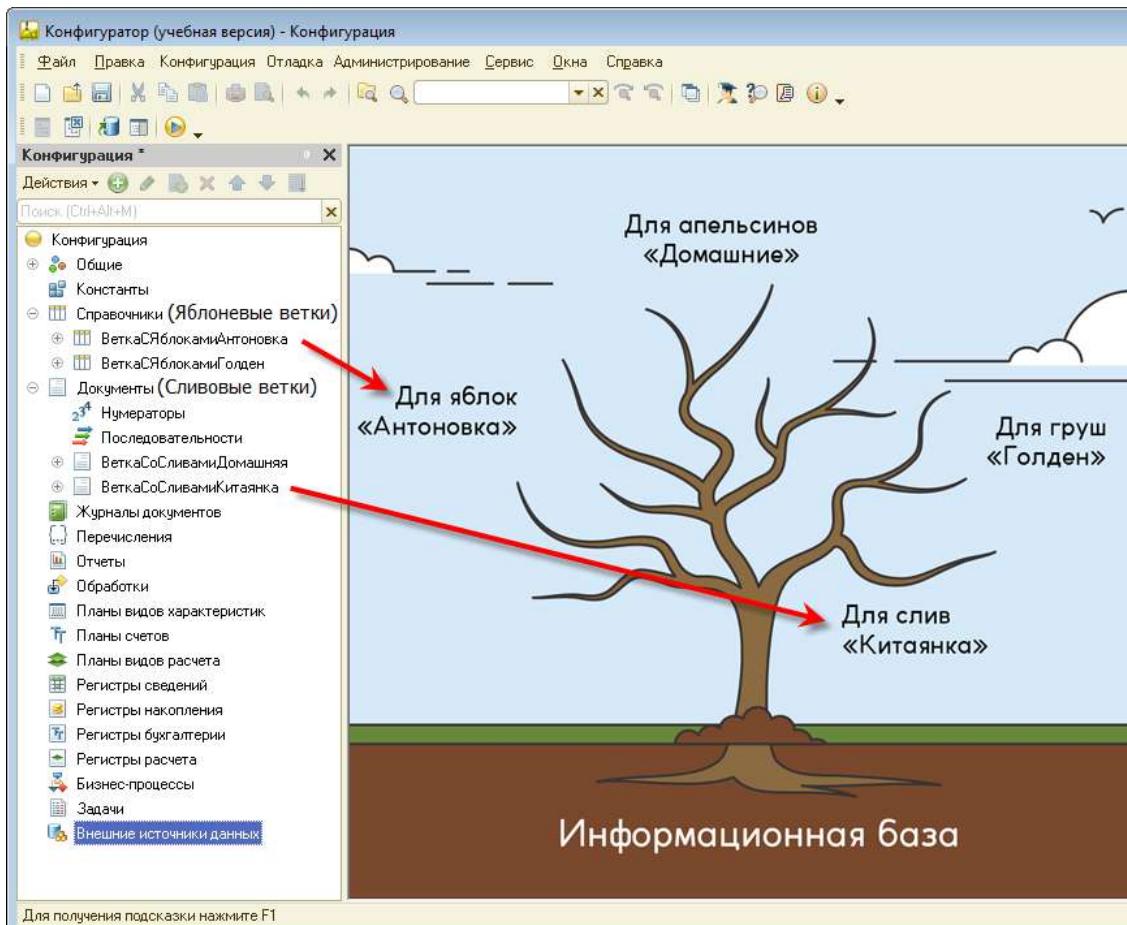


Рисунок 2.22. Объекты конфигурации — ветки плодовых деревьев

Потом, когда пользователи начнут работать с вашим прикладным решением в режиме 1С:Предприятие, они развесят на этих ветках подходящие фрукты — *данные* (рисунок

2.23).



Рисунок 2.23. Конфигурация с данными

Примечание

На что похожи объекты конфигурации? Они похожи на ветки плодовых деревьев. Причём ветки от разных деревьев и от разных сортов. В конфигураторе, в теплице, вы видите только голые ветки. В режиме 1С:Предприятие на этих ветках висят плоды — данные (рисунок 2.24). Эту картинку надо запомнить, она сильно пригодится вам в дальнейшем.

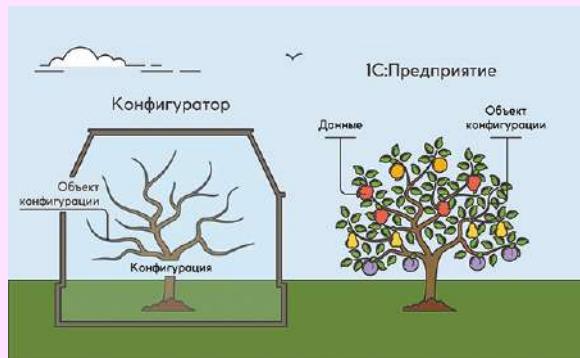


Рисунок 2.24. В конфигураторе — ствол, в режиме 1С:Предприятие — плоды

Теперь по-другому представьте, на что ещё похожи объекты конфигурации. Дерево — это хорошо, но не совсем точно. Забудьте про дерево.

Представьте, что вместо дерева вы строите дом. Одноэтажный дом. Каждый объект конфигурации — это комната. Комнаты, по своему назначению, бывают разные. Гостиная, спальня, кухня, ванная. И дом может быть большой, а значит, в нём может быть несколько спален, разных. Может быть несколько кухонь, разных. И так далее.

В конфигураторе вы рисуете план дома. Какие комнаты в нём будут. Например, две спальни и две гостиные (рисунок 2.25). Каждая комната — это объект конфигурации.

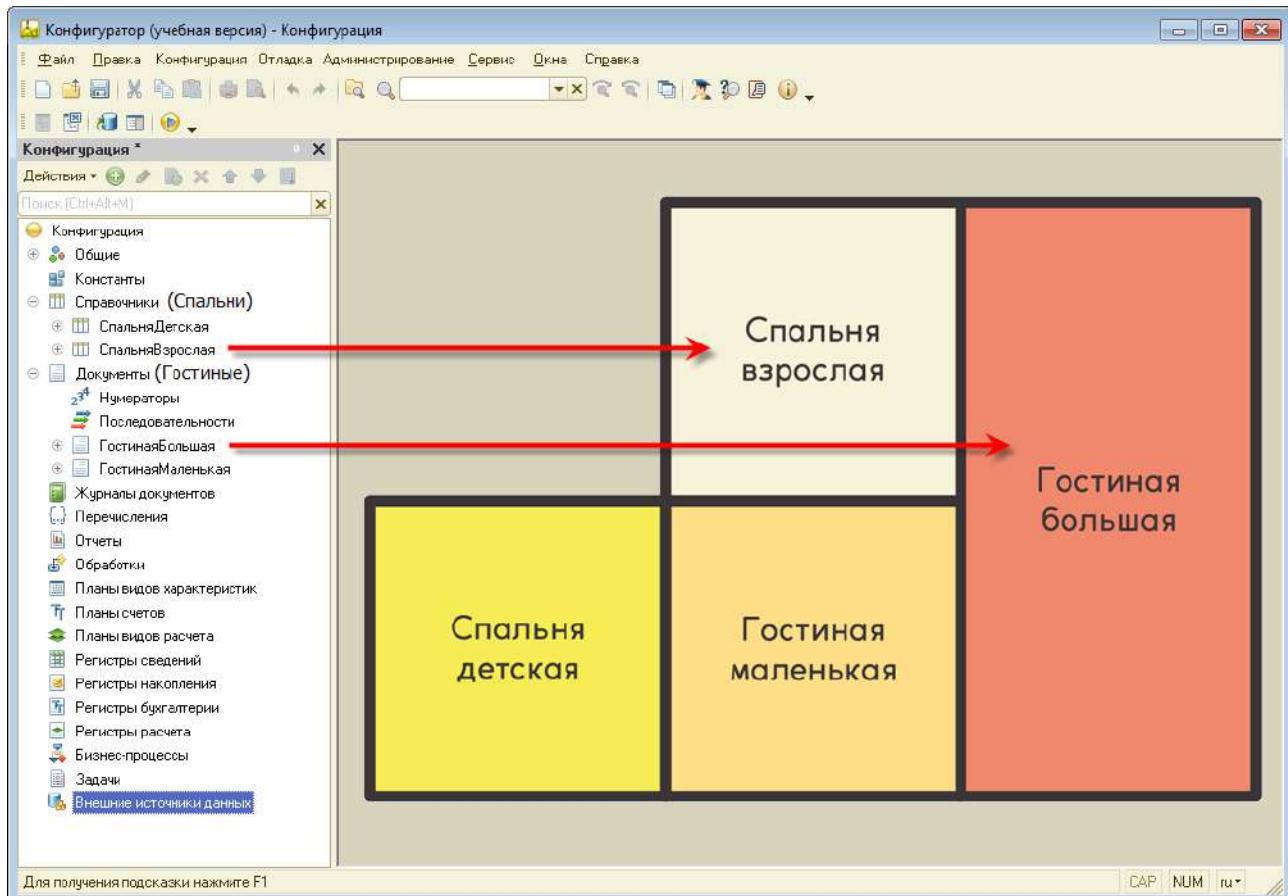


Рисунок 2.25. Объекты конфигурации — комнаты

А в режиме 1С:Предприятие, когда пользователи начнут работать с вашим прикладным решением, они заполнят этот дом людьми — *данными* (рисунок 2.26).

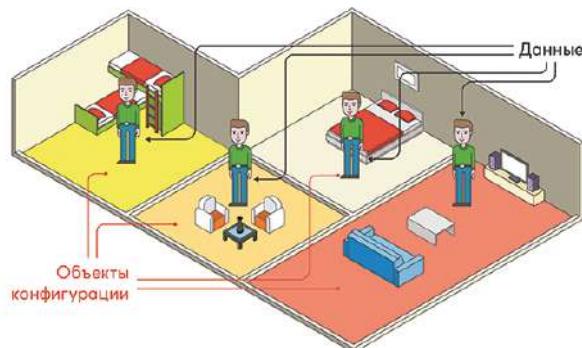


Рисунок 2.26. Конфигурация с данными

Примечание

В конфигураторе у вас только план дома, пустой дом. Он состоит из отдельных комнат — объектов конфигурации. А в режиме 1С:Предприятие этот дом уже наполнен людьми. В каждой комнате находятся какие-то люди, то есть данные (рисунок 2.27).

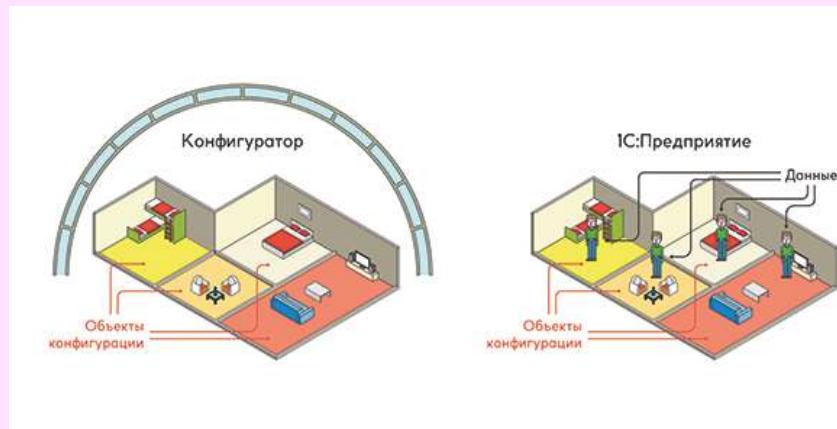


Рисунок 2.27. В конфигураторе — стены, в режиме 1С:Предприятие — люди

Другими словами, прикладное решение — это дом, в котором живут данные. В конфигураторе вы строите стены, комнаты. А в режиме 1С:Предприятие заселяете этот дом данными.

Подробнее

Подробнее вы можете прочитать о дереве объектов конфигурации в документации «Руководство разработчика 8.3. Раздел 2.7. Дерево объектов конфигурации».

2.5 Какие объекты конфигурации можно добавлять

Наверняка у вас уже возник вопрос: «Какие комнаты можно добавлять в этот дом?»

Какие угодно комнаты добавлять нельзя. Нельзя добавить что-то, выдуманное из головы. Можно добавить только те комнаты, те объекты конфигурации, которые знает платформа 1С:Предприятие.

А какие объекты она знает? Все объекты, которые знает платформа, перечислены в дереве объектов конфигурации (рисунок 2.28).

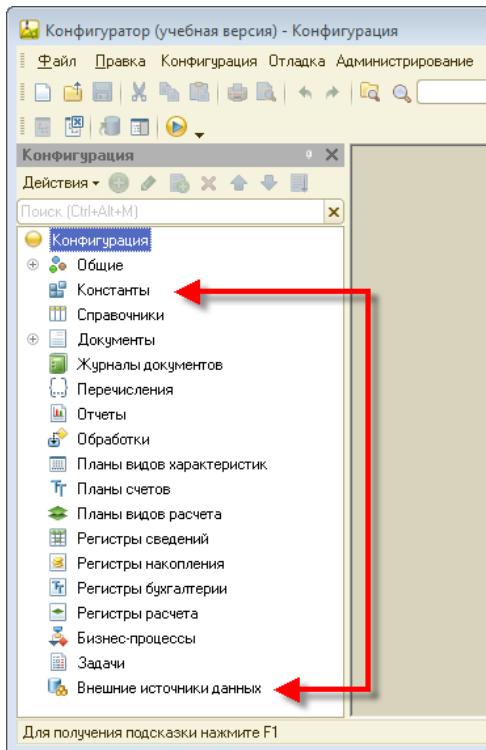


Рисунок 2.28. Доступные объекты конфигурации

Это Константы, Справочники, Документы, Журналы документов, Перечисления, Отчёты и так далее. Всего вы их видите 17 штук. Но это не все объекты. Ещё есть объекты конфигурации, которые находятся в ветке Общие.

Я уже говорил раньше, что система 1С:Предприятие — специализированная. Она не предназначена для того, чтобы решать любые задачи, создавать любые программы. Так вот, этот набор объектов конфигурации придуман таким образом, чтобы их было достаточно для решения тех задач, которые стоят перед прикладными решениями 1С:Предприятия.

Конечно, сложные задачи вы в этой книге выполнять не будете. Поэтому из всего множества объектов конфигурации я познакомлю вас только с некоторыми, самыми главными.

2.6 Красота, или какой объект выбрать

Вы ещё не знакомы ни с одним объектом конфигурации, но уже понимаете, что они разные. Наверное, они чем-то отличаются друг от друга. И это действительно так.

Сейчас, пока вы не начали углубляться в детали, очень хороший момент для того, чтобы усвоить одно золотое правило.

Когда вы разрабатываете прикладное решение, вы добавляете в него объекты конфигурации. Тем самым в компьютере вы пытаетесь «изобразить» что-то похожее на окружающую вас действительность. Например, в этой книге вы будете создавать в компьютере подобие школьного дневника, существующего в действительности в виде бумажной тетрадки.

Естественно, действительность вокруг вас самая разная. А объектов конфигурации в компьютере, грубо говоря, всего 17 штук. Поэтому очень важно выбирать такие объекты конфигурации, которые максимально подходят для вашей задачи.

«Для этого в них нужно хорошо разбираться», — скажете вы. Да. И этому вы будете учиться.

Но есть и другой способ. Не менее важный. А иногда и более важный. Он основывается на вашей интуиции, и обозначить его можно одним словом — красота.

Красивая программа хорошо работает. Эта закономерность известна многим программистам. Но какая красота может быть в компьютере? Это же не картина, не фотография. Что должно быть красивым? Та картинка, которую вы видите на экране? Или что-то другое?

Объяснить это коротко нельзя. Поэтому по ходу ваших занятий я иногда буду говорить о красоте. Разбирать разные примеры. Как красиво. Как некрасиво. Сейчас как раз один из таких удобных моментов.

Когда вы выбираете те или иные объекты конфигурации, вспомните картинку, на которой вы представляли их в виде комнат (рисунок 2.29).

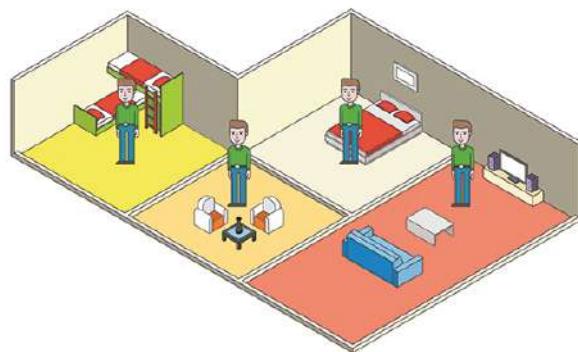


Рисунок 2.29. Объекты конфигурации и данные

Например, вы выбираете между спальней и гостиной. Они похожи. И там, и там можно посидеть с друзьями. И там, и там при желании можно спать. Но интуитивно вы понимаете, что назначение этих комнат разное.

Если добавить только спальню, это будет некрасивое решение. Гости не смогут разместиться в ней удобно и комфортно. Если добавить только гостиную, то это тоже некрасивое решение. Когда кто-то из вашей семьи захочет отдохнуть, он просто не сможет уединиться, пока гости не уйдут. Поэтому красивым решением будет добавить и спальню, и гостиную. Чтобы всем было удобно.

Другой пример. И в ванной, и на кухне есть раковина. Помыть руки можно и там, и там. Казалось бы, можно обойтись без ванной. Но это некрасивое решение. И вам, и гостям при этом будет очень неудобно.

Поэтому красивое решение — добавить и кухню, и ванную. А если гостей у вас бывает много или они останавливаются на длительный срок, то хорошо бы иметь не одну ванную, а две. Одну — для себя, а другую — для гостей. Чтобы они не испытывали неудобства.

Примечание

Выбирая те или иные объекты конфигурации для построения своей программы, не забывайте о красоте. Когда объекты конфигурации используются по своему прямому назначению — это **красиво**. И это хорошо работает. Такую программу вам легко разрабатывать. Когда через полгода или год вы решите что-то изменить в такой программе, вы быстро и легко вспомните, как она устроена.

Используя объекты конфигурации не по назначению, вы, конечно, поразите других программистов своей оригинальностью. Но хорошо работать такое решение не будет. Вам с ним будет сложно. И вполне может быть, что через некоторое время вы и сами уже не вспомните, как оно устроено.

Поэтому не бойтесь быть проще и старайтесь делать красивые вещи.
Сложно и некрасиво у вас само получится, для этого и стараться не надо.

2.7 Данные

Чтобы правильно выбрать объекты конфигурации, нужно понять, какие данные вы собираетесь использовать. Я уже употреблял это слово — «данные». Но не объяснял, что это такое. Сейчас пора на них посмотреть, на данные.

Вспомните свою задачу. Вы в компьютере хотите сделать аналог школьного дневника. Школьный дневник выглядит так (рисунок 2.30).

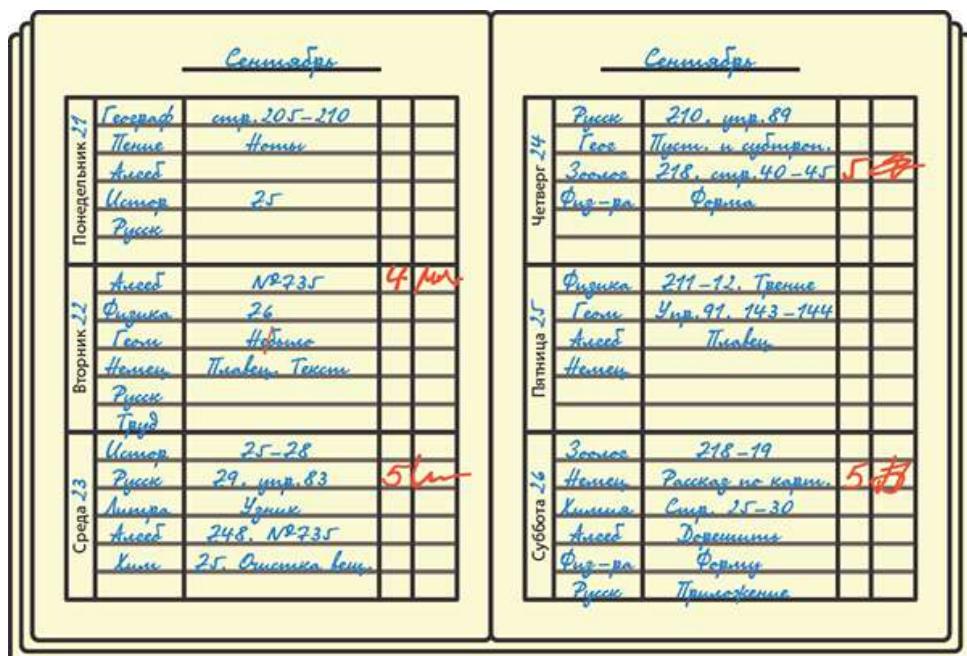


Рисунок 2.30. Школьный дневник

Это тетрадка, в которую вы что-то записываете. Так вот, то, что вы туда записываете, — это и есть **данные**.

Для тетрадки данные выглядят просто как разные надписи. Но они написаны в определённых местах тетрадки. Поэтому мы, люди, можем понять, где написано название предмета, а где — домашнее задание.

Но компьютер так не умеет. Поэтому, чтобы компьютер не запутался и мог отличать названия предметов от домашних заданий, от оценок и другой информации, вы должны каким-нибудь образом разложить все похожие друг на друга данные по отдельным «кучкам». Потом для хранения каждой кучки выбрать подходящий объект конфигурации. Добавить эти объекты в свою конфигурацию. И в режиме 1С:Предприятие записать в компьютер эти данные (рисунок 2.31).

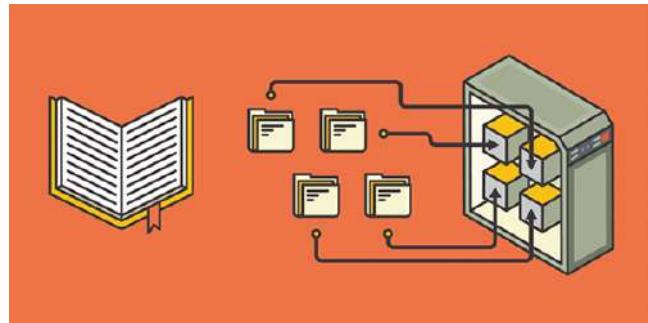


Рисунок 2.31. Как данные попадут в компьютер

Вы уже совершенно точно можете назвать одну такую «кучку» данных. Это предметы. Вы должны иметь в компьютере список тех предметов, которые вы изучаете.

Данные:

- предметы.

Дальше. Ваш компьютерный журнал будет лучше, чем бумажная тетрадка. Поэтому в компьютерном журнале на каждый день у вас будет записано не только расписание уроков, но и указано, какой учитель ведёт этот предмет. И в каком классе проходят занятия.

Поэтому вы, наверное, уже догадываетесь: у вас вырисовываются ещё две «кучки» данных. Это учителя. И это кабинеты.

Данные:

- предметы,
- учителя,
- кабинеты.

Ещё дальше. Допустим, в компьютере у вас уже есть предметы, учителя, кабинеты. Но дневника из этого не получается. Чего не хватает? А не хватает такой таблички на каждый день, в которой будет написано, какие есть уроки, что по ним задано и какие оценки получены. То есть вырисовывается ещё одна «кучка» данных, которую можно назвать «учебные дни».

Данные:

- предметы,
- учителя,
- кабинеты,
- учебные дни.

Пока этого достаточно. Пора создавать объекты конфигурации. Я буду рассказывать вам о некоторых (самых нужных) объектах конфигурации, а вы будете их создавать.

2.8 Справочник

Сейчас, хочу вам напомнить, вы находитесь в лесу и стоите перед деревом (рисунок 2.32).

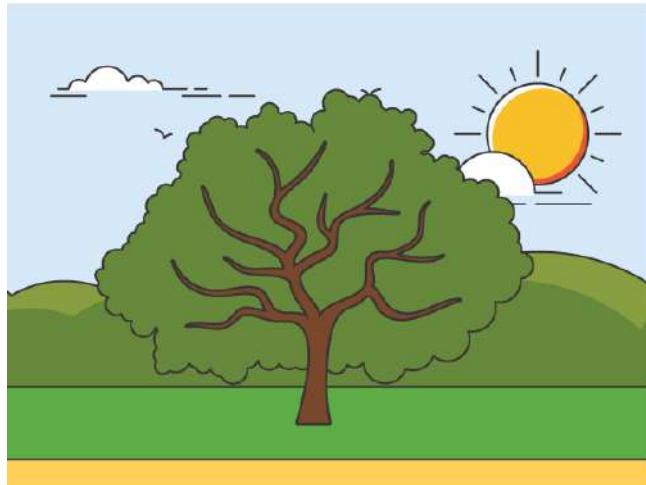


Рисунок 2.32. Дерево

Ствол этого дерева вам уже известен. Вы знаете, что к нему крепятся большие ветки. Сейчас вы будете изучать одну из больших веток этого дерева — справочники.

Есть такой объект конфигурации, который называется *справочник*. Существуют два признака, по которым можно понять, что для хранения ваших данных нужен справочник.

Во-первых, справочник подходит для того, чтобы хранить в компьютере данные, которые в обычной жизни вы представляете себе в виде какого-то списка. И в этом же виде их используете.

Например, ваши контакты. Они есть в каждом телефоне. То есть весь список контактов — это справочник. А каждый отдельный контакт — это элемент справочника. Как вы используете контакты? Именно как список. Вы находите в нём какого-то человека и потом звоните ему или отправляете сообщение.

Примечание

Если данные выглядят как список и вы работаете с ними как со списком, это первый верный признак того, что для их хранения нужен справочник.

Второй признак сложнее. Чтобы его понять, нужно ещё раз посмотреть в школьный дневник. В дневнике вы можете в одном месте написать «Географ.», а в другом — «Геог.» (рисунок 2.33).

Сентябрь		
Понедельник 27	Географ	чтв. 205-210
	Пение	Ноты
	Алгебр	
	Информ	25
	Русск	
Четверг 29	Русск	210, упр. 89
	Геог	Путеш. и губерн.
	Зоолог	218, упр. 40-45
	Физ-ра	Форма

Рисунок 2.33. География

И в том и в другом случае вы понимаете, что речь идёт о предмете «География».

Или в одном месте вы можете написать «Хим.», а в другом — «Химия». Тут вы тоже понимаете, что речь идёт о химии (рисунок 2.34).

The image shows two school timetables for Chemistry. The left timetable is for Wednesday (Среда) and the right one is for Friday (Суббота). Both timetables have a grid of 5 columns and 6 rows.

	Имя	25-28		
Среда 23	Русин	29, учр. 83	5	лн
	Людмила	Ученик		
	Алексей	248, № 735		
	Кимин	25. Олимпиада химии		

	Занятие	218-19		
Суббота 26	Начало	Рассказ по карточкам	5	лн
	Химия	Спирт, 25-30		
	Алексей	Дорогами		
	Физ-ра	Формулы		
	Русин	Применение		

Рисунок 2.34. Химия

Но компьютер так не умеет. Если вы так запишете в компьютере, он подумает, что это два разных предмета: «Хим.» и «Химия».

Как люди обычно выходят из такой ситуации? Мы решаем, что всегда и везде нужно писать одинаково и правильно: «Химия». Тогда и вы, и я, и любой другой человек, и компьютер точно не ошибётся.

Но это неудобно. Не все знают, что писать надо именно так. Если название длинное, а вы спешите, наверняка вы его как-то сократите.

Чтобы не возникало ошибок и разночтений, в компьютере такие данные тоже удобно хранить в справочнике. Зачем? Затем, чтобы в том месте, где нужно будет написать «Химия», не писать каждый раз «Химия», а просто *сослаться* на элемент справочника, который называется «Химия».

Это значит, что вы запишете в дневник не сами данные, не название предмета «Химия», а *ссылку* на элемент справочника. Этот элемент уже есть где-то в компьютере, он называется «Химия». И таким образом в каждом месте, в каждой клетке, где вы будете ссылаться на этот элемент, автоматически появится его правильное название.

Не удивляйтесь, но со ссылками вы очень часто имеете дело в жизни. Только не обращаете на это внимания.

Смотрите, например друг спрашивает вас: «Какой у тебя телефон?» Вы достаёте из кармана телефон, показываете ему и говорите: «Вот такой» (рисунок 2.35). Это вы ему даёте данные.



Рисунок 2.35. «Вот такой у меня телефон» — данные

А теперь другой случай. Он спрашивает вас: «Какой у тебя телефон?» А вы показываете рукой на стол, где лежит ваш телефон, и говорите: «Вон такой, иди, посмотри» (рисунок 2.36).

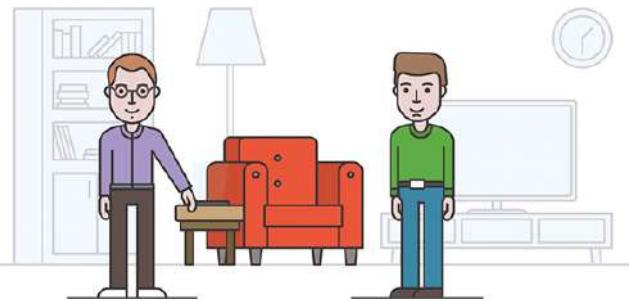


Рисунок 2.36. «Мой телефон вон там» — ссылка

Это вы дали ему ссылку. То есть сказали, где лежат данные.

Примечание

Если один и тот же элемент ваших данных нужно использовать (вводить, записывать, выбирать, подставлять) в разных местах программы, то это другой верный признак того, что для хранения этих данных нужен объект конфигурации, который называется *справочник*.

Подробнее

Подробнее вы можете прочитать про справочник в документации «[Руководство разработчика 8.3. Раздел 5.8. Справочники](#)».

2.9 Кабинеты

Начните с самых простых ваших данных, с кабинетов. Подходит ли справочник для хранения таких данных? Давайте разберёмся.

Нужен ли вам список кабинетов? Важно ли вам знать, какие вообще есть кабинеты? Нет. Значит, по первому признаку не подходит.

Надо ли указывать один и тот же кабинет в разных местах программы? Да. Потому что рядом с названием урока вы хотите записывать и номер кабинета, в котором проходит урок. И таких мест много, потому что в одном и том же кабинете вы можете заниматься в разные дни. И в понедельник, и в среду, например.

Значит, добавьте в свою программу справочник. Перейдите в конфигуратор, он должен быть всё ещё открыт у вас на экране (рисунок 2.37).

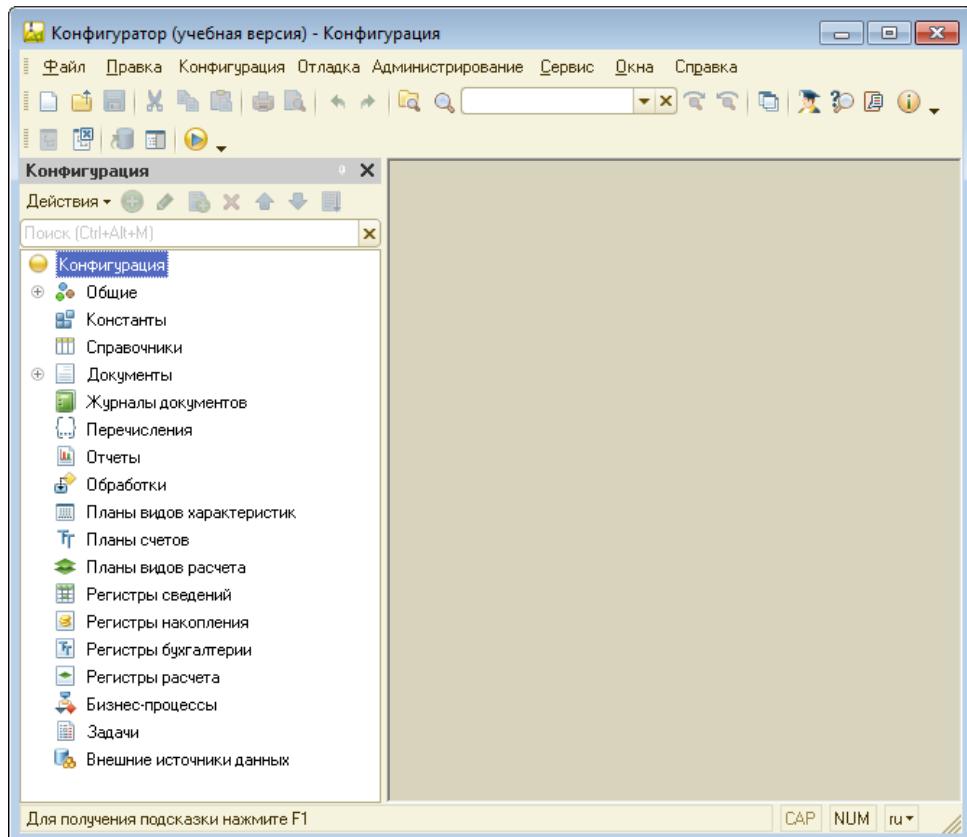


Рисунок 2.37. Пустая конфигурация

Сразу скажу: когда вы работаете в конфигураторе, почти всегда есть несколько способов сделать одно и то же действие. Например, добавить объект конфигурации.

Я буду показывать всегда только один способ. Самый простой и удобный. Если потом вы прочитаете или узнаете, что то же самое можно было сделать другим способом — не удивляйтесь и не расстраивайтесь. Я специально не рассказываю обо всём, чтобы вы не запутались.

Итак, проще всего *добавить объект конфигурации* с помощью контекстного меню. *Контекстное меню* — это список команд, который появляется в результате нажатия на правую кнопку мыши.

Вы хотите добавить справочник, значит, покажите мышью на *Справочники* и нажмите правую кнопку мыши (рисунок 2.38).

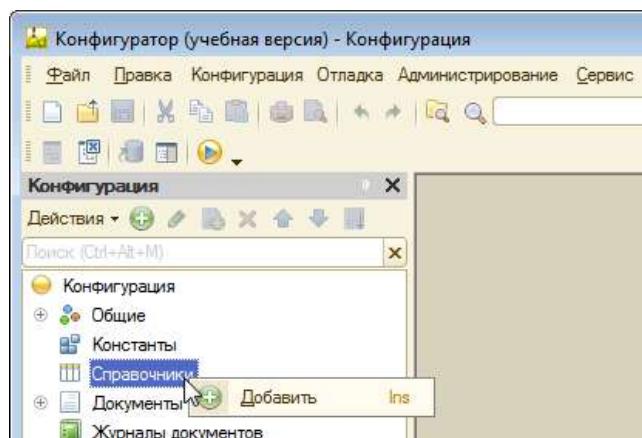


Рисунок 2.38. Добавить справочник

В контекстном меню вы видите всего одну команду — *Добавить*. Нажмите её.

В дереве конфигурации появится *Справочник1*, и откроется новое окно (рисунок 2.39).

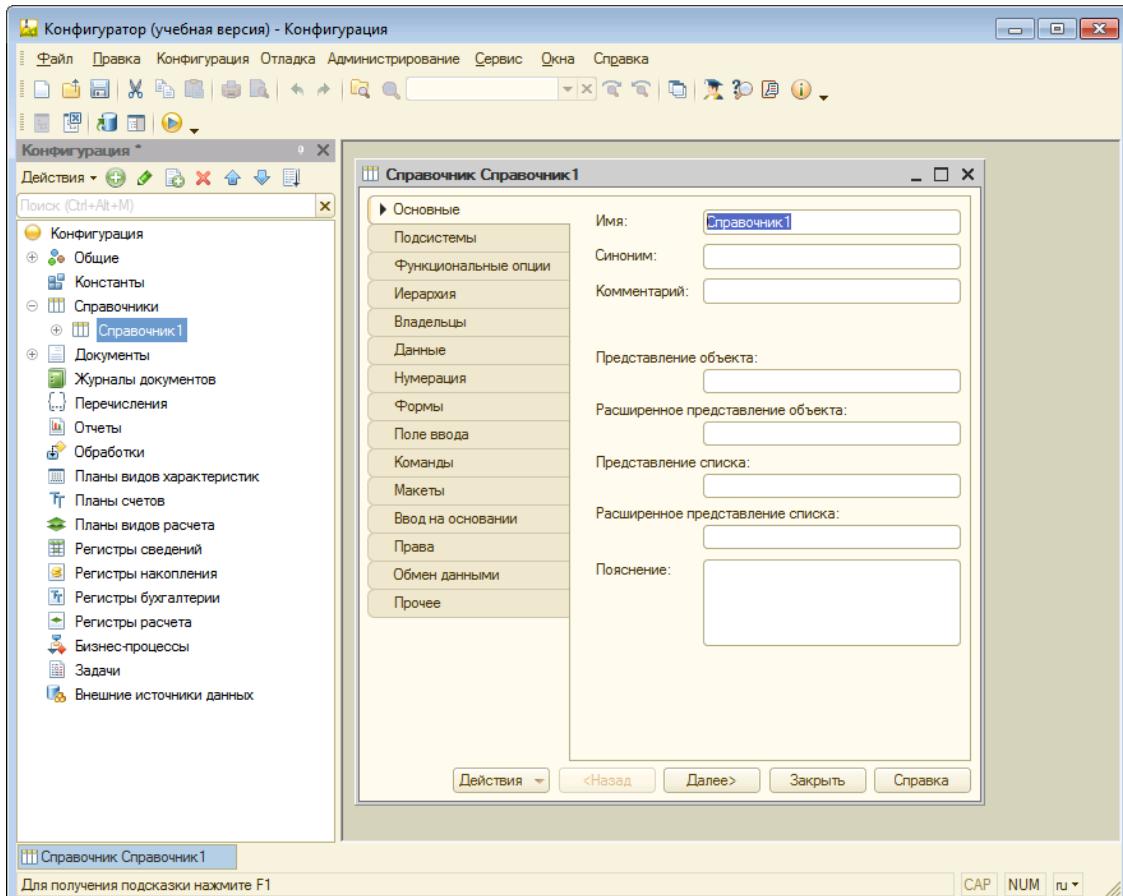


Рисунок 2.39. Окно редактирования справочника

Это окно называется *окно редактирования объекта конфигурации*. С его помощью можно изменить и настроить объект конфигурации так, как вам нужно.

Смотрите: новый объект конфигурации платформа создала сама и поэтому назвала его *Справочник1*. Но вам такое название не подходит. Назовите его *Кабинеты*.

Для этого в поле *Имя* вместо *Справочник1* напишите *Кабинеты*.

После этого нажмите на клавиатуре TAB, чтобы перейти к следующему полю (рисунок 2.40).

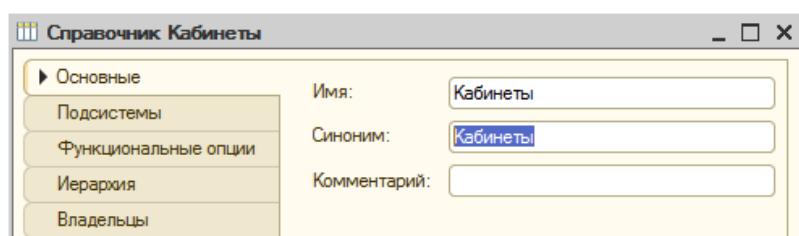


Рисунок 2.40. Имя справочника — «Кабинеты»

Зачем нажимать клавишу TAB? Это нужно для того, чтобы платформа поняла, что вы закончили изменять значение поля. Иногда после этого платформа выполняет какие-то дополнительные действия. Например, сейчас, после того как вы закончили изменять значение поля *Имя*, платформа сама автоматически сформировала синоним для этого объекта конфигурации и подставила его в поле *Синоним*.

Примечание

Когда в конфигураторе вы изменяете значения каких-то полей, не забывайте в конце нажимать клавишу TAB.

Подробнее

Подробнее вы можете прочитать про сочетания клавиш для работы с формой во встроенной справке (командная панель сверху): *Справка — Содержание справки — Сочетания клавиш (Конфигуратор) — Форма*.

И раз уж я упомянул *сионим*, объясню, что это такое. Сейчас вы находитесь в конфигураторе и дали объекту конфигурации *имя Кабинеты*. Это имя удобно и понятно для вас. Но в режиме 1С:Предприятие, когда пользователи будут работать с вашей программой, вполне может оказаться так, что *имя Кабинеты* для них будет непонятно, некрасиво, неудобно и так далее. Поэтому вы можете в поле *Синоним* написать то название, которое будет понятно и удобно для пользователей. То есть *имя* — это для вас, для разработчика, а *сионим* — это для пользователей.

Сейчас получилось так, что имя и синоним совпадают. Но обычно они отличаются, вы увидите это позже.

Продолжим заниматься разработкой. Обратите внимание: когда вы что-то меняете в конфигурации, платформа сразу это замечает и показывает звёздочку рядом с названием конфигурации (рисунок 2.41).

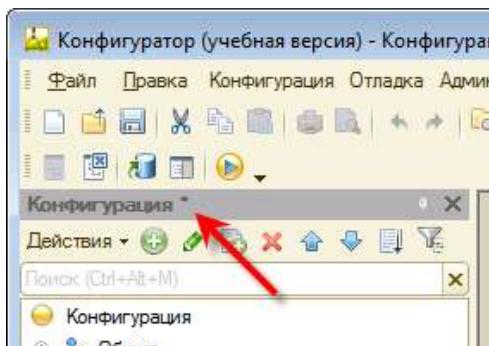


Рисунок 2.41. Конфигурация была изменена

Если звёздочки нет, значит, в конфигурации ничего не менялось. Если звёздочка есть, значит, вы что-то изменили в конфигурации. Это помогает вам случайно не потерять свои изменения.

Если сейчас вы случайно (или специально) попробуете закрыть конфигуратор, платформа не даст это сделать. Попробуйте.

Закройте конфигуратор. Вы увидите такое сообщение (рисунок 2.42).

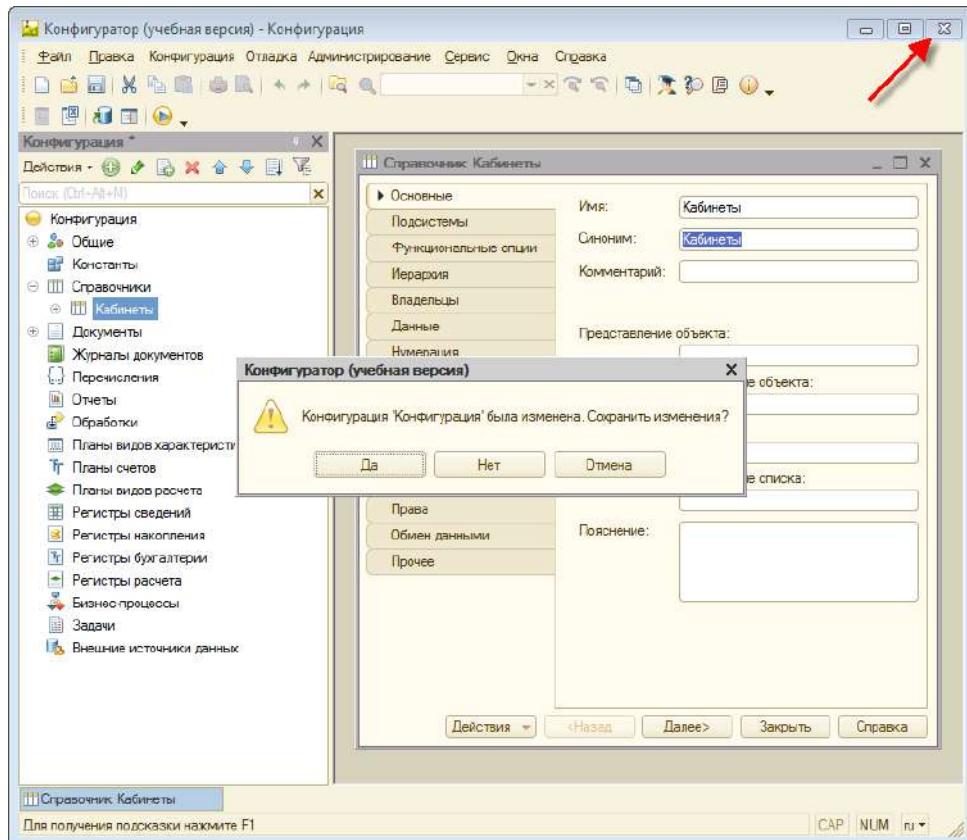


Рисунок 2.42. Предупреждение о том, что конфигурация была изменена

Нажмите Отмена.

Теперь вы знаете: после того как вы что-то изменили в конфигурации, эти изменения нужно сохранить. Чтобы *сохранить изменения конфигурации*, нажмите кнопку *Сохранить* (рисунок 2.43).

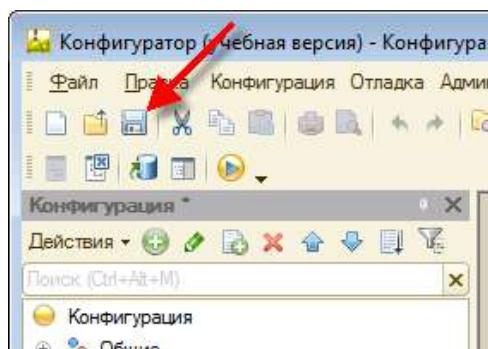


Рисунок 2.43. Сохранить изменения конфигурации

Ну что? Вроде бы вы всё сохранили. Хотите посмотреть, как это работает в режиме 1С:Предприятие?

Не закрывайте конфигуратор. Ещё раз запустите 1С:Предприятие и в списке информационных баз нажмите 1С:Предприятие. Вы увидите такое сообщение (рисунок 2.44).

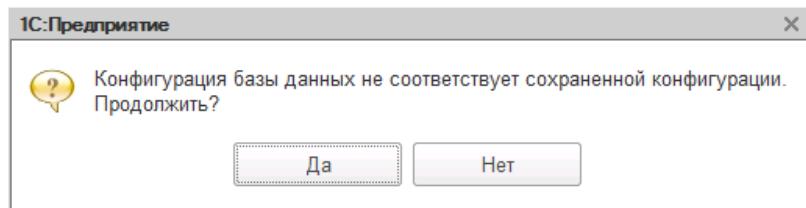


Рисунок 2.44. Конфигурация базы данных не соответствует сохранённой конфигурации

Если вы нажмёте *Нет*, программа просто закроется. Если вы нажмёте *Да*, то программа откроется и вы увидите ту же картинку, что вы видели в самый первый раз. Когда только добавили новую информационную базу и хотели посмотреть, как она выглядит в режиме 1С:Предприятие (рисунок 2.45).

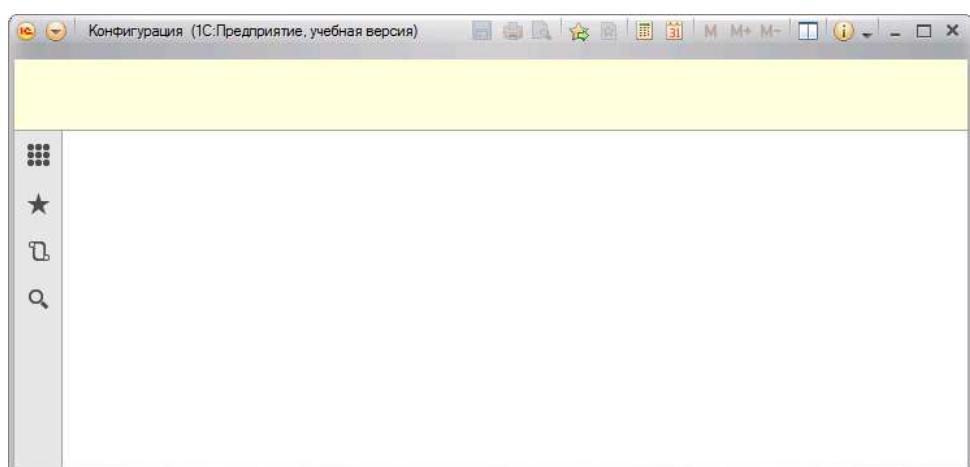


Рисунок 2.45. Пустая конфигурация в режиме 1С:Предприятие

То есть ничего нового в ней не появилось. А ведь вы добавили справочник *Кабинеты*. Где же он?

Тут нужно подробнее объяснить, как устроена информационная база.

2.9.1 Информационная база

Может быть, это объяснение покажется вам сложным. Но не расстраивайтесь. Это объяснение нужно только для того, чтобы вы не чувствовали себя «дрессированными обезьянками, нажимающими на клавиши». Если вы не поймёте его до конца, это не страшно. Потому что в дальнейшем платформа всё будет делать за вас, а вам нужно будет только нажимать на кнопки.

Итак. Вспомните, когда вы в первый раз столкнулись с информационной базой и добавляли новую базу. Я сказал тогда, что информационная база — это такая посадочная яма, в которую вы будете сажать ваше дерево — конфигурацию. И когда вы создали новую информационную базу, в ней уже есть ствол. Без веток (рисунок 2.46).

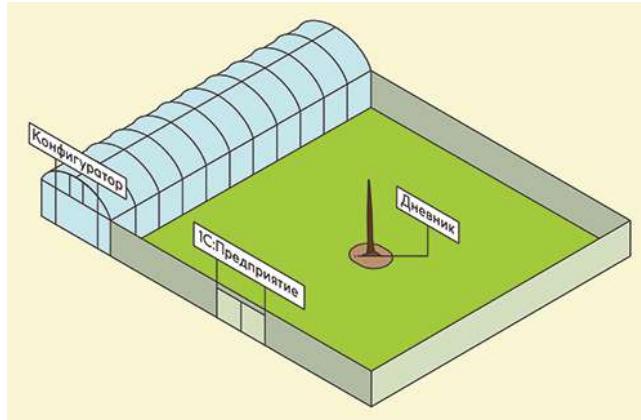


Рисунок 2.46. Ваш сад

Так вот. Дело в том, что эта посадочная яма есть не только в 1С:Предприятии, но и в конфигураторе (рисунок 2.47).

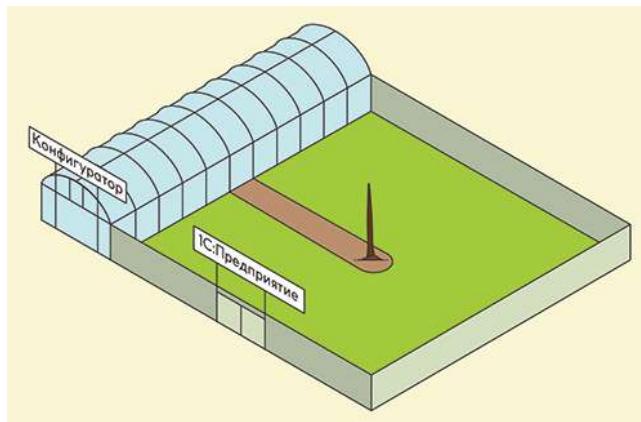


Рисунок 2.47. Информационная база в саду и под теплицей

И если посмотреть на эту конструкцию в разрезе, то вы увидите, что в посадочной яме не одна, а две конфигурации, два ствола (рисунок 2.48).



Рисунок 2.48. Основная конфигурация и конфигурация базы данных

С основной конфигурацией работают разработчики. То есть вы. Вы изменяете её в конфигураторе. Например, сейчас вы добавили к ней справочник Кабинеты (рисунок 2.49).



Рисунок 2.49. Основная конфигурация изменена

А *конфигурация базы данных* — это конфигурация, с которой работают пользователи. В ней пока, как вы увидели, никаких изменений не произошло. Как в ней не было ничего, так в ней и сейчас ничего нет (рисунок 2.50).



Рисунок 2.50. Конфигурация базы данных — для пользователей

Кстати, именно об этом вам и говорила платформа, когда вы запускали приложение в режиме 1С:Предприятие (рисунок 2.44). «Конфигурация базы данных не соответствует сохранённой конфигурации».

Зачем нужны две конфигурации? Чтобы вы могли изменять прикладное решение даже в то время, когда с ним работают пользователи. А потом, когда все изменения будут готовы, просто взмахнули волшебной палочкой, и у пользователей конфигурация стала точно такой же, как у вас.

Где же эта волшебная палочка? Что же нужно сделать, чтобы изменения, которые вы выполнили в основной конфигурации, появились и в конфигурации базы данных?

Для этого нужно *обновить конфигурацию базы данных*.

Закройте сеанс 1С:Предприятия, если он у вас ещё открыт, и вернитесь в конфигуратор. Волшебная палочка — это кнопка *Обновить конфигурацию базы данных* (рисунок 2.51).

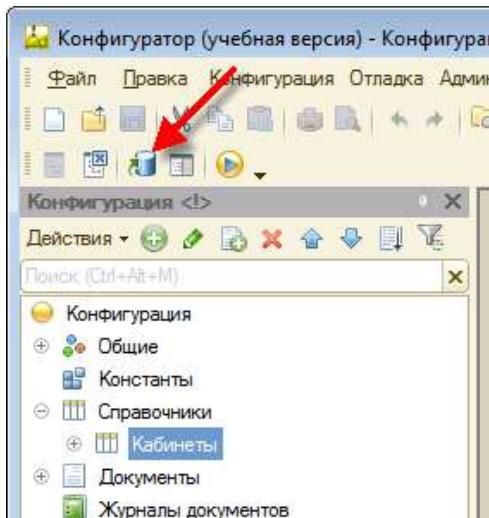


Рисунок 2.51. Обновить конфигурацию базы данных

Но прежде чем её нажимать, обратите внимание, что рядом с названием конфигурации вы видите восклицательный знак в кавычках (рисунок 2.52).

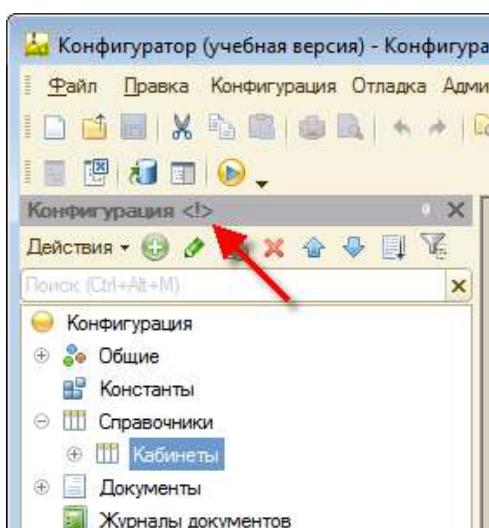


Рисунок 2.52. Основная конфигурация отличается от конфигурации базы данных

Этот значок как раз и говорит, что основная конфигурация (которую вы перед этим изменили и сохранили) отличается от конфигурации базы данных.

Теперь нажмите кнопку *Обновить конфигурацию базы данных* (рисунок 2.51). Платформа проанализирует все отличия, которые есть между двумя конфигурациями, и сообщит вам о тех, которые являются наиболее важными (рисунок 2.53).

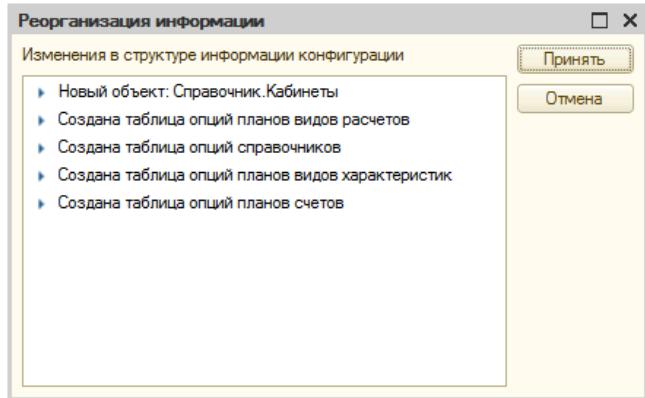


Рисунок 2.53. Изменения в структуре конфигурации

Это нужно только для вашего самоконтроля. Чтобы случайно в конфигурацию базы данных не попали какие-то ваши эксперименты, которые вы делали, но забыли потом отменить в основной конфигурации. Поэтому обычно здесь всегда нажимают *Принять*. Нажмите и вы.

Всё. Теперь конфигурация базы данных стала такой же, как основная конфигурация, которую вы изменили. И, кстати, рядом с названием конфигурации теперь снова нет никаких значков. Это значит, что основная конфигурация не изменилась и что она в точности соответствует конфигурации базы данных.

Теперь я объясню всё то же самое ещё раз по шагам и на картинке. Чтобы вы могли проверить, как вы поняли.

Вы зашли в конфигуратор. Рядом с названием конфигурации нет значков (рисунок 2.54).



Рисунок 2.54. Обе конфигурации одинаковы.

Вы добавили новый объект конфигурации. Рядом с названием конфигурации появилась звёздочка. Она говорит о том, что конфигурация была изменена (рисунок 2.55).



Рисунок 2.55. Основная конфигурация изменена

Вы сохранили основную конфигурацию. Рядом с названием конфигурации появился восклицательный знак. Он говорит о том, что основная конфигурация отличается от конфигурации базы данных (рисунок 2.56).



Рисунок 2.56. Основная конфигурация отличается от конфигурации базы данных

Вы обновили конфигурацию базы данных. Конфигурации стали идентичными. А рядом с названием конфигурации снова нет никаких значков (рисунок 2.57).



Рисунок 2.57. Конфигурации идентичны

Подробнее

Подробнее вы можете прочитать про конфигурацию в документации «[Руководство разработчика 8.3. Раздел 2.1. Общая информация](#)».

Вот теперь снова запустите прикладное решение в режиме 1С:Предприятие и посмотрите, изменилось в нём что-то или нет (рисунок 2.58).

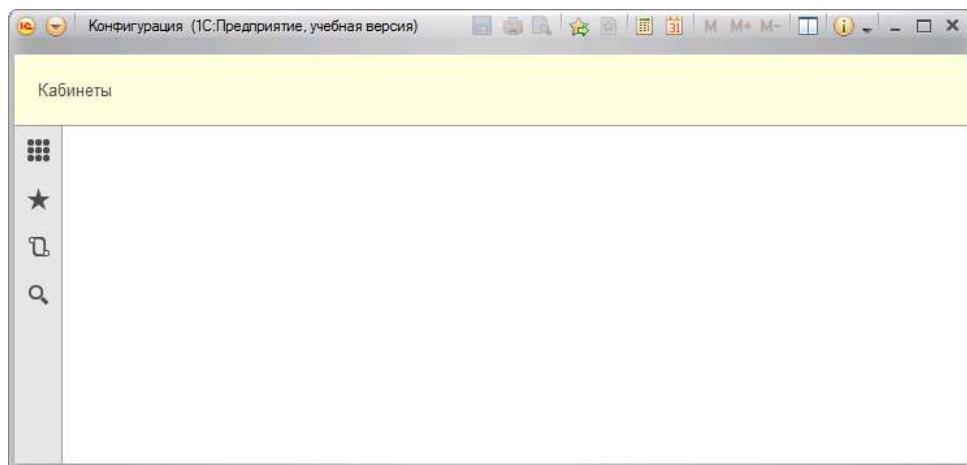


Рисунок 2.58. Прикладное решение в режиме 1С:Предприятие

Да, изменилось. На жёлтой полосе появилась надпись *Кабинеты*. Уже совсем скоро вы узнаете, зачем нужна эта надпись и что с ней делать.

А пока закройте 1С:Предприятие, вернитесь в конфигуратор. Нужно разобраться с тем, как вы будете изменять, и сохранять, и запускать 1С:Предприятие в процессе всех занятий в этой книжке.

Как вы это делали только что? Вы запускали платформу. Заходили в конфигуратор. Что-то там меняли. Потом опять запускали платформу. Запускали 1С:Предприятие и смотрели, что получилось. То есть выглядело это так (рисунок 2.59).

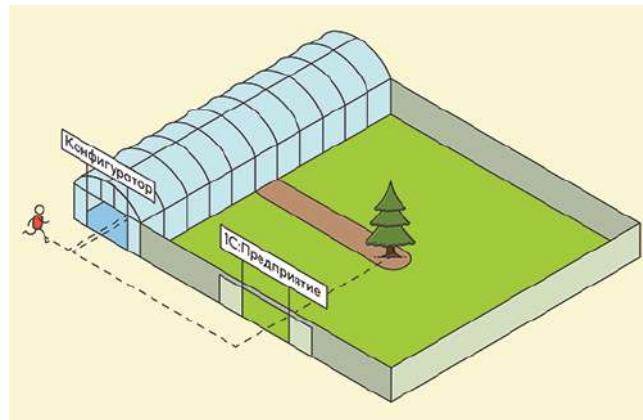


Рисунок 2.59. Режим «Конфигуратор» и режим «1С:Предприятие»

Но есть более простой и короткий путь. В конфигураторе есть дверь прямо в 1С:Предприятие (рисунок 2.60).

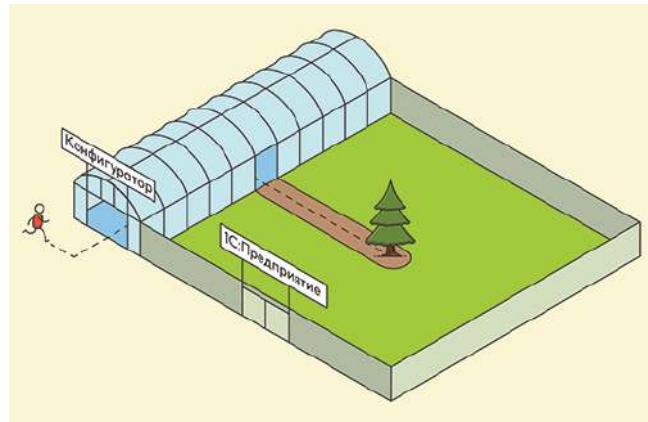


Рисунок 2.60. Запуск 1С:Предприятия из конфигуратора

То есть, зайдя в конфигуратор и изменив там что-то, вы можете сразу запустить 1С:Предприятие и посмотреть, что получилось. При этом платформа сама сохранит конфигурацию, обновит конфигурацию базы данных и запустит потом 1С:Предприятие.

Посмотрите, как запустить *1С:Предприятие в режиме отладки* из конфигуратора.

Перейдите в конфигуратор. Закройте окно редактирования объекта конфигурации. Пока оно вам не понадобится.

Внесите одно полезное изменение в конфигурацию. Сейчас она называется у вас *Конфигурация*. Это стандартное название для новой конфигурации, и оно вам ни о чём не говорит. Назовите её *Дневник*.

Для этого откройте контекстное меню в корне конфигурации (рисунок 2.61).

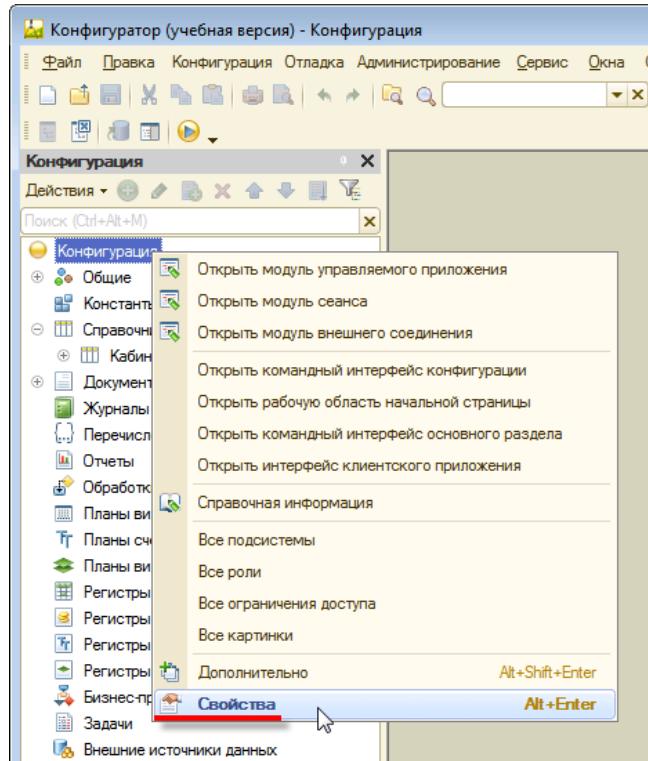


Рисунок 2.61. Контекстное меню в корне конфигурации

В самом низу будет команда *Свойства*. Нажмите на неё.

Откроется новый элемент конфигуратора — *палитра свойств* (рисунок 2.62).

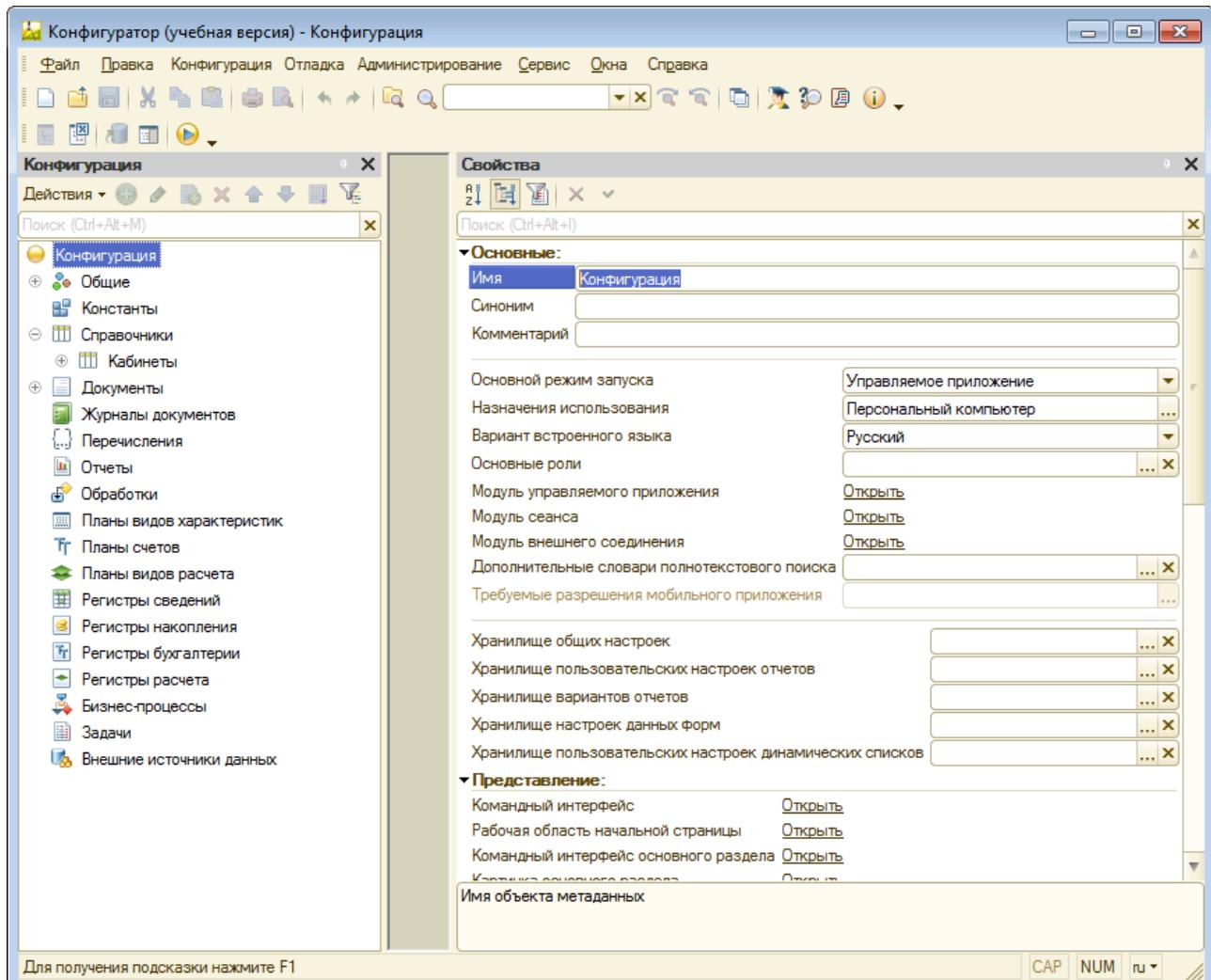


Рисунок 2.62. Палитра свойств

Если у вас палитра свойств не такая широкая, как на картинке, покажите мышью на её левый край. Курсор изменится. Теперь вы можете нажать на левую кнопку мыши и потянуть левый край на такую ширину, которая вам нужна.

Палитра свойств — это ещё один инструмент, с помощью которого вы можете изменять конфигурацию. Она позволяет работать с тем элементом конфигурации, который выделен слева, в дереве конфигурации. Сейчас у вас выделен корень всей конфигурации, значит, вы можете поменять что-то, что относится ко всей конфигурации в целом.

Измените имя конфигурации с *Конфигурация* на *Дневник*. Не забудьте про клавишу TAB (рисунок 2.63).

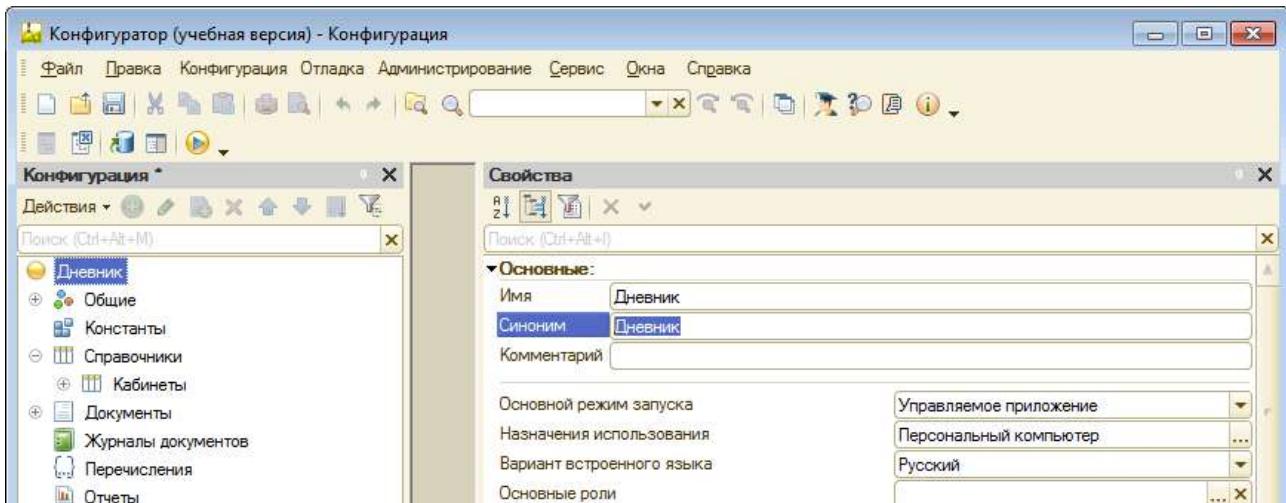


Рисунок 2.63. Имя конфигурации — «Дневник»

Рядом с названием конфигурации появилась звёздочка. Вы изменили конфигурацию.

Теперь не сохраняйте, не обновляйте конфигурацию, а просто нажмите кнопку *Начать отладку* (рисунок 2.64).

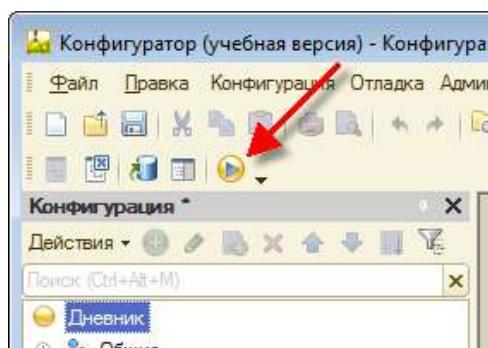


Рисунок 2.64. Начать отладку

Платформа предложит вам обновить конфигурацию базы данных (рисунок 2.65). Согласитесь.

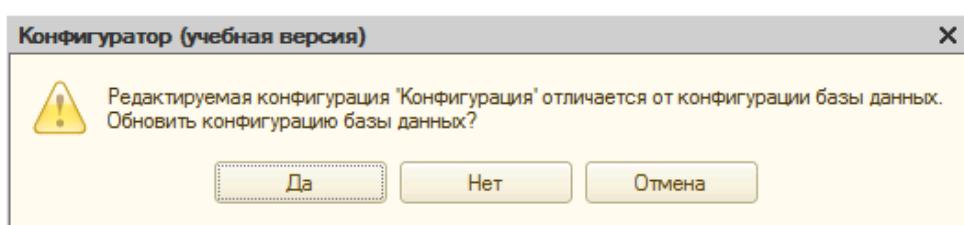


Рисунок 2.65. Обновить конфигурацию базы данных?

После этого она запустится в режиме 1С:Предприятие, и вы увидите своё прикладное решение (рисунок 2.66).

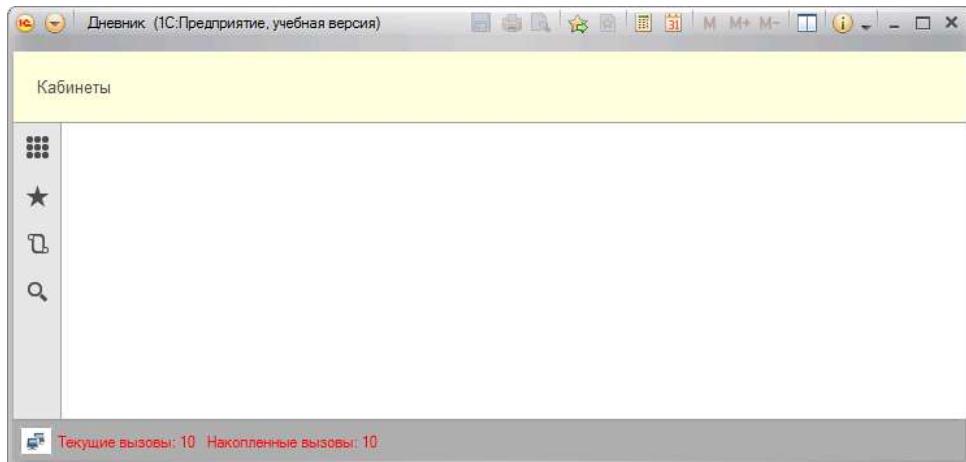


Рисунок 2.66. 1С:Предприятие в режиме отладки

Примечание

Именно этот способ запуска 1С:Предприятия вы будете использовать на протяжении всей книги. Он специально сделан для разработчиков. Потому что в процессе разработки постоянно приходится запускать 1С:Предприятие, чтобы посмотреть, проверить, как работают сделанные вами изменения.

2.9.2 Режим отладки

Вы изменили имя конфигурации. Где это видно?

Во-первых, в заголовке прикладного решения (рисунок 2.67).

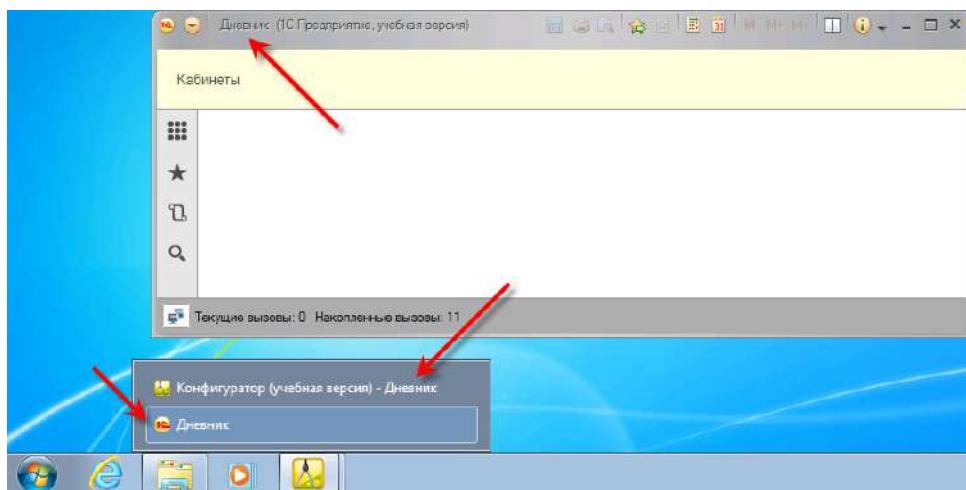


Рисунок 2.67. Название конфигурации

Во-вторых, в панели задач и у 1С:Предприятия, и у конфигуратора теперь указано название конфигурации. Поэтому, если у вас будет запущено несколько прикладных решений 1С:Предприятия, вы сможете отличить их друг от друга.

Что значит начать отладку? Что значит запустить 1С:Предприятие в режиме отладки? *Режим отладки* — это специальный режим 1С:Предприятия для разработчиков. Внешне он полностью похож на «обычный» режим 1С:Предприятие. Разница только в том, что в этом режиме прикладное решение не просто выполняется само по себе. Оно остаётся связано с конфигуратором. И если случится какая-то ошибка, то вы сможете легко перейти в конфигуратор и посмотреть, в каком именно месте вы ошиблись. Почему произошла эта ошибка. Как её исправить.

Кстати, если вы внимательны, то, наверное, обратили внимание, что в режиме отладки внизу появилась серая полоса, на которой написано: «Текущие вызовы... Накопленные вызовы...» Это как раз один из инструментов, позволяющих проследить за работой прикладного решения. Но в этой книге он вам не понадобится.

Поэтому сейчас закройте 1С:Предприятие и измените в конфигураторе пару настроек, для того чтобы в режиме отладки 1С:Предприятие у вас выглядело так же, как и у пользователей. И чтобы вам было удобно выполнять будущие задания по программированию.

Перейдите в конфигуратор и выполните команду *Сервис — Параметры* (рисунок 2.68).

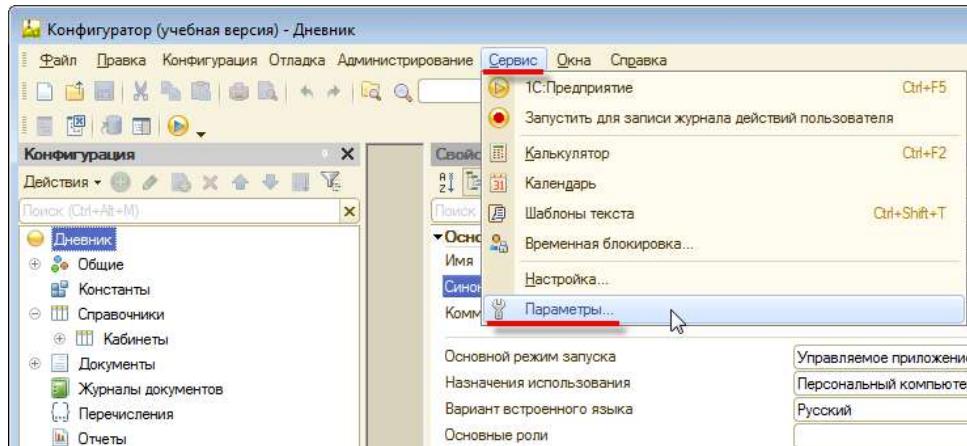


Рисунок 2.68. Сервис — Параметры

Откроется окно настройки параметров конфигуратора (рисунок 2.69).

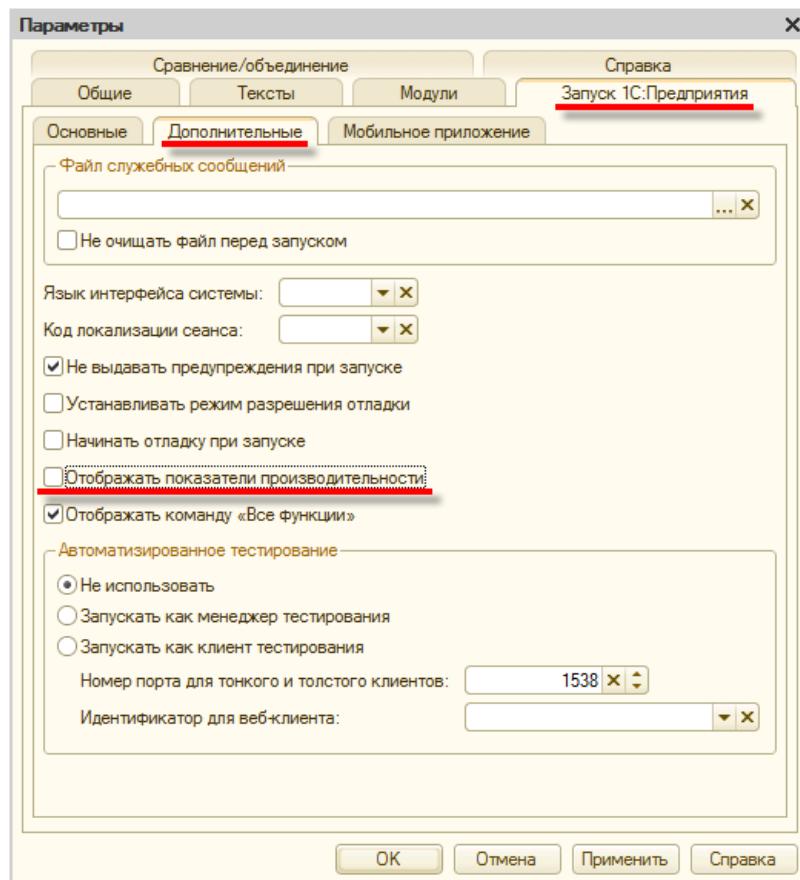


Рисунок 2.69. Параметры запуска

Здесь перейдите на закладку Запуск 1С:Предприятия.

Потом на закладку Дополнительные.

И затем снимите флажок Отображать показатели производительности.

Это вы избавились от серой полосы внизу прикладного решения, на которой написано про количество текущих и накопленных вызовов (рисунок 2.66).

Теперь сделайте ещё одну настройку. Она пригодится вам в дальнейшем, когда вы начнёте писать команды на встроенным языке.

Перейдите на закладку Модули.

Затем на закладку Редактирование (рисунок 2.70).

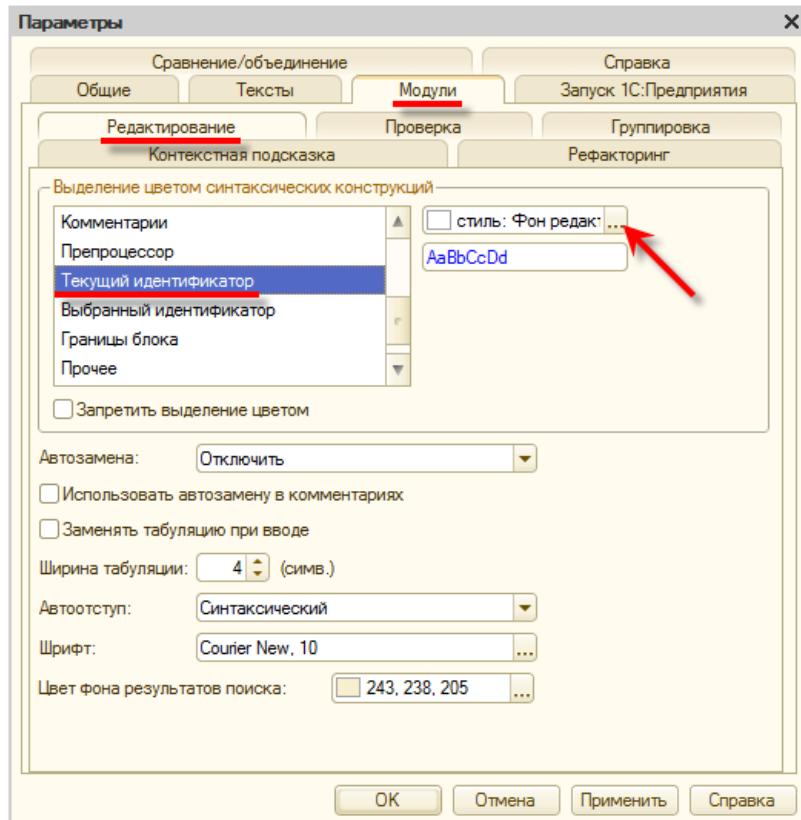


Рисунок 2.70. Параметры редактирования модулей

Прокрутите список и выделите в нём строку Текущий идентификатор.

В поле справа, в котором сейчас написано стиль: Фон редактирования, нажмите кнопку выбора.

Откроется диалог выбора цвета (рисунок 2.71).

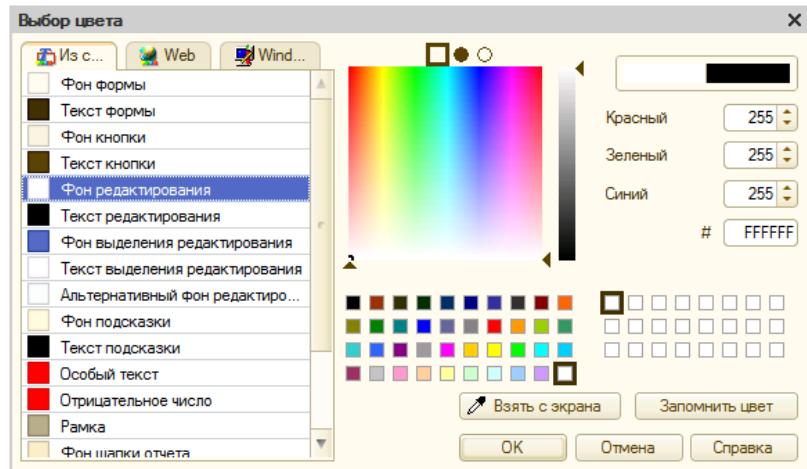


Рисунок 2.71. Диалог выбора цвета

В этом диалоге перейдите на закладку *Web*, нажмите мышью на любую строку списка и наберите на клавиатуре «жел» (рисунок 2.72).

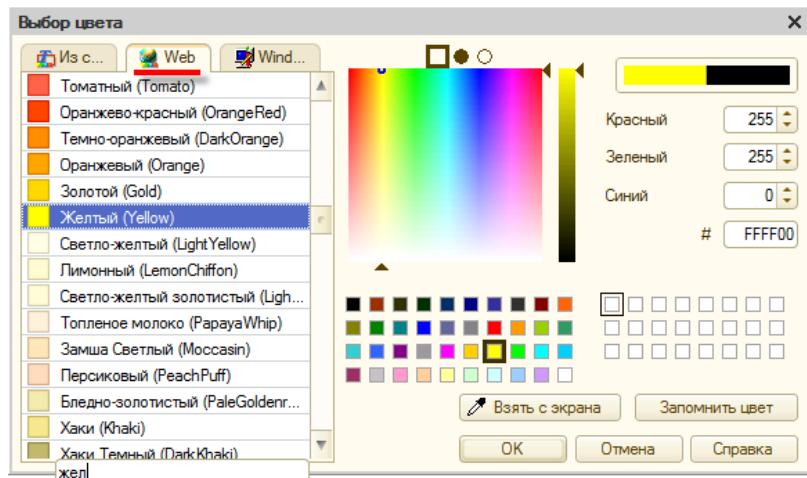


Рисунок 2.72. Выбор жёлтого цвета

В списке будет выбран жёлтый цвет. Нажмите *OK*.

В окне параметров (рисунок 2.70) тоже нажмите *OK*.

Теперь у вас всё готово для того, чтобы продолжить разработку прикладного решения.

2.9.3 Добавление данных

Итак, что у вас уже есть? Вы добавили объект конфигурации — справочник *Кабинеты*. Вы видели, что он появился в режиме 1С:Предприятие.

Теперь посмотрите, как с этим справочником работают пользователи. Как они добавляют в него данные.

Сейчас вы добавите один кабинет. А потом я расскажу вам о том, «что это было» и «как вы это сделали».

Запустите конфигурацию в режиме отладки. Вы увидите такую картинку (рисунок 2.73):

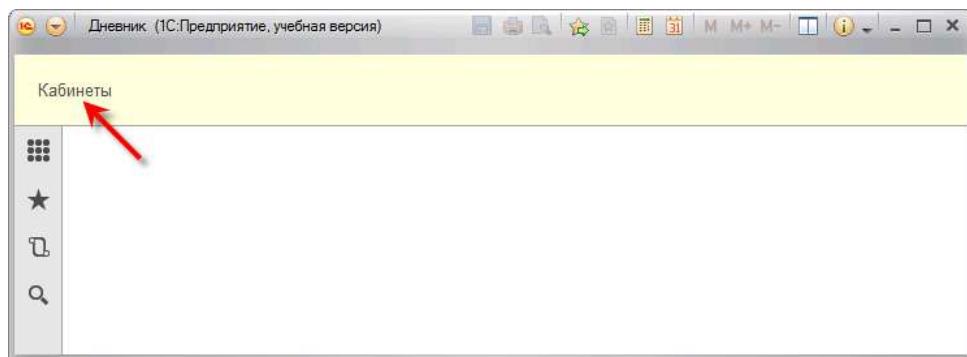


Рисунок 2.73. Команда «Кабинеты»

Нажмите на надпись *Кабинеты*, которая находится на жёлтой полосе. На месте белого прямоугольника появится такая картинка (рисунок 2.74).

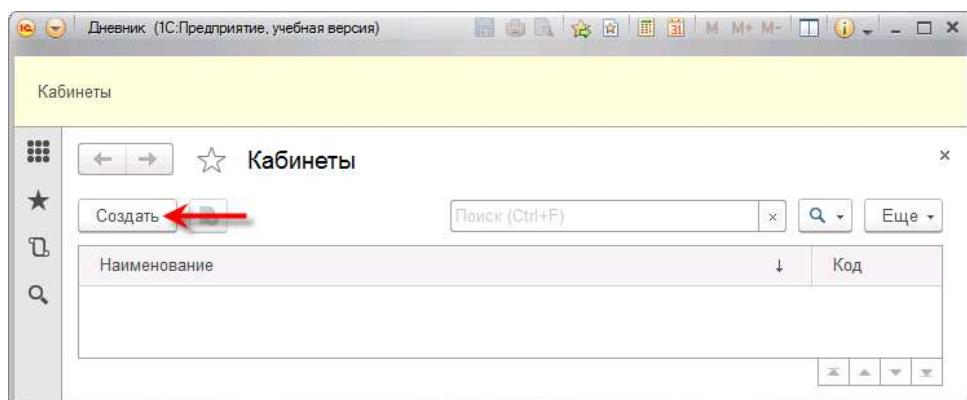


Рисунок 2.74. Результат выполнения команды «Кабинеты»

Нажмите на кнопку *Создать*. Откроется новое окно (рисунок 2.75).

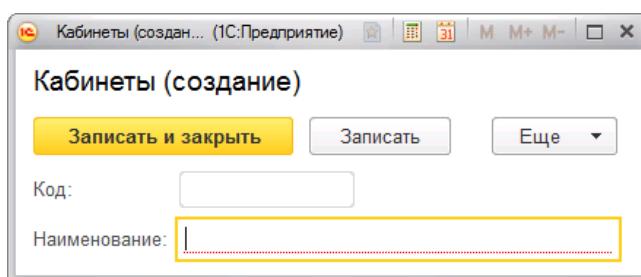


Рисунок 2.75. Окно «Кабинеты (создание)»

Тут в поле *Наименование* напишите номер кабинета. Например, 101.

Нажмите кнопку *Записать и закрыть*. У вас получится так (рисунок 2.76).

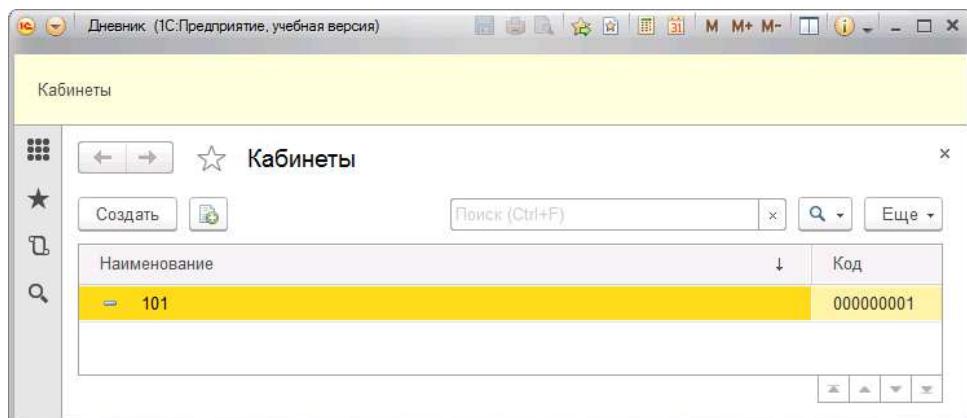


Рисунок 2.76. Кабинет 101

Всё. Вы создали новый кабинет.

Он сохранился в программе. Об этом вас проинформировало всплывающее окно, которое появилось (а затем исчезло) в правом нижнем углу экрана (рисунок 2.77).

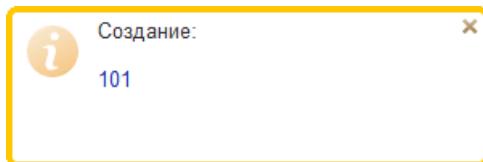


Рисунок 2.77. Всплывающее окно

2.9.4 Объект данных

Может быть, вы не заметили, но вы написали свою первую программу. Создали своё первое прикладное решение в системе 1С:Предприятие! Поздравляю!

Это действительно так. Посмотрите. У вас есть программа. Её можно запустить. Вы можете записать в эту программу какие-то данные. Эти данные сохраняются в компьютере. Когда вы в следующий раз включите компьютер и запустите программу, вы опять увидите свои данные.

А что вы сделали? Да, в общем-то, ничего! Добавили объект конфигурации. Создали кабинет с номером 101. И всё.

На самом деле всё остальное за вас сделала платформа 1С:Предприятия. Раз в конфигурации есть справочник, который вы назвали *Кабинеты*, то платформа знает, что делать дальше.

В режиме 1С:Предприятие она сама нарисует нужные надписи, кнопки. Сама будет открывать разные окна, списки. Платформа сама знает, как выглядят ваши данные, которые вы собираетесь добавлять в этот справочник. Она сама знает, куда и в каком виде их нужно записать.

То есть вы ещё не написали ни одной строчки программы, а у вас уже получилась работающая программа. Как же это называется?

А называется это *визуальное конструирование*. В системе 1С:Предприятие многие вещи разрабатываются в конфигураторе только «с помощью мышки». Без написания каких-то команд на каком-то языке программирования.

Теперь внимательно посмотрите, что произошло, когда вы создали в режиме 1С:Предприятие кабинет с номером 101. Это важный момент, и вы должны хорошо его понимать.

Когда вы были в конфигураторе, вы добавили справочник *Кабинеты*. Вы знаете, что эта вещь, которую вы добавили, называется *объект конфигурации*.

Когда вы были в пользовательском режиме, вы добавили кабинет с номером 101. Так вот, эту вещь тоже называют объектом. *Объектом данных*.

Вообще, в 1С:Предприятии очень любят слово «объект». Оно часто используется. И оно обозначает разные «вещи». Поэтому, чтобы не запутаться в разных «объектах», с самого начала запомните две картинки. Они пригодятся вам в будущем.

Итак. Помните, вы представляли конфигурацию как ствол, а данные — как фрукты? Тогда то, что у вас есть сейчас, выглядит следующим образом (рисунок 2.78).

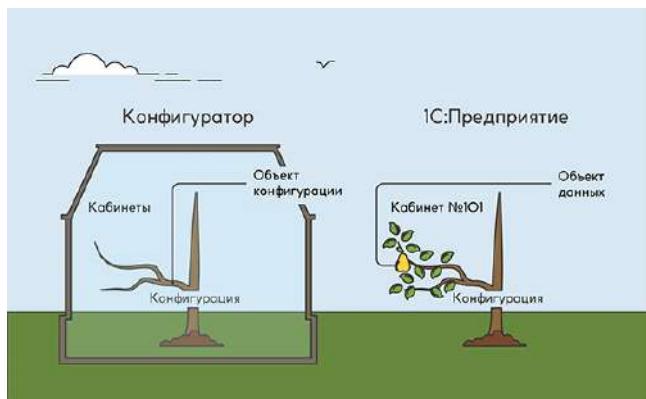


Рисунок 2.78. Объект конфигурации и объект данных

В конфигурации у вас есть ветка *Кабинеты*. Объект конфигурации *Кабинеты*.

А в пользовательском режиме вы добавили фрукт *101*. Объект данных *101*.

Теперь вспомните, что вы представляли конфигурацию как дом, а данные — как людей. Тогда ваша картинка будет выглядеть так (рисунок 2.79).

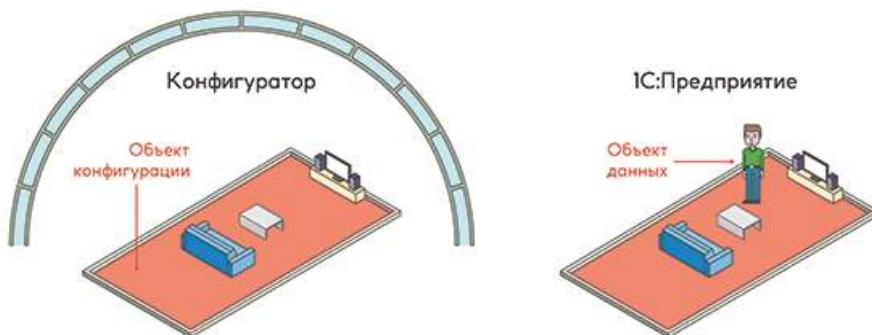


Рисунок 2.79. Объект конфигурации и объект данных

В конфигурации объектом у вас является комната. Она называется *Кабинеты*. А в 1С: Предприятии объектом у вас является человек, которого вы поместили в эту комнату. Человечек с именем *101*.

2.9.5 Объект конфигурации описывает, как будут выглядеть его данные

А теперь вспомните одно интересное наблюдение. Обычно от назначения комнаты зависит то, как одеты люди, которые в ней находятся. Например, люди, находящиеся в

спальне, скорее всего одеты в пижамы. А люди, которые находятся в гостиной, скорее всего одеты в платья, брюки и рубашки.

То же самое и в 1С:Предприятии. Объект конфигурации содержит описание того, как выглядят данные, которые в нём будут храниться. Измените кое-что прямо сейчас в объекте конфигурации, и вы увидите, как и где это повлияет на кабинет 101, который вы создали в режиме 1С:Предприятие.

Закройте 1С:Предприятие. Перейдите в конфигуратор.

Раскройте окно конфигуратора на весь экран. Обычно с ним работают именно так. Для этого нажмите кнопку, которая находится рядом с кнопкой закрытия окна (рисунок 2.80).



Рисунок 2.80. Раскрыть на весь экран

Если палитра свойств у вас очень широкая, помните, что её можно двигать за левую границу.

Откройте окно редактирования справочника Кабинеты. Когда вы добавляли справочник, это окно открылось само. А чтобы открыть его для того объекта, который уже есть в дереве конфигурации, дважды щёлкните мышью на этом объекте. То есть на справочнике Кабинеты.

В окне редактирования вы увидите четыре поля (рисунок 2.81).

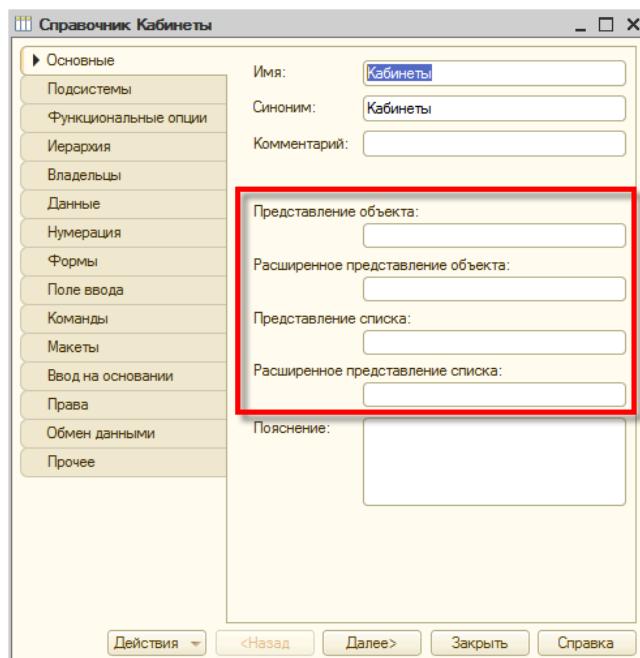


Рисунок 2.81. Представления объектов

Они говорят про какие-то объекты. Так вот, это как раз и имеются в виду объекты данных. Те объекты, которые будут храниться в этом справочнике. То есть это про тот

кабинет № 101, который вы добавили в пользовательском режиме. И про все другие кабинеты, которые вы потом добавите.

Во всех четырёх случаях в названии есть слово *Представление*. Что это такое?

Представление — это то, что вы увидите на экране.

Чем вы занимаетесь с самого начала книги? Вы воображаете, **представляете** себе разные картинки. Как выглядит это. Как выглядит то. То есть для вас картинка — это представление чего-то. Компьютер тоже в принципе может нарисовать на экране картинку. Но если он всё будет представлять вам только в виде картинок, то на экране не хватит места. Да и запутаться очень легко в разных картинках.

Поэтому компьютер **представляет** что-то в виде надписи, текста.

Значит, что вы напишете в этих полях? Вы напишете какие-то слова, которые в режиме 1С:Предприятие будут обозначать, представлять вам объекты данных из этого справочника. Либо один объект, либо несколько объектов. Про несколько объектов — это в тех полях, где есть слово «список».

Сейчас вы напишите без объяснений, а в следующем разделе вы поймёте, почему надо было написать именно так.

В поле *Представление объекта* напишите *Кабинет*. Потому, что он один.

В поле *Расширенное представление объекта* напишите *Учебный кабинет*.

В поле *Представление списка* напишите *Кабинеты*. Потому, что их много.

В поле *Расширенное представление списка* напишите *Учебные кабинеты* (рисунок 2.82).

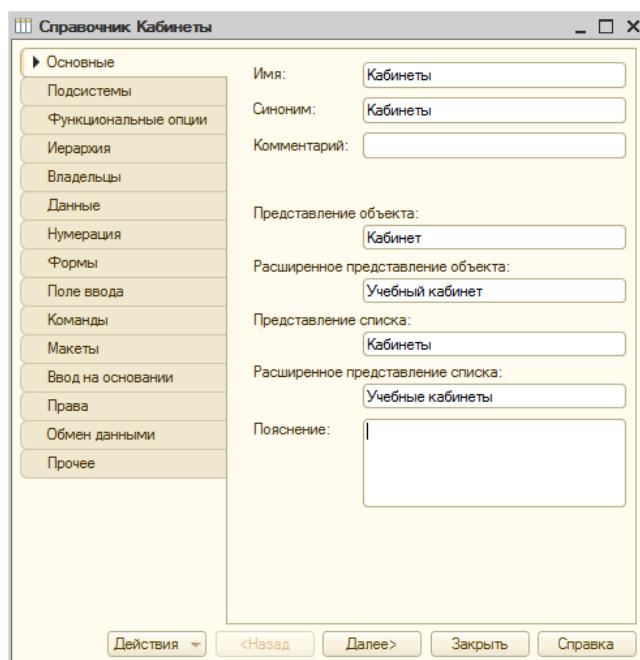


Рисунок 2.82. Заполненные представления объектов

И ещё одно изменение сделайте, чтобы все эти представления были вам видны в следующем разделе. Смысл этого изменения я тоже очень скоро вам расскажу.

В корне конфигурации откройте контекстное меню и выполните команду *Открыть командный интерфейс основного раздела* (рисунок 2.83).

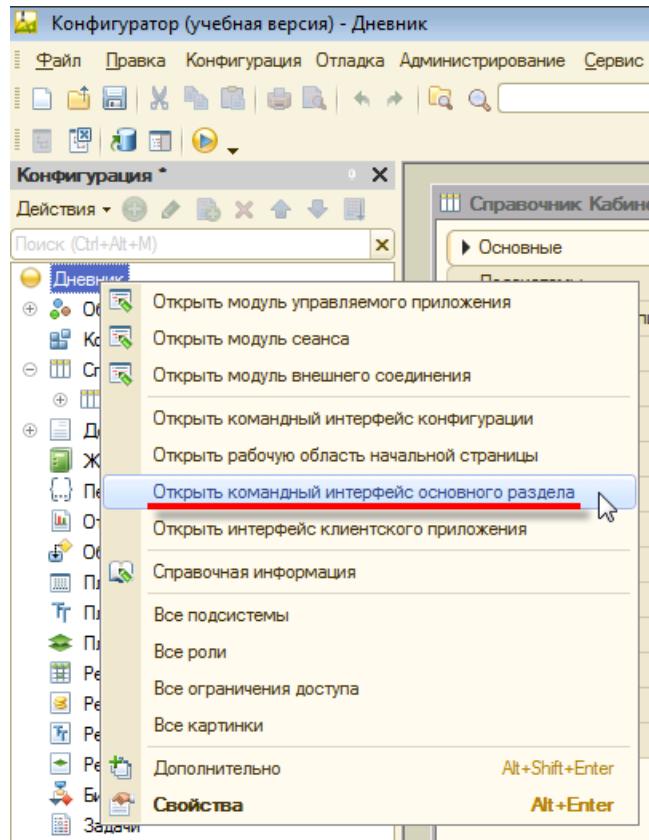


Рисунок 2.83. Открыть командный интерфейс основного раздела

Вы увидите на экране один из редакторов командного интерфейса (рисунок 2.84).

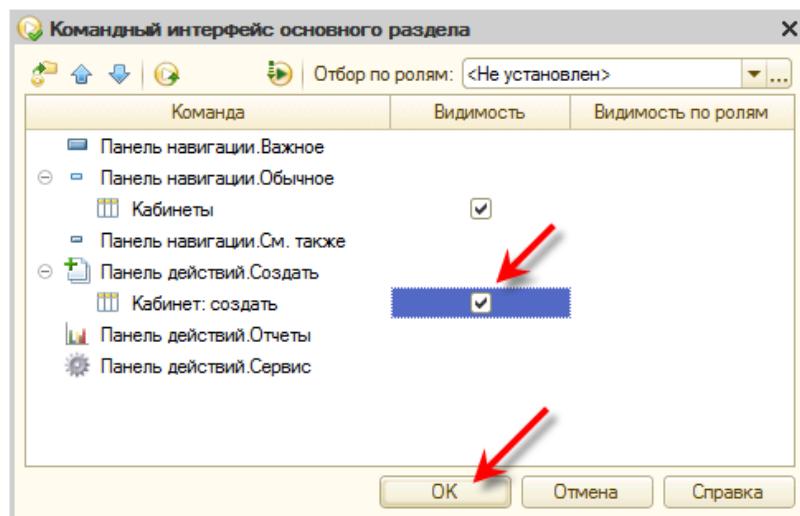


Рисунок 2.84. Редактор командного интерфейса

В нём установите флажок напротив *Кабинет: создать*.

После этого нажмите *OK*.

Запустите конфигурацию в режиме отладки.

Подробнее

Подробнее вы можете прочитать про редактор командного интерфейса основного раздела в документации «Руководство разработчика 8.3. Раздел 27.5. Редактор командного интерфейса основного раздела».

2.9.6 Интерфейс

Продолжим обсуждать тот факт, что в режиме 1С:Предприятие вы добавили кабинет № 101. Посмотрите ещё раз, как это происходило.

Сейчас ваше прикладное решение должно выглядеть так (рисунок 2.85).

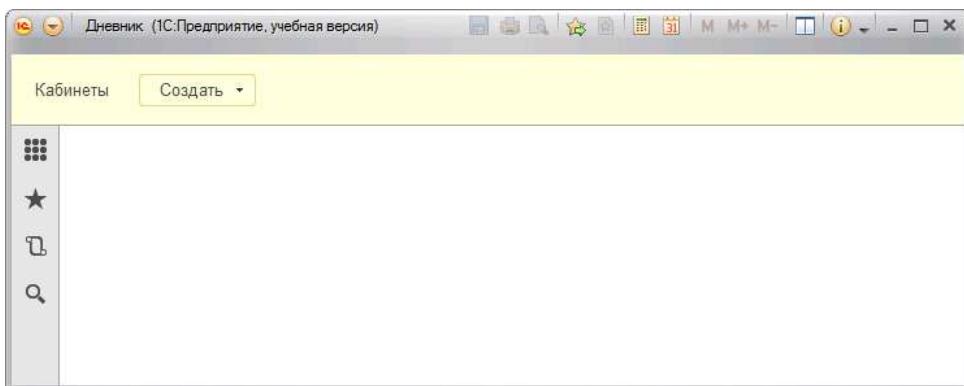


Рисунок 2.85. Прикладное решение

Вы наверняка заметили, что появилась кнопка *Создать*, которой раньше не было. Это всё потому, что вы изменили командный интерфейс. Но давайте по порядку.

Раз уж я сказал слово *интерфейс*, нужно объяснить, что это такое. Программисты любят слово «интерфейс» и применяют его к разным вещам. Это слово никогда не употребляется само по себе. Но применительно к любой вещи оно всегда обозначает одно и то же.

Включайте воображение.

У вас есть интерфейс. Это то, как вы выглядите. Какая на вас рубашка, брюки, платье. Какой язык вы понимаете. Какие просьбы можете выполнить. Это то, что вы можете сказать в ответ. Всё вместе это ваш интерфейс.

У телефона есть интерфейс. Это то, как он выглядит. Что он может показать на экране. Это кнопки, которые можно на нём нажать, и он что-то сделает в ответ. Или у него нет кнопок, а нажимать надо прямо на экран. Всё вместе это интерфейс телефона.

У собаки есть интерфейс. Это то, как она выглядит. То, какие команды человека она понимает (если понимает). Это то, на каком языке с ней общаются другие собаки. Всё вместе это интерфейс собаки.

У дома есть интерфейс. Это то, как он выглядит снаружи. Какие в нём окна. Какие на окнах шторы видно с улицы. Как в нём с улицы открывается дверь. Нужно потянуть за ручку, нужно набрать код, или, может быть, она сама откроется при вашем приближении. Всё вместе это интерфейс дома.

Что общего в этих четырёх примерах? Всегда есть что-то, с чем хочется взаимодействовать (вы, телефон, собака, дом). Интерфейс — это все те возможности взаимодействия с чем-то, которые имеются у этого чего-то «снаружи».

Ключевое слово здесь «снаружи». Когда говорят об интерфейсе чего-то, подразумевают, что внутреннее устройство этого чего-то неизвестно и не интересует.

Программа живёт в компьютере. Внутри компьютера есть другие программы. Снаружи компьютера есть пользователь (рисунок 2.86).

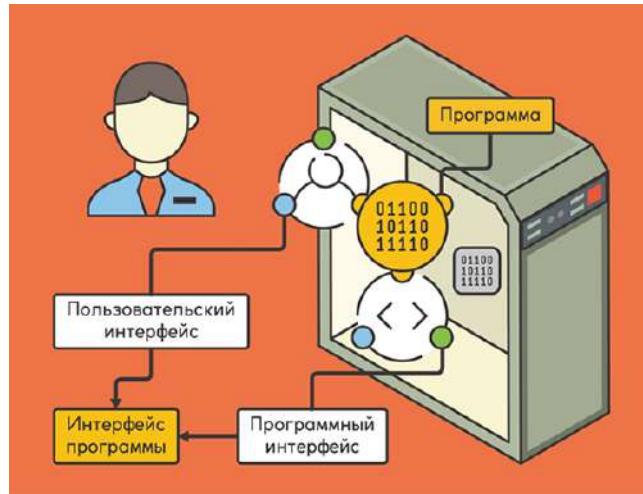


Рисунок 2.86. Интерфейс программы

То, что есть у программы, чтобы с ней могли взаимодействовать другие программы, называется *программный интерфейс*. То, что есть у программы, чтобы с ней мог взаимодействовать пользователь, называется *пользовательский интерфейс*.

Все вместе они составляют *интерфейс программы* вообще.

То, что вы видите сейчас на экране перед собой или на рисунке 2.85, — это пользовательский интерфейс вашего прикладного решения.

В нём есть места, на которые можно нажать мышкой, и что-то произойдёт. Например, вы уже нажимали на надпись *Кабинеты*. И есть места, на которые сколько ни нажимай, ничего не произойдёт. Например, куда-нибудь на жёлтую полоску, где нет надписей.

Теперь представьте и мысленно соберите вместе все места, на которые можно нажать, и что-то произойдёт. Представили? Вот всё это вместе называется *командный интерфейс* прикладного решения. То есть это часть пользовательского интерфейса, которая состоит только из одних команд.

Команда — это действие, которое может выполнить программа по инициативе пользователя. Например, команда *Кабинеты*. Когда вы нажимали на эту команду, программа что-то показывала вам на экране.

Теперь наконец-то я могу рассказать вам «правильными словами» о том, что вы делали, когда создавали комнату с номером 101. Какие команды вы выполняли.

Как устроен пользовательский интерфейс?

Пользовательский интерфейс любого прикладного решения 1С:Предприятия похож на тетрадку с несколькими страницами. Каждая страница этой тетрадки называется *раздел* (рисунок 2.87).



Рисунок 2.87. Разделы прикладного решения

Одна страница в этой тетрадке есть всегда. Это первая страница. Она называется *основной раздел*. Когда вы запускаете прикладное решение, оно всегда открывается на этой странице — на основном разделе (рисунок 2.88).



Рисунок 2.88. Основной раздел

Другие страницы могут быть, а могут и не быть. Их добавляет разработчик.

Ваше прикладное решение небольшое. Поэтому всё, что вы будете делать в этой книге, поместится в основном разделе. То есть на первой странице тетрадки.

Все разделы устроены одинаково. Часть пространства по краям занимают панели (рисунок 2.89).

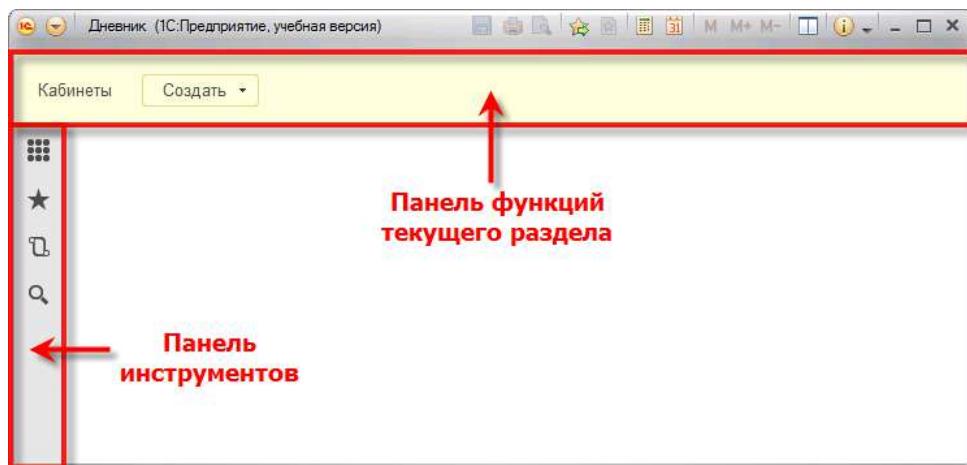


Рисунок 2.89. Панель инструментов и панель функций текущего раздела

В вашем случае слева находится *панель инструментов*. С её помощью вы можете посмотреть, например, самые важные для вас команды, которые вы сохранили в избранном. Можете посмотреть, когда и какие данные вы изменяли в программе. Можете найти что-то.

Сверху находится *панель функций текущего раздела*. В ней платформа показывает команды, с помощью которых вы можете работать с данными. Помните? С помощью команды *Кабинеты* вы добавили кабинет с номером 101.

Подробнее

И пользователь, и разработчик могут менять состав и расположение панелей. Вы этим заниматься не будете, но если вам интересно, то вы можете почитать об этом в документации и на сайте:

- «[Руководство пользователя. Интерфейс Такси 8.3. Раздел 8.1.6. Редактор панелей](#)».
- «[Редактор интерфейса клиентского приложения](#)».

Самая большая, центральная, часть раздела называется *рабочая область* (рисунок 2.90).

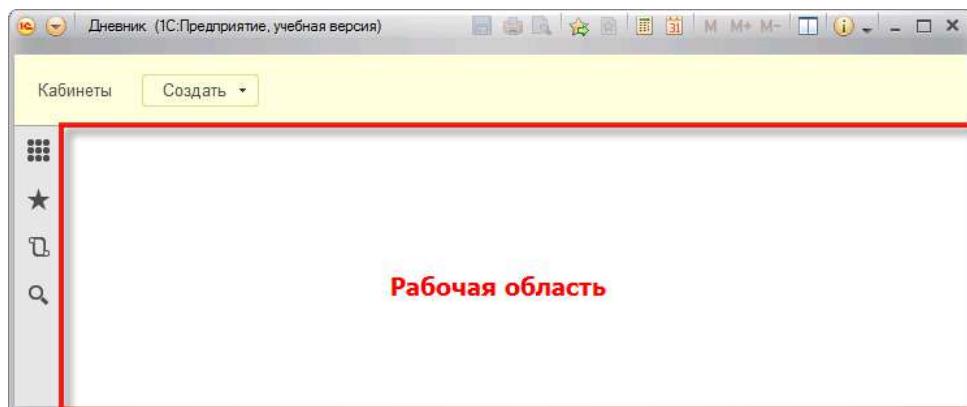


Рисунок 2.90. Рабочая область

Рабочая область — это основное место, в котором происходит вся работа с данными. В рабочей области платформа показывает *формы*.

Подробнее

Подробнее вы можете почитать про интерфейс в документации «[Руководство разработчика 8.3. Глава 3. Интерфейс приложения](#)».

2.9.7 Что такое формы?

Формы вы уже видели. Можете посмотреть на них ещё раз. Выполните команду *Кабинеты*. В рабочей области появится форма (рисунок 2.91).

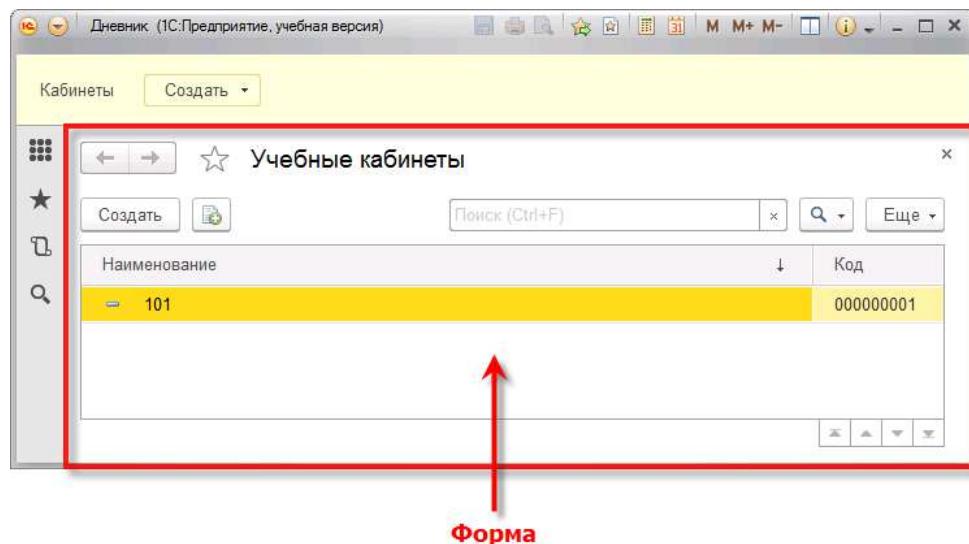


Рисунок 2.91. Форма в рабочей области

А теперь откройте элемент, который называется *101*. Чтобы это сделать, дважды щёлкните мышью на строке.

Откроется новое окно, и в нём тоже будет форма (рисунок 2.92).

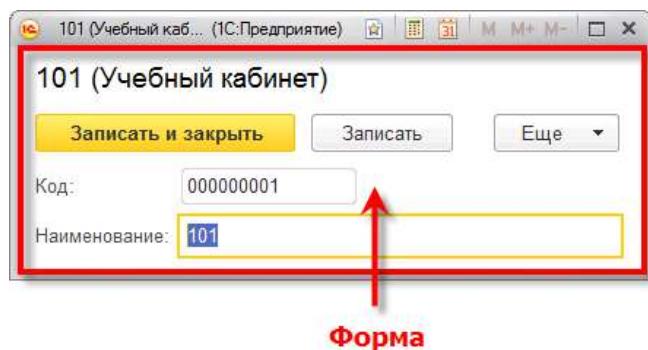


Рисунок 2.92. Форма в отдельном окне

Закройте эту форму.

Форма — это элемент пользовательского интерфейса. Она показывает данные в том виде, который удобен пользователю. Для чего удобен? Для того чтобы просмотреть, какие данные есть, изменить их, удалить или добавить новые данные.

Я уже говорил: после того как вы добавили в конфигурацию справочник Кабинеты, всё остальное платформа сделала за вас сама. В том числе, когда в режиме 1С:Предприятие вы выполняете разные команды, она сама автоматически создаёт все нужные формы. И вам их показывает. Такие формы называют *автогенерируемыми*.

Откуда платформа знает, какие нужны формы? Оказывается, для каждого объекта конфигурации уже заранее известно, какие формы могут потребоваться. Например, для справочника могут потребоваться всего 5 различных форм. Они полностью обеспечивают 99% всех ваших потребностей. И все эти формы платформа умеет создавать сама, без вашего участия. Но если вдруг вы захотите чего-то особенного, тогда вы можете создать собственную форму и пользоваться ей.

Не все 5 форм справочника требуются одинаково часто. Активно, практически всегда, используются только две формы. Остальные используются гораздо реже.

Обе эти формы вы как раз и видели. Одна из них называется *форма списка* (рисунок 2.91), а другая — *форма объекта* (рисунок 2.92). Для чего они нужны?

Это очень легко понять, если представить, что справочник — это школьный класс. Все ученики, которые есть в классе. А каждый отдельный ученик — это элемент справочника. Объект данных, как вы называли его раньше.

Тогда форма списка — это групповая фотография класса. А форма объекта — это индивидуальное фото, на котором только один ученик (рисунок 2.93).



Рисунок 2.93. Форма списка и форма объекта

В чём разница между этими двумя фотографиями?

Групповая фотография нужна, чтобы посмотреть на всех учеников. При этом вам не будут видны все детали про каждого ученика. Какие у него ботинки или брюки. Потому что его закрывает стоящий перед ним ученик. Но главное, что вы, глядя на эту фотографию, точно можете отличить одного ученика от другого. Потому что вы видите лицо каждого из них.

А на индивидуальной фотографии ученика вы видите абсолютно все детали. Как он одет, какая у него обувь, и так далее.

То же самое и с формами. Форма списка может не показывать абсолютно всю информацию про каждый из элементов. Но зато она позволяет просмотреть все элементы и точно отличить один элемент от другого.

Форма объекта, напротив, показывает всю информацию, которая известна про объект, и позволяет эту информацию изменять.

Посмотрите на своё прикладное решение. В панели функций текущего раздела сейчас есть две команды. Одна — это команда *Кабинеты*. Она открывает форму списка справочника *Кабинеты*. Для каждого справочника платформа автоматически создаёт одну такую команду и помещает её в командный интерфейс (рисунок 2.94).

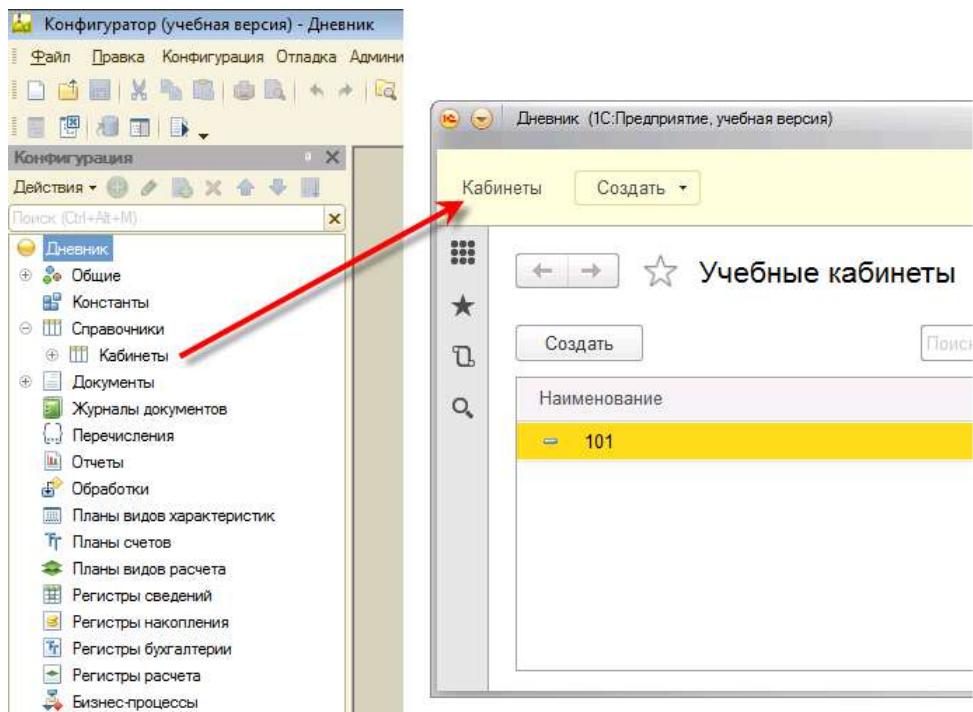


Рисунок 2.94. Команда перехода к списку

Потом, когда вы в форме списка нажимаете *Создать* или дважды щёлкаете мышью по существующему элементу, платформа открывает форму объекта. Для того чтобы вы могли создать новый объект. Или для того чтобы вы могли изменить данные уже существующего элемента справочника.

Всё это вы помните. Всё это вы делали, когда создавали кабинет с номером 101.

Получается, чтобы добавить новый элемент, нужна такая последовательность действий:

- нажать *Кабинеты* и открыть форму списка;
- нажать *Создать* и открыть форму нового элемента;
- ввести данные и нажать *Записать* и закрыть.

Не очень удобно. Особенно неудобно это тогда, когда новые элементы вам нужно добавлять часто и их много.

Чтобы посмотреть другой вариант, *закройте форму* списка. Для этого нажмите на крестик в правом верхнем углу формы (рисунок 2.95).

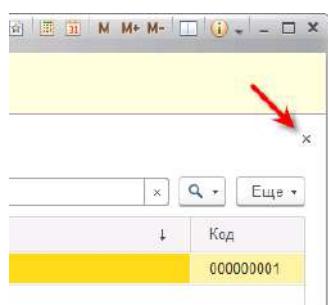


Рисунок 2.95. Закрыть форму

Теперь прикладное решение выглядит так, как будто вы его только что запустили.

Так вот. У справочника есть ещё одна команда. Она называется *открыть форму нового элемента*. Платформа сама, автоматически, не помещает эту команду в интерфейс. Но вы можете сделать это самостоятельно. И вы это уже сделали (рисунок 2.84).

Именно поэтому в вашей панели функций текущего раздела появилась кнопка *Создать*. Если вы нажмёте на неё, то увидите команду *Кабинет* (рисунок 2.96).

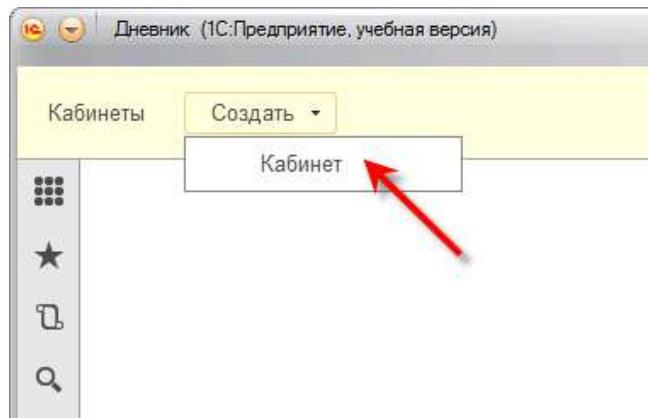


Рисунок 2.96. Команда «Кабинет»

Можете нажать на неё, и откроется форма нового элемента справочника. Ничего не меняйте и просто закройте эту форму.

2.9.8 Представления объекта конфигурации в интерфейсе

Теперь самое время вспомнить, что некоторое время тому назад вы в конфигураторе задавали разные представления для справочника *Кабинеты* (рисунок 2.82). Теперь вы можете увидеть, для чего они нужны.

Начнём по порядку.

Представление объекта показывает, как будет выглядеть команда создания нового объекта (рисунок 2.97).

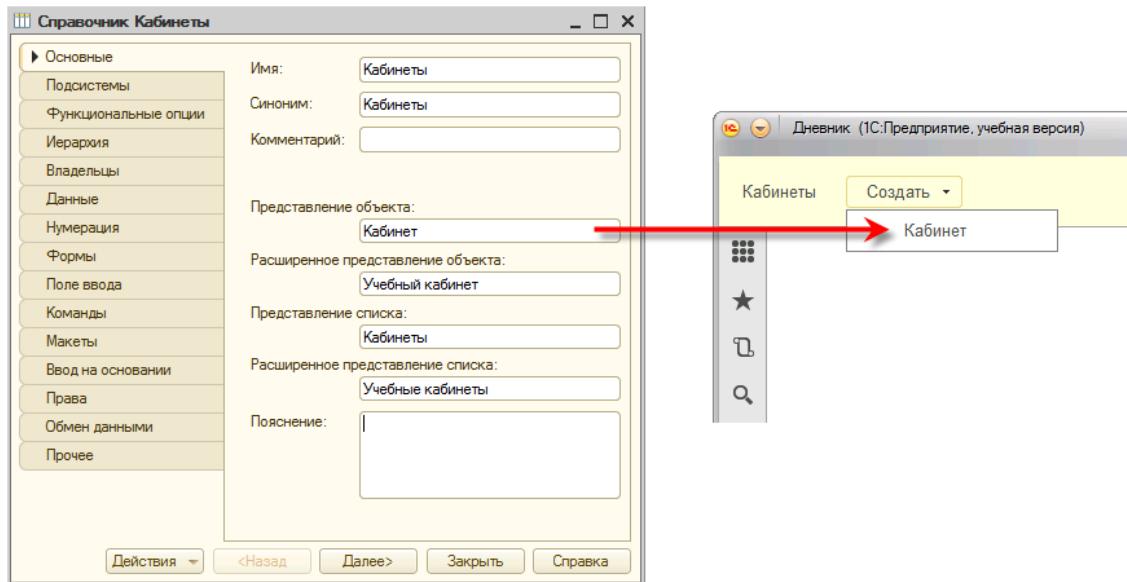


Рисунок 2.97. Представление объекта

Поэтому вы и писали её в единственном числе.

Если вы собираетесь использовать в интерфейсе команду создания нового объекта, то лучше задайте представление объекта. Ведь если какое-нибудь представление не задано, то платформа будет использовать синоним. И у вас получится *Создать: Кабинеты*. А это некрасиво. И неправильно.

Расширенное представление объекта показывает, что будет написано в заголовке формы объекта (рисунок 2.98).

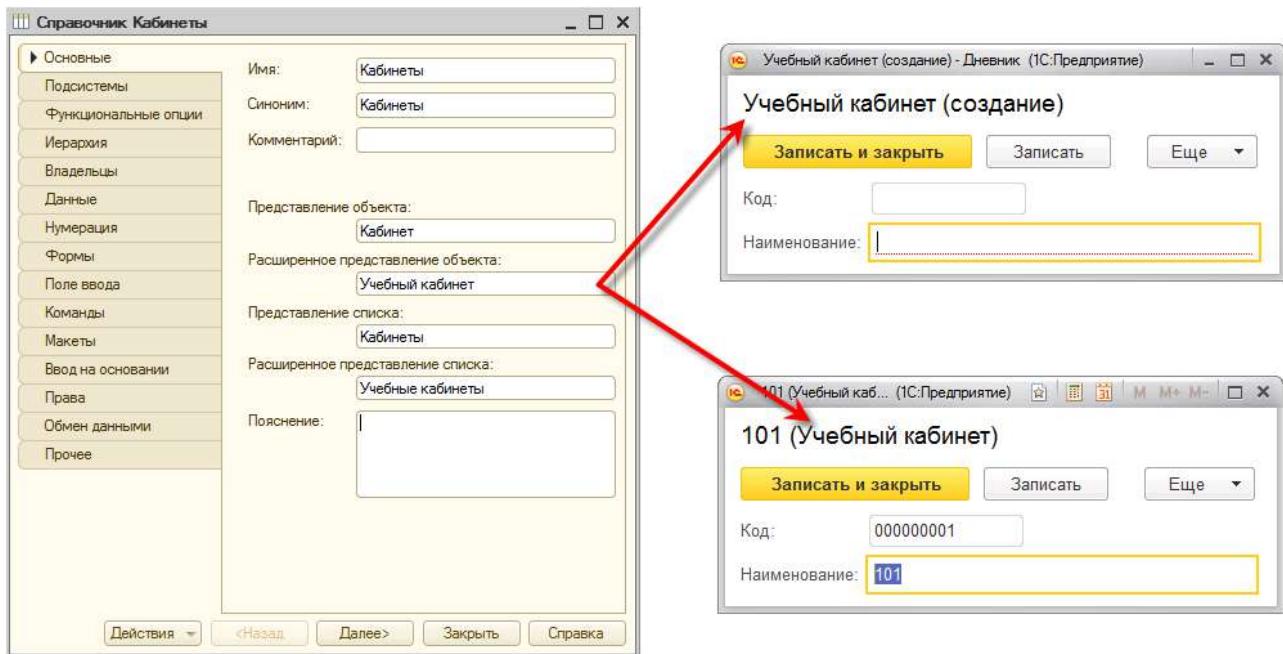


Рисунок 2.98. Расширенное представление объекта

Для справочников расширенное представление почти всегда задавать нужно. Потому что справочники вы называете во множественном числе, например *Кабинеты*. А в форме объекта речь идёт про один элемент этого справочника, то есть про кабинет. Значит, расширенное представление объекта вам нужно задать в единственном числе.

Можете задать его так, чтобы оно совпадало с представлением объекта: *Кабинет*. Можете задать его так, чтобы оно отличалось от представления объекта: *Учебный кабинет*. Это дело вкуса. Тут нужно учитывать только одну особенность.

В командном интерфейсе, там, где используется представление объекта, места мало. А в форме, там, где используется расширенное представление объекта, места много. Поэтому очень желательно, чтобы представление объекта было кратким. А вот расширенное представление может быть кратким, а может быть и не кратким (рисунок 2.99).

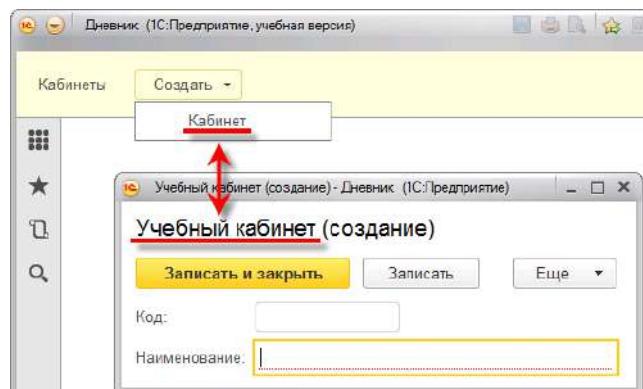


Рисунок 2.99. Представление объекта и расширенное представление объекта

Представление списка показывает, как будет выглядеть команда перехода к списку справочника (рисунок 2.100).

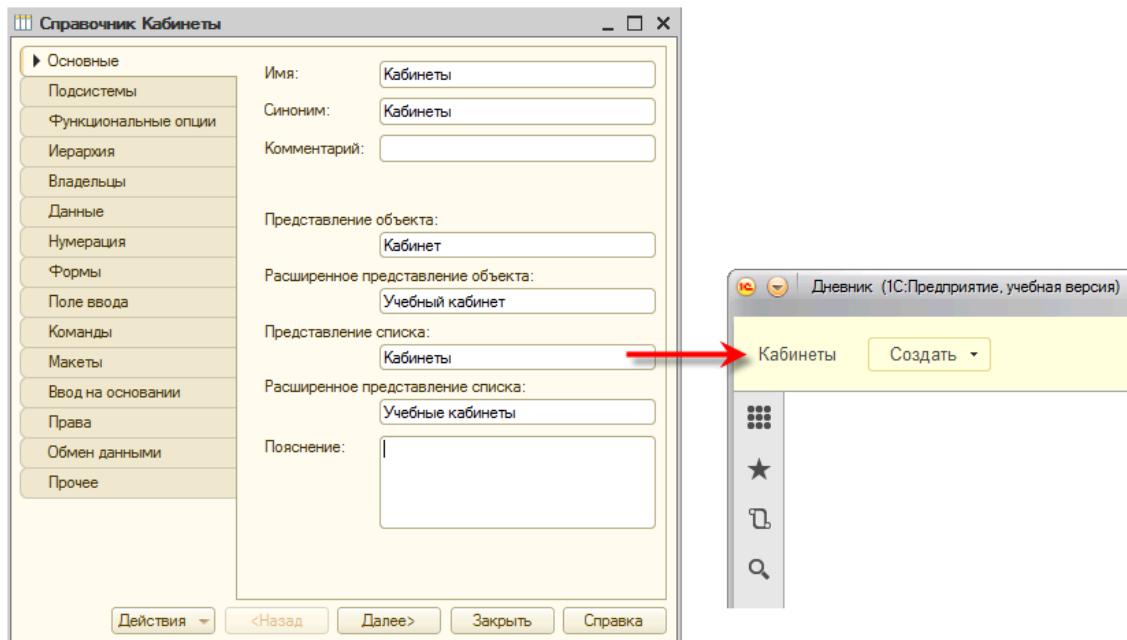


Рисунок 2.100. Представление списка

Для справочника вы можете её не задавать. Потому что справочники обычно называются во множественном числе и это хорошо подходит для команды списка.

И, наконец, *расширенное представление списка* показывает, что будет написано в заголовке формы списка (рисунок 2.101).

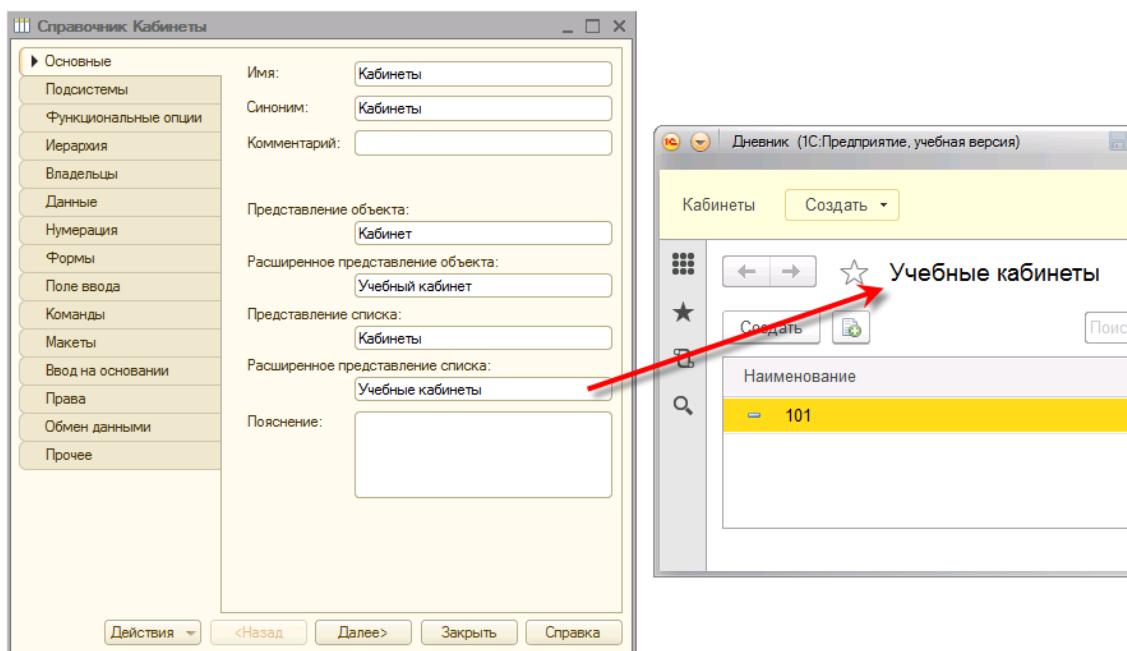


Рисунок 2.101. Расширенное представление списка

Здесь правило то же самое, что и с представлением объекта. Представление списка стремитесь делать коротким. А расширенное представление списка может быть и коротким, и длинным. Как хотите.

Задание 2.1

Для выполнения этих заданий добавьте новую информационную базу. Откройте её в конфигураторе.

Добавьте справочник *ЛюбимыеИгры*. В этом справочнике будут храниться ваши любимые компьютерные игры.

Задайте для этого справочника: представление объекта, расширенное представление объекта, представление списка и расширенное представление списка.

В командный интерфейс основного раздела добавьте команду создания элемента этого справочника. Запустите 1С:Предприятие в режиме отладки. Посмотрите все элементы интерфейса, в которых используются заданные вами представления.

Задание 2.2

В той же информационной базе, в которой вы выполняли задание 2.1, создайте ещё несколько справочников:

- **Одноклассники.** В этом справочнике будут храниться фамилии и имена всех, кто учится с вами в одном классе.
- **СловарьИностранныхСлов.** В этом справочнике будут храниться все иностранные слова, с которыми вам приходится сталкиваться. А также будет храниться их перевод на русский язык.
- **Ученики.** В этом справочнике будут храниться фамилии и имена всех, кто учится в вашей школе.

Для каждого из этих справочников задайте те представления, которые, по вашему мнению, нужны. Объясните, почему вы задали именно такое представление. Также объясните, почему вы не стали задавать представление.

Включите в интерфейсе команды создания элементов этого справочника. Запустите 1С:Предприятие в режиме отладки. Посмотрите, как выглядят ваши представления в интерфейсе приложения.

2.9.9 Наименование и код

Теперь я хочу показать вам одну интересную особенность. Для того чтобы вы её увидели, создайте ещё один кабинет. Но создайте его «модным» способом — с помощью команды *Создать* в панели функций текущего раздела (рисунок 2.102).

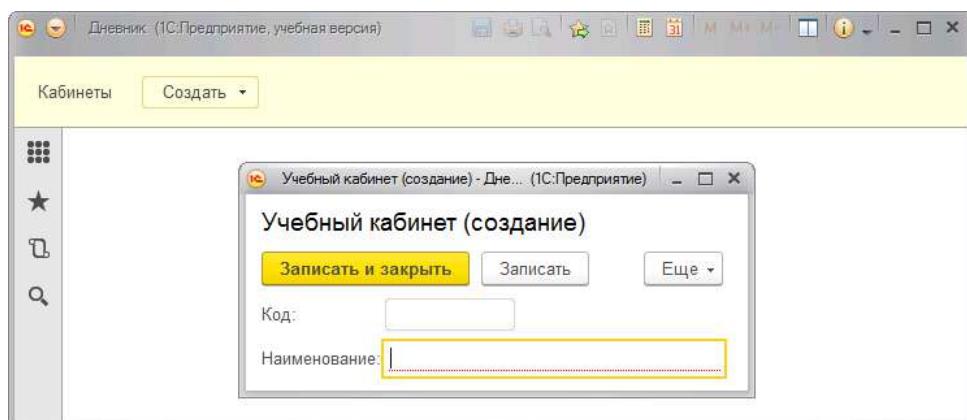


Рисунок 2.102. Добавление нового кабинета

Теперь задайте наименование 101.

Нажмите *Записать и закрыть*.

Откройте список всех кабинетов (рисунок 2.103).

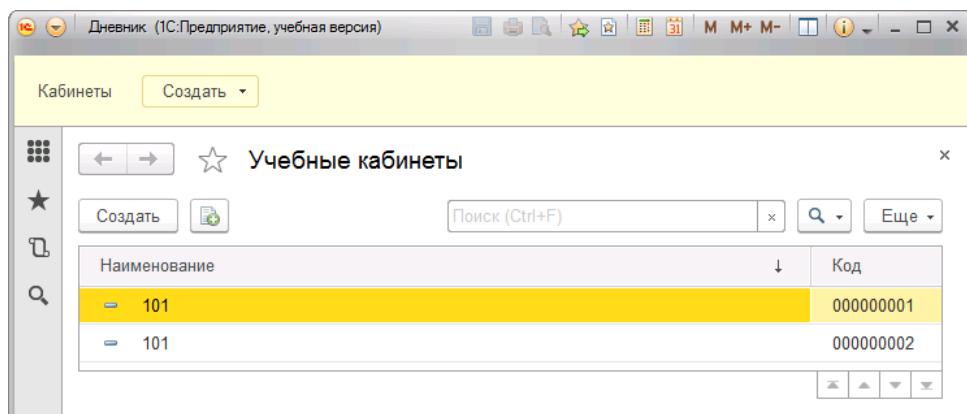


Рисунок 2.103. Два кабинета с одинаковым номером

У вас получилось два кабинета с одним и тем же номером 101. Это плохо. Так не может быть.

Но это две разные строчки. А чем они отличаются? Они отличаются колонкой *Код*. В одном случае там 000000001, а в другом — 000000002.

Когда вы создавали кабинеты, вы вводили только наименование. Код платформа представила сама. Может быть, вы неправильно используете эти поля? Может быть, 101 нужно было писать туда, где код? И вообще, зачем нужен *Код*? И что такое *Наименование*? Да-вайте разберёмся.

Вспомните эту картинку (рисунок 2.104).

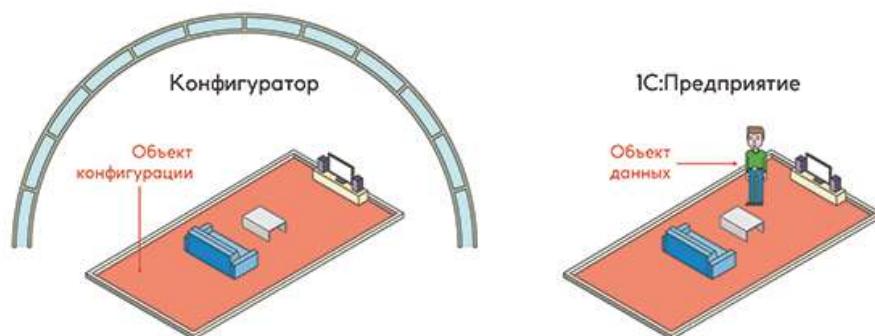


Рисунок 2.104. Объект конфигурации и объект данных

Я уже говорил, что объект конфигурации не просто описывает стены, в которых будут жить данные. Ещё он описывает то, как эти данные будут выглядеть. Он содержит описание того, что вы хотите знать о каждом из объектов данных.

Сейчас вы видели, что стандартно, если ничего не менять, про каждый из объектов данных платформа хочет знать его код и его наименование. Каждая такая вещь, «которую вы (или платформа) хотите знать про объект данных», описывается в объекте конфигурации с помощью *реквизита*. То есть у вашего объекта конфигурации *Кабинеты* сейчас есть два реквизита: *Код* и *Наименование*. Посмотрите на них.

Закройте сеанс 1C:Предприятие.

Перейдите в конфигуратор.

Раскройте ветку *Кабинеты* (рисунок 2.105).

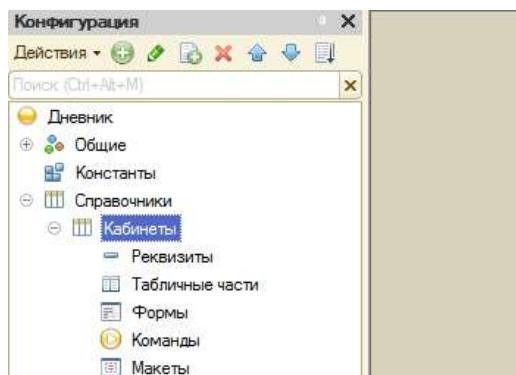


Рисунок 2.105. Реквизиты справочника

Вы видите ветку *Реквизиты*. Но эта ветка пуста. Где же *Код* и *Наименование*?

Дело в том, что в этой ветке будут находиться те реквизиты, которые вы добавите самостоятельно. А *Код* и *Наименование* вы не добавляли. Их платформа создала сама. В тот момент, когда вы добавляли справочник *Кабинеты*.

Такие реквизиты, которые платформа создаёт сама, стандартным образом, не спрашивая вас, называются *стандартные реквизиты*. Чтобы их увидеть, откройте контекстное меню у объекта конфигурации и выполните команду *Стандартные реквизиты* (рисунок 2.106).

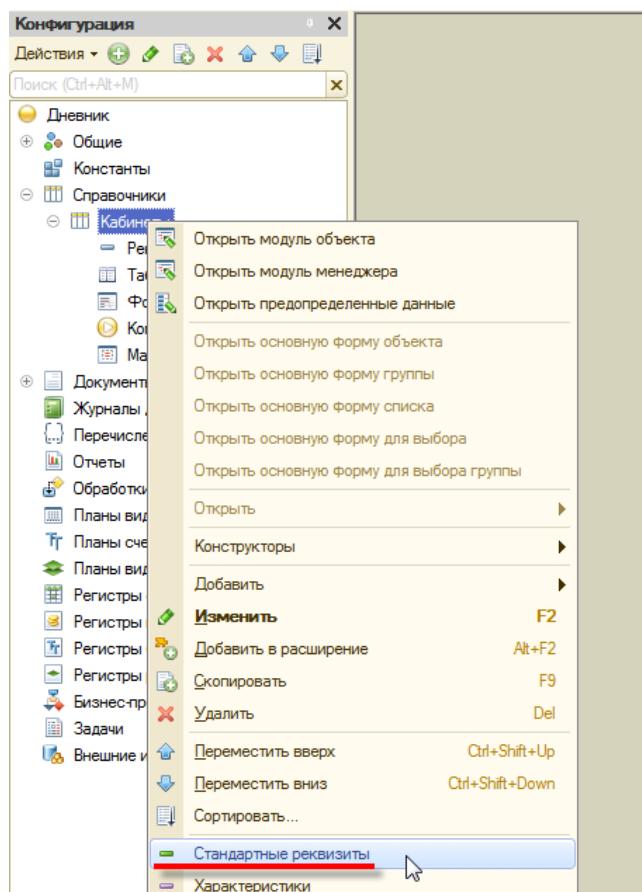


Рисунок 2.106. Команда «Стандартные реквизиты»

На экране, в отдельном окне, появятся все стандартные реквизиты, которые могут быть у справочника (рисунок 2.107).

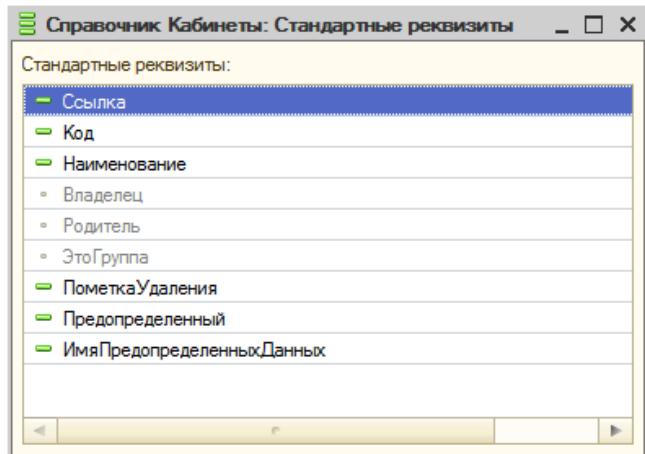


Рисунок 2.107. Стандартные реквизиты справочника

Тут их много. Но, когда вы создавали кабинеты, вы видели только *Код* и *Наименование*. А для чего остальные?

Во-первых, тут все реквизиты. Вообще все, которые могут быть у справочника. Это не значит, что все они есть у вашего справочника *Кабинеты*. Например, серым цветом обозначены реквизиты *Владелец*, *Родитель* и *ЭтоГруппа*. Серые они потому, что у вашего справочника *Кабинеты* их нет.

Во-вторых, не все реквизиты предназначены для того, чтобы пользователь менял их вручную. Но они могут понадобиться вам, разработчику, когда вы будете писать программу на встроенным языке 1С:Предприятия. Это такие реквизиты, как *Ссылка*, *ПометкаУдаления*, *Предопределенный* и *ИмяПредопределенныхДанных*.

Что остаётся? Остаются *Код* и *Наименование*, которые вы и видели.

Примечание

Благодаря тому, что у объекта конфигурации есть реквизит *Код* и реквизит *Наименование*, вы имеете возможность для каждого объекта данных (элемента справочника) указать код этого элемента и наименование этого элемента (рисунок 2.108).

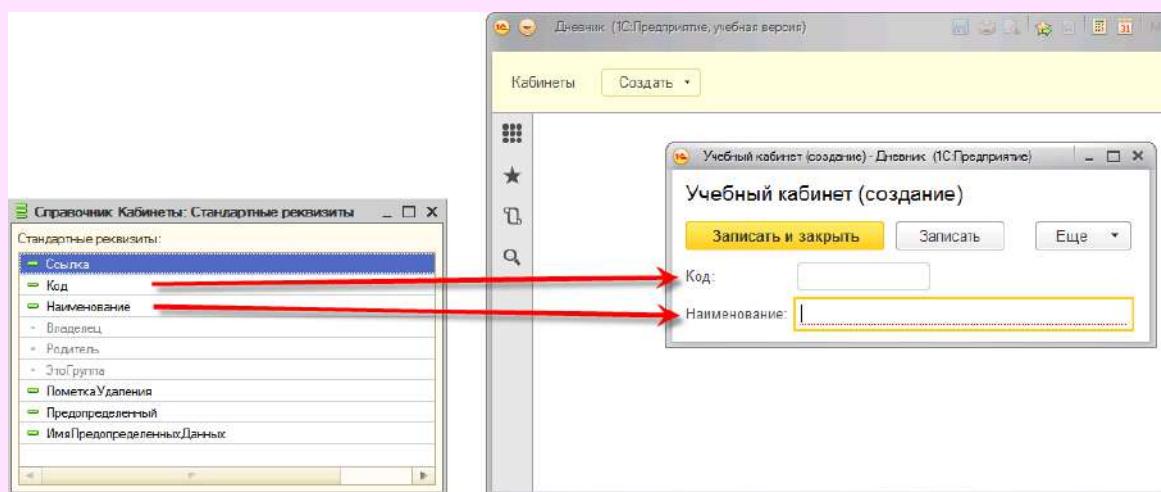


Рисунок 2.108. Реквизиты «Код» и «Наименование»

Очень хорошо. Теперь вы знаете, где живут стандартные реквизиты. Пока они вам больше не понадобятся, поэтому можете закрыть это окно.

Теперь посмотрите, как так получилось, что платформа сама придумала какие-то коды

для ваших кабинетов. В одном случае там 000000001, а в другом — 000000002.

Откройте окно редактирования справочника *Кабинеты*.

Перейдите на закладку *Нумерация* (рисунок 2.109).

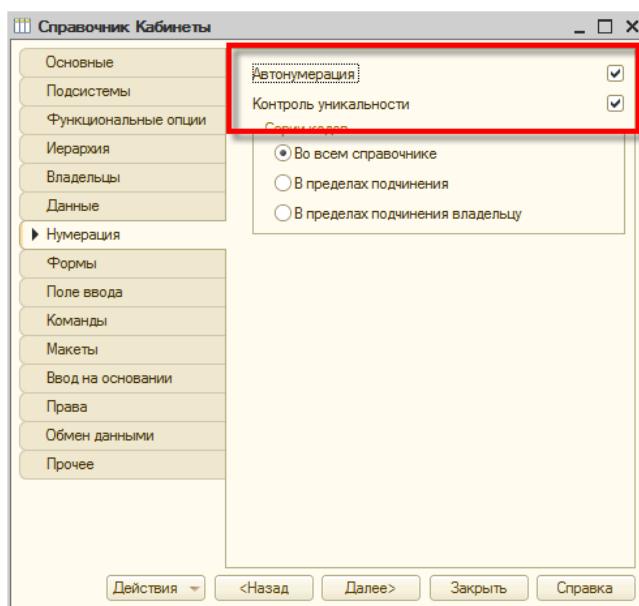


Рисунок 2.109. Закладка «Нумерация»

Тут вас будут интересовать два флажка: *Автонумерация* и *Контроль уникальности*. Оба эти флажка рассказывают о том, как будет использоваться реквизит *Код*.

Автонумерация означает, что платформа сама будет придумывать код — по порядку, друг за другом. И вам не нужно вводить его вручную (рисунок 2.110).

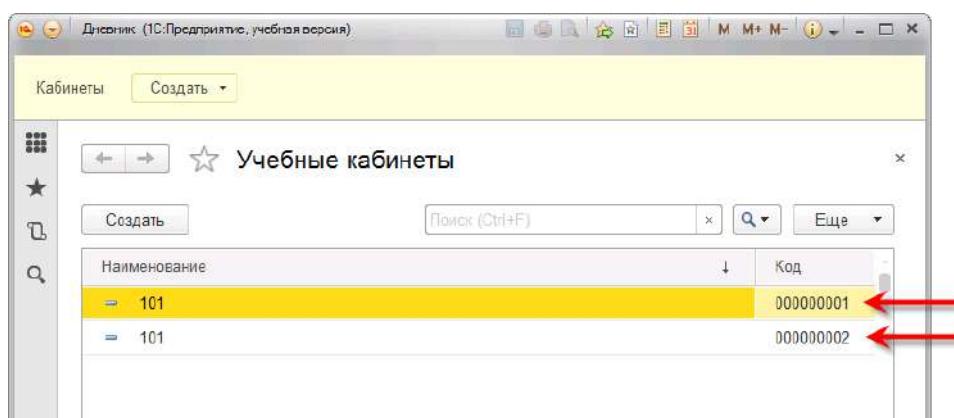


Рисунок 2.110. Код в режиме 1С:Предприятие

Контроль уникальности означает, что платформа будет следить за тем, чтобы коды не повторялись. Неважно, сама она их придумывает или вы их вводите. В любом случае она проследит за тем, чтобы не было двух одинаковых кодов.

Вот так устроен стандартный справочник. У каждого элемента есть наименование и код. Наименование может быть каким угодно. А код платформа придумывает сама, и он всё время разный.

То есть код — это что-то особенное, уникальное для каждого элемента. А наименование — это просто какое-то обозначение, которое может быть одинаковым у разных элементов.

На что это похоже в жизни?

Например, на список ваших друзей. Наименование — это имя друга. Может быть несколько друзей с одинаковыми именами. А что у них разное? А разные у них фамилии (таблица 2.1).

Таблица 2.1. Имя и фамилия

Наименование	Код
Юра	Никитин
Ваня	Трофимов
Юра	Пестов

Но может быть, что среди ваших друзей два Юры Никитина. Как быть тогда? Тогда лучше в качестве уникального признака, в качестве кода, использовать не фамилию, а, например, адрес (таблица 2.2).

Таблица 2.2. Имя, фамилия и адрес

Наименование	Код
Юра Никитин	г. Москва, ул. Тверская, д. 5
Ваня Трофимов	г. Москва, ул. Мытная, д. 2
Юра Пестов	г. Москва, ул. Полярная, д. 7
Юра Никитин	г. Минск, ул. Достоевского, д. 3

Итак, это стандартное поведение справочника.

А что нужно вам? Вам нужен просто список кабинетов. Каждый кабинет обозначается своим номером. И этот же номер отличает один кабинет от другого. Двух кабинетов с одинаковым номером быть не может.

Значит, номер кабинета — это уникальный признак. Код. А наименование вам вообще не нужно.

Что ещё не похоже на стандартное поведение справочника? Код вы хотите вводить самостоятельно. Вам не нужно, чтобы его придумывала платформа. Но при этом нужно, чтобы она следила за тем, чтобы номера кабинетов не повторялись.

Всё это вы сейчас и сделаете.

Вы не хотите, чтобы платформа сама придумывала код, значит, флажок Автонумерация нужно сбросить (рисунок 2.111).

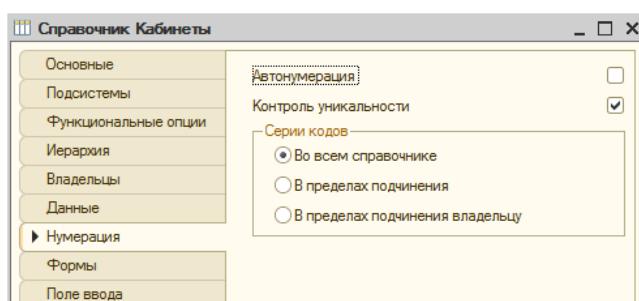


Рисунок 2.111. Сбросить свойство «Автонумерация»

А контроль уникальности нужно оставить, чтобы не было возможности ввести два одинаковых номера кабинета.

Дальше нужно избавиться от реквизита *Наименование*. Переходите на закладку *Данные* (рисунок 2.112).

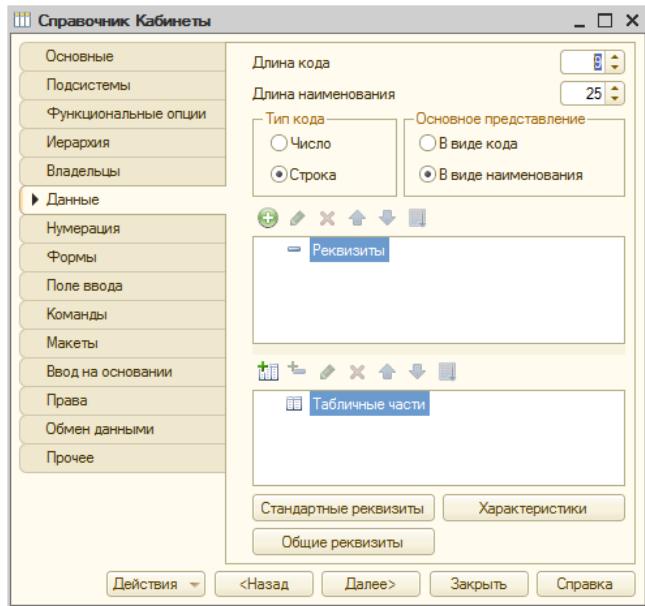


Рисунок 2.112. Закладка «Данные»

Чтобы избавиться от наименования, установите *длину наименования* равной 0.

Длина кода. Сейчас она 9 символов. Вам столько не нужно. Для номера кабинета обычно хватает трёх символов. Или четырёх. Установите 4.

Последнее, что вам понадобится изменить, — это *тип кода*. Он может быть или числом, или строкой. Число — это 101, 203 и так далее. А строка может понадобиться, если у вас есть кабинеты 101а или 203б. Для простоты будем считать, что таких кабинетов у вас нет. Поэтому установите тип кода *Число*.

В результате у вас должно получиться так (рисунок 2.113).

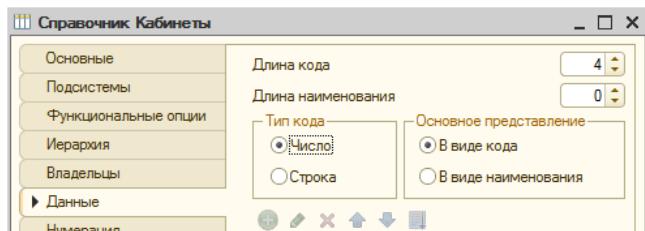


Рисунок 2.113. Длина наименования равна 0

Теперь, чтобы не было ошибок, нужно сделать одно изменение, которое платформа не умеет делать сама. Вы убрали наименование, но есть одно место, где оно осталось. Там его нужно убрать вручную.

Перейдите на закладку *Поле ввода*.

Нажмите кнопку выбора у поля *Ввод по строке* (рисунок 2.114).

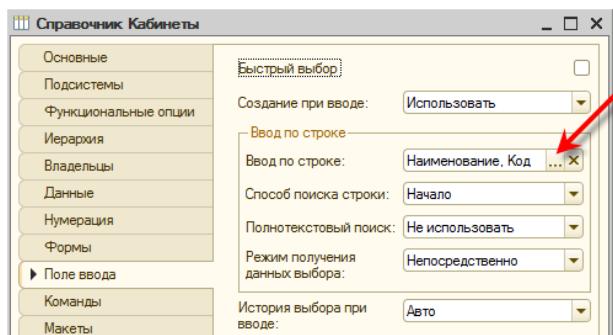


Рисунок 2.114. Ввод по строке

Здесь нажмите кнопку *Исключить* поле из списка, чтобы поле *Наименование* пропало. И осталось только поле *Код* (рисунок 2.115).

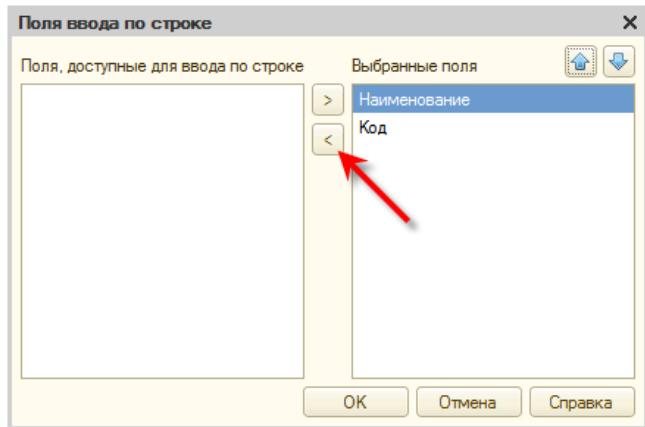


Рисунок 2.115. Поля ввода по строке

После этого нажмите *OK*.

В результате в окне редактирования справочника *Кабинеты* у вас должно быть так (рисунок 2.116).

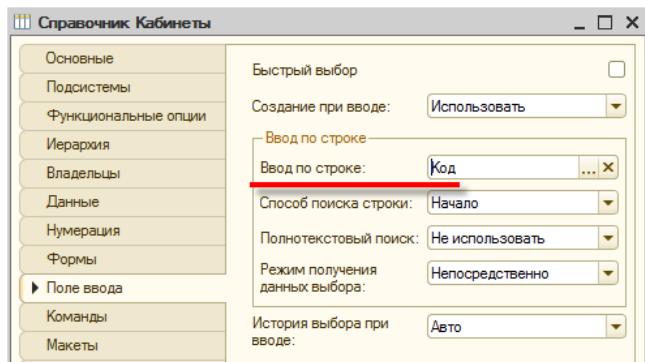


Рисунок 2.116. Ввод по строке — «Код»

Итак, наименование вы убрали совсем. Остался только код. Его вы и будете использовать. В режиме 1С:Предприятие, вы видели это, он так и будет называться — Код.

Но это некрасиво и непонятно. Ведь на самом деле это просто номер кабинета, а никакой не код. Вы можете это исправить!

Откройте стандартные реквизиты справочника *Кабинеты*. Кстати, обратите внимание, что реквизит *Наименование* тоже стал серым. Его нет теперь.

Откройте контекстное меню у реквизита *Код* (рисунок 2.117).

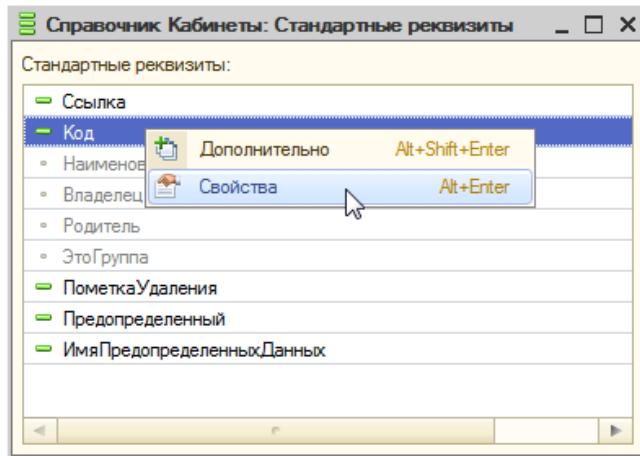


Рисунок 2.117. Контекстное меню реквизита «Код»

Выполните команду *Свойства*. Как вы, наверное, уже догадываетесь, откроется палитра свойств. В ней укажите Синоним для поля *Код* — *Номер кабинета* (рисунок 2.118).

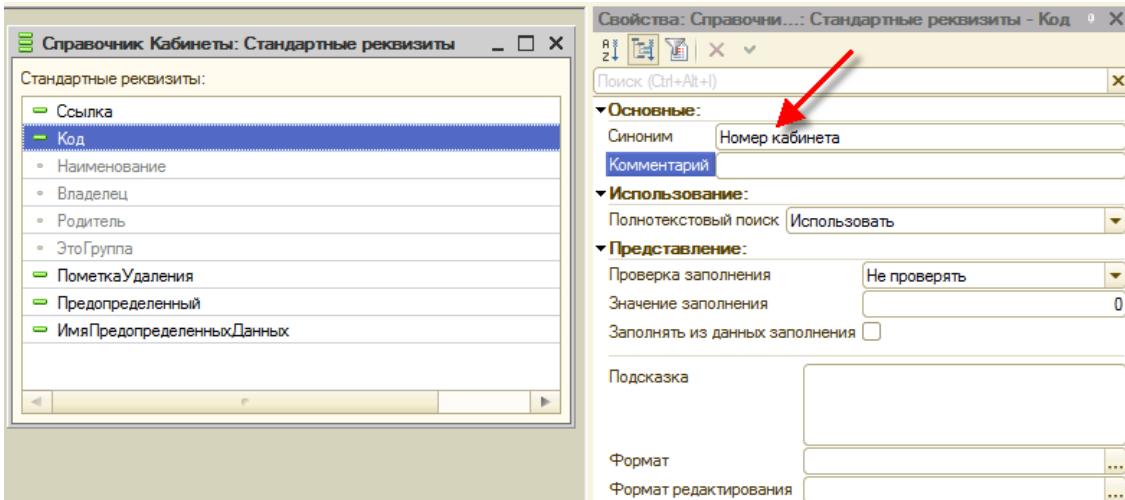


Рисунок 2.118. Синоним поля «Код»

Теперь, когда вы запустите 1С:Предприятие, везде вместо *Код* будет написано *Номер кабинета*.

Проверьте.

Запустите конфигурацию в режиме отладки. Откройте список справочника *Кабинеты*. Теперь вы увидите такую картинку (рисунок 2.119).

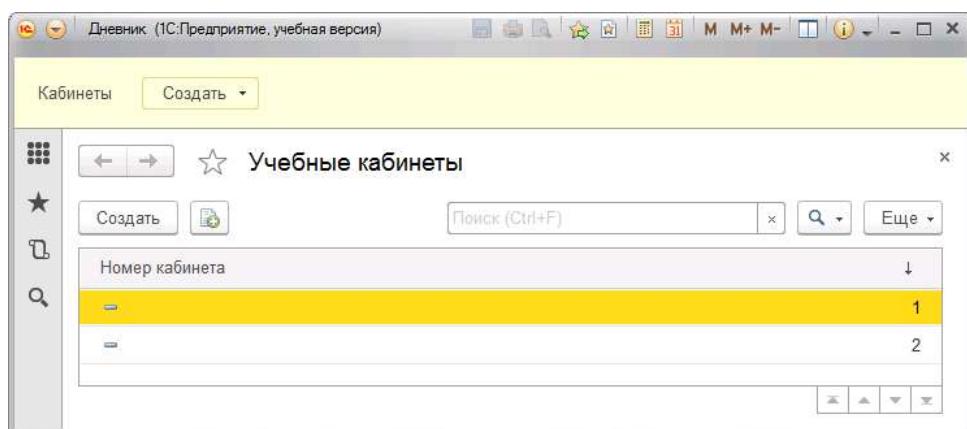


Рисунок 2.119. Список кабинетов

Что изменилось?

Во-первых, нет колонки *Наименование*.

Во-вторых, колонка *Код* теперь называется *Номер кабинета*.

В третьих, код раньше был строкой. Теперь он у вас число. Поэтому платформа преобразовала, как смогла, код 000000001 в число 1, а код 000000002 в число 2. Числа платформа всегда прижимает к правому краю, поэтому они оказались справа.

Но ваши кабинеты имеют другие номера. Измените их. Первому кабинету задайте номер 101, а второму — 127 (рисунок 2.120).

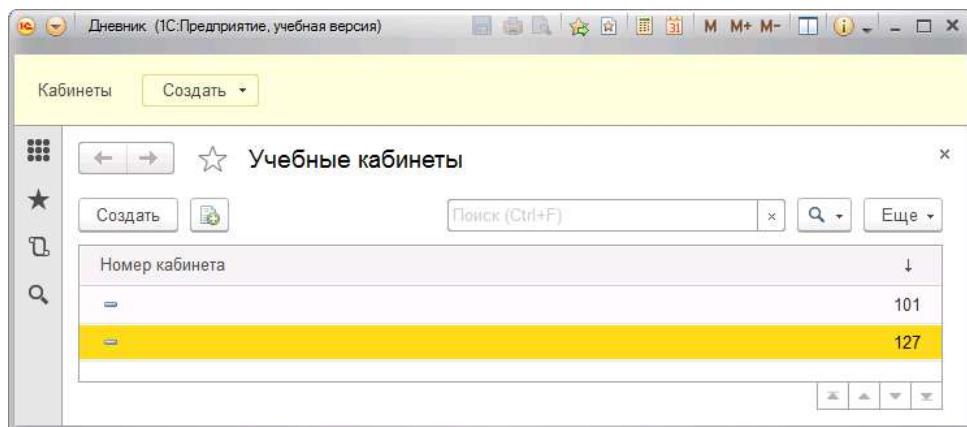


Рисунок 2.120. Новые номера кабинетов

А теперь попробуйте создать новый кабинет с тем же номером 101. Каким хотите способом. С помощью команды *Создать*, которая есть в форме списка. Или с помощью команды *Создать: Кабинет*, которая есть в панели функций текущего раздела.

Платформа не даст вам это сделать и сообщит об ошибке (рисунок 2.121).

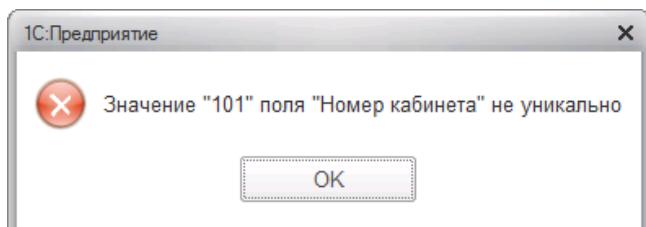


Рисунок 2.121. Кабинет 101 уже есть

Тогда напишите номер кабинета 131 и нажмите *Записать* и закрыть. Он вам тоже пригодится.

Ну что? Здорово?

Подробнее

Подробнее вы можете прочитать о свойствах справочника в документации «[Руководство разработчика 8.3. Раздел 5.8.2. Свойства справочника](#)».

Смотрите, сколько всего интересного вы уже сделали! Вы запрограммировали справочник таким образом, что он отслеживает уникальность номера кабинета, который вы создаёте. И не позволяет вам создавать кабинеты с теми номерами, которые уже есть.

А ведь при этом вы не написали ни одной строчки программы! Всё только с помощью визуального конструирования!

Надо ли создавать в этом справочнике все кабинеты, которые есть в школе? Нет, не надо. Ведь вам понадобятся только те кабинеты, в которых проходят ваши занятия. А как узнать, какие это кабинеты?

Оказывается, что специально узнавать это не нужно. Очень скоро вы увидите, что само собой получится так, что в вашем прикладном решении окажутся только те кабинеты, которые вам нужны.

2.10 Учителя

Вспомните, какие данные вы собирались хранить в программе:

- предметы,
- учителя,
- кабинеты,
- учебные дни.

Для хранения кабинетов вы создали справочник. Теперь придумайте, как хранить список учителей. Справочник подходит для этого?

Вспомните, какие признаки подсказывают вам, что нужно использовать справочник.

Нужен ли вам список учителей сам по себе? Важно ли вам знать, какие вообще есть учителя в школе? Нет. Значит, по первому признаку не подходит.

Надо ли указывать одного и того же учителя в разных местах программы? Да. Потому что рядом с названием урока вы хотите записывать и фамилию учителя, который проводит урок. И таких мест много, потому что один и тот же учитель может проводить занятия в разные дни. Более того, один и тот же учитель может вести разные предметы.

Поэтому для учителей нужно создать справочник. Вы уже достаточно хорошо ориентируетесь в конфигураторе, поэтому вполне можете сделать это самостоятельно.

Добавьте новый справочник и назовите его Учителя. Посмотрите и решите самостоятельно, какие нужно задать представления для этого справочника, чтобы команды в интерфейсе выглядели хорошо. Чтобы заголовки форм, которые будут открываться, тоже выглядели хорошо и правильно.

После того как вы это сделаете, можете посмотреть мой вариант (рисунок 2.122).

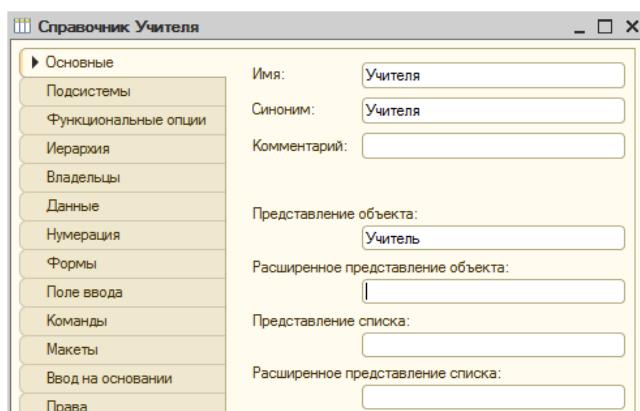


Рисунок 2.122. Справочник «Учителя»

Я добавил только представление объекта — Учитель. Чтобы в форме объекта был правильный заголовок (рисунок 2.123).

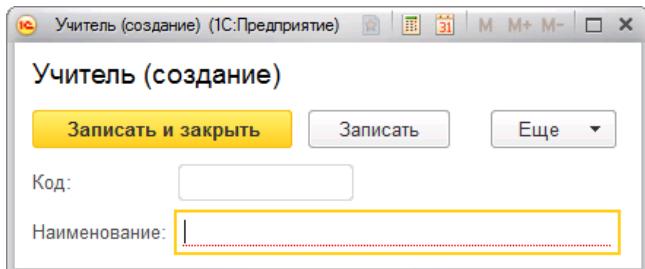


Рисунок 2.123. Форма объекта для справочника «Учителя»

Заметка

Тут вы можете возразить, что в заголовке формы ведь используется расширенное представление. Да, вы правы. Используется оно, если оно есть. А если его нет, то представление объекта. То есть принцип такой. В заголовке формы отображается:

- расширенное представление объекта;
- если оно не указано, то представление объекта;
- если и оно не указано, то синоним;
- если и синоним не указан, то имя объекта конфигурации.

В остальных местах интерфейса (в панели функций текущего раздела, в форме списка) подходит тот синоним, который есть у справочника, — Учителя. Поэтому каких-то особых представлений я не задавал.

Теперь нужно решить вопрос с кодом и наименованием. В наименование удобно записывать фамилию и инициалы учителя. А нужен ли код? Есть вероятность, что в школе окажутся два учителя с одинаковыми фамилией и инициалами? Наверное, совсем исключать это нельзя.

Значит, пусть поле *Код* остаётся. Тем более что платформа заполняет его сама. Если вдруг появятся два одинаковых учителя, вы сможете отличить их по коду.

Что ещё полезно сделать из того, что вы умеете?

Полезно посмотреть, какая длина наименования у справочника Учителя. Когда вы добавляли справочник, платформа установила стандартную длину наименования 25 символов. Этого хватит, чтобы записать фамилию и инициалы. Я так и буду делать. Но если вы хотите записывать имя и отчество полностью, тогда увеличьте длину наименования до 50 символов, например.

Полезно поменять синоним у стандартного реквизита *Наименование*. Ведь в это поле вы будете записывать фамилию и инициалы? Значит, задайте ему синоним *ФИО*.

Как всё это делать, вы знаете. Сделайте это самостоятельно.

В результате у вас должно получиться так (рисунок 2.124).

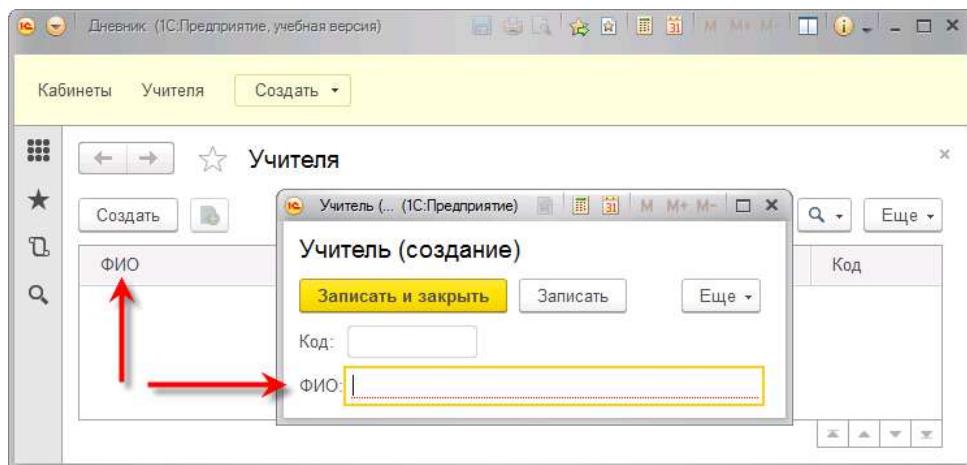


Рисунок 2.124. Синоним для поля «Наименование»

И в форме объекта, и в форме списка вместо *Наименование* должно появиться *ФИО*.

Теперь можно заполнить справочник данными. Всех учителей сейчас добавлять не надо. Вы сделаете это позже, совсем скоро. Сейчас добавьте только одного учителя, чтобы посмотреть, как это выглядит. Пусть это будет Евсеева Ольга Юрьевна (рисунок 2.125).

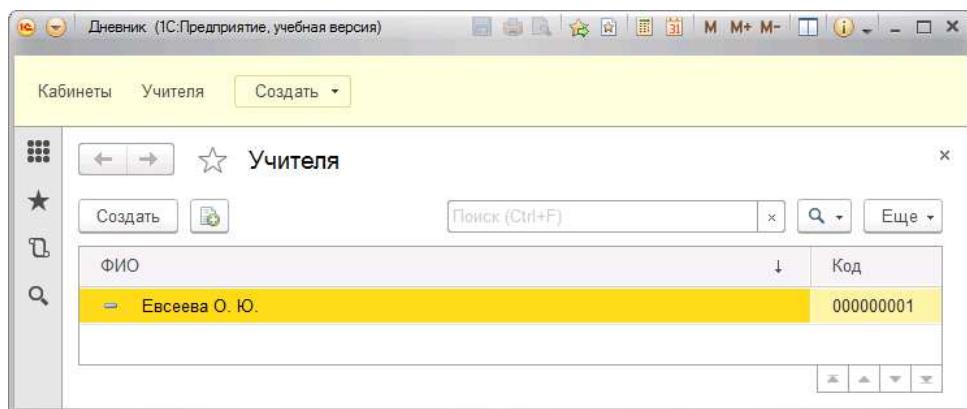


Рисунок 2.125. Евсеева Ольга Юрьевна

2.11 Предметы

Снова вспомните, какие данные вы собирались хранить в программе:

- предметы,
- учителя,
- кабинеты,
- учебные дни.

У вас уже есть справочник *Кабинеты* и справочник *Учителя*. Теперь придумайте, как хранить список предметов.

Я знаю, что для этого нужен справочник. А вы обоснуйте, почему. Какой признак говорит о том, что нужен справочник?

Многие действия вы уже умеете делать самостоятельно. Поэтому попробуйте сами добавить нужный справочник и изменить в его стандартном устройстве всё то, что вы считаете нужным изменить.

И только после этого прочтайте то, что написано дальше. И сравните это со своим вариантом.

Итак, мой справочник выглядит следующим образом (рисунок 2.126).

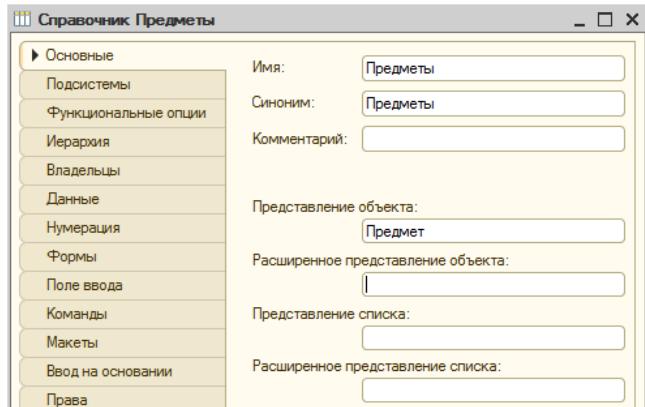


Рисунок 2.126. Справочник «Предметы: Основные»

Его имя и синоним одинаковы — *Предметы*.

Представление объекта — *Предмет*. Другие представления я задавать не стал, потому что для них полностью подходит синоним.

Наименование я использовать не буду. Потому что каждый предмет однозначно характеризуется своим названием: «история», «информатика», «литература». И двух разных предметов с одинаковым названием быть не может. Поэтому длину поля *Наименование* я сделал 0 (рисунок 2.127).

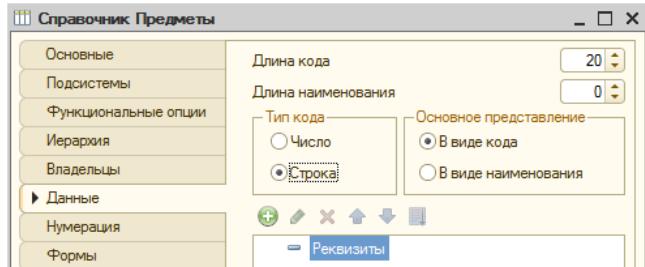


Рисунок 2.127. Справочник «Предметы: Данные»

Название предмета я буду записывать в поле *Код*. Поэтому его длину я сделал 20 символов. Чтобы поместилось название предмета, даже длинное. А тип кода я сделал *Строка*.

Название предмета (поле *Код*) я буду заполнять самостоятельно. Поэтому на закладке *Нумерация* я снял флагок *Автонумерация*. А флагок *Контроль уникальности* я оставил, потому что названия предметов не должны повторяться (рисунок 2.128).

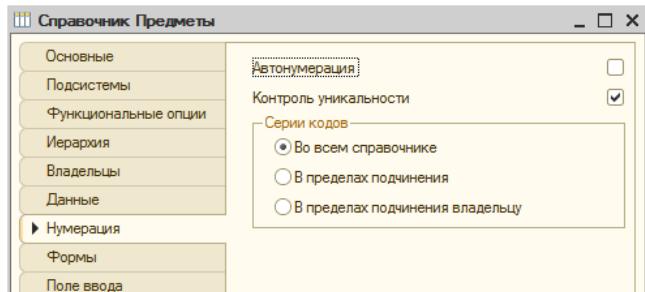


Рисунок 2.128. Справочник «Предметы: Нумерация»

После этого я поправил ввод по строке. Раз наименования у меня теперь нет, я убрал его из ввода по строке на закладке *Поле ввода* (рисунок 2.129).

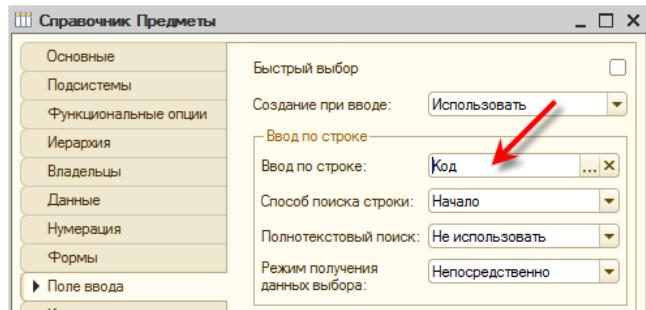


Рисунок 2.129. Справочник «Предметы: Поле ввода»

И, наконец, я изменил синоним поля Код. Чтобы в режиме 1С:Предприятие оно называлось *Название предмета* (рисунок 2.130).

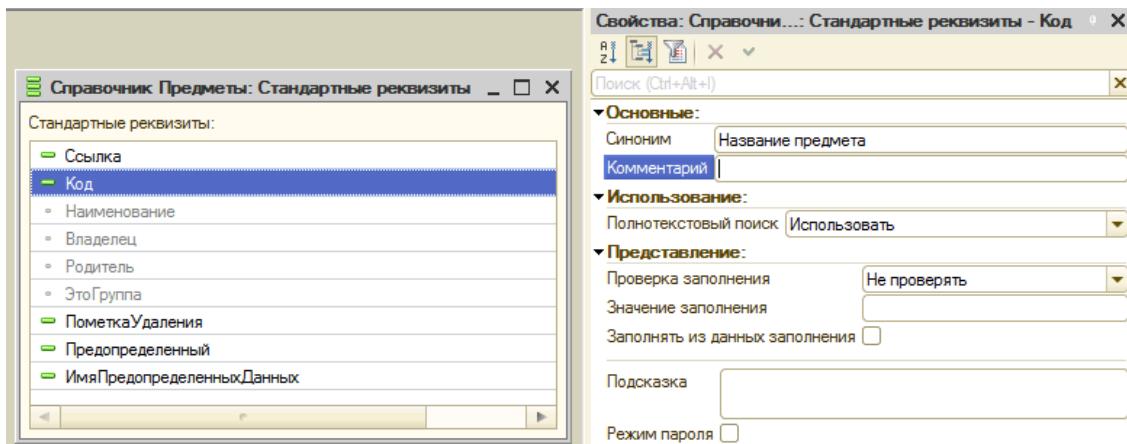


Рисунок 2.130. Синоним поля «Код»

В результате в режиме 1С:Предприятие у меня получилось так (рисунок 2.131).

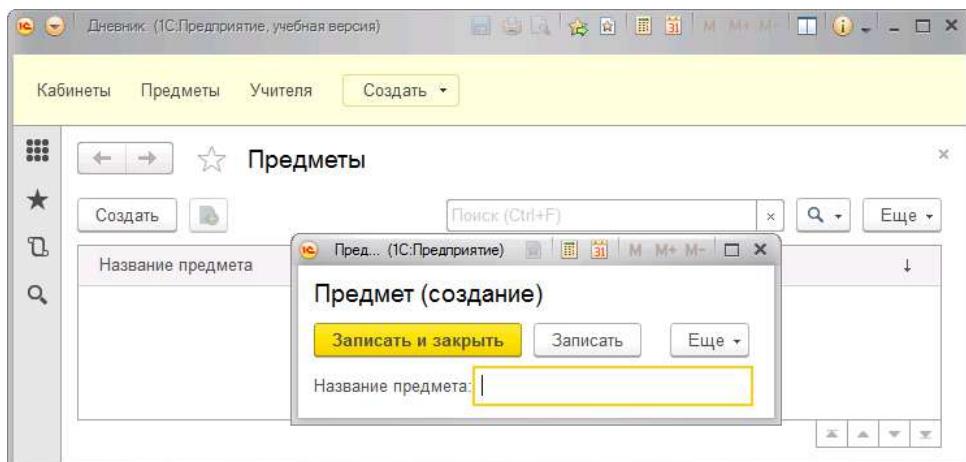


Рисунок 2.131. Справочник «Предметы» в режиме 1С:Предприятие

Если у вас получилось так же — отлично! Поздравляю!

Если не так — ничего страшного! Прочитайте ещё раз, что делал я, и сделайте у себя то же самое.

И пока никакие предметы создавать не нужно.

2.11.1 Реквизиты

А теперь обратите внимание на одну очень интересную особенность.

У вас есть три справочника: *Кабинеты*, *Учителя* и *Предметы*. Сейчас они похожи друг на друга. Но на самом деле один из них, по отношению к реальной жизни, значительно отличается от двух других. Какой?

Подсказка: это справочник *Предметы*. Чем он может отличаться от справочников *Учителя* и *Кабинеты*?

Тогда ещё подсказка.

Что вы хотите знать про каждый кабинет? Его номер. Больше ничего.

Что вы хотите знать про каждого учителя? Его фамилию и инициалы. Больше ничего.

А что вы хотите знать про каждый предмет? Его название. Всё? А может быть, что-то ещё?

Да. Что-то ещё. Я говорил об этом раньше.

Про каждый учебный предмет вы хотите знать, в каком кабинете проходит занятие и какой учитель этот предмет преподаёт.

Если вы уже сообразили, что нужно делать дальше, — отлично. Если нет, то я снова напомню вам картинку про комнату и людей в ней (рисунок 2.132).

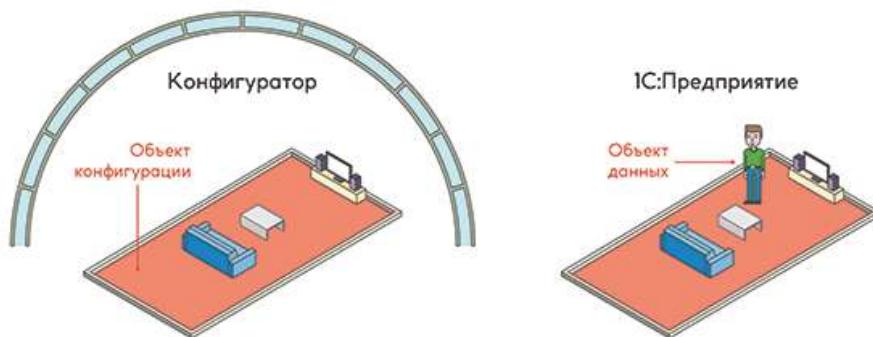


Рисунок 2.132. Объект конфигурации и объект данных

Объект конфигурации не просто описывает стены, в которых будут жить данные. Ещё он описывает то, как эти данные будут выглядеть. Он содержит описание того, что вы хотите знать о каждом из объектов данных.

Каждая такая вещь, «которую вы (или платформа) хотите знать про объект данных», описывается в объекте конфигурации с помощью *реквизита*. С двумя реквизитами вы уже хорошо знакомы и умеете их использовать. Это стандартные реквизиты *Код* и *Название*.

Но теперь про каждый учебный предмет вы хотите знать, какой кабинет и какой учитель. Как сделать так, чтобы у каждого предмета появилось ещё два поля, в которые можно это записать?

Очень просто! Вам нужно создать в этом справочнике два своих реквизита: *Кабинет* и *Учитель*. Тогда в режиме 1С:Предприятие у каждого предмета появятся ещё два поля помимо поля *Название* предмета. В одном вы запишете, в каком кабинете проходят занятия, в другом — какой учитель ведёт этот предмет.

Если это не совсем понятно, не страшно. Сейчас вы это сделаете, а потом ещё раз посмотрите на картинке.

Как добавить к справочнику собственный реквизит? Наверняка вы уже догадались. В конфигураторе. В дереве конфигурации.

Когда вы что-то хотите добавить к дереву конфигурации, это всегда можно делать одним и тем же способом. Так, как вы добавляли новые объекты конфигурации. Выделяете в дереве ту ветку, куда вы хотите добавить, и вызываете контекстное меню. В контекстном меню будет команда *Добавить*.

Сейчас вы хотите добавить реквизит справочника *Предметы*, значит, вам нужна ветка *Реквизиты*, которая есть в этом справочнике (рисунок 2.133).

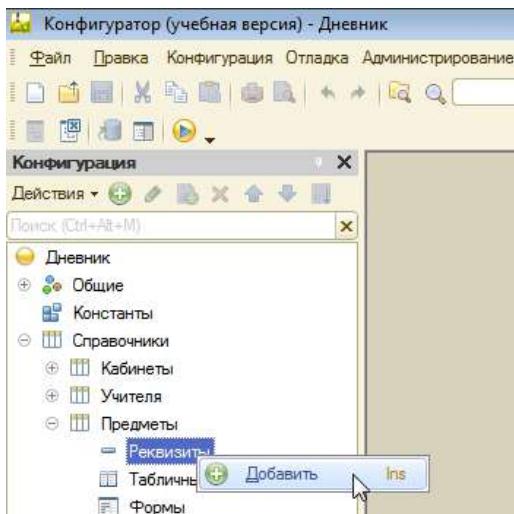


Рисунок 2.133. Добавить реквизит справочника

В дереве конфигурации появится реквизит с именем *Реквизит1*, а для изменения его свойств откроется палитра свойств (рисунок 2.134).

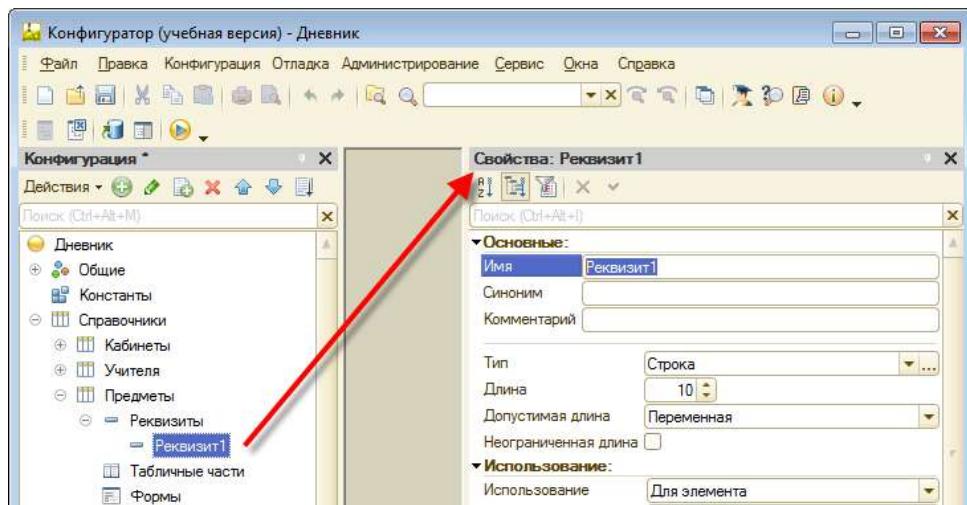


Рисунок 2.134. Новый реквизит справочника

Что нужно изменить?

Ну, прежде всего, конечно, имя. Стандартное имя *Реквизит1* вам не подходит. Измените его на *Кабинет*.

Что ещё? Ещё нужно изменить тип. Вы уже сталкивались со словом *тип*. Когда изменяли тип кода у справочников.

Вспомните: когда вам нужно было хранить числа (номера кабинетов), вы указывали, что тип кода — *Число*. А когда вам нужно было хранить названия учебных предметов, вы указывали, что тип кода — *Строка*.

Интуитивно вам было понятно, почему в одном случае *Число*, а в другом — *Строка*. Поэтому я и не рассказывал о том, что такое тип.

Сейчас другая ситуация. Сейчас без объяснений не получится. Но и рассказывать о типе тоже рано. Про типы я много расскажу вам в следующей части книги. Поэтому сейчас обратитесь к тем картинкам, которые вы уже знаете.

Когда я рассказывал вам про справочник, я говорил, что бывают такие ситуации, когда нужно указывать не само название учебного предмета, а ссылку на него. Чтобы не было путаницы, если в одном месте предмет называли «Хим.», а в другом — «Химия». И я показывал две картинки.

Одна картинка — когда друг спрашивает вас: «Какой у тебя телефон?» Вы достаёте из кармана телефон, показываете ему и говорите: «Вот такой» (рисунок 2.135). Это вы ему даёте данные.



Рисунок 2.135. «Вот такой у меня телефон» — данные

Так вот эта картинка про то, когда вы указывали, что тип кода *Число* или *Строка*. То есть тем самым вы говорили, что в поле *Код* будет лежать какое-то число или какая-то строка. Которые сами и являются данными.

А сейчас, для реквизита *Кабинет*, вам нужно сделать по-другому. Друг спрашивает вас: «Какой у тебя телефон?» А вы показываете рукой на стол, где лежит ваш телефон, и говорите: «Вон такой, иди, посмотри» (рисунок 2.136).



Рисунок 2.136. «Мой телефон вон там» — ссылка

То есть вам нужно указать, что в этом поле *Кабинет* будет находиться *ссылка*. Ссылка на что? На один из кабинетов, которые есть в вашей программе. То есть на один из элементов справочника *Кабинеты*.

Чтобы это сделать, нажмите кнопку выпадающего списка у поля *Тип* (рисунок 2.137).

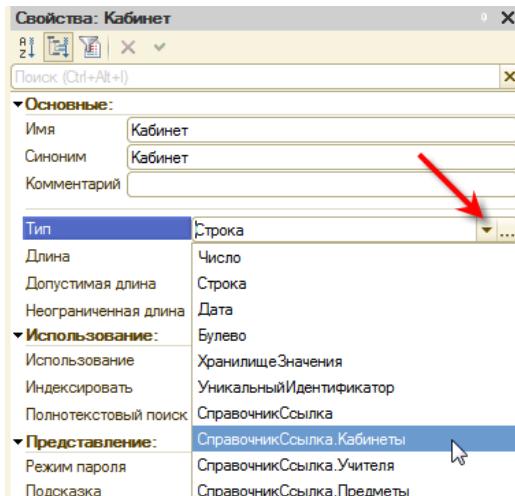


Рисунок 2.137. Выбор типа

Тут написано много непонятных слов. Не пугайтесь. Потом вы их узнаете.

А пока вы знаете, что вам нужно что-то про кабинеты со словом «ссылка». Такая строчка есть, это *СправочникСсылка.Кабинеты*. Выбирайте эту строчку (рисунок 2.138).

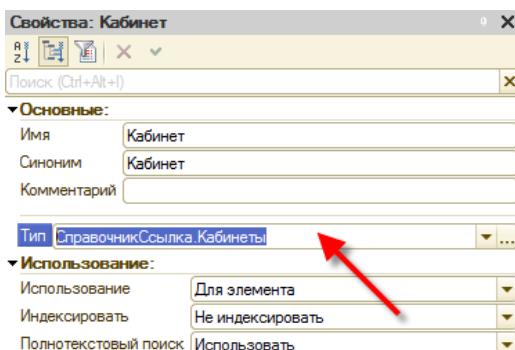


Рисунок 2.138. Тип «СправочникСсылка.Кабинеты»

Запустите конфигурацию в режиме отладки и посмотрите, что получилось. Откройте список справочника *Предметы* и нажмите *Создать* (рисунок 2.139).

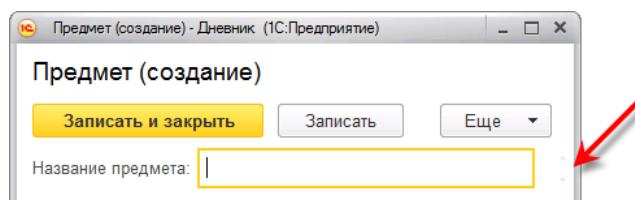


Рисунок 2.139. Форма объекта справочника «Предметы»

Бывает так, что платформа не всегда поспевает за вами, когда вы меняете объекты конфигурации. Например, как на этом рисунке. Тут видно, что в форме есть ещё одно поле, но оно не поместилось в видимую часть. О том, что видно не всё, говорят две стрелки «вверх — вниз», которые есть у правого края формы.

Если у вас тоже получилось так, восстановите положение окна. Для этого нажмите на клавиатуре сочетание клавиш Alt+Shift+R, и форма примет тот размер, который и должен у неё быть (рисунок 2.140).

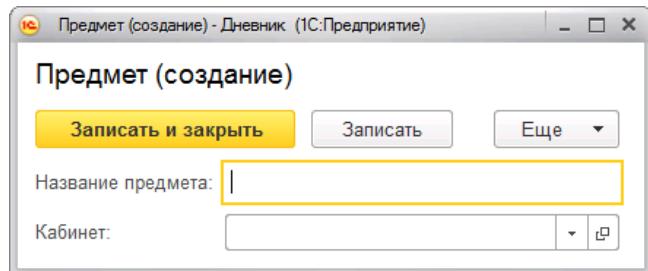


Рисунок 2.140. Форма правильного размера

Смотрите: помимо поля *Название предмета* у вас теперь есть ещё одно поле, которое называется *Кабинет*.

Примечание

Итак, благодаря тому, что у объекта конфигурации есть стандартный реквизит *Код (Название предмета)* и реквизит *Кабинет* (созданный вами), вы имеете возможность для каждого объекта данных (элемента справочника) указать название предмета и кабинет, в котором проходят занятия (рисунок 2.141).

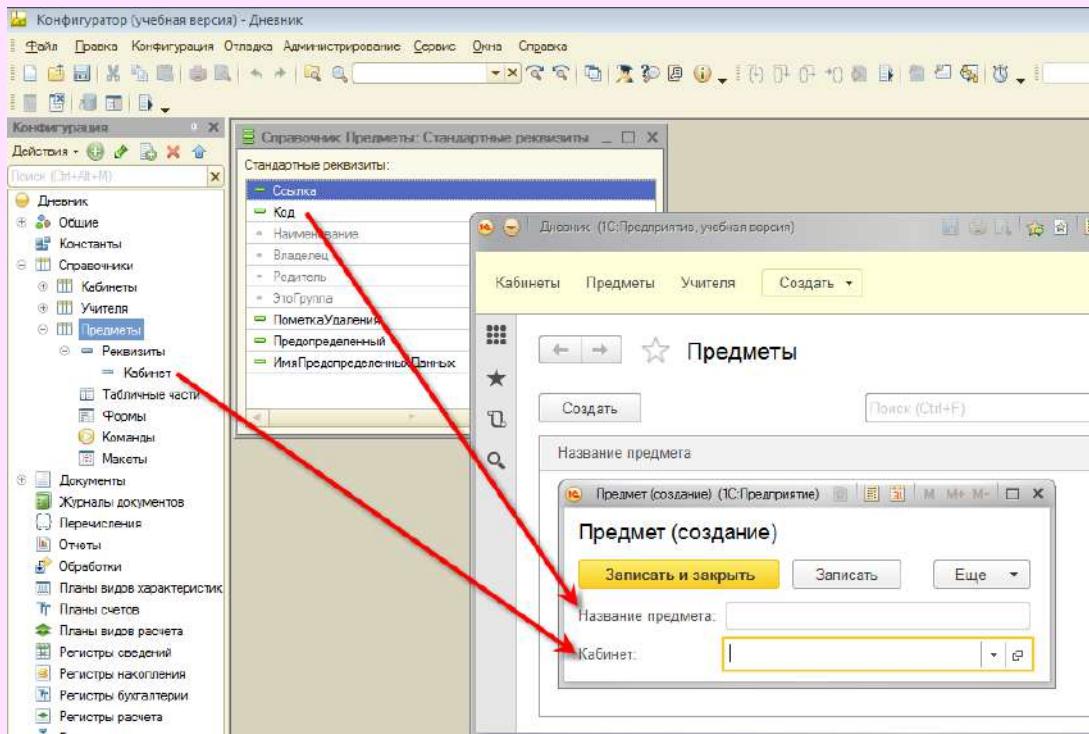


Рисунок 2.141. Реквизиты «Код» и «Кабинет»

Закройте 1С:Предприятие. Вернитесь в конфигуратор.

Совет

Зачем я прошу вас в этот момент закрыть 1С:Предприятие? Это техническая особенность.

Когда вы работаете в режиме 1С:Предприятие, вы можете изменить размеры любой формы. А платформа запоминает размеры всех форм. Для того чтобы при следующем запуске в режиме 1С:Предприятие показать их в том же размере. Но запоминает она их только в момент обычного, штатного, завершения сеанса 1С: Предприятия.

Поэтому, если вы восстанавливали положение окна с помощью **Alt+Shift+R**, нужно завершить сеанс 1С:Предприятия, чтобы платформа запомнила новый размер этой формы.

Если вы из конфигуратора выполните команду *Завершить отладку*, или перезапустите отладочный сеанс, или каким-нибудь другим способом завершите работу в режиме 1С:Предприятие, платформа не запомнит новые размеры окна. И в следующий раз в этой форме опять не будет видно всех полей.

Подробнее

Подробнее вы можете прочитать про сочетания клавиш для управления окнами во встроенной справке: *Главное меню – Справка – Содержание справки – Сочетания клавиши (1С:Предприятие) – Управление окнами*.

Итак, один реквизит, *Кабинет*, вы создали. Теперь создайте второй реквизит — *Учитель*.

Как это делать, вы теперь знаете. Сделайте это самостоятельно.

А чтобы вы могли себя проверить, я покажу, как это делал я (рисунок 2.142).

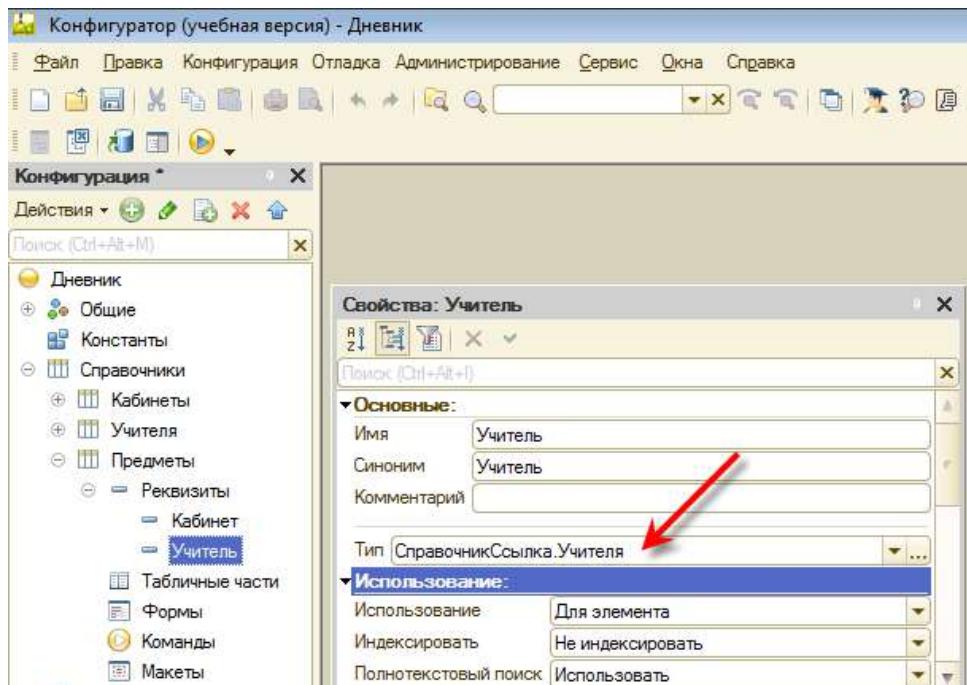


Рисунок 2.142. Реквизит «Учитель»

К справочнику *Предметы* я добавил реквизит, который назвал *Учитель*. Тип этого реквизита я задал как *СправочникСсылка.Учителя*. Для того чтобы иметь возможность сослаться на какого-то учителя, который есть в справочнике *Учителя*.

Теперь, если вы запустите конфигурацию в режиме отладки и откроете форму нового предмета, вы увидите такую картинку (рисунок 2.143). Возможно, вам опять понадобится нажать **Alt+Shift+R**.

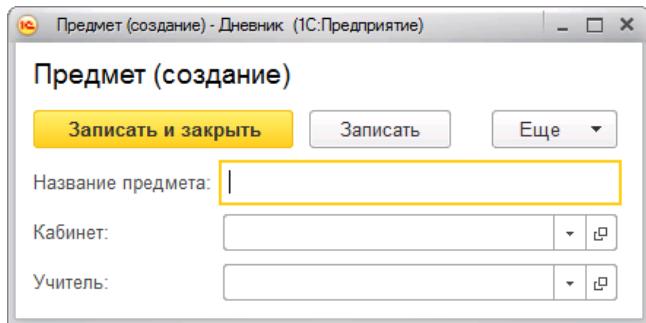


Рисунок 2.143. Поля «Кабинет» и «Учитель»

Получилось всё, как вы хотели. Про каждый предмет вы можете записать его название, кабинет, в котором проходят занятия, и учителя, который преподает этот предмет.

Можно начать записывать? Да! И это будет очень интересно!

Но чтобы это было еще и красиво, наведите порядок в командном интерфейсе.

2.11.2 Командный интерфейс раздела

Закройте сеанс 1С:Предприятия, вернитесь в конфигуратор.

Сейчас командный интерфейс выглядит так (рисунок 2.144).

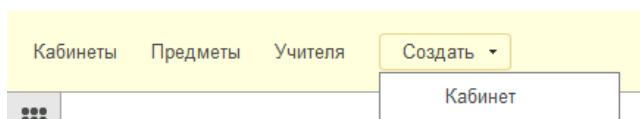


Рисунок 2.144. Командный интерфейс

Это не очень хорошо. Команды лучше расположить в порядке их значимости. Сначала самые «важные», в конце — самые «неважные».

Самым важным для вас является, наверное, список предметов. Потому что в нем есть информация и об учителях, и о кабинетах. Самым неважным для вас является, наверное, список кабинетов. Его можно поставить на последнее место. Значит, список учителей окажется на втором месте.

Команду создания нового кабинета вы добавляли только для учебных целей. Если же говорить о работе, то лучше вместо неё добавить команду создания нового предмета.

Всё это вы можете сделать самостоятельно. Для этого вам понадобится *открыть командный интерфейс основного раздела* (рисунок 2.145). Если вы забыли, как это делается, посмотрите в разделе 2.9.5 «Объект конфигурации описывает, как будут выглядеть его данные» на страницах 82–83.

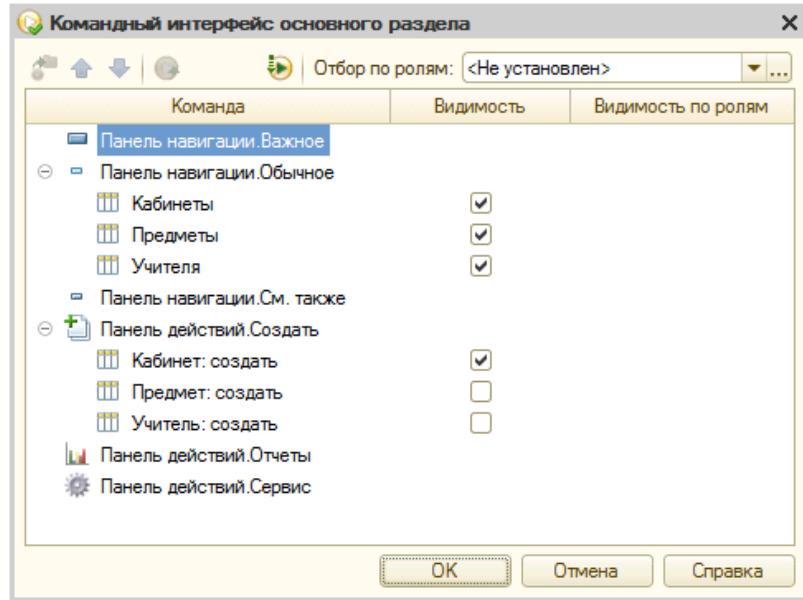


Рисунок 2.145. Командный интерфейс основного раздела

Что тут нужно сделать? Имеющиеся команды нужно перенести в правильные ветки. И у тех команд, которые вы хотите видеть, надо включить флажки.

«Какая разница между этими ветками?» — спросите вы. Это совсем несложно. Просто сейчас там есть названия, на которые вам не нужно обращать внимания. Это такие слова, как «панель навигации» и «панель действий». Уберите их и представьте, что ветки называются просто *Важное*, *Обычное* и так далее.

Тогда *Важное*, *Обычное* и *См. также* работают так (рисунок 2.146).

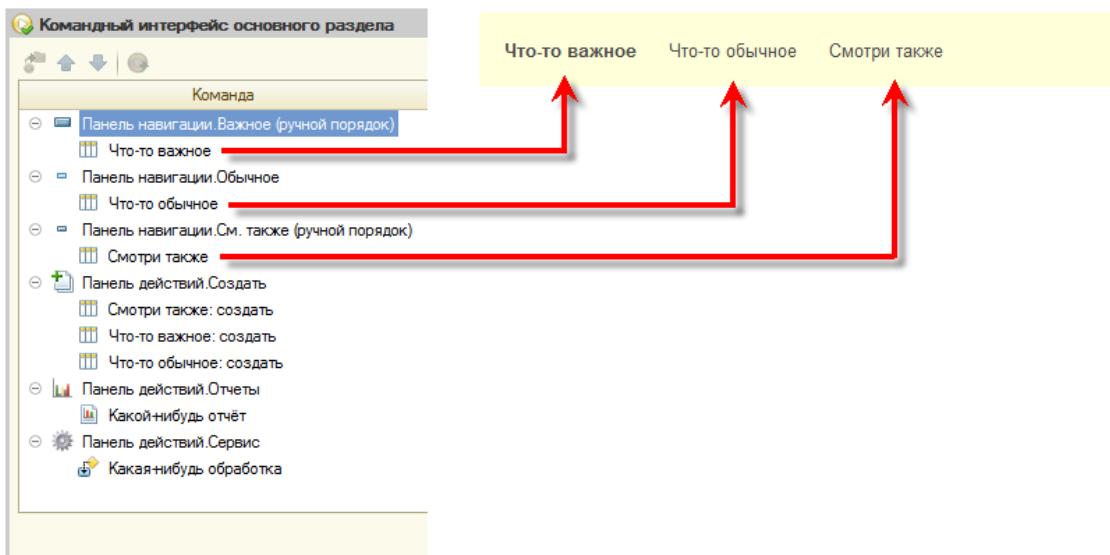


Рисунок 2.146. Группы «Важное», «Обычное» и «См. также»

В начале отображаются команды из группы *Важное*. И они выделяются жирным шрифтом.

Затем отображаются команды из группы *Обычное*. После них — из группы *См. также*. Команды этих двух групп внешне ничем не отличаются, поэтому можно использовать только группу *Обычное*.

В ветки *Важное*, *Обычное* и *См. также* платформа автоматически собирает команды перехода к спискам.

Для ветки *Создать* создаётся кнопка, «под которой» друг за другом размещаются команды создания объектов данных (рисунок 2.147).

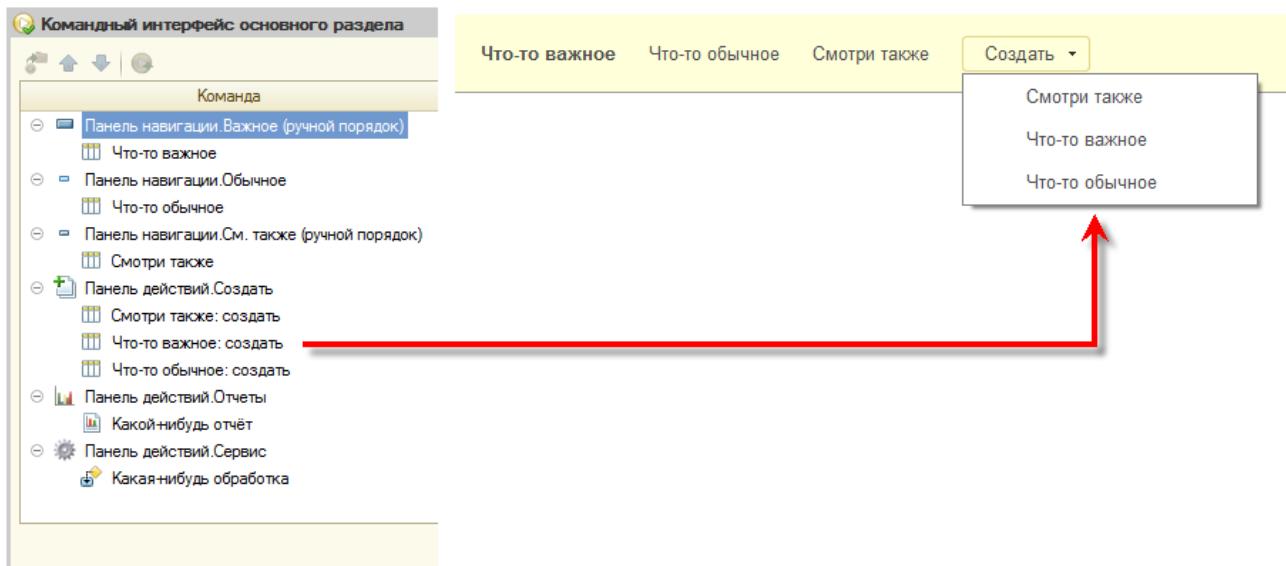


Рисунок 2.147. Создать

Для веток *Отчёты* и *Сервис* тоже создаются свои кнопки. В ветке *Отчёты* размещаются команды для работы с отчётами. Есть такие объекты конфигурации. Вы с ними познакомитесь позже (рисунок 2.148).

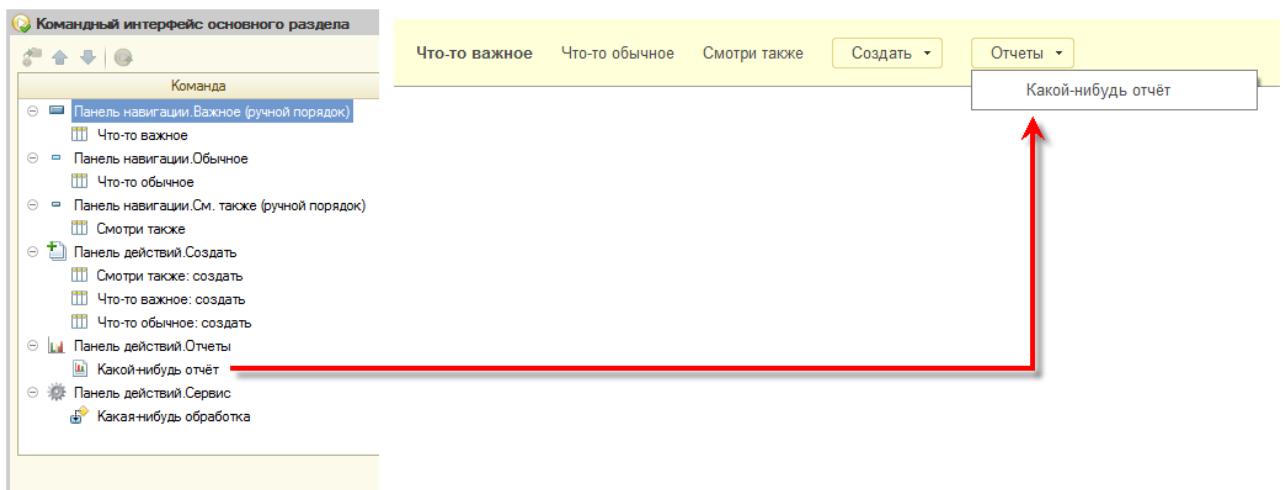


Рисунок 2.148. Отчёты

А в ветке *Сервис* размещаются команды для запуска обработок. *Обработка* — это тоже объект конфигурации. Обычно они используются в служебных целях. Для того чтобы «навести порядок» в ваших данных. Или для того чтобы автоматически загрузить данные из другой программы (рисунок 2.149).

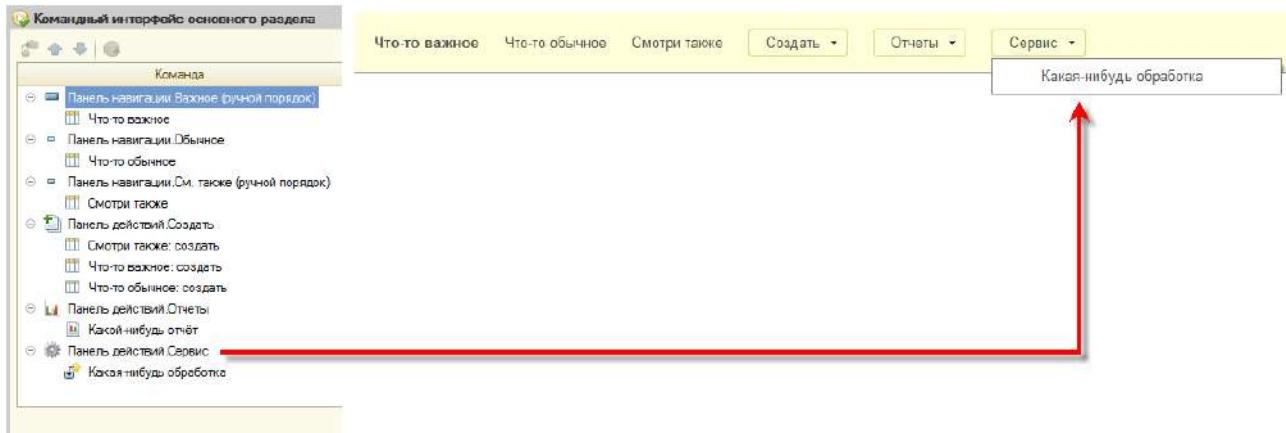


Рисунок 2.149. Сервис

Теперь, когда вы знаете, как устроен командный интерфейс раздела, расставьте команды.

Чтобы перенести команду в другую ветку, перетащите её мышью. А чтобы изменить порядок команд, используйте синие стрелки *Переместить вверх* и *Переместить вниз* (рисунок 2.150).



Рисунок 2.150. Переместить вверх и вниз

А мой вариант, для сравнения, вы можете посмотреть на рисунке 2.151.

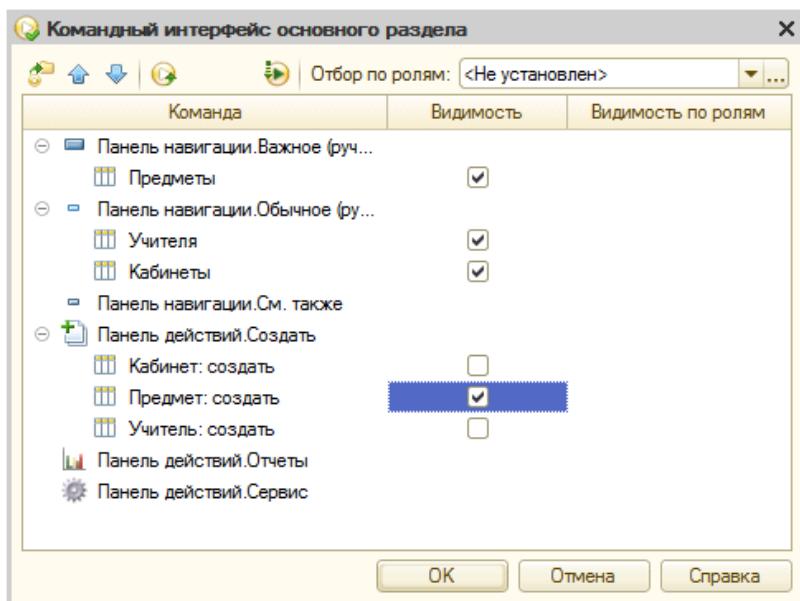


Рисунок 2.151. Командный интерфейс основного раздела

Команду *Предметы* я перетащил в *Важное*. Она будет жирной. За ней будут команды *Учителя* и *Кабинеты*. А в конце будет кнопка *Создать*, в которой будет команда для создания нового предмета.

Нажмите *OK*. Запустите конфигурацию в режиме отладки и проверьте (рисунок 2.152).

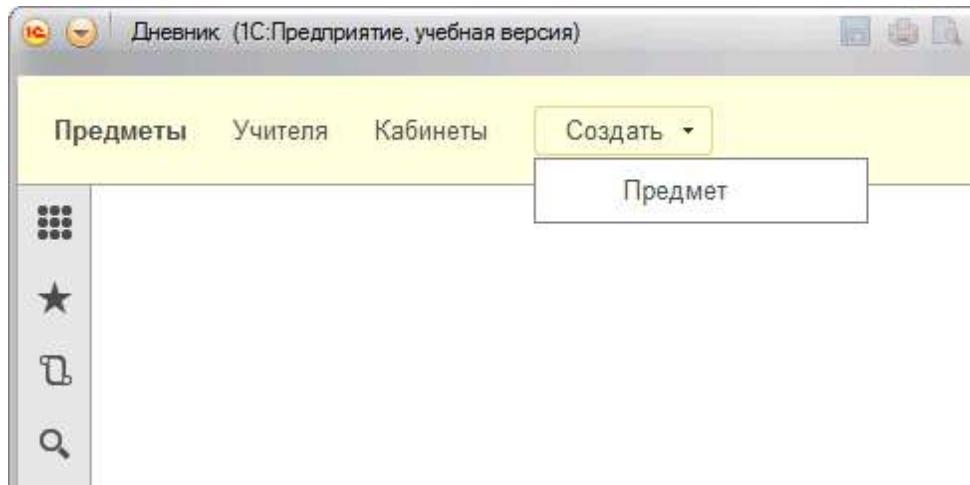


Рисунок 2.152. Новый командный интерфейс

2.11.3 Ввод по строке

Теперь у вас наконец-то есть возможность ввести большую часть данных. Сначала я покажу, как это удобнее всего делать. А потом вы самостоятельно введёте всю информацию.

Совет

Если хотите, можете создавать свои собственные учебные предметы, учителей и кабинеты. Не такие, как я предлагаю в книге. Тогда в дальнейшем вам всё время придётся учитывать, что ваши данные отличаются от того, что вы видите на рисунках.

Если вам не важно, какие данные будут в вашем прикладном решении, можете не вводить абсолютно все данные до конца. Возьмите уже готовую информационную базу «01 ПредметыУчителяКабинеты.dt», в которой все эти данные есть. Как её подключить, написано в разделе [А.1 «Как подключить демонстрационную базу»](#) на странице [543](#).

Сейчас вы будете создавать предметы. Нажмите *Создать – Предмет*.

Первым предметом будет Музыка. Ведёт его Евсеева Ольга Юрьевна. Кабинет № 131. Напишите название предмета и нажмите клавишу TAB (рисунок 2.153).

Рисунок 2.153. Название предмета заполнено

Теперь вам нужно ввести номер кабинета. Наберите на клавиатуре 131 (рисунок 2.154).

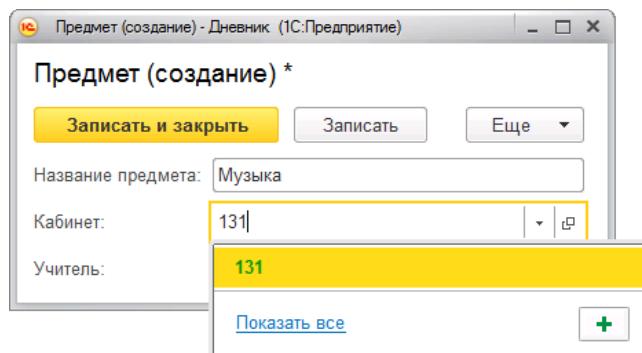


Рисунок 2.154. Введён номер кабинета

Что произошло? Вы помните, что, когда вы создавали реквизит *Кабинет*, вы выбрали для него тип ссылки на справочник *Кабинеты*. Чтобы он ссылался на какой-то элемент этого справочника.

Сейчас вы ввели номер кабинета, и платформа сама нашла в справочнике *Кабинеты* кабинет с таким номером. Он там уже есть, вы создавали его раньше.

Теперь вам осталось только нажать клавишу **Enter**, и платформа подставит найденную ссылку в поле *Кабинет* (рисунок 2.155).

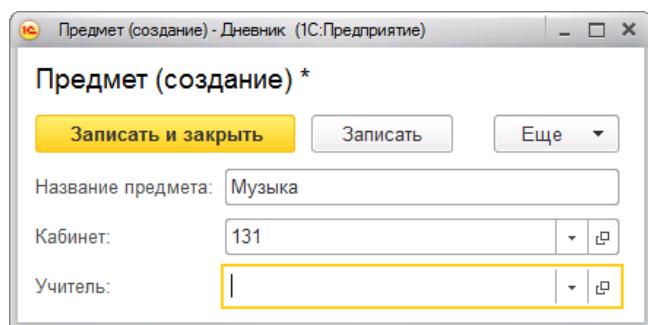


Рисунок 2.155. Кабинет заполнен

Такой способ ввода называется *ввод по строке*. Когда вам нужно в какое-то поле поместить, например, ссылку на элемент справочника, вы не ищете этот элемент в других списках. Не просматриваете все элементы, которые имеются. Вы просто начинаете вводить в поле либо наименование, либо код, которые вам известны. И платформа сама находит и предлагает вам подходящие элементы.

Поле *Учитель* заполните точно так же. Начните набирать на клавиатуре «евсе». Платформа найдёт и предложит вам подходящий элемент (рисунок 2.156).

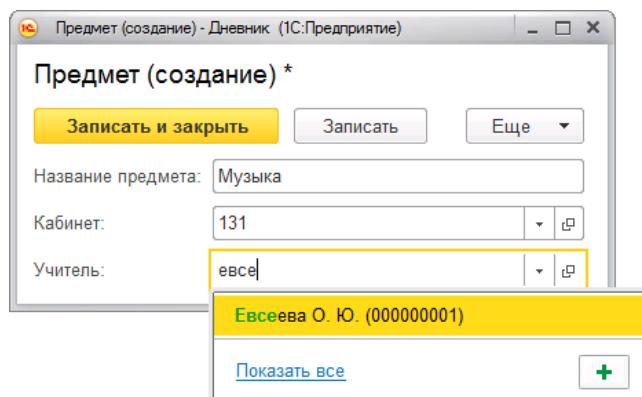


Рисунок 2.156. Введено начало фамилии

Поскольку платформа уже нашла подходящий элемент, вы можете не продолжать набор фамилии. Просто нажмите **Enter**, и платформа подставит найденную ссылку в поле Учитель (рисунок 2.157).

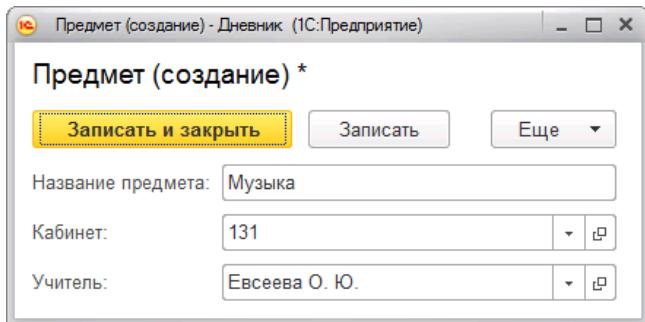


Рисунок 2.157. Учитель заполнен

Теперь нажмите **Записать и закрыть**.

Всё. Первый предмет вы создали. Теперь создайте второй.

Для разнообразия сделайте это с помощью другой команды. Откройте список предметов и в нём нажмите **Создать**.

Вторым предметом будет **ИЗО**. Ведёт его Казаков Кирилл Данилович. Кабинет № 101. Название предмета и кабинет вы уже можете заполнить самостоятельно.

А когда вы начнёте вводить фамилию учителя, вы увидите такую картинку (рисунок 2.158).

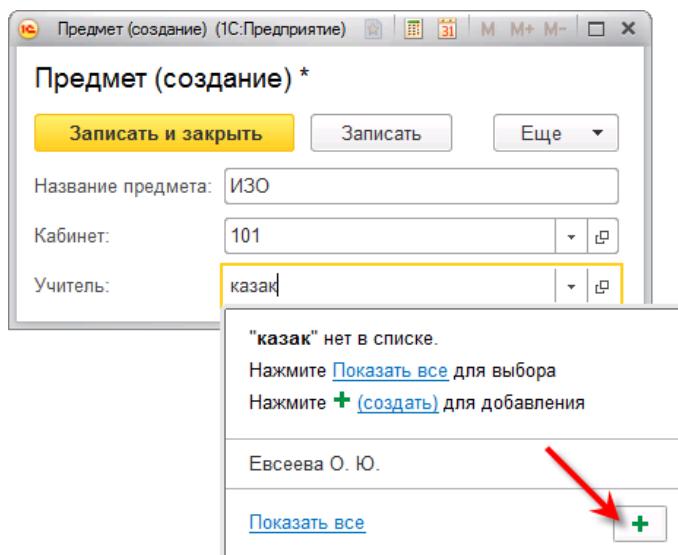


Рисунок 2.158. Подходящего учителя нет в справочнике

Платформа скажет вам, что в списке учителей нет ни одного элемента, который начинался бы на «казак». То есть нет смысла вводить фамилию до конца, всё равно подходящего учителя нет. Что же делать?

Очень просто. Такого учителя надо создать. И для этого не нужно открывать список учителей, создавать в нём новый элемент... Достаточно всего лишь нажать на кнопку «+», которая есть в этом же окне.

Платформа сама откроет форму для создания нового учителя и даже подставит в неё ту строку, которую вы начали вводить (рисунок 2.159).

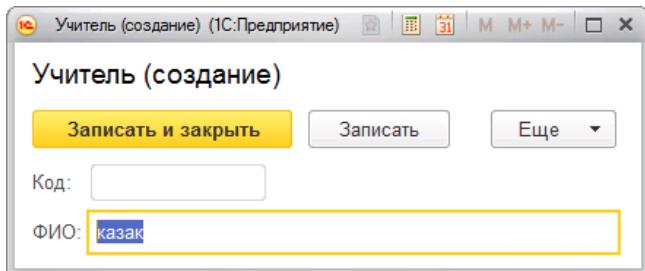


Рисунок 2.159. Форма для создания нового учителя

Вам останется только исправить *ФИО* так, как они должны выглядеть, Казаков К. Д., и нажать *Записать и закрыть*.

После этого платформа сохранит нового учителя и сразу же подставит ссылку на него в то поле, которое заполняли (рисунок 2.160).

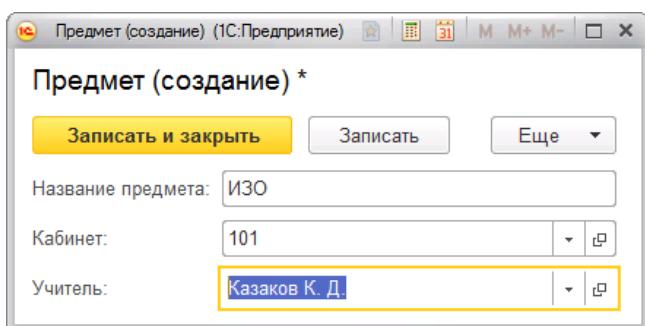


Рисунок 2.160. Учитель заполнен

Нажмите *Записать и закрыть* — и вот у вас создано уже два предмета (рисунок 2.161).

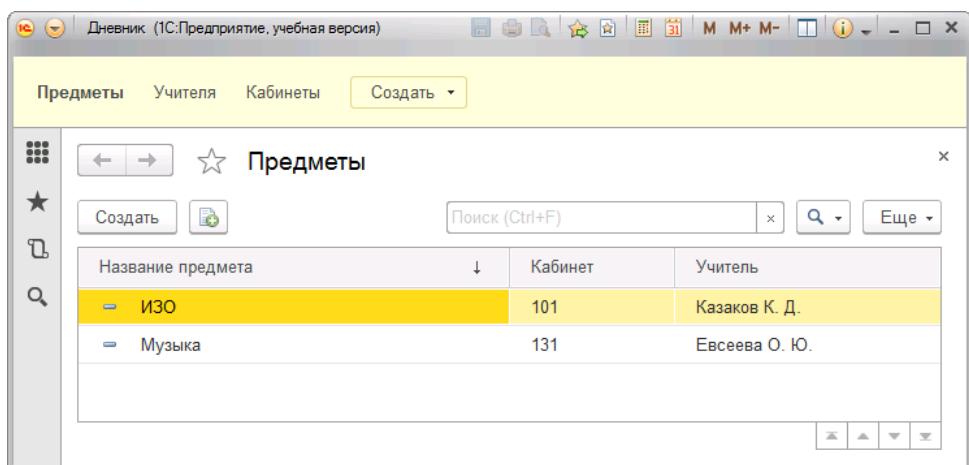


Рисунок 2.161. Два предмета создано

Точно таким же образом создайте сами все остальные предметы. Начинайте вводить кабинет или учителя. Если подходящий есть — выбирайте его. Если подходящего нет — создавайте новый.

Создайте такие предметы:

- Английский язык. Ведёт его Николаева Татьяна Яковлевна. Кабинет № 401.
- Информатика. Ведёт её Давыдова Александра Вадимовна. Кабинет № 418.
- История. Ведёт её Герасимова Екатерина Сергеевна. Кабинет № 127.
- Кл. час. Ведёт его Рыбакова Анастасия Егоровна. Кабинет № 311.
- Литература. Ведёт её Авдеева Валерия Матвеевна. Кабинет № 409.
- Математика. Ведёт её Рыбакова Анастасия Егоровна. Кабинет № 311.

- **Природоведение.** Ведёт его Филиппова Лариса Валерьевна. Кабинет № 220.
 - **Русский язык.** Ведёт его Авдеева Валерия Матвеевна. Кабинет № 409.
 - **Физ. культура.** Ведёт её Крюков Валерий Валерьевич. Кабинет № 223.
- В результате у вас получится такой список (рисунок 2.162).

Название предмета	Кабинет	Учитель
Английский язык	401	Николаева Т. Я.
ИЗО	101	Казаков К. Д.
Информатика	418	Давыдова А. В.
История	127	Герасимова Е. С.
Кл. час	311	Рыбакова А. Е.
Литература	409	Авдеева В. М.
Математика	311	Рыбакова А. Е.
Музыка	131	Евсеева О. Ю.
Природоведение	220	Филиппова Л. В.
Русский язык	409	Авдеева В. М.
Физ. культура	223	Крюков В. В.

Рисунок 2.162. Учебные предметы

Ради интереса можете открыть список учителей или кабинетов. Вы увидите, что эти списки содержат все те элементы, которые вы создавали в процессе добавления предметов.

2.12 Документ

Информационная база

Прикладное решение, которое должно получиться у вас к этому моменту, содержится в демонстрационной базе «01 ПредметыУчителяКабинеты.dt». Как её подключить, написано в разделе A.1 «Как подключить демонстрационную базу» на странице 543.

Пришло время познакомиться ещё с одним объектом конфигурации. Этот объект конфигурации называется *документ*. Так же как это было со справочником, есть два признака, по которым можно понять, что для хранения определённого вида данных нужен документ.

Во-первых, документ подходит для того, чтобы хранить в компьютере данные, которые привязаны к дате или ко времени. Такие данные, про которые важно знать их расположение на оси времени.

Например, вы ведёте список дел. Когда и что вам нужно сделать. То есть ваши данные — это отдельные дела. Вы должны знать, в какой день и в какое время необходимо выполнить каждое дело. Купить подарок нужно обязательно в пятницу. Потому что в субботу вы идёте на день рождения и этот подарок вам понадобится (рисунок 2.163).

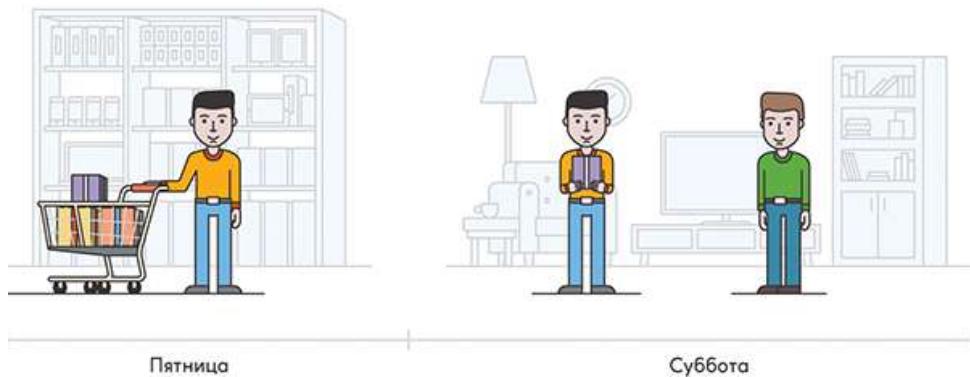


Рисунок 2.163. Данные, привязанные ко времени

Во-вторых, привязка данных к конкретным датам может быть не важна. Но очень важно, чтобы эти данные располагались по времени друг за другом в определённой последовательности.

Например, вы хотите навестить своих друзей. То есть ваши данные — это отдельные встречи с друзьями. Вы можете сделать это сегодня, или завтра, или на следующей неделе — это не имеет значения. Но для вас важно, что сначала нужно заехать к Сергею, а потом к Ивану. Потому что они живут рядом. И только после этого ехать к Борису, который живёт в другом районе (рисунок 2.164).

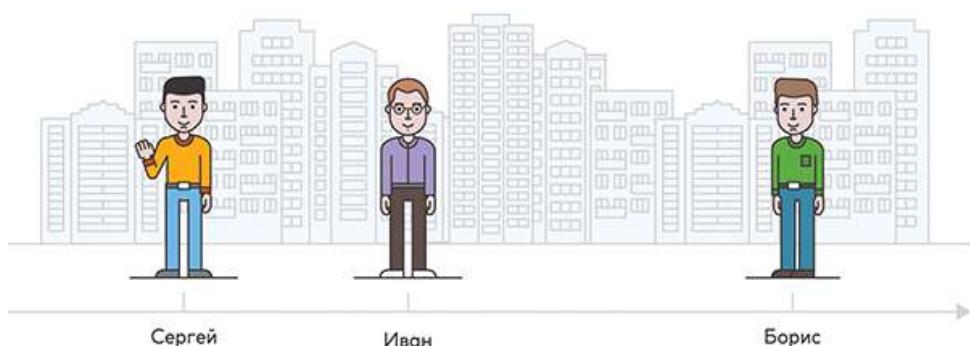


Рисунок 2.164. Последовательность данных во времени

Примечание

Если данные привязаны ко времени или важна последовательность данных на оси времени, это первый верный признак того, что для их хранения нужен объект конфигурации, который называется *документ*.

Другой признак — это наличие в ваших данных какой-то информации, которую вы хотите сложить, накопить, проанализировать.

Например, данные — это ваши друзья. Про каждого друга вы знаете, сколько ему лет. Интересно ли вам знать, сколько лет всем вашим друзьям вместе? Наверное, нет. Поэтому вы храните их в справочнике.

Другой случай. Ваши данные — это поездки к друзьям. Про каждую поездку вы знаете, сколько денег вы потратили. К Сергею вы доехали на трамвае за 35 рублей, к Ивану дошли пешком, а к Борису ехали на такси за 1 000 рублей. Интересно ли вам знать, сколько денег вы потратили на поездки к друзьям? Наверняка! Значит, для хранения таких данных, поездок к друзьям, вам нужен документ.

Примечание

Если каждый элемент данных содержит информацию, которую нужно накапливать, складывать, анализировать в совокупности по всем элементам данных — это второй верный признак того, что для хранения таких данных нужен объект конфигурации, который называется *документ*.

Подробнее

Подробнее вы можете прочитать про документ в документации «Руководство разработчика 8.3. Раздел 5.9. Документы».

2.13 Учебные дни

Какие данные остались у вас неохваченными? Предметы, учителей и кабинеты вы уже записали в программу. Остались учебные дни.

Напомню: каждый учебный день — это табличка, в которой написано, какие уроки проходят, что по ним задано и какие оценки получены (рисунок 2.165). Подходит ли документ для хранения таких данных? Давайте разберёмся.

Среда 23	История	25-28	
	Русский	29, учр. 83	5/6
	Литера	Узник	
	Англ	248, № 735	
	Хим	25. Очистка кемп.	

Рисунок 2.165. Учебный день

Привязаны ли эти данные ко времени? Конечно. Каждый учебный день однозначно связан с конкретной датой.

Содержит ли каждый учебный день какую-то информацию, которую вы хотите накапливать и анализировать? Да. И такой информации много.

Во-первых, это полученные вами оценки. Все они нужны для того, чтобы посчитать средний балл по каждому предмету.

Во-вторых, это домашние задания. Их считать не нужно, но все их нужно анализировать. Чтобы в каждый момент времени у вас была возможность узнать, какие задания нужно выполнить на завтра. На послезавтра. И так далее.

И в-третьих, будет интересно знать количество прошедших занятий по каждому предмету. Сейчас, может быть, эта информация кажется вам неважной. Но позже вы увидите, что она может быть очень полезной.

Значит, добавьте в свою программу новый объект конфигурации *Документ*. Таким же способом, которым вы добавляли справочники. Платформа добавит в дерево объект конфигурации с именем *Документ1* и откроет окно для его редактирования (рисунок 2.166). Как видите, все действия с разными объектами конфигурации выполняются похожим образом.

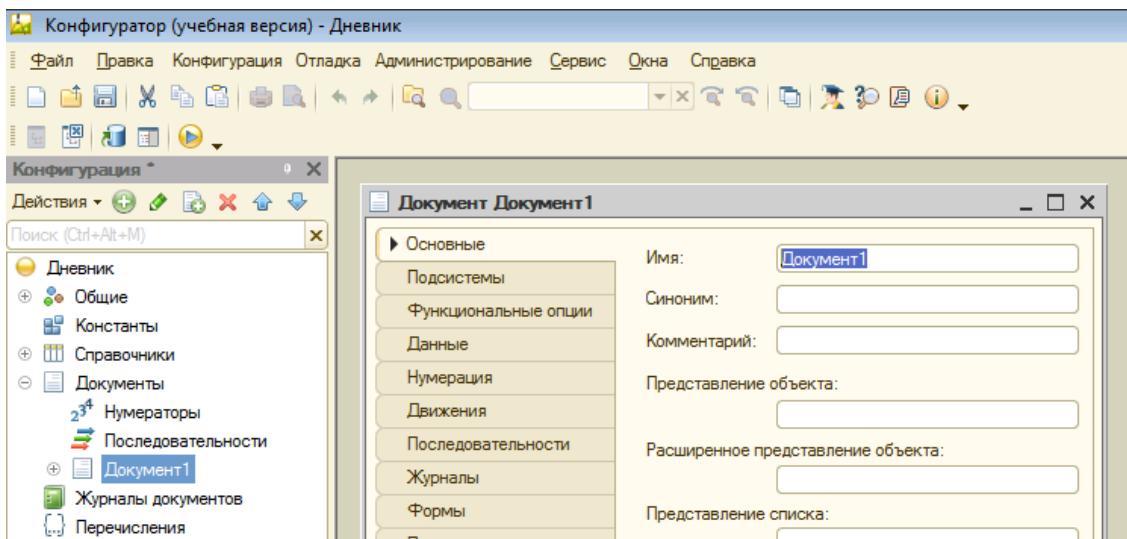


Рисунок 2.166. Новый документ

Нужно дать новому объекту конфигурации имя. Тут в случае с документом (и только с ним) есть одна маленькая особенность.

Как вы помните, объект конфигурации описывает хранилище, в котором будут находиться объекты данных. Вспомните комнаты и людей в комнатах (рисунок 2.167).

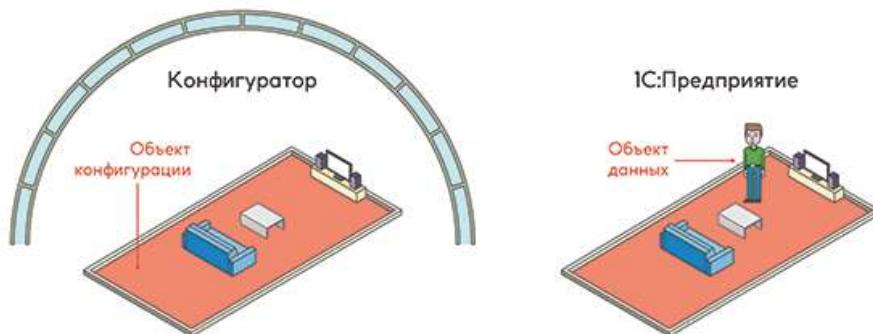


Рисунок 2.167. Объект конфигурации и объект данных

Когда вы создавали справочники, вся комната называлась у вас «справочник». А каждый человек — «элемент справочника».

Сейчас, когда вы имеете дело с документами, каждый человечек — это «документ». А все человечки, находящиеся в одной комнате, — это «набор документов одного вида». То есть объект конфигурации правильно было называть «набор документов одного вида».

Но так сложилось исторически, что объект конфигурации в 1С:Предприятии тоже называется «документ». То есть вся комната называется «документ», и каждый человек в этой комнате тоже называется «документ».

Поэтому, раз каждый документ у вас называется «учебный день», то объект конфигурации назовите тоже «учебный день». В единственном числе (рисунок 2.168).

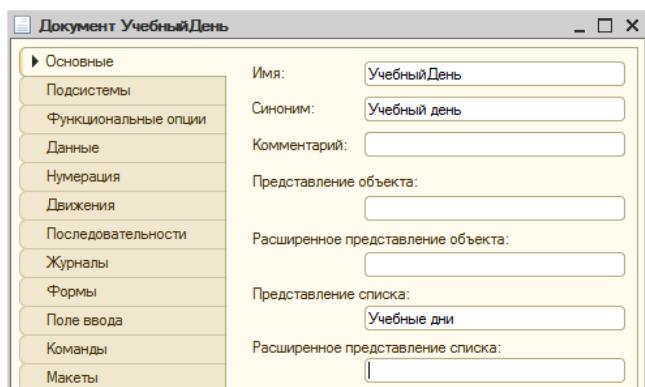


Рисунок 2.168. Документ «Учебный день: Основные»

Обратите внимание на то, как я написал имя: *УчебныйДень*. Раньше вы с этим не сталкивались.

Я уже говорил о том, что *имя* — это обозначение, которое предназначено для вас, для разработчика. А ещё оно предназначено для платформы. Именно потому, что имена использует платформа, они не могут быть «какими угодно». Они должны быть составлены по определённым правилам:

- имя — это всегда одно слово. В имени не может быть пробелов;
- имя всегда начинается с буквы;
- имя может содержать только буквы, цифры и символ подчёркивания «_»;
- в именах вместо буквы «ё» используется буква «е».

Раньше вы уже писали имена, но они всегда состояли из одного слова, и вопросов не возникало. Теперь вам понадобилось составить имя из нескольких слов. В таких случаях принято поступать просто.

Пробелы между словами выбрасывают, каждое слово начинают с прописной буквы, союзы, предлоги тоже выбрасывают. Таким образом, например, из названия «Учёт доходов и расходов» получается имя «УчетДоходовРасходов».

Если вы напишете имя по этим правилам, то из такого имени платформа сама сформирует правильный синоним. Разделит имя на отдельные слова и заменит прописные буквы строчными. Так получилось в данном случае, платформа сама заполнила синоним — *Учебный день*.

Этот синоним в единственном числе. Поэтому он хорошо подходит для представления объекта (одного документа). Значит, специально задавать представление объекта не нужно.

А для представления списка (нескольких документов) я задал представление во множественном числе — *Учебные дни*.

Все объекты конфигурации во многом похожи между собой. Документ имеет много общих черт со справочником. У справочника, вы знаете, есть два самых важных реквизита. Это *Код* и *Наименование*.

Так же и у документа есть два самых важных реквизита. Это *номер* и *дата*.

Номер у документа имеет такой же смысл, как и код у справочника. Это уникальный признак, который позволяет отличить один документ от другого. Он может быть числом или строкой. Платформа умеет самостоятельно присваивать документам новые номера и умеет следить за тем, чтобы номера не повторялись.

Дата документа указывает его положение на оси времени. Дата содержит в себе не только календарный день, но и время с точностью до секунды. Например, дата может выглядеть так: «01.09.2015 9:30:00». Это значит: «9 часов 30 минут первого сентября 2015 года».

Подумайте, какие из этих двух реквизитов, *Номер* и *Дата*, вам понадобятся.

Дата нужна, так как вам нужно указывать, к какому дню относится набор занятий, содержащийся в документе.

А номер не нужен. Потому что в один день у вас будет только один документ. Не может быть двух документов для одного дня, двух разных наборов занятий в один и тот же день. Поэтому каждый документ вы сможете совершенно точно отличить от другого по его дате.

Чтобы избавиться от номера, перейдите на закладку *Нумерация* и установите длину номера равной нулю (рисунок 2.169).

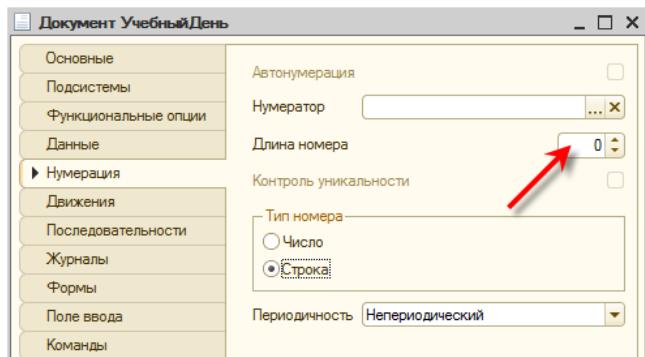


Рисунок 2.169. Длина номера равна нулю

Также это поле, *Номер*, нужно теперь вручную удалить из свойства *Ввод по строке* (рисунок 2.170). Вспомните, как вы делали это для справочника Кабинеты. Если забыли, можете посмотреть в разделе 2.9.9 «Наименование и код» на страницах 100–101.

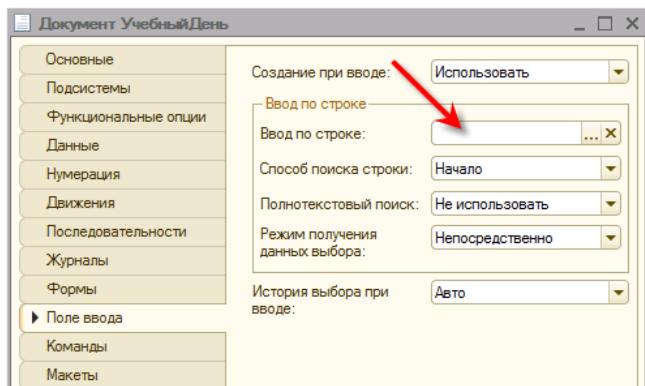


Рисунок 2.170. Свойство *Ввод по строке*

Теперь запустите конфигурацию в режиме отладки и посмотрите, что получилось.

В панели функций текущего раздела у вас появилась новая команда *Учебные дни*. Она открывает список учебных дней. Вы можете создать в этом списке новый учебный день. Его форма будет выглядеть так (рисунок 2.171).

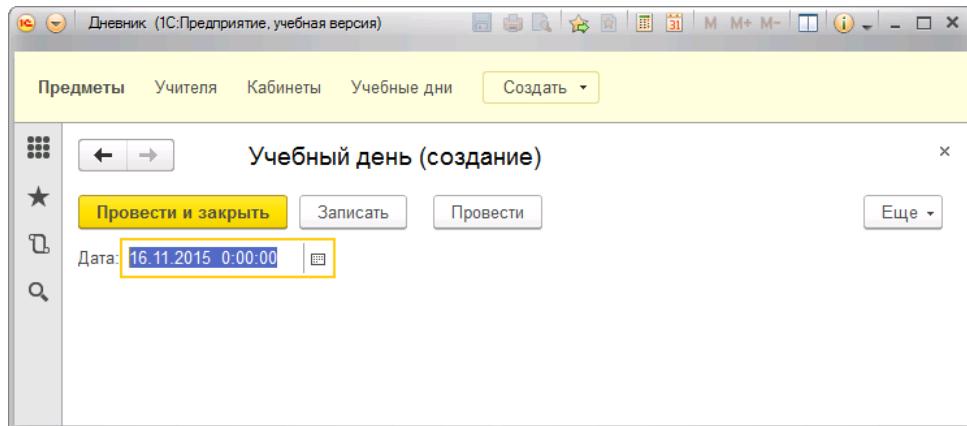


Рисунок 2.171. Форма нового учебного дня

Что вы хотите знать про этот учебный день кроме его даты? Вы хотите знать, какие занятия проходят у вас в этот день. Как их записать сюда?

Вспомните: я говорил, что каждая «вещь», которую вы хотите знать про объект данных, описывается в дереве конфигурации с помощью реквизита объекта конфигурации. Именно так появились реквизиты *Кабинет* и *Учитель* у справочника *Предметы*. Потому что про каждый предмет вы хотели знать, в каком кабинете проходят занятия и какой учитель их ведёт (рисунок 2.172).

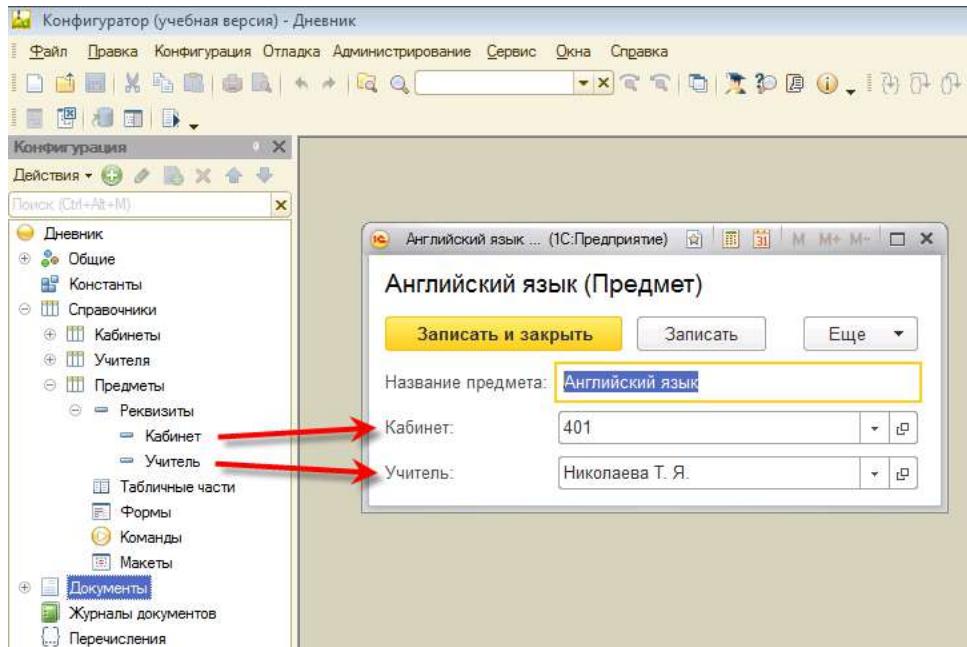


Рисунок 2.172. Реквизиты «Кабинет» и «Учитель»

Логично предположить, что точно так же нужно для каждого занятия создать отдельный реквизит в документе *Учебные дни*. Первый урок, второй урок, третий урок...

Но сколько таких реквизитов создавать? Сегодня может быть пять уроков. А завтра — шесть. А послезавтра вообще три. Каждый день количество уроков может быть разным, и вы не знаете его заранее.

Специально для таких случаев у объектов конфигурации в 1С:Предприятии есть удобная вещь, которая называется *табличная часть*. Если вы вспомните, что объект конфигурации — это комната, а объекты данных — это люди в комнате, то табличная часть — это портфель, сумка, которая есть у каждого человека в комнате (рисунок 2.173).

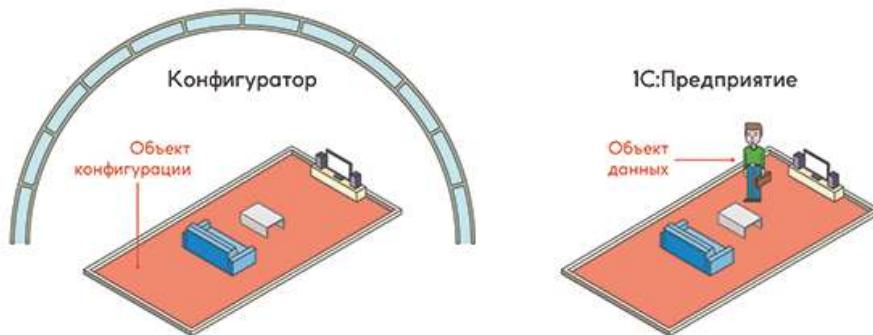


Рисунок 2.173. Портфель у каждого человека в комнате

В этих портфелях у них лежат однотипные вещи, например учебники. Но количество учебников у каждого человека может быть разное.

В вашем случае человек — это документ Учебный день, а в портфеле у него (в табличной части) будут храниться записи о том, какие занятия проходят в этот день. У каждого учебного дня количество записей в его табличной части может быть разным.

Закройте 1С:Предприятие, вернитесь в конфигуратор. Документу Учебный день добавьте табличную часть. Для этого у него (и у других объектов конфигурации) есть ветка Табличные части (рисунок 2.174).

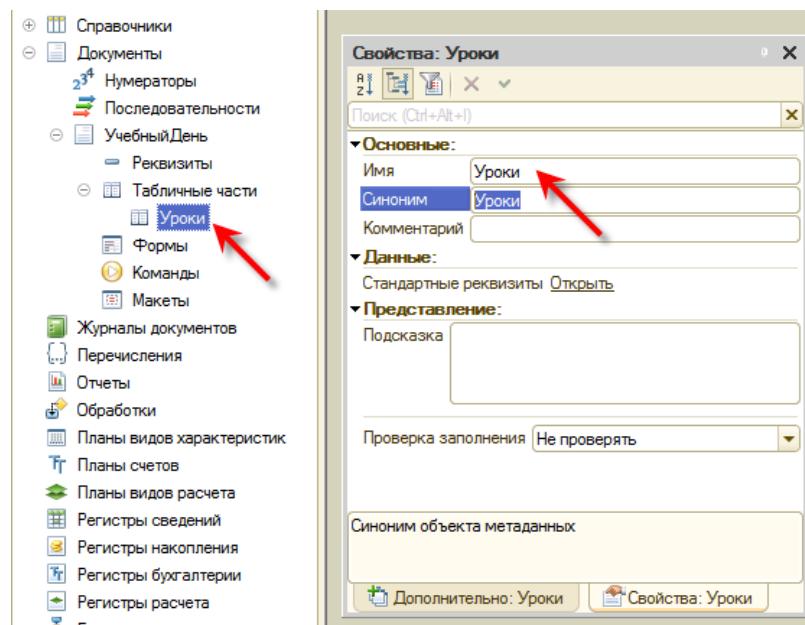


Рисунок 2.174. Табличная часть «Уроки»

Назовите табличную часть Уроки.

Портфель вы сделали. Теперь нужно каким-то образом «рассказать» 1С:Предприятию, что будет находиться в этом портфеле. Только что я говорил о том, что там будут находиться «записи о том, какие занятия проходят в этот день». Что представляет собой такая запись? Для этого нужно вспомнить дневник (рисунок 2.175).

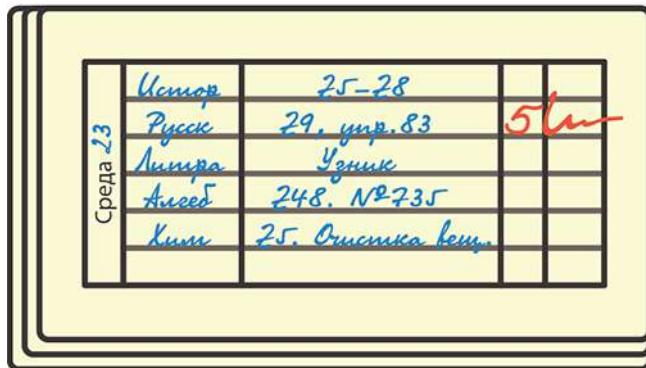


Рисунок 2.175. Расписание уроков на день

Смотрите. Кроме названия предмета каждая запись содержит:

- домашнее задание;
- оценку.

А иногда бывает ещё и комментарий учителя. Значит, теперь в табличную часть вам нужно добавить 4 реквизита:

- Предмет. Он будет ссылаться на один из предметов, которые есть в справочнике. Значит, нужно установить ему тип *СправочникСсылка.Предметы*.
- ДомашнееЗадание. Это просто строка текста, которая описывает то, что задано на дом. Значит, тип этого реквизита будет *Строка*.
- Оценка. Это число. От единицы до пяти. Конечно бывает так, что учитель ставит «4+» или «5-». Но сейчас будем считать, что учитель ставит только «целые» оценки. Поэтому тип этого реквизита будет *Число*.
- КомментарийУчителя. Это тоже будет просто строка текста. Значит, тип реквизита — *Строка*.

Добавить реквизиты вы вполне можете самостоятельно. А я сделаю то же самое, чтобы вы могли себя проверить.

Реквизит *Предмет* будет выглядеть следующим образом (рисунок 2.176).

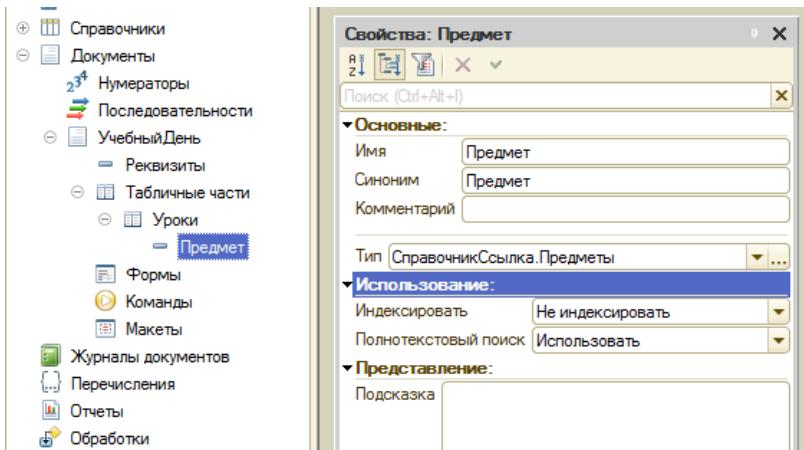


Рисунок 2.176. Реквизит «Предмет»

Имя реквизита — *Предмет*. Тип реквизита — *СправочникСсылка.Предметы*. Другие свойства изменять не нужно.

Реквизит для хранения домашнего задания будет выглядеть так (рисунок 2.177).

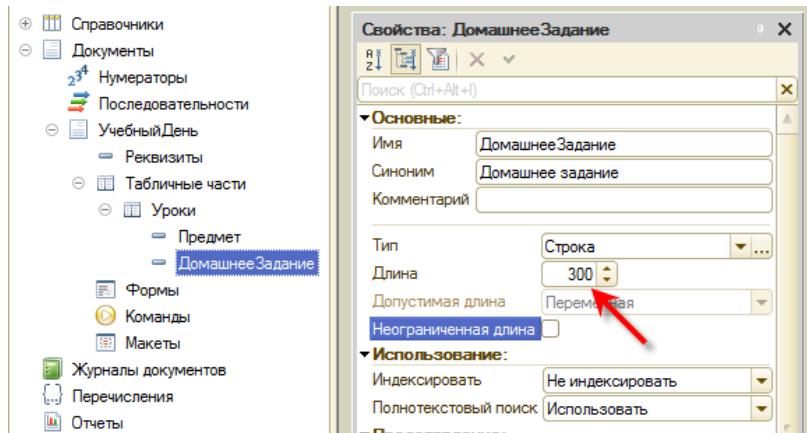


Рисунок 2.177. Реквизит «ДомашнееЗадание»

Имя реквизита — *ДомашнееЗадание*. Тип — *Строка*. И кроме этого я установил длину 300 символов. 300 символов вполне должно хватить для того, чтобы записать домашнее задание. Если вам этого мало — можете увеличить.

Заметка

Вы можете установить флажок *Неограниченная длина*. Тогда размер записи о домашнем задании не будет ограничен ничем. Но такой способ нужно использовать только в крайних случаях, когда размер очень большой (больше 1024 символов) или действительно может быть любым.

Например, если вы решите хранить в строковом реквизите сочинение на тему «Как я провёл лето», то для него, конечно, нужно будет установить неограниченную длину.

Реквизит *Оценка* будет выглядеть следующим образом (рисунок 2.178).

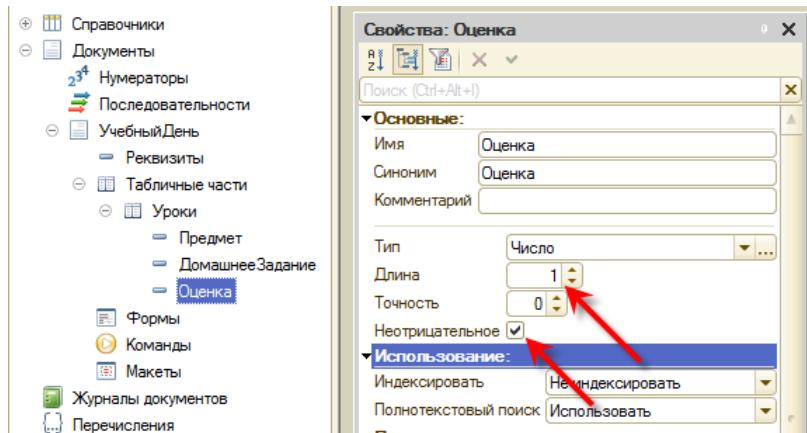


Рисунок 2.178. Реквизит «Оценка»

Имя реквизита — *Оценка*. Тип — *Число*. Кроме этого я установил длину (1 разряд) и флажок *Неотрицательное*. Потому что оценка не может быть отрицательной и не может состоять из нескольких разрядов.

И, наконец, реквизит для хранения комментария учителя будет выглядеть так (рисунок 2.179).

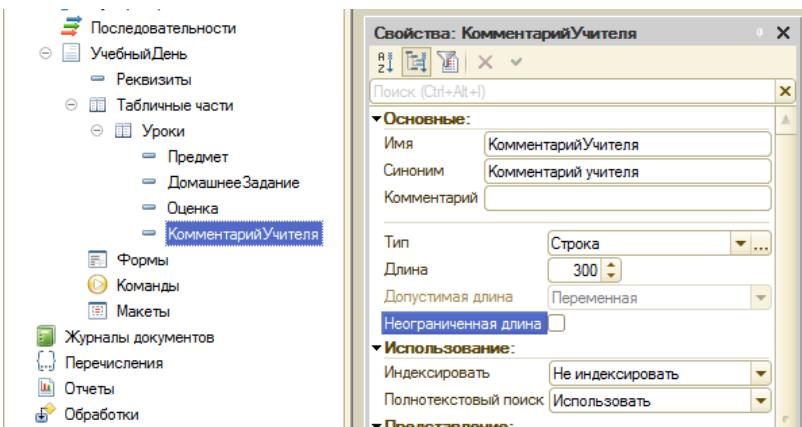


Рисунок 2.179. Реквизит «КомментарийУчителя»

Имя реквизита — *КомментарийУчителя*. Тип — *Строка*. Длина — 300.

Теперь запустите конфигурацию в режиме отладки, откройте список учебных дней и создайте новый учебный день (рисунок 2.180).

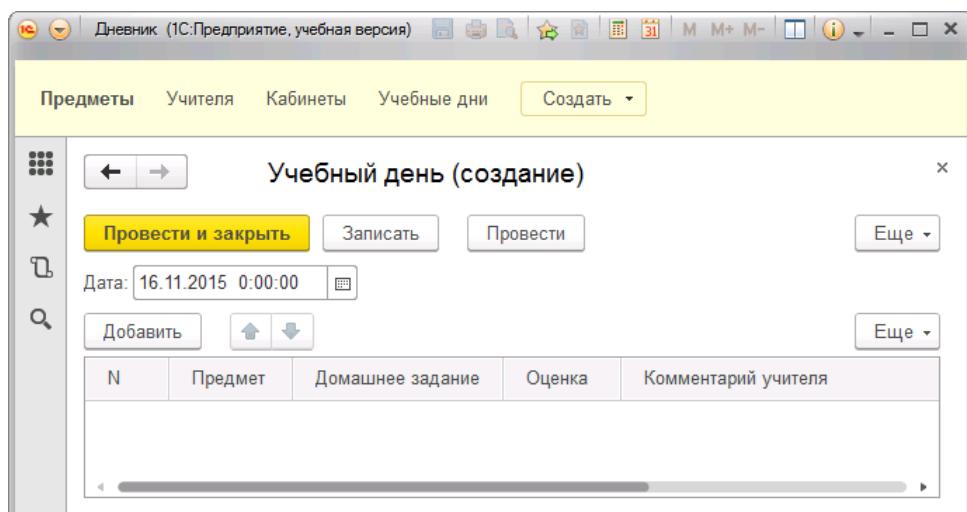


Рисунок 2.180. Учебный день

Совет

Может случиться так, что не все колонки оказались вам видны. Например, потому что колонка *Домашнее задание* очень широкая. Тогда вы можете выбрать для себя один из двух вариантов, какой вам больше нравится.

Вы можете раскрыть окно 1С:Предприятия на весь экран.

Или вы можете оставить размер окна таким, какой он есть, и отрегулировать ширину колонок. Чтобы изменить ширину колонки, достаточно потянуть мышью её границу (рисунок 2.181).

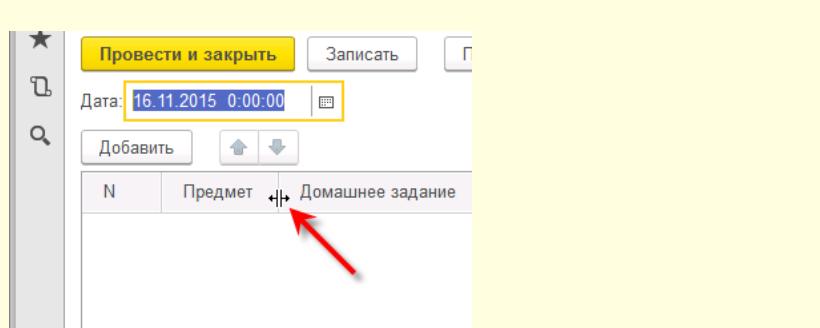


Рисунок 2.181. Изменение ширины колонки

А для того чтобы увидеть колонки, которые не поместились в видимую область, вы можете прокрутить табличную часть вправо, потянув мышью за бегунок внизу (рисунок 2.182).



Рисунок 2.182. Прокрутка табличной части

Стало гораздо интереснее! Теперь в документе есть таблица с колонками. В таблице нет ни одной строки, но вы можете их добавить.

Пусть, для примера, это будет первое сентября. Сначала установите дату.

Дату, конечно, можно написать вручную прямо в поле *Дата*. Но проще всего нажать кнопку выбора, выбрать, месяц *Сен*, а затем дату — 1 (рисунок 2.183).

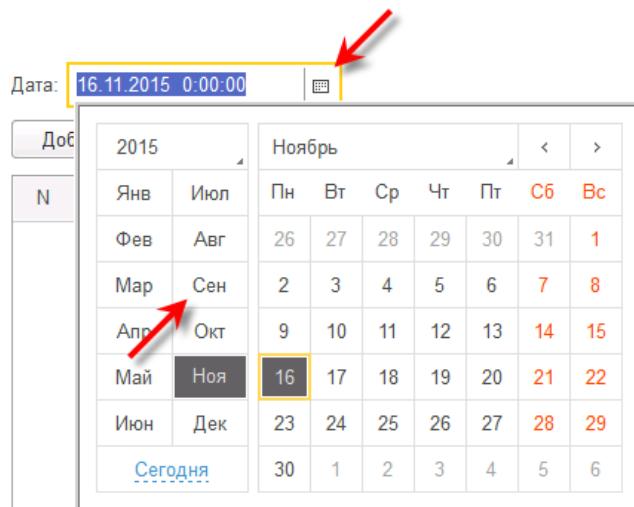


Рисунок 2.183. Ввод даты

После этого нажмите *Добавить* (рисунок 2.184) и введите предмет *Кл. час.*

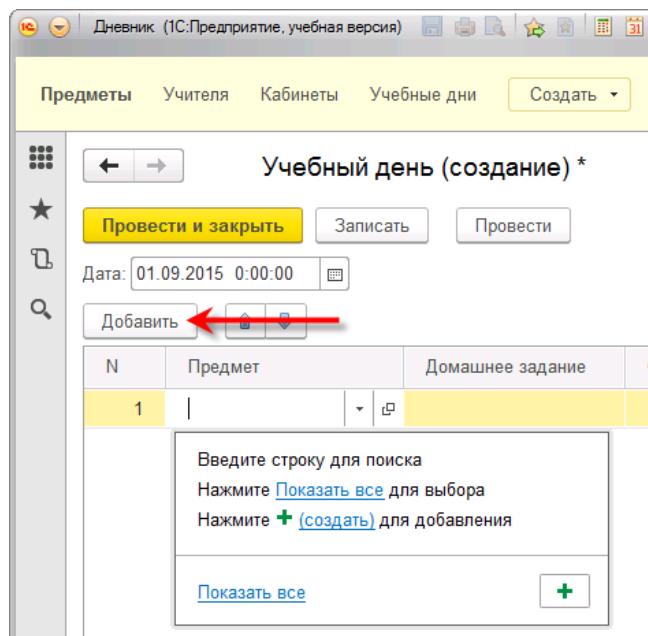


Рисунок 2.184. Ввод новой строки

Домашних заданий, оценок и комментариев учителя первого сентября не будет. Поэтому четыре раза нажмите клавишу TAB, чтобы пропустить эти поля. 1С:Предприятие автоматически добавит новую строку и снова предложит выбрать предмет (рисунок 2.185).

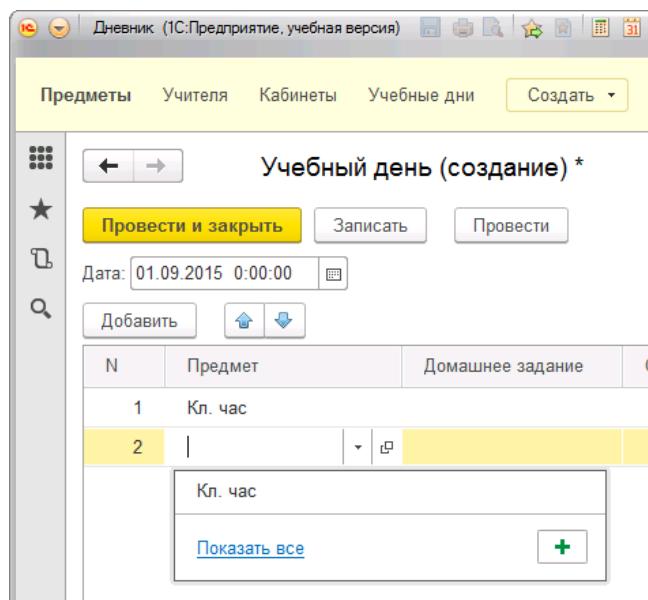


Рисунок 2.185. Автоматический ввод новой строки

Точно таким же образом добавьте предметы:

- Кл. час,
- Английский язык,
- Музыка,
- Математика,
- Русский язык.

Когда у вас появится седьмая строчка (рисунок 2.186), не выбирайте предмет, а нажмите два раза клавишу Esc, чтобы завершить редактирование.

N	Предмет	Домашнее задание	Оценка
1	Кл. час		
2	Кл. час		
3	Английский язык		
4	Музыка		
5	Математика		
6	Русский язык		
7	Русский язык Математика Музыка Английский язык		

Рисунок 2.186. Завершение редактирования

Итого первого сентября у вас должно получиться шесть предметов (рисунок 2.187).

N	Предмет	Домашнее задание
1	Кл. час	
2	Кл. час	
3	Английский язык	
4	Музыка	
5	Математика	
6	Русский язык	

Рисунок 2.187. Шесть предметов первого сентября

В табличной части всегда существует определённый порядок строк. Поэтому у каждой строки в первой колонке указан её номер. Если вы захотите поменять порядок строк, то для этого есть две команды, которые находятся рядом с кнопкой *Добавить* (рисунок 2.188).

N	Предмет	Домашнее задание
1	Кл. час	
2	Кл. час	
3	Английский язык	
4	Музыка	
5	Математика	
6	Русский язык	

Рисунок 2.188. Команды перемещения строк

Теперь нужно сохранить данные, которые вы ввели. Тут есть особенность. Документ отличается от справочника. Вспомните: чтобы сохранить данные элемента справочника, вы нажимали *Записать и закрыть* (рисунок 2.189).

Предмет (создание)		
Записать и закрыть	Записать	Еще ▾
Название предмета: <input type="text"/>		
← → Учебный день (создание) * → x		
Провести и закрыть	Записать	Провести
Дата: 01.09.2015 0:00:00 <input type="button"/>		

Рисунок 2.189. Команды справочника и документа

Это главная, основная команда для элемента справочника. Команда, которая используется чаще всего. Поэтому она выделена жёлтым цветом и называется *кнопка по умолчанию*.

У документа самой главной является другая команда — *Провести и закрыть*. Что значит «проводи», я расскажу позже, когда вы до конца сконструируете документ (смотрите в разделе 5.1 «Зачем нужны регистры» на странице 376). А сейчас правильнее будет нажать *Записать*, а потом закрыть форму, нажав на крестик.

Если вы нажмёте *Провести и закрыть*, ничего страшного не случится, данные тоже сохранятся. Но лучше пока этого не делать.

Вот так выглядит в списке первый учебный день 1 сентября (рисунок 2.190).

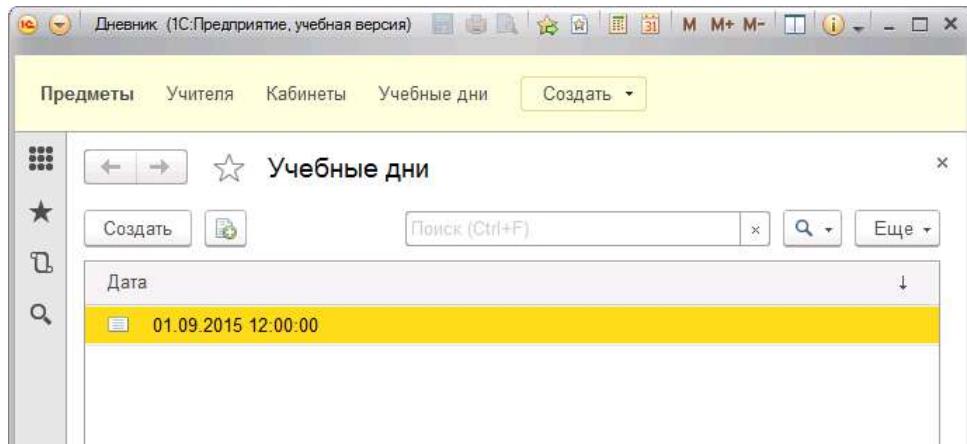


Рисунок 2.190. Учебный день в списке

Итак, чтобы вы хорошо запомнили, как создаётся и выглядит табличная часть, посмотрите ещё раз на дерево конфигурации и документ (рисунок 2.191).

N	Предмет	Домашнее задание	Оценка	Комментарий учителя
1	Кл. час			
2	Кл. час			
3	Английский язык			
4	Музыка			
5	Математика			
6	Русский язык			

Рисунок 2.191. Табличная часть в конфигураторе и документе

Табличная часть Уроки — это таблица, которая будет внутри каждого документа. А реквизиты табличной части — это, по сути, колонки, которые будут у этой таблицы.

Теперь вы можете самостоятельно добавить оставшиеся дни первой недели сентября. Если хотите, можете заполнять их своими данными. А если хотите, можете взять готовую информационную базу «02 ПерваяНеделяСентября.dt», в которой все эти данные есть. Как её подключить, написано в разделе A.1 «Как подключить демонстрационную базу» на странице 543.

2.14 Редактирование форм

Информационная база

Прикладное решение, которое должно получиться у вас к этому моменту, содержится в демонстрационной базе «02 ПерваяНеделяСентября.dt». Как её подключить, написано в разделе [A.1 «Как подключить демонстрационную базу»](#) на странице [543](#).

Сейчас вы будете продвигаться ещё глубже «внутрь» 1С:Предприятия. Поэтому, прежде чем «бросаться в бой», сообразите, где вы находитесь.

Перед этим вы стояли рядом с деревом и изучали большие ветки, которые прикрепляются к стволу. Таких веток вы изучили две: справочники и документы.

А сейчас вы заберёtesь на одну из этих крупных веток и займёtesь изучением более мелких веток, которые крепятся к ней (рисунок [2.192](#)).

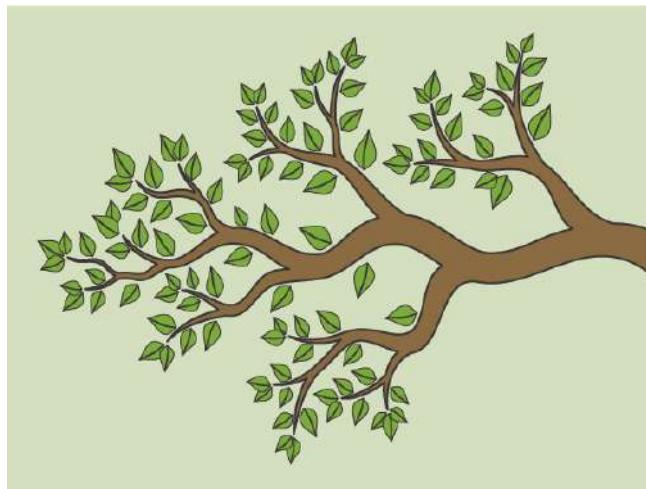


Рисунок 2.192. Сейчас вы тут

Сначала нужно понять, почему вообще вдруг вы «полезли на ветку». Понятно, что я вас об этом попросил. Но я это сделал не просто так.

Ваше прикладное решение хорошее, но недостаточно удобное. Хочется сделать его более красивым и дружественным.

Что в нём можно улучшить? Я покажу вам несколько таких моментов.

Во-первых, когда вы запускаете прикладное решение, вы видите пустую рабочую область (рисунок [2.193](#)).

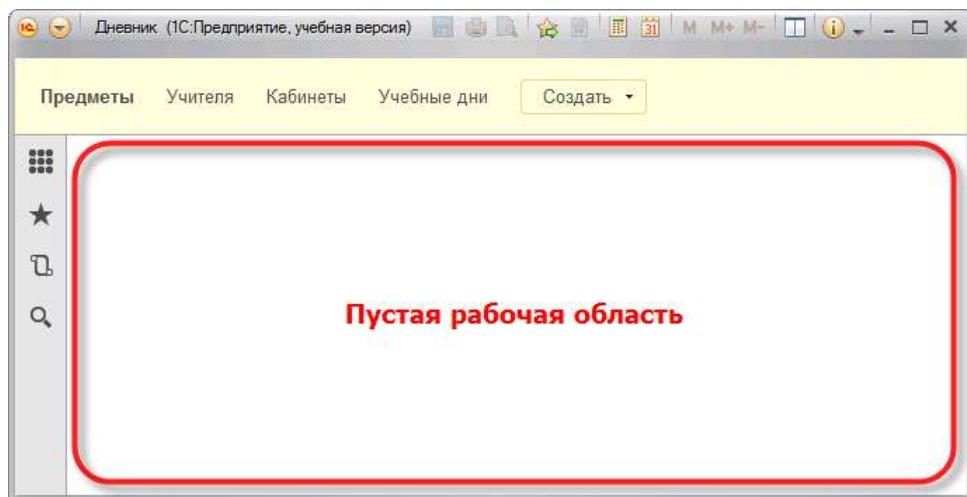


Рисунок 2.193. Пустая рабочая область

Это некрасиво. Хочется, чтобы она была заполнена чем-нибудь полезным. Например, чтобы на ней был список учебных дней. Ведь именно с ними вы работаете чаще всего.

Во-вторых, в списке учебных дней каждый из них обозначен неудобно (рисунок 2.194).

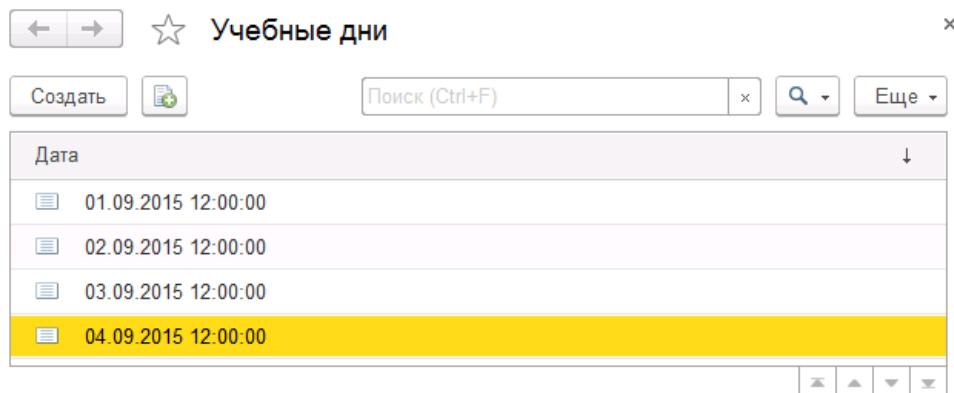


Рисунок 2.194. Список учебных дней

Время вам знать не нужно. Для одной даты всегда будет только один документ. И его время значения не имеет.

Кроме того, дата документа показана неудобно. Хочется, чтобы месяц был написан словами, и не помешало бы знать, какой это день недели.

В-третьих, в форме учебного дня мало информации (рисунок 2.195).

The screenshot shows a configuration interface for a 'School Day'. At the top, there are navigation buttons (back, forward, star), a title 'Учебный день от 01.09.2015 12:00:00', and several action buttons: 'Провести и закрыть' (Post and Close), 'Записать' (Save), 'Провести' (Post), and 'Еще' (More). Below these are buttons for 'Добавить' (Add), 'Up', and 'Down'. A date field shows '01.09.2015 12:00:00' with a calendar icon. The main area is a table with columns: N (Number), Предмет (Subject), Домашнее задание (Homework), Оценка (Grade), and Комментарий учителя (Teacher's Comment). The table contains six rows, each with a subject name: 1. Кл. час, 2. Кл. час, 3. Английский язык, 4. Музыка, 5. Математика, and 6. Русский язык.

N	Предмет	Домашнее задание	Оценка	Комментарий учителя
1	Кл. час			
2	Кл. час			
3	Английский язык			
4	Музыка			
5	Математика			
6	Русский язык			

Рисунок 2.195. Форма учебного дня

Сейчас вы видите только предметы. А ведь вы говорили, что в расписании занятий хочется видеть и номер кабинета, в котором проходит урок, и учителя, который этот урок ведёт.

Всё это можно исправить. Но для этого вам понадобится создать собственные формы для документов *УчебныйДень*.

2.14.1 Добавление формы

До сих пор вы использовали автогенерируемые формы. Платформа создаёт их сама. Они удобные. Но, конечно же, платформа не может на 100% предугадать ваши желания и ваши потребности. Поэтому, если вы хотите, чтобы форма выглядела по-другому, нужно добавить в конфигурацию собственную форму. И сделать её такой, как вам нужно.

Начните с простого. Перейдите в конфигуратор. Документу *УчебныйДень* добавьте форму списка.

Как добавлять элементы в дерево объектов конфигурации, вы знаете. После выполнения команды откроется *конструктор формы* (рисунок 2.196).

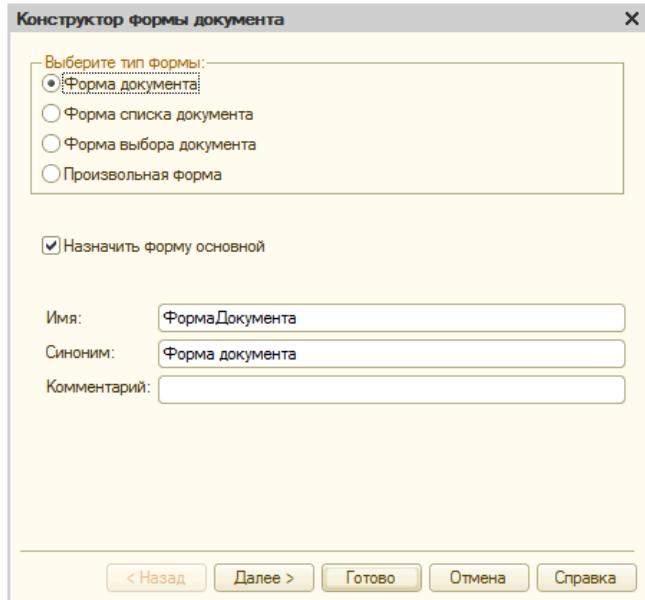


Рисунок 2.196. Конструктор формы

Он совсем простой. В подавляющем большинстве случаев здесь достаточно выбрать тип формы и нажать *Готово*. Сделайте это самостоятельно, выбрав пункт *Форма списка документа*.

После этого в дереве конфигурации у документа *УчебныйДень* появится форма списка. Эту форму платформа сконструировала сама, и она очень похожа на автогенерируемую форму. Так сделано для того, чтобы вам не нужно было создавать свою форму «с нуля». Это может быть довольно трудоёмко. Гораздо проще внести изменения в готовую форму.

А в рабочей области будет открыт редактор этой формы (рисунок 2.197). Чтобы вы сразу могли изменить её.

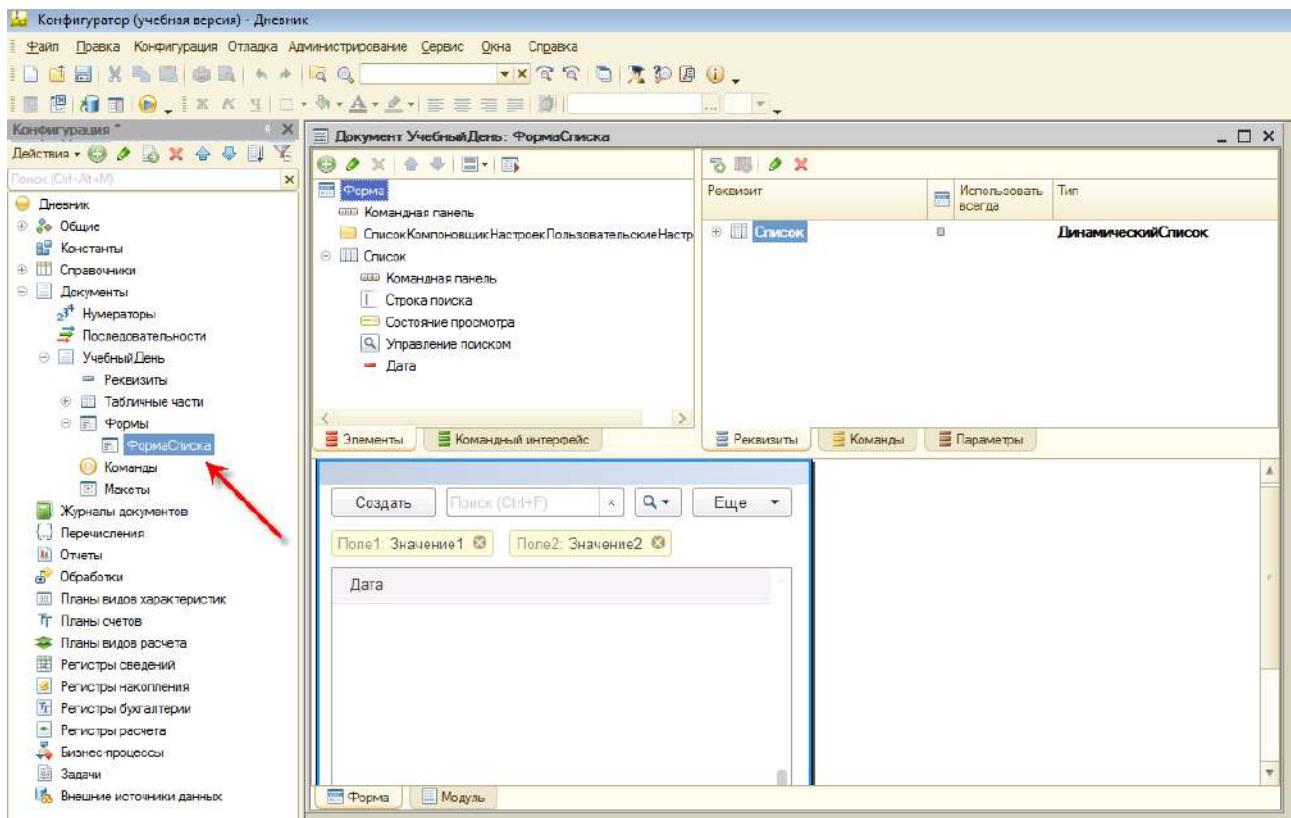


Рисунок 2.197. Редактор формы

Уже совсем скоро я расскажу, как пользоваться этим редактором. А пока закройте его. Сначала вы используете эту собственную форму, не внося в неё никаких изменений.

Как платформа узнает, что теперь она должна открывать вашу форму, а не генерировать свою? По имени формы? Нет.

Откройте окно редактирования документа *УчебныйДень* и перейдите на закладку *Формы* (рисунок 2.198).

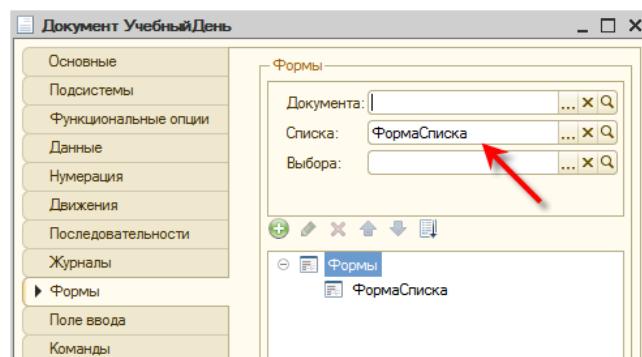


Рисунок 2.198. Документ «УчебныйДень: Формы»

У документа есть три свойства, в которых можно указать, какую форму должна использовать платформа при отображении данных этого документа. Если никакая форма не указана, то платформа будет самостоятельно генерировать форму. А если форма указана, то будет использоваться именно она.

Когда вы добавляли документу *УчебныйДень* форму списка, платформа сразу же представила её в это свойство документа. Такие формы, которые вы создали сами и которые указаны в свойствах объекта конфигурации, называются *основными формами*.

Основные они потому, что кроме них могут быть и другие формы. Такие, которые платформа самостоятельно не использует. Которые открываете только вы сами с помощью встроенного языка. Позже вы научитесь это делать.

А сейчас сделайте так, чтобы после запуска вашего прикладного решения вы видели не пустую рабочую область, а список учебных дней. Сделать это просто. Нужно сказать, чтобы платформа показывала в этом месте форму списка справочника Учебные Дни. Почему вы не могли сделать этого раньше? Да просто потому, что там нельзя указать автогенерируемую форму. Обязательно нужна форма, которую вы добавили сами.

Теперь у вас такая форма есть. Поместите её туда. «Куда туда?» — спросите вы. Объясню.

Когда я рассказывал про интерфейс 1С:Предприятия, я сказал, что после запуска прикладного решения всегда открывается основной раздел (рисунок 2.199).



Рисунок 2.199. Основной раздел

И вы даже умеете изменять командный интерфейс основного раздела. Но вы не интересовались тем, что может находиться у него в рабочей области.

Теперь пришло время об этом сказать. В рабочей области основного раздела может находиться *начальная страница* (рисунок 2.200).

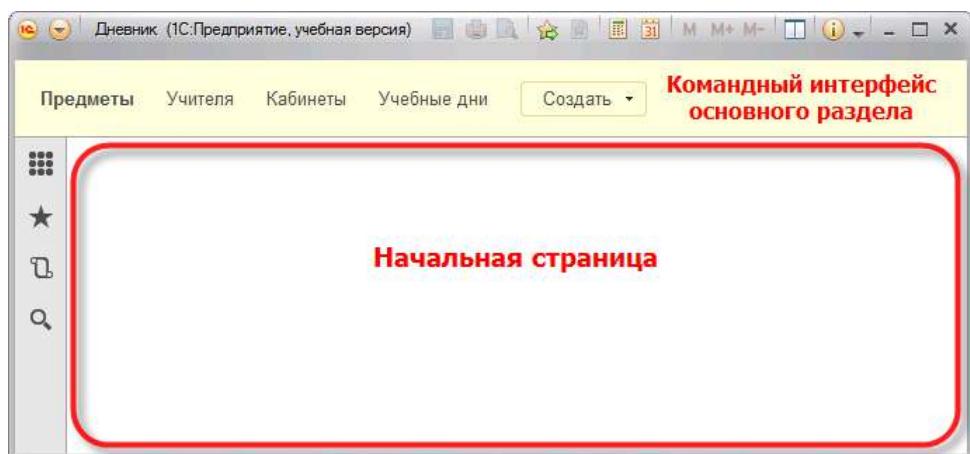


Рисунок 2.200. Начальная страница

Если начальная страница задана, после запуска прикладного решения будет показана именно она.

Задайте её. Для этого вызовите контекстное меню в корне конфигурации и выполните команду *Открыть рабочую область начальной страницы* (рисунок 2.201).

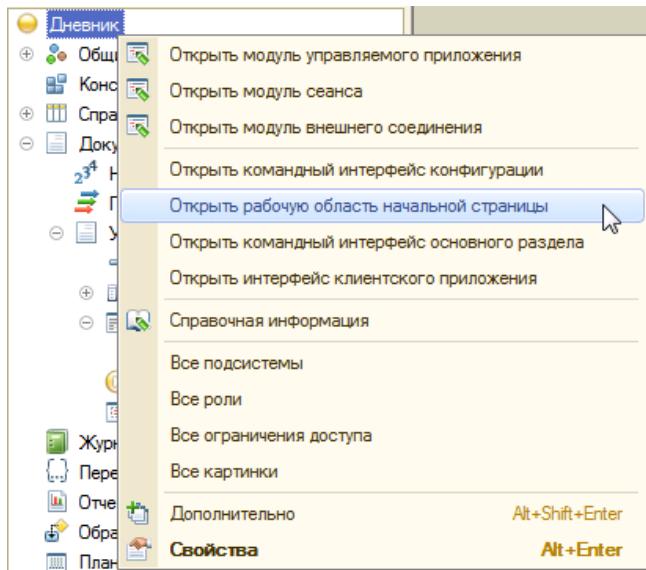


Рисунок 2.201. Открыть рабочую область начальной страницы

Откроется *редактор рабочей области начальной страницы* (рисунок 2.202).

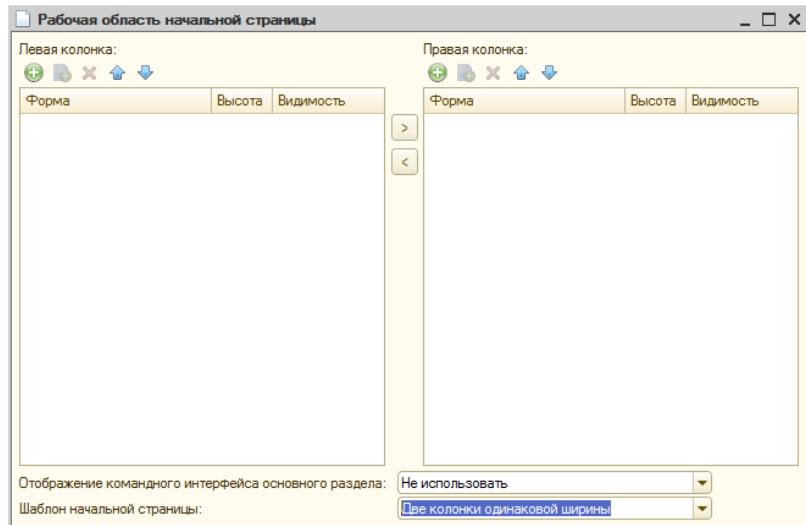


Рисунок 2.202. Редактор рабочей области начальной страницы

Начальная страница может состоять из одной колонки, а может из двух: правой и левой. Стандартно редактор предлагает использовать две колонки.

Сейчас вам две колонки не понадобятся. Но исправлять это нет необходимости. Пустую колонку платформа показывать не будет. Поэтому пусть остаются две колонки.

Форму списка документа УчебныйДень добавьте в правую колонку. Чтобы добавить форму, нажмите зелёную кнопку *Добавить* (рисунок 2.203).

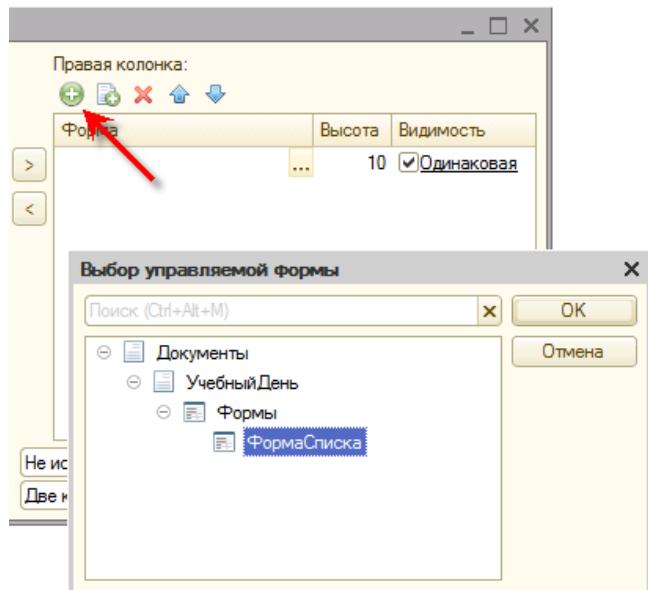


Рисунок 2.203. Добавление формы в рабочую область

Выберите форму списка документа УчебныйДень.

Нажмите **OK**.

После этого закройте окно редактора (рисунок 2.204).

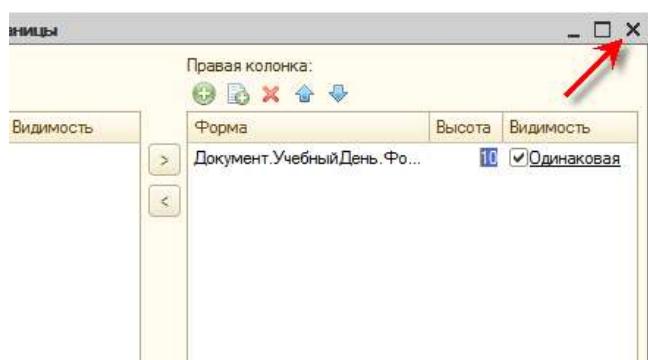


Рисунок 2.204. Закрыть окно редактора

Запустите конфигурацию в режиме отладки, и вы сразу же увидите список учебных дней на начальной странице (рисунок 2.205).

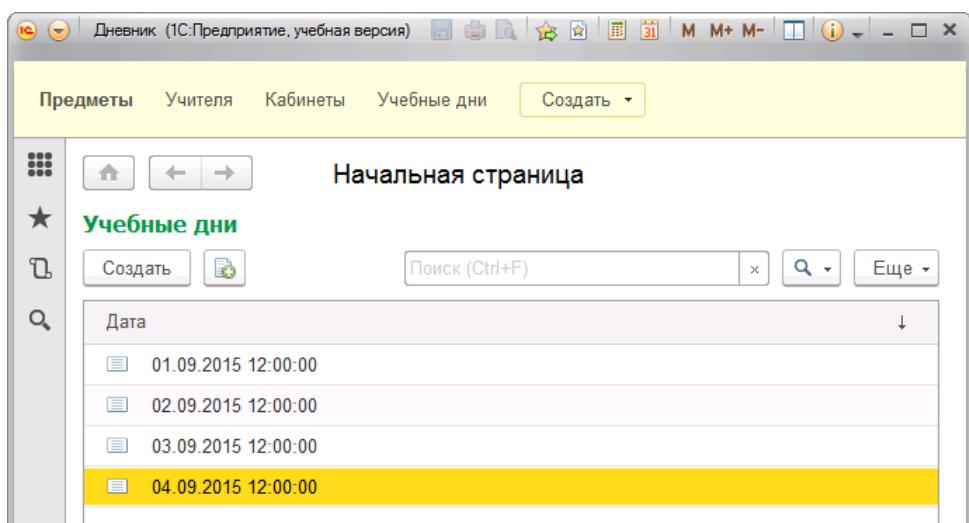


Рисунок 2.205. Список учебных дней на начальной странице

С первой задачей вы справились. Начальная страница не пустая.

Теперь можно заняться второй задачей. Чтобы в списке учебных дней каждый из них был обозначен более понятно и удобно, чем сейчас.

Прежде чем это сделать, вам нужно хотя бы в общих чертах познакомиться с тем, как устроен редактор формы.

2.14.2 Редактор формы

Закройте 1С:Предприятие, вернитесь в конфигуратор. Откройте в редакторе форму списка, которая есть у документа УчебныеДни. Для этого нужно дважды щёлкнуть на ней мышью (рисунок 2.206).

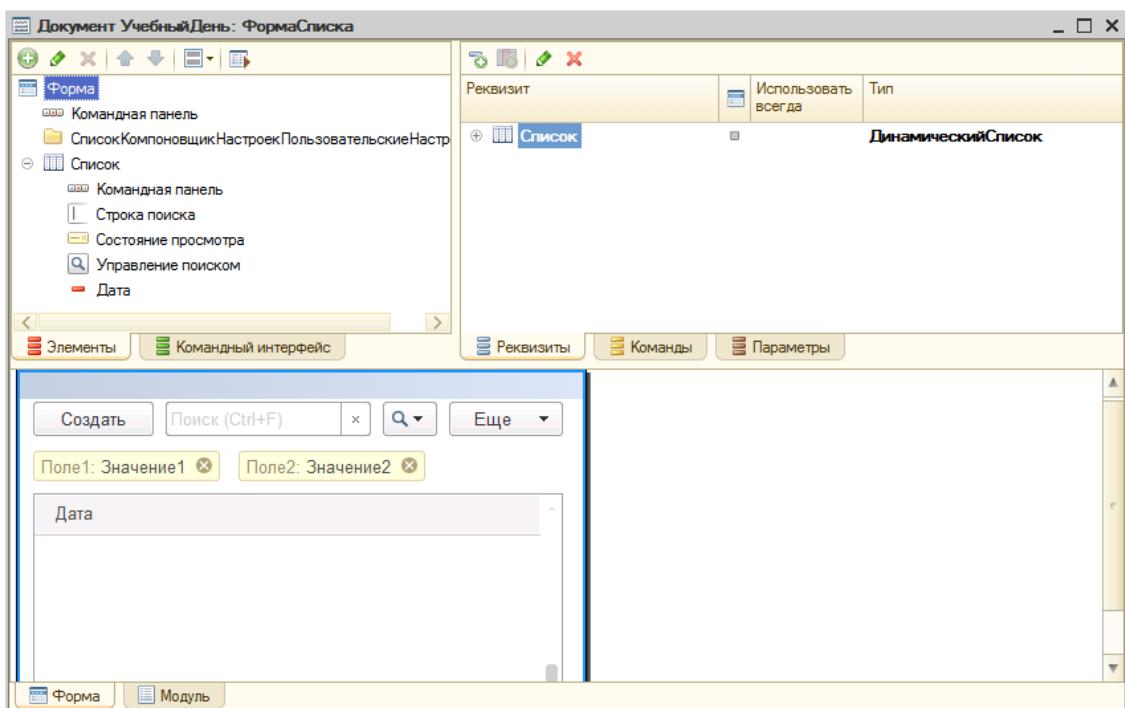


Рисунок 2.206. Редактор формы

Вообще, форма очень похожа на дом с окнами (рисунок 2.207).



Рисунок 2.207. Дом с окнами

В доме есть окна, за окнами горят лампочки. У каждого окна есть своя лампочка. Окна разные, лампочки тоже разные. Поэтому где-то окно прямоугольное и оранжевое, а где-то окно овальное и белое.

В форме тоже есть свои «окна» и свои «лампочки» за этими окнами. Лампочки — это *реквизиты* формы. Они содержат данные, которые должны быть показаны в форме (рисунок 2.208).

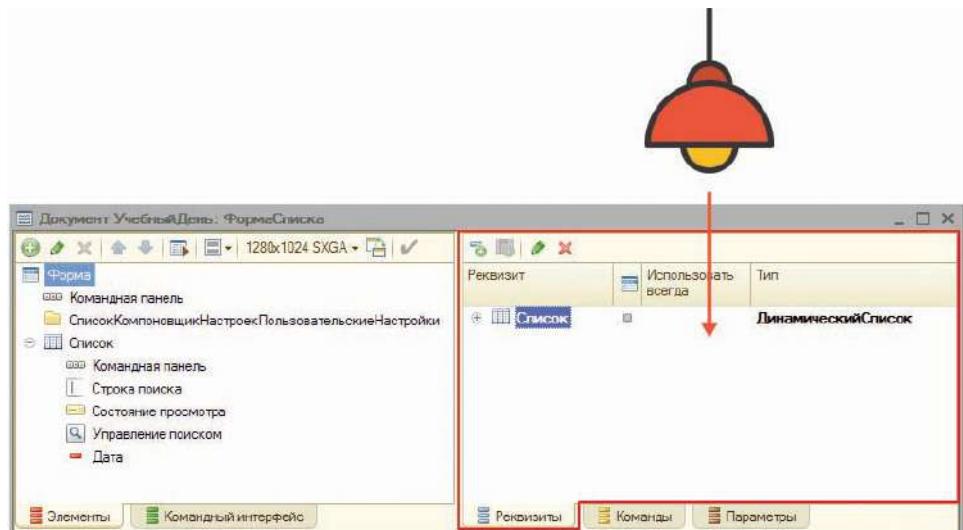


Рисунок 2.208. Реквизиты формы

Все реквизиты формы перечислены в правом верхнем окне редактора, на закладке *Реквизиты*. Сейчас в вашей форме всего один реквизит. Он называется *Список*. Он содержит все документы УчебныйДень, которые есть в вашей базе.

Окна — это элементы *формы*. Они показывают пользователю то, что находится в реквизитах. А кроме этого элементы позволяют пользователю изменить значение, находящееся в реквизите (рисунок 2.209).

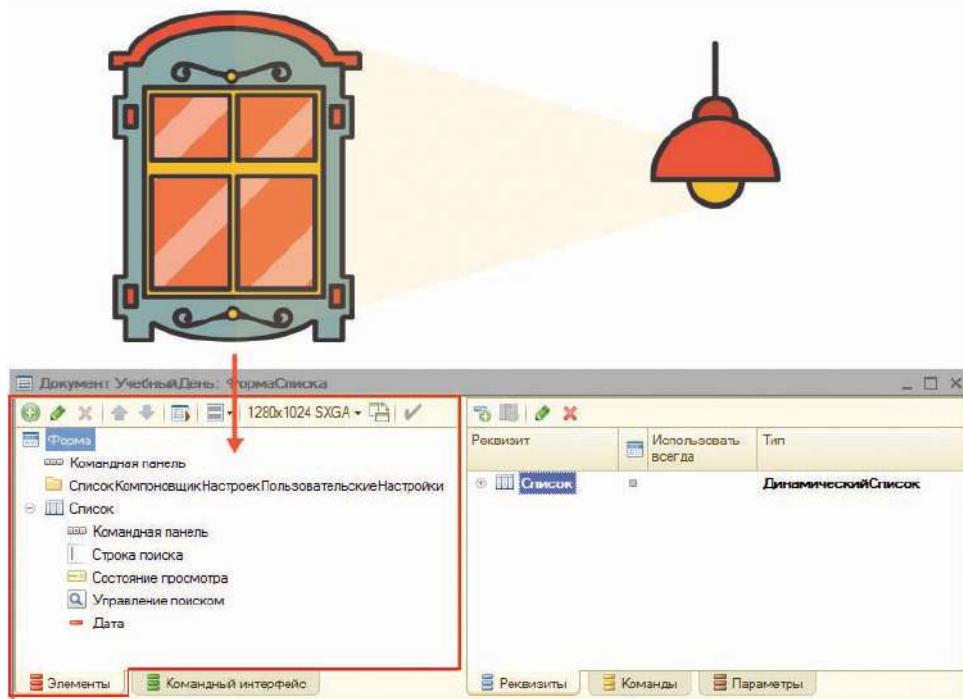


Рисунок 2.209. Элементы формы

Все элементы формы перечислены в левом верхнем окне редактора, на закладке Элементы. Сейчас в вашей форме всего один элемент. Он тоже называется *Список*. Он умеет показывать список каких-нибудь объектов данных. В данном случае — список документов УчебныйДень.

Когда вы смотрите на дом, за каждым окном горит какая-то лампочка. Точно так же каждый реквизит формы связан с каким-то элементом. Для этого у элемента есть свойство *ПутьКДанным*.

Вы можете выделить элемент *Список* и в палитре свойств посмотреть, какое значение у этого свойства (рисунок 2.210). Если у вас палитра свойств закрыта, открыть её можно из контекстного меню на элементе *Список* командой *Свойства*.

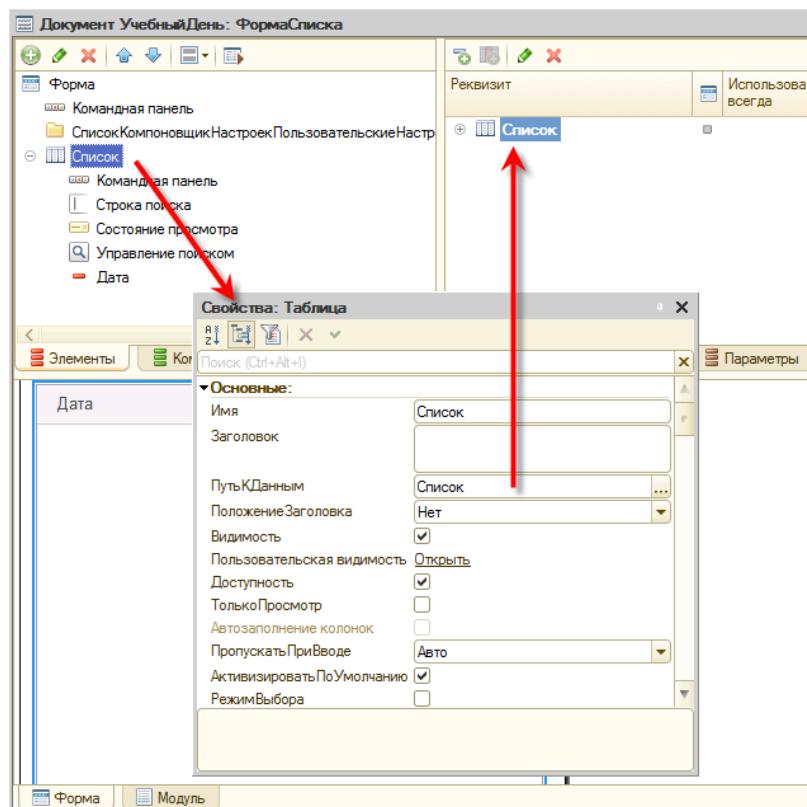


Рисунок 2.210. Свойство «ПутьКДанным»

В нижнем окне редактора формы показывает, как примерно, будет выглядеть форма, когда пользователь увидит её в режиме 1С:Предприятие (рисунок 2.211).

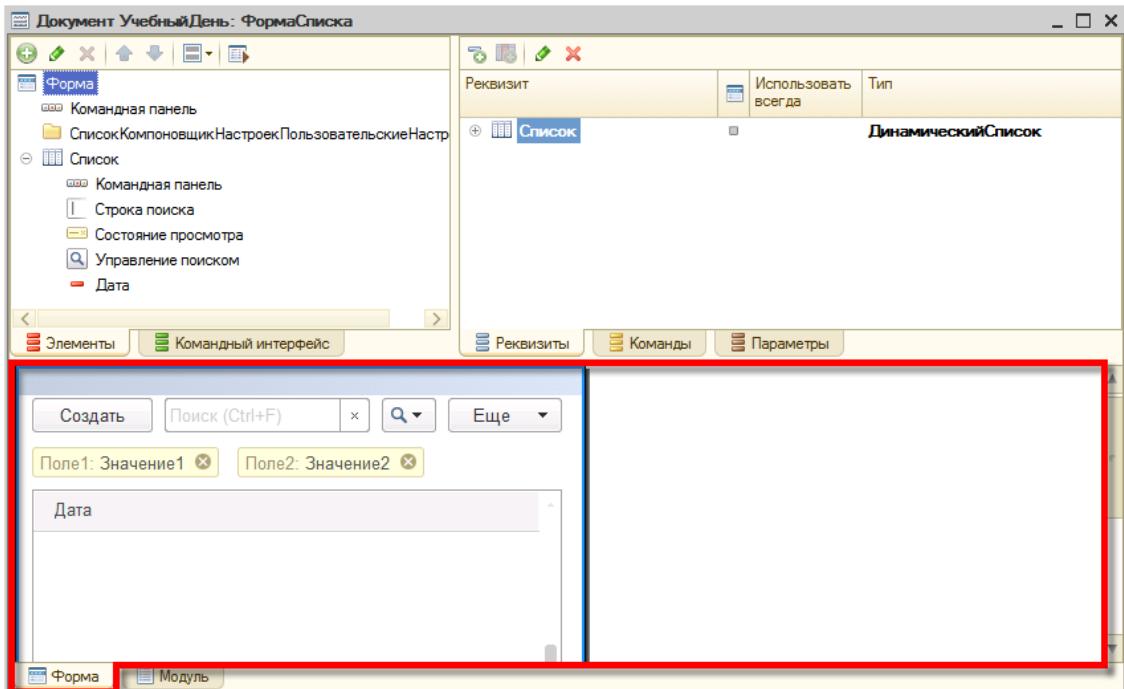


Рисунок 2.211. Внешний вид формы

Подробнее

Подробнее вы можете прочитать о формах в документации «[Руководство разработчика 8.3. Глава 7. Формы](#)».

Теперь обратите внимание на одну интересную особенность. Платформа сама, без вашего участия, размещает элементы в форме некоторым образом. Таким образом, чтобы ими было удобно пользоваться. То есть вам не нужно указывать, в каком именно месте формы какой элемент должен находиться. Какую точную высоту или ширину он должен иметь.

У платформы есть список элементов (наверху), и она сама по некоторым правилам размещает их в форме (внизу). Как она это делает? Это интересный момент.

Проще всего представить, что вся форма состоит из «строк» и «колонок». Например, сама форма может быть либо «строкой», либо «колонкой». Если вы выделите в дереве элементов форму (корень дерева), то в палитре свойств увидите, что у неё есть свойство **Группировка** (рисунок 2.212).

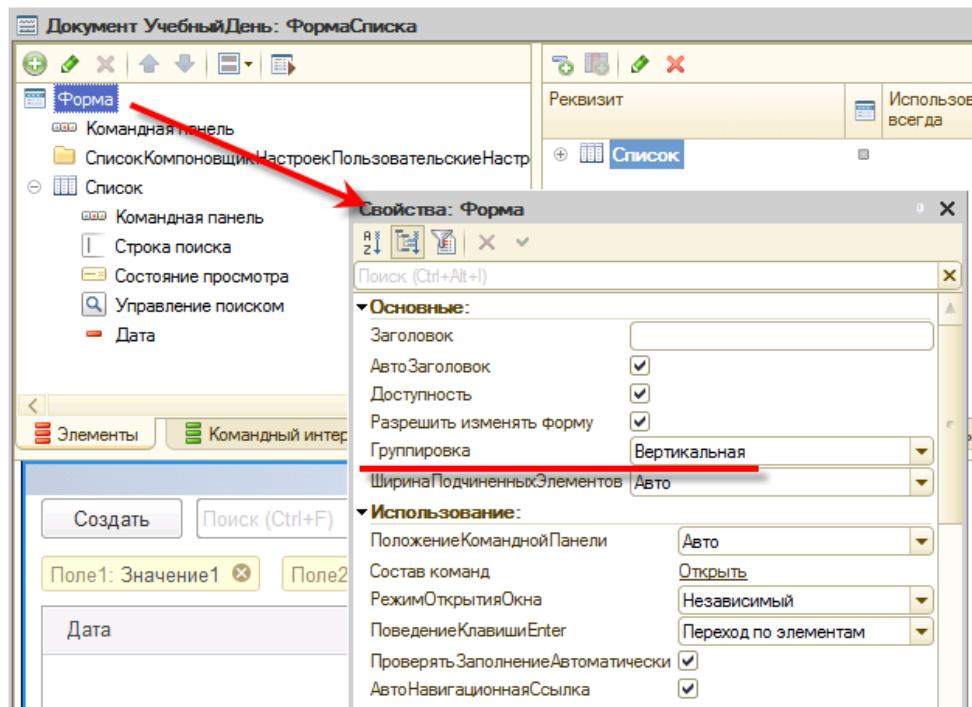


Рисунок 2.212. Свойство формы «Группировка»

Сейчас оно имеет значение *Вертикальная*. Что это значит?

Это значит, что форма сама сейчас является «колонкой». И если вы поместите в неё несколько элементов, то они будут расположены один под другим (рисунок 2.213).

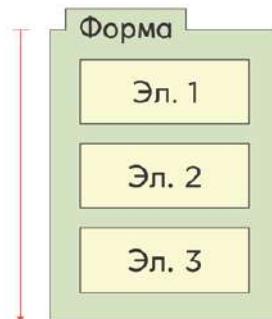


Рисунок 2.213. Группировка «Вертикальная»

А если вы измените это свойство на *Горизонтальная*, то сама форма станет «строкой». И те же самые элементы будут расположены уже друг за другом, слева направо (рисунок 2.214).

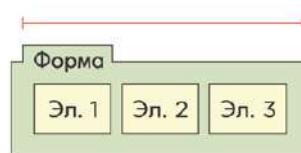


Рисунок 2.214. Группировка «Горизонтальная»

Не только сама форма может быть «строкой» или «колонкой». Внутри формы вы тоже можете сделать «строки» и «колонки». Для этого служит элемент, который называется *Группа*.

Чаще всего его не видно. Потому что он используется для того, чтобы собрать вместе несколько элементов формы. Чтобы расположить эти элементы в виде «строки» или в виде «колонки».

Например, сама форма может иметь вертикальную группировку, то есть быть «колонкой». И внутри неё будут расположены два элемента и группа (рисунок 2.215).

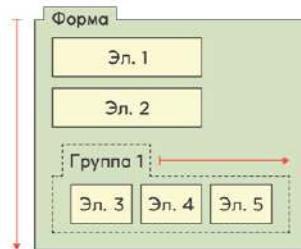


Рисунок 2.215. Форма с группой

А группа может иметь горизонтальную группировку, то есть быть «строкой». И тогда элементы, находящиеся в этой группе, будут расположены друг за другом.

Получается *иерархия*. То есть подчинение одних элементов другим.

Элемент1, Элемент2 и Группа1 подчинены форме. А Элемент3, Элемент4 и Элемент5 подчинены элементу группы Группа1.

Или, если рассматривать эту конструкцию в другую сторону, то форма состоит из элемента 1, элемента 2 и группы 1. А группа 1 состоит из элемента 3, элемента 4 и элемента 5.

Чтобы представить иерархию компактным образом, удобно использовать дерево. Например, в моём примере иерархическое дерево элементов будет выглядеть следующим образом (рисунок 2.216).

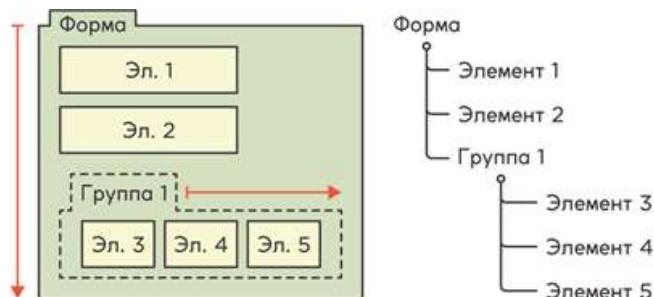


Рисунок 2.216. Иерархия элементов формы

Так вот, в редакторе формы элементы перечислены не просто списком, а в виде такого дерева (рисунок 2.217).

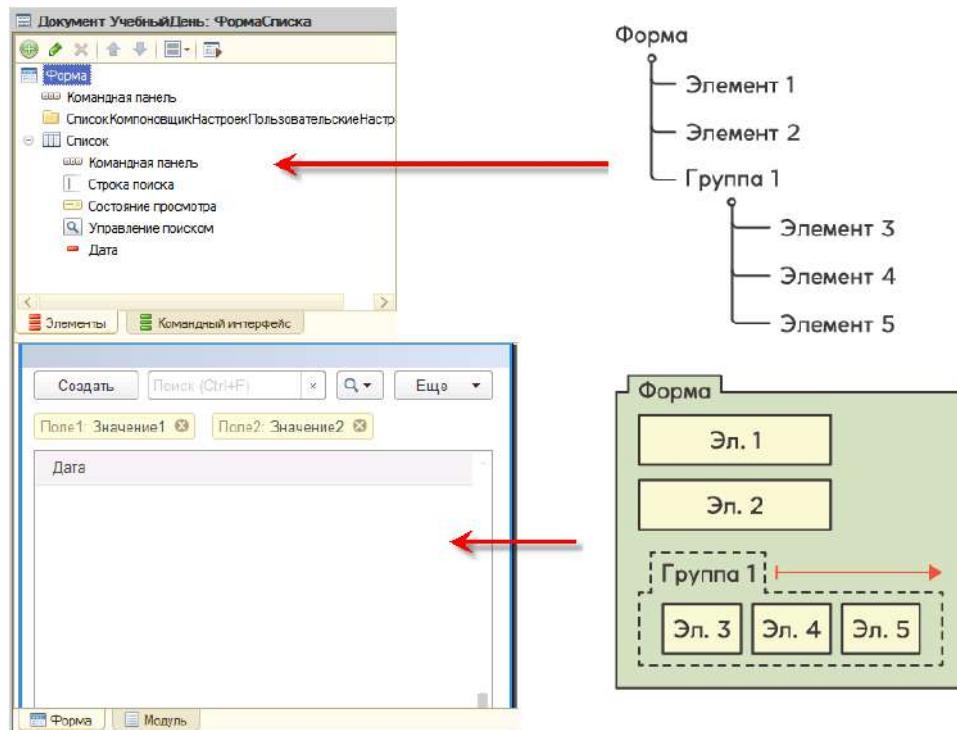


Рисунок 2.217. Дерево элементов в редакторе формы

Сейчас в вашей форме всего один элемент — это *Список*. Но в дереве вы видите много строк. Всё дело в том, что некоторые элементы бывают довольно сложными. Они содержат в себе некоторые важные части, которые в дереве показываются отдельными строками.

Например, форма всегда имеет командную панель. Она может быть видна пользователю или не видна пользователю. Но вы, разработчики, всегда видите её в дереве элементов (рисунок 2.218).

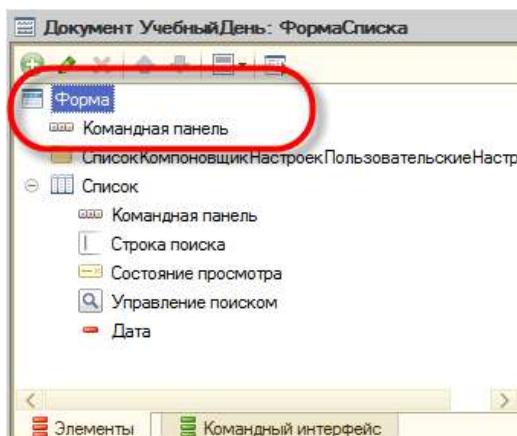


Рисунок 2.218. Форма и её командная панель

А элемент *Список*, который является таблицей, наверное, один из самых сложных элементов. Поэтому у него есть несколько дополнительных строк и внутри, и снаружи, которые помогают разработчику его настраивать (рисунок 2.219).

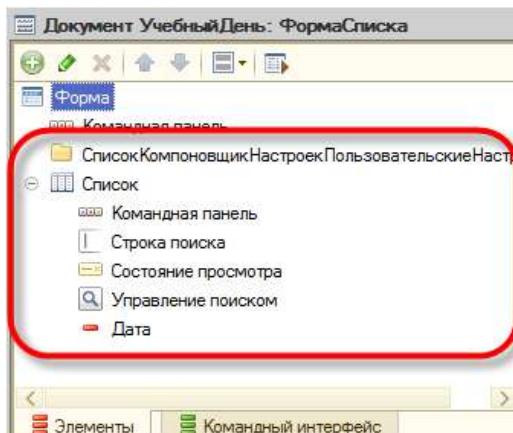


Рисунок 2.219. Таблица «Список»

Как не запутаться во всех трёх окнах редактора? Как понять, какой элемент отображает этот реквизит? Или где в дереве находится тот элемент, который вы видите в окне предварительного просмотра?

Тут всё просто. Сейчас я покажу вам несколько приёмов. Выделите в дереве элементов корень *Форма*.

Первый пример. Вы хотите узнать, с каким элементом связан реквизит *Список*. Для этого откройте его контекстное меню и выполните команду *Перейти*.

В дереве элементов будет выделен элемент *Список*, а в окне просмотра синей рамкой будет показано, где этот список находится в форме (рисунок 2.220).

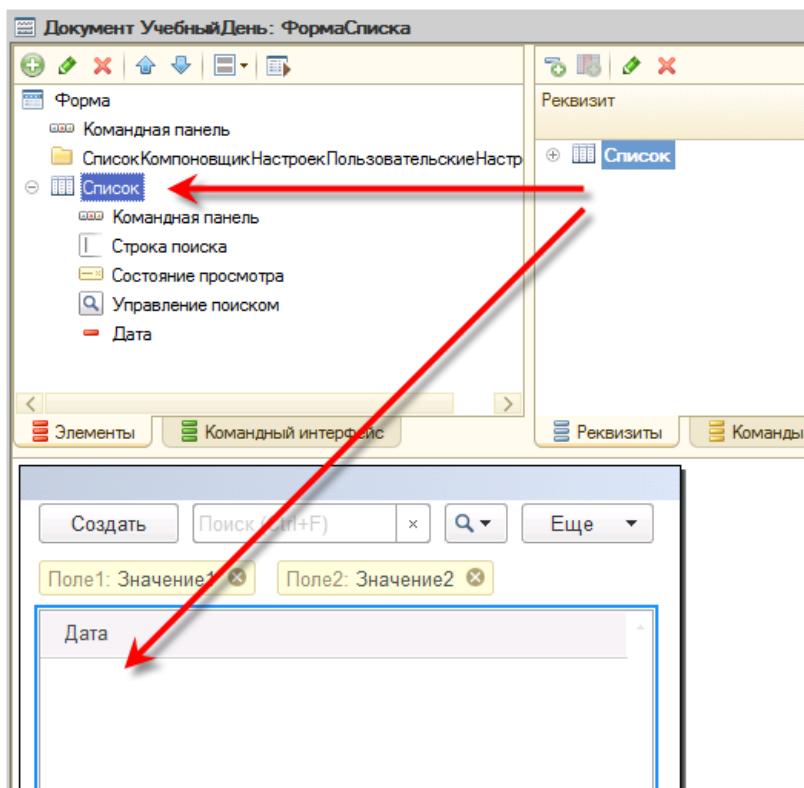


Рисунок 2.220. Перейти к элементу формы

Не все реквизиты связаны с элементами. Например, если вы раскроете реквизит *Список*, то у него будет много подчинённых колонок. Колонка *Дата* есть в форме, и к ней вы можете перейти.

А колонка *Ссылка* не показывается в форме. Поэтому и команда *Перейти* для неё недоступна (рисунок 2.221).

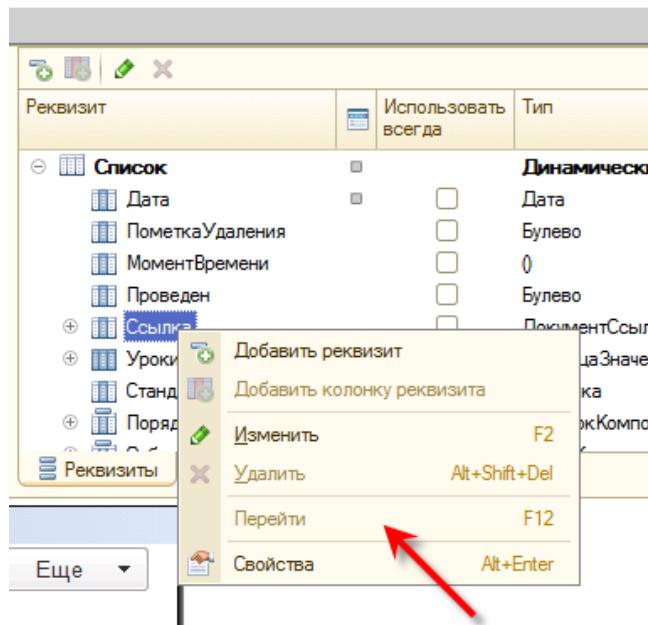


Рисунок 2.221. Команда «Перейти» недоступна

Второй пример. Вы хотите узнать, где в форме окажется некоторый элемент, который есть в дереве элементов формы. Для этого просто выделите этот элемент в дереве, и внизу, в окне просмотра, он тоже будет выделен синей рамкой. Например, *Состояние просмотра* (рисунок 2.222).

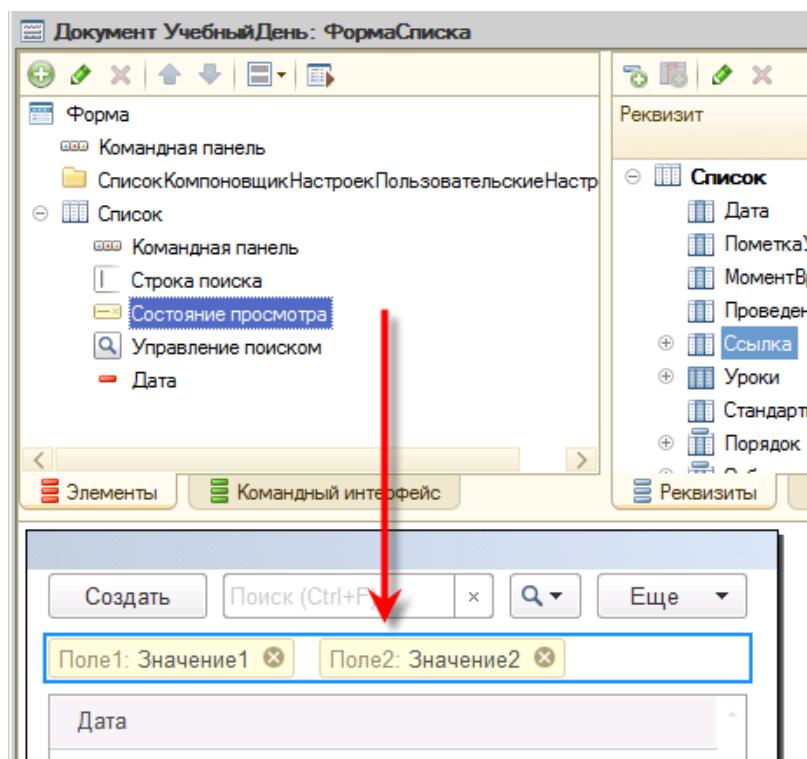


Рисунок 2.222. Показать элемент в форме

Не все элементы видны в форме. Например, если вы выделите командную панель, которая принадлежит списку, вы её не увидите внизу. А командную панель формы уви-

дите. Такова особенность формы списка. Она всегда использует только командную панель формы.

Чтобы показать следующий пример, сверните содержимое реквизита *Список*, чтобы он снова стал одной строкой.

Теперь в другую сторону. Вы хотите узнать, с каким реквизитом связан элемент формы. Для этого откройте его контекстное меню и выполните команду *Перейти*.

Например, если вы выполните эту команду для элемента *Дата*, то в окне реквизитов будет показан этот реквизит (рисунок 2.223).

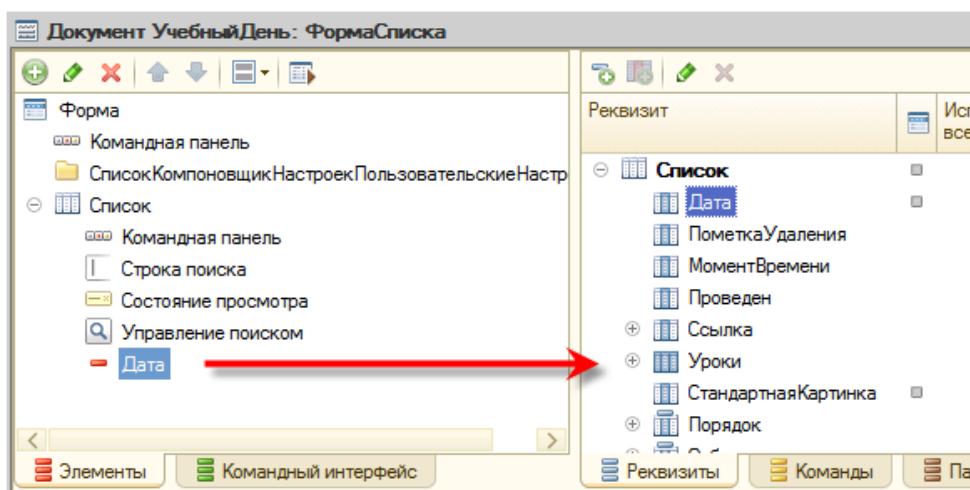


Рисунок 2.223. Перейти к реквизиту формы

Иногда, у сложных элементов, бывает так, что перейти можно не только к самому реквизиту, но и к какой-то его составной части. Важной.

Например, если вы выполните команду *Перейти* для элемента *Список*, то платформа предложит вам такой выбор (рисунок 2.224):

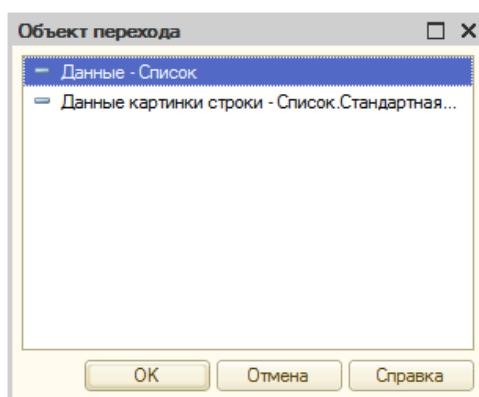


Рисунок 2.224. Выбор объекта перехода

Если вы выберете *Данные — Список*, то перейдёте на сам реквизит *Список*.

А если вы выберете *Данные картинки строки — Список.СтандартнаяКартинка*, то перейдёте к той колонке реквизита *Список*, которая отвечает за картинку, показываемую рядом с элементом списка.

Ещё бывает так, что элемент не связан с реквизитом, а связан с другим элементом формы. Например, если вы попробуете выполнить команду *Перейти* для элементов *Строка поиска* или *Состояние просмотра*, то вы перейдёте к элементу *Список*. Потому что именно этот элемент управляет и строкой поиска, и состоянием просмотра.

Теперь последний пример. Вы хотите узнать, где в дереве элементов тот элемент, который вы видите в нижнем окне, в окне просмотра. Для этого нужно просто выделить тот элемент, который вас интересует, и он будет подсвечен в дереве элементов.

Например, если вы нажмёте мышью на заголовок колонки *Дата*, то в дереве будет выделен элемент *Дата* (рисунок 2.225).

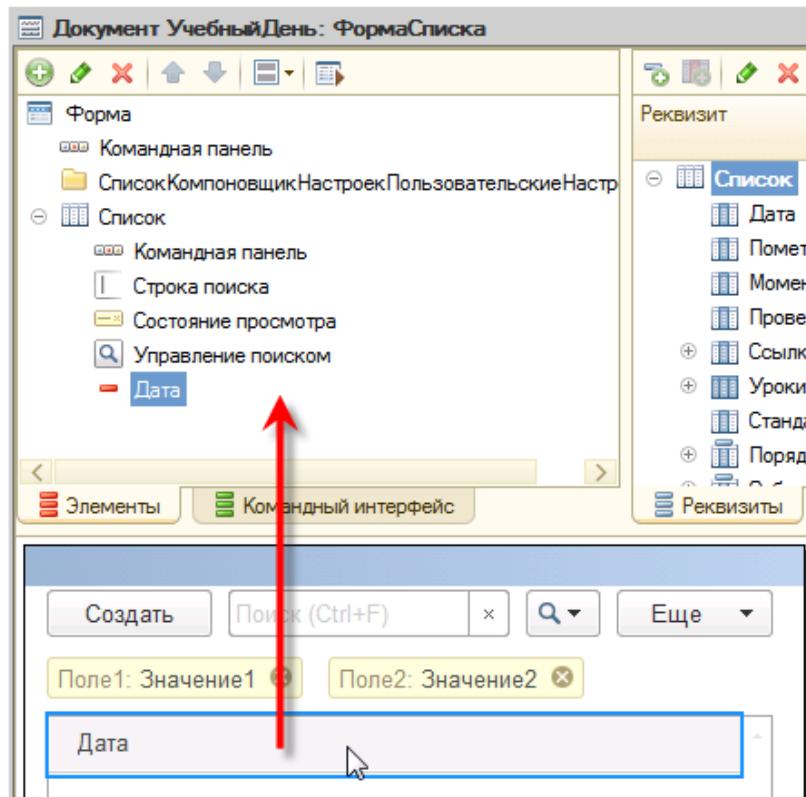


Рисунок 2.225. Показать элемент в дереве

А если нажмёте мышью на белое поле под заголовком *Дата*, то в дереве будет выделен элемент *Список* (рисунок 2.226).

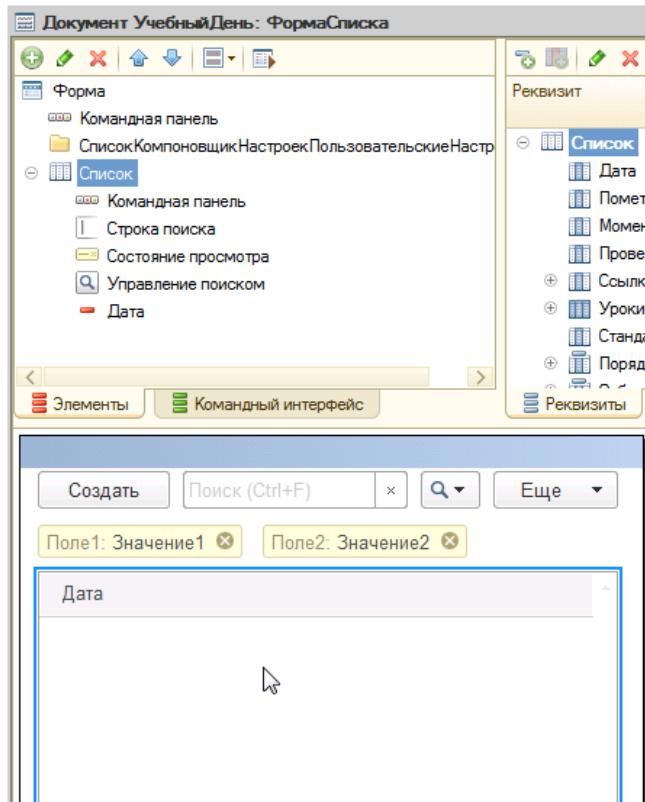


Рисунок 2.226. Показать список в дереве

Чтобы перейти к реквизиту, связанному с тем элементом, который вы видите в окне просмотра, нужно из контекстного меню выполнить команду *Перейти*.

Например, если вы выполните эту команду для заголовка *Дата*, то в окне реквизитов будет выделен реквизит *Дата* (рисунок 2.227).

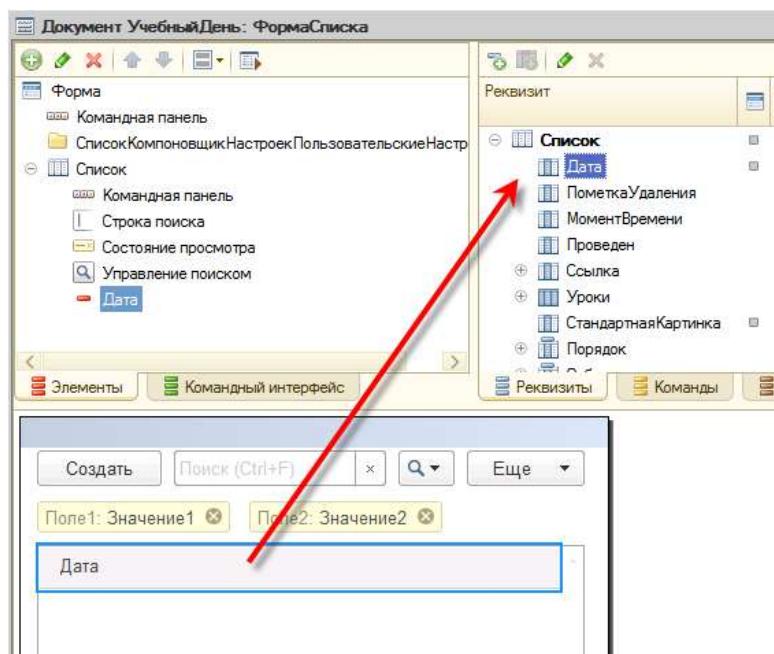


Рисунок 2.227. Перейти к реквизиту

И ещё одно маленькое замечание. При редактировании форм всегда используются два инструмента. Это сам редактор формы, с которым вы только что познакомились. И палитра свойств.

Палитра свойств нужна для того, чтобы изменять свойства реквизитов или элементов формы. Если палитра свойств у вас по каким-то причинам оказалась закрыта, открыть её можно командой *Свойства* из контекстного меню. А контекстное меню можно вызвать у любого элемента или реквизита формы.

Некоторое неудобство, которое может возникать у вас в первое время, связано с тем, что палитра свойств показывает свойства того элемента или реквизита, который выделен. Но разница между активным и неактивным выделением может быть не очень заметна для вас на первый взгляд (рисунок 2.228).

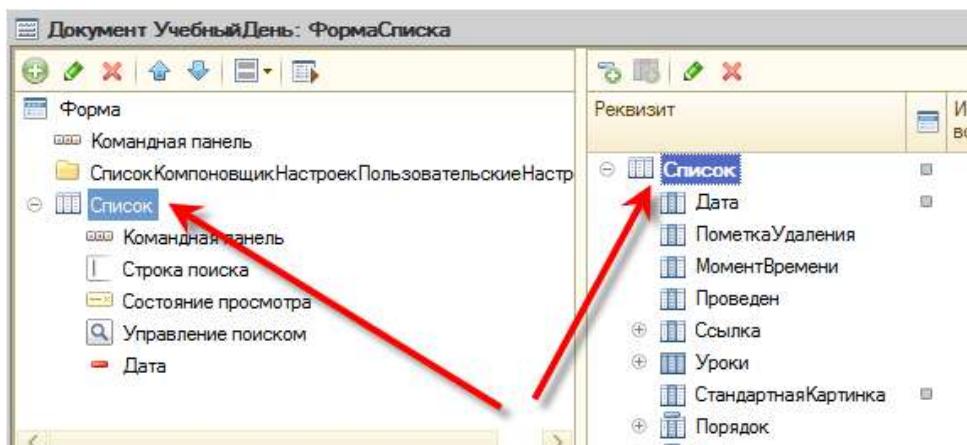


Рисунок 2.228. Активное и неактивное выделение

Конечно, в заголовке палитры свойств и в том, и в другом случае будет написано, чьи свойства она показывает (рисунок 2.229). Но на это вы тоже можете не обратить внимание.

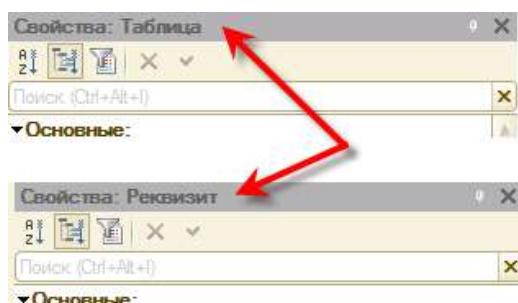


Рисунок 2.229. Заголовок палитры свойств

Примечание

Для верности, особенно в первое время, лучше поступать так.

Прежде чем изменять что-то в палитре свойств или расстраиваться, что вы не нашли нужное свойство, сначала выделите мышью нужный элемент или реквизит, а потом уже идите в палитру свойств.

Тогда в ней наверняка будут свойства того элемента или того реквизита, который вам нужен.

2.14.3 Изменение формы списка

Теперь, когда вы познакомились с редактором формы, можно заняться второй задачей. Чтобы в списке учебных дней каждый из них был обозначен более понятно и удобно, чем сейчас (рисунок 2.230).

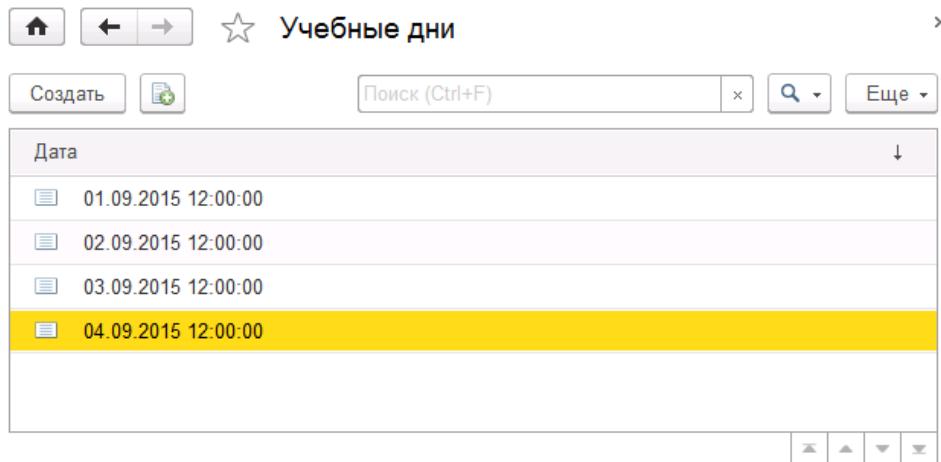


Рисунок 2.230. Список учебных дней

Что не нравится? Время вам знать не нужно. Для одной даты всегда будет только один документ. И его время значения не имеет.

Кроме того, дата показана неудобно. Хочется, чтобы месяц был написан словами, и не помешало бы знать, какой это день недели.

Для этого нужно изменить то, каким образом дата представляется в этом списке.

Форма списка документа УчебныйДень должна быть открыта у вас в конфигураторе. Если нет — откройте её.

В списке, как вы видите, у вас всего одна колонка. Это колонка Дата.

Откройте свойства элемента Дата. Среди них есть свойство Формат (рисунок 2.231).

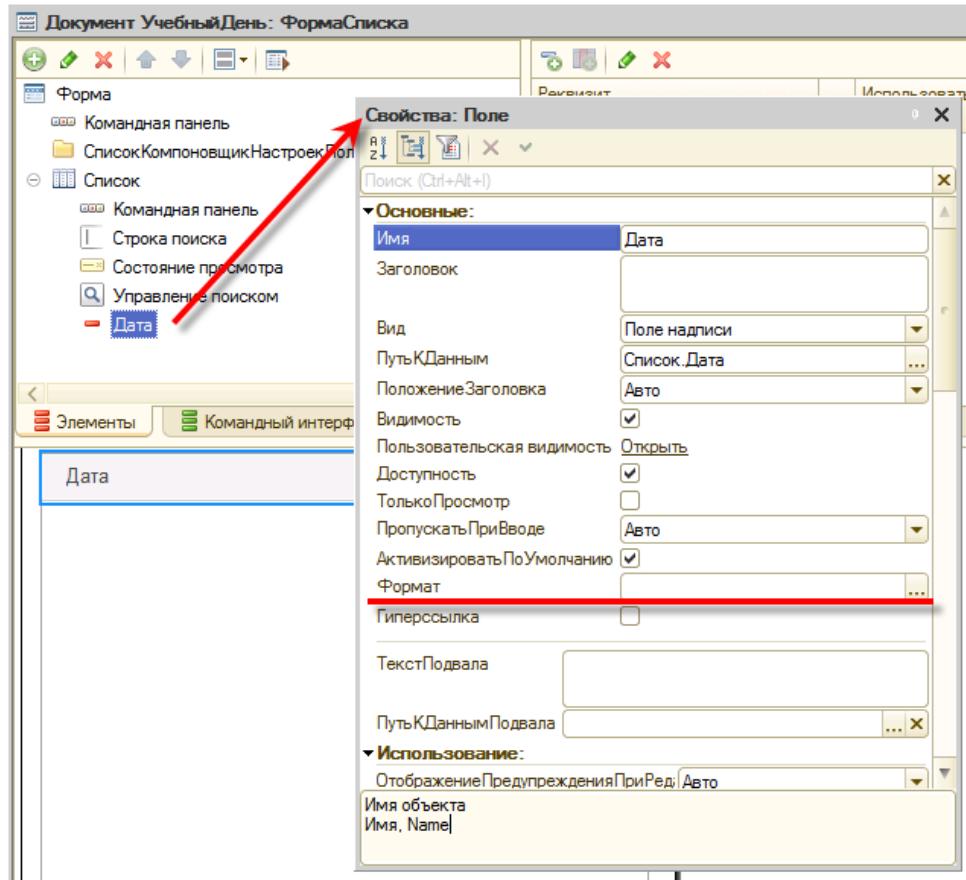


Рисунок 2.231. Свойство «Формат»

Это свойство позволяет задать, каким образом значение, находящееся в этом поле, будет показано пользователю.

Есть определённые правила, по которым формируется эта форматная строка. Правила довольно сложные. Но для некоторых самых простых способов в 1С:Предприятии существует *конструктор форматной строки*. Также он помогает и в сложных случаях, если вы знаете, какие параметры есть у форматной строки.

Но вы этого не знаете. Поэтому вы воспользуетесь справкой. Прежде чем выполнять дальнейшие действия, убедитесь, что в палитре свойств у вас открыты свойства поля *Дата*.

Теперь, чтобы найти в справке нужную статью, выполните команду *Справка — Поиск по справке* (рисунок 2.232).

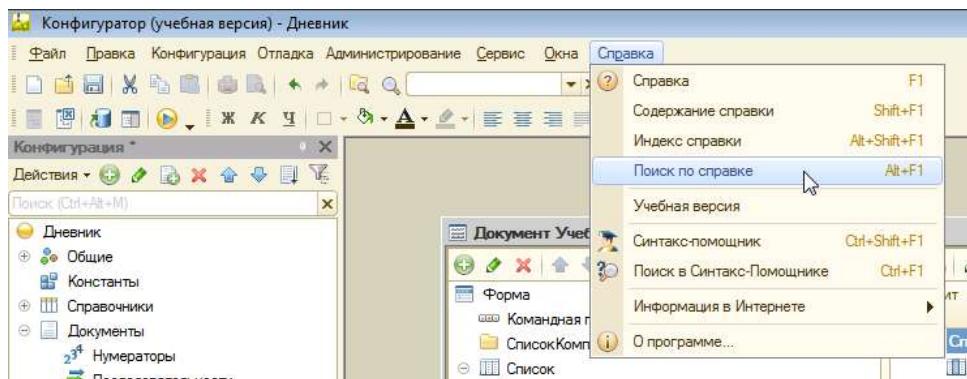


Рисунок 2.232. Поиск по справке

В открывшемся окне напишите: *форматная строка* (рисунок 2.233).

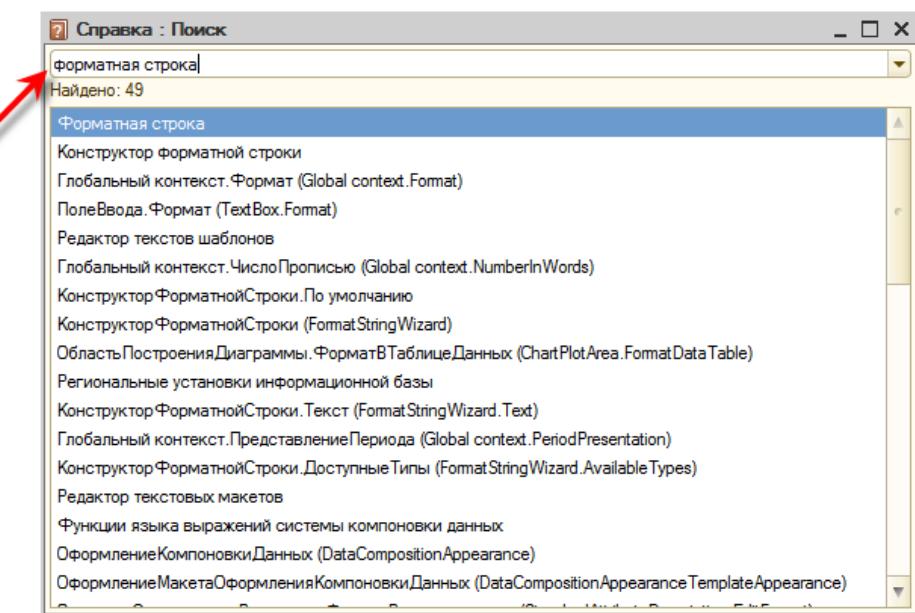


Рисунок 2.233. Поиск «форматная строка»

Платформа сразу же найдёт вам статьи, в которых есть это предложение. Вам нужна самая первая статья *Форматная строка*. Выберите её двойным нажатием мыши.

Прокрутите статью почти до самого конца. До того места, где описывается параметр *ДФ (DF)* (рисунок 2.234).

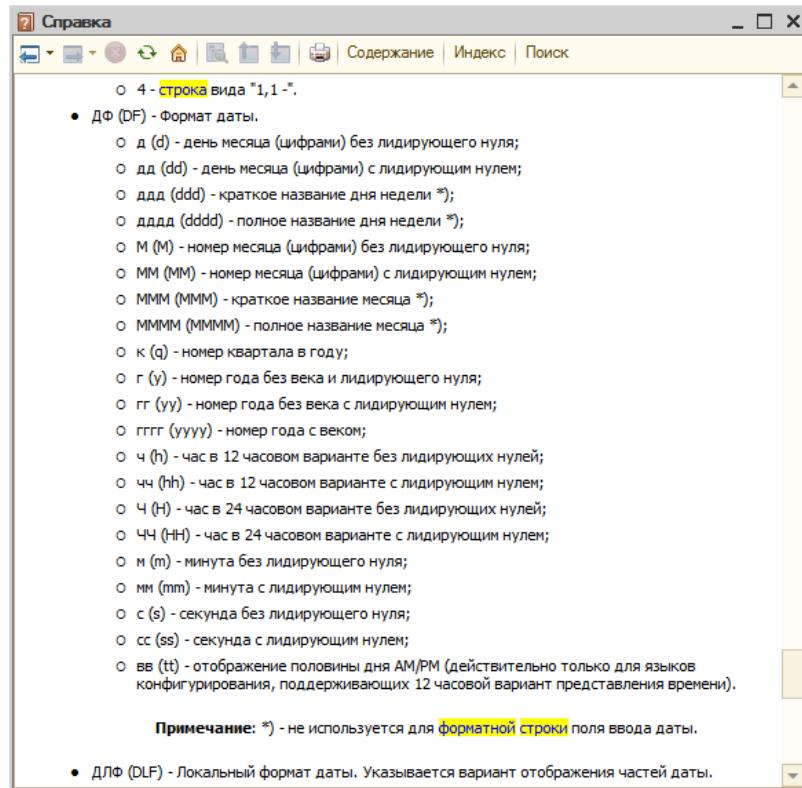


Рисунок 2.234. Описание параметра «ДФ»

Что писать, вы теперь знаете. Осталось узнать, куда это нужно написать.

Сейчас в палитре свойств у вас открыты свойства поля *Дата*. Чтобы открыть *конструктор форматной строки*, нажмите кнопку выбора в поле *Формат* (рисунок 2.235).

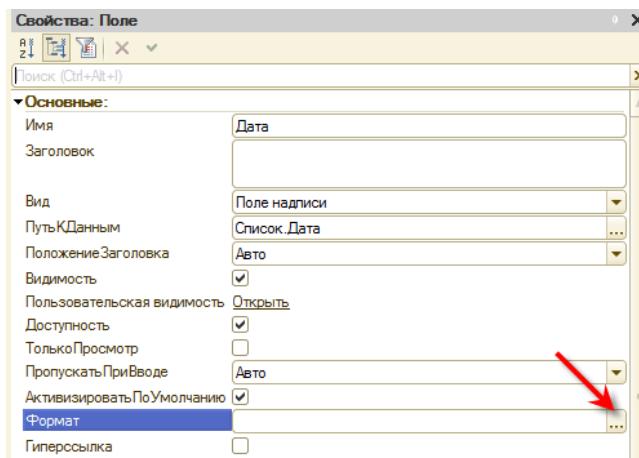


Рисунок 2.235. Кнопка выбора

Появится конструктор (рисунок 2.236).

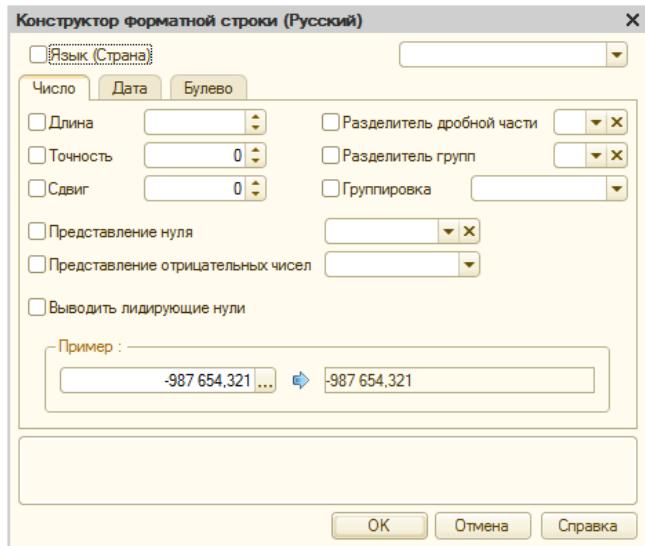


Рисунок 2.236. Конструктор форматной строки

У него есть три закладки: *Число*, *Дата* и *Булево*. Вы хотите задать, как будет выглядеть дата, значит, переходите на закладку *Дата* (рисунок 2.237).

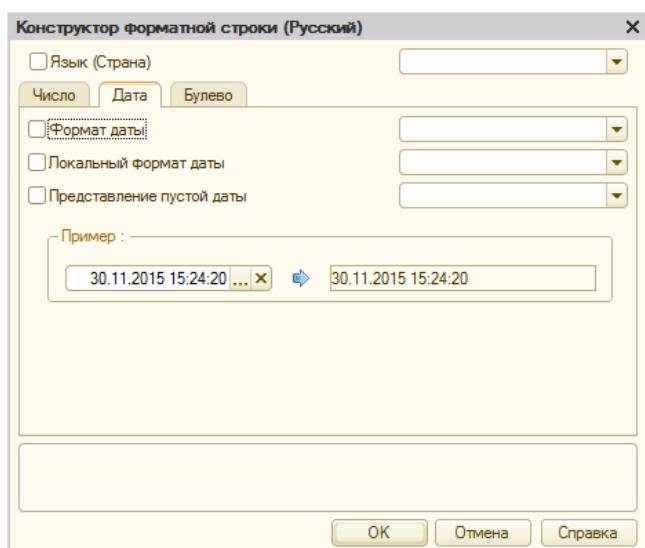


Рисунок 2.237. Закладка «Дата»

Здесь есть два способа задать внешний вид для значения *Дата*: формат даты и локальный формат даты. Воспользуйтесь первым способом и установите флажок *Формат даты* (рисунок 2.238).

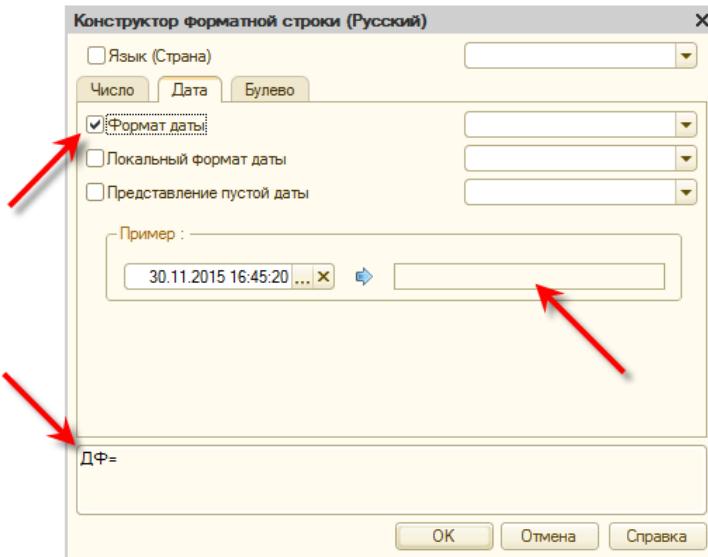


Рисунок 2.238. Формат даты

Обратите внимание, что в нижней части окна конструктор уже начал формировать форматную строку: **ДФ=**. А в средней части, в рамке *Пример*, он будет показывать то представление даты, которое у вас получится.

Теперь в поле рядом с *Формат даты* напишите те параметры форматной строки, которые нужны (рисунок 2.239).

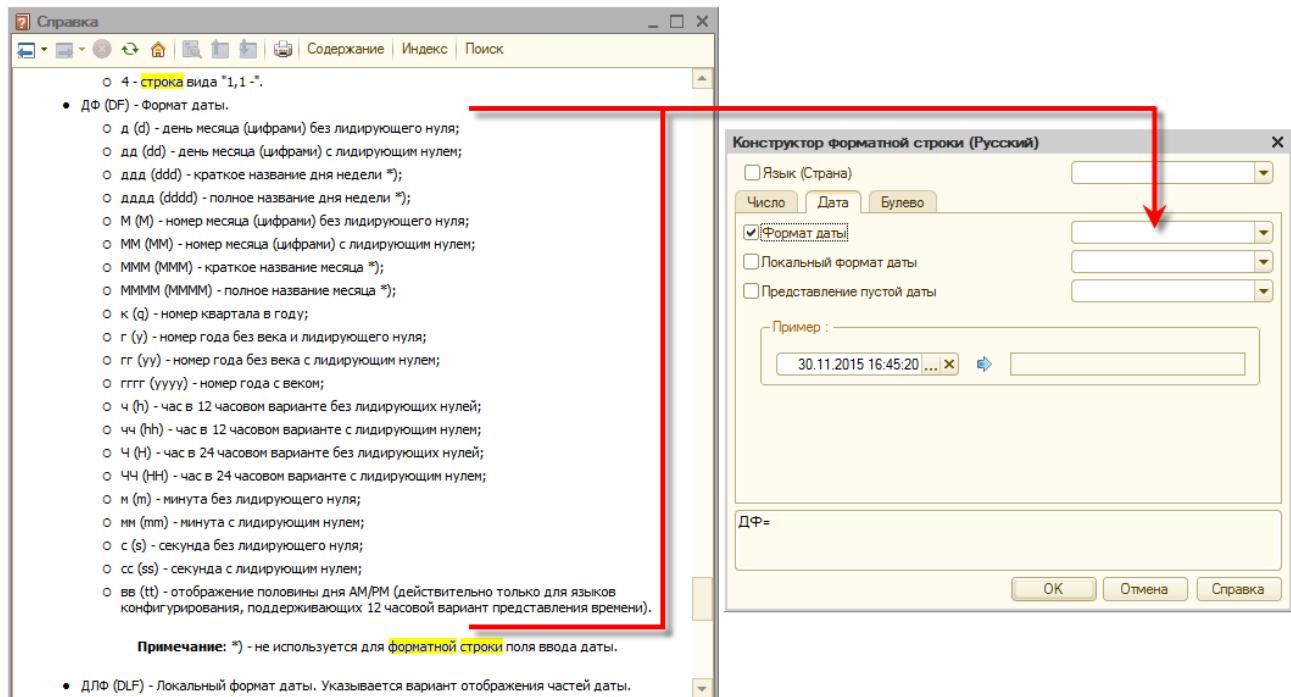


Рисунок 2.239. Заполнение параметров форматной строки

Как вы хотите представить дату? Например, как «4 сентября 2015, пятница». Значит, сначала нужно написать **д**. Это день месяца без лидирующего нуля. Затем, через пробел, **ММММ**. Это полное название месяца.

И так далее. Вся строка должна выглядеть у вас следующим образом:

д ММММ гггг, дддд

После этого нажмите **ТАБ**, и вы увидите, что получилось (рисунок 2.240).

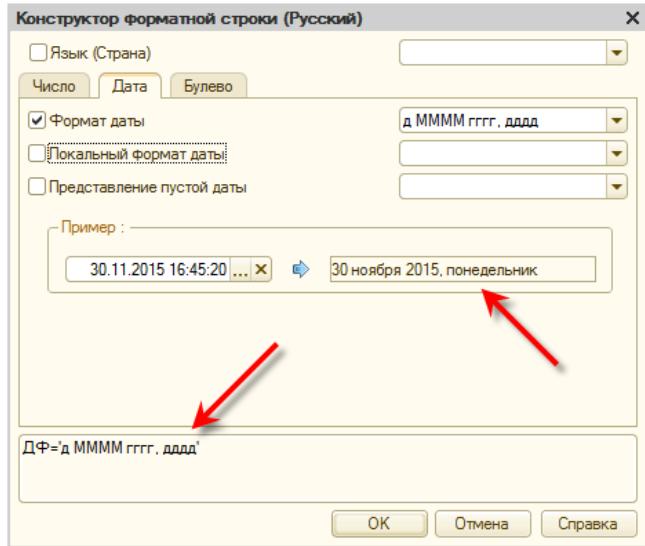


Рисунок 2.240. Форматная строка

Теперь вам остаётся только нажать *OK*, и эта форматная строка подставится в свойство *Формат* (рисунок 2.241).

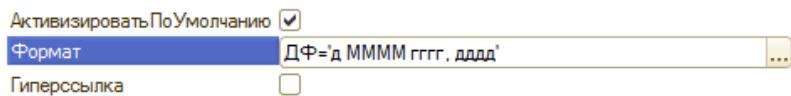


Рисунок 2.241. Свойство «Формат» заполнено

Окна справки можно закрыть. Они пока вам больше не понадобятся.

А конфигурацию можно запустить в режиме отладки и посмотреть, что получилось (рисунок 2.242).

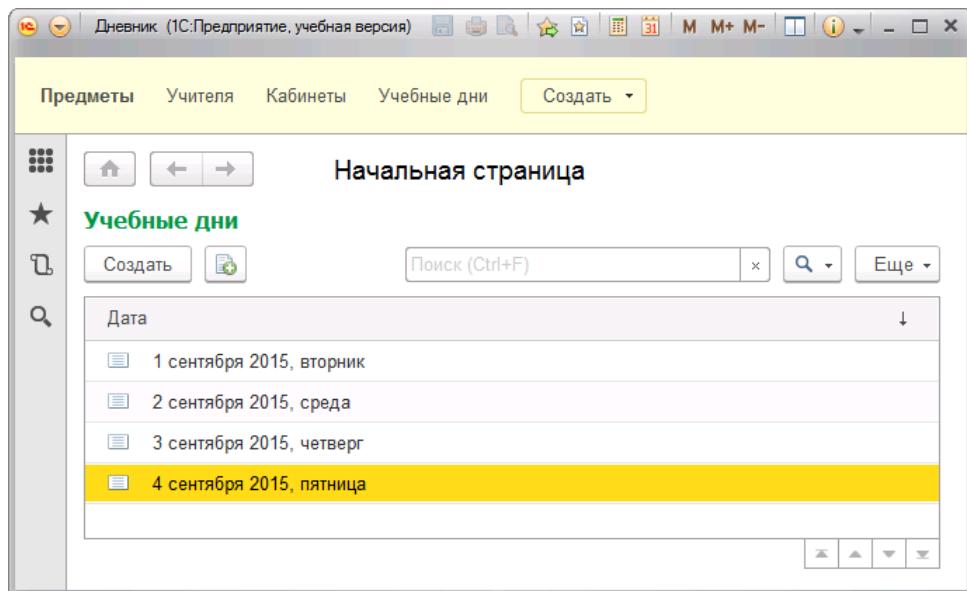


Рисунок 2.242. Список учебных дней

Отлично! Обозначения учебных дней стали гораздо удобнее.

2.14.4 Изменение формы объекта

Теперь вы можете заняться третьей задачей. Напомню. В форме учебного дня мало информации (рисунок 2.243).

N	Предмет	Домашнее задание	Оценка	Комментарий учителя
1	Кл. час			
2	Кл. час			
3	Английский язык			
4	Музыка			
5	Математика			
6	Русский язык			

Рисунок 2.243. Форма учебного дня

Сейчас вы видите только предметы. А ведь вы говорили, что в расписании занятий хочется видеть и номер кабинета, в котором проходит урок, и учителя, который этот урок ведёт.

Чтобы это изменить, вернитесь в конфигуратор и добавьте форму документа для документа УчебныйДень. В дереве конфигурации у вас появится ещё одна форма (рисунок 2.244).

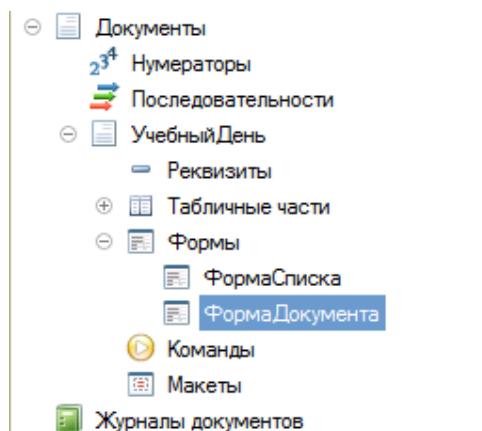


Рисунок 2.244. Форма документа в дереве

А в рабочей области откроется конструктор этой формы (рисунок 2.245).

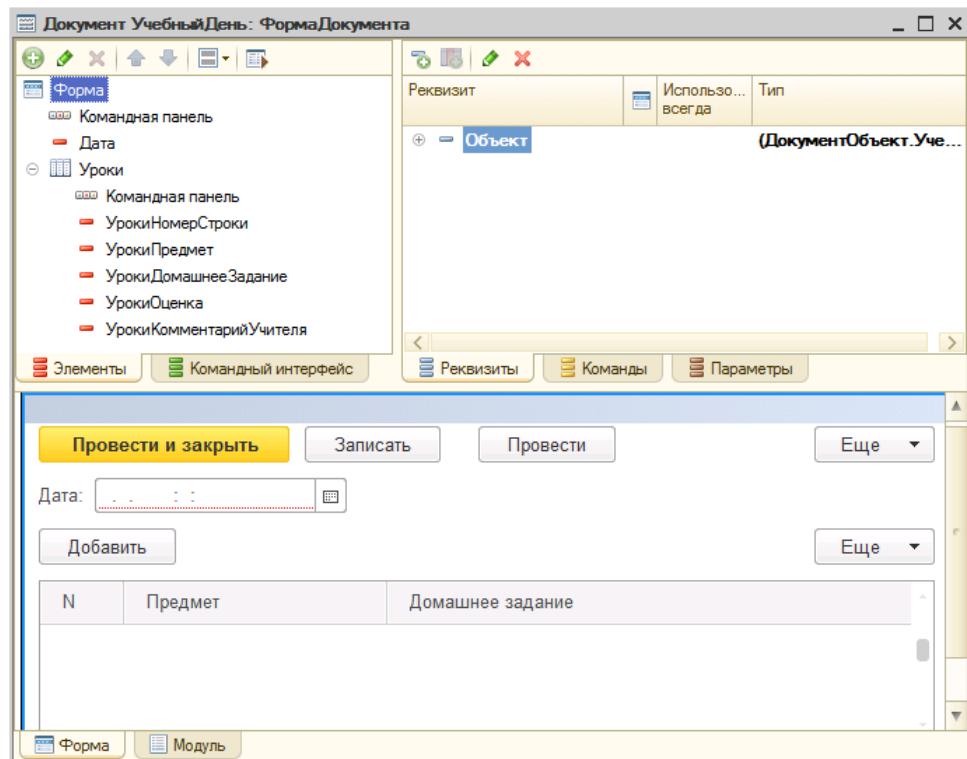


Рисунок 2.245. Форма документа

Как вы видите, платформа создала такую же форму, которую она генерирует автоматически. Вам остаётся только изменить её в соответствии со своими желаниями.

В предыдущем задании вы работали с элементами формы. Для одного из элементов вы изменили свойство *Формат*. А в этом задании вам понадобятся реквизиты формы.

Реквизитов у формы может быть много. Среди них всегда есть один самый важный. Он называется *основной реквизит*.

Заметка

Бывают формы без основного реквизита, но вы их рассматривать не будете.

Основной реквизит для формы — это как велосипедист для велосипеда. От него полностью зависит, как поедет велосипед, что он будет делать (рисунок 2.246).

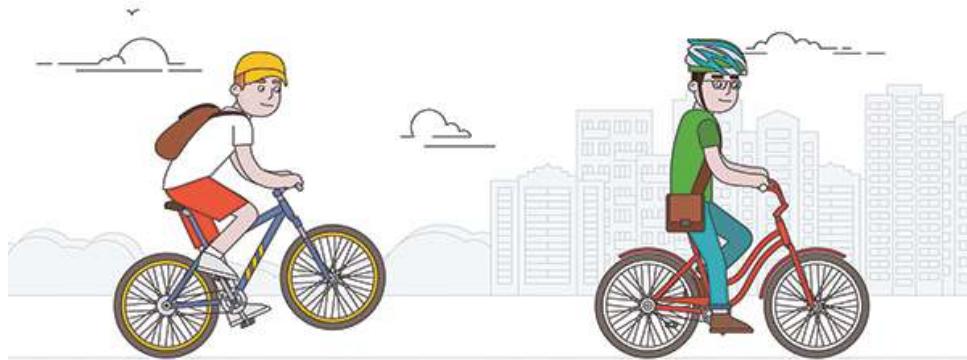


Рисунок 2.246. Взрослый велосипедист и молодой велосипедист

Если велосипедист взрослый, он будет ехать не спеша, аккуратно. А молодой велоси-

пешест на том же самом велосипеде будет быстро разгоняться, резко тормозить, перепрыгивать через препятствия.

Точно так же основной реквизит формы определяет, какие действия умеет выполнять форма и как она выглядит. Если основной реквизит — это список, то форма будет уметь создавать новые элементы в списке, находить имеющиеся элементы и так далее (рисунок 2.247).

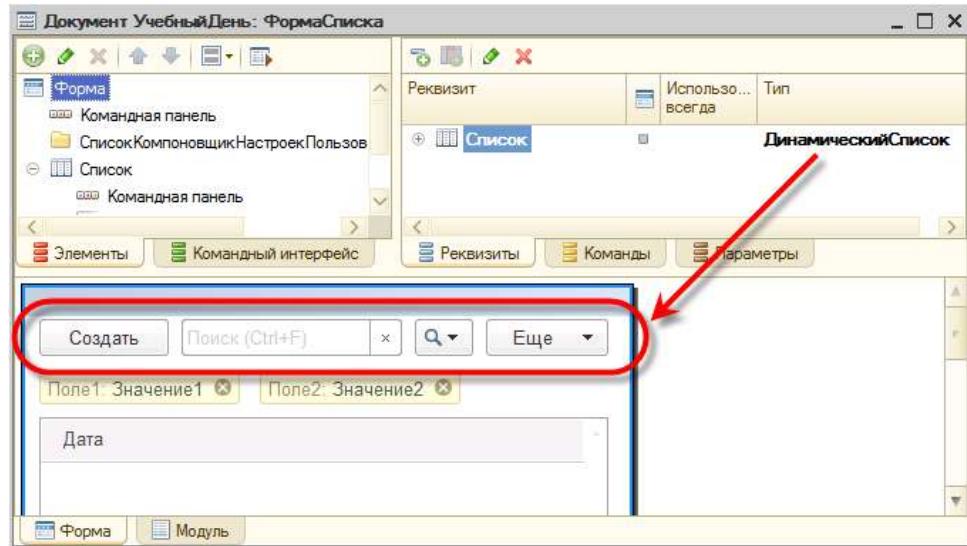


Рисунок 2.247. Основной реквизит — список

Если основной реквизит — это объект, то форма будет уметь записывать данные, если это документ — то ещё и проводить их (рисунок 2.248). Что такое «проводить», я расскажу позже.

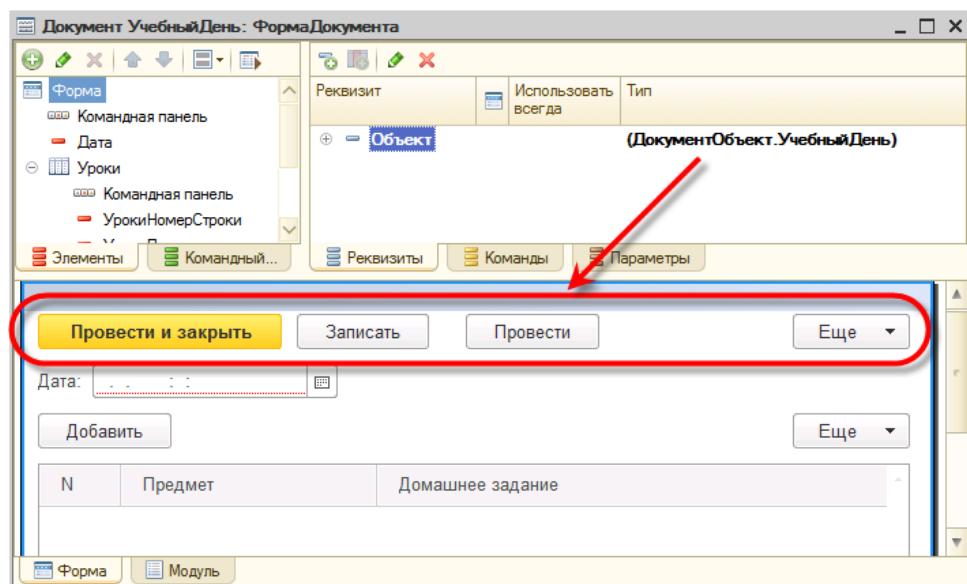


Рисунок 2.248. Основной реквизит — объект

То есть основной реквизит — это «хозяин» всей формы. Поэтому то, что форма показывает с помощью своих элементов, — это, главным образом, данные, содержащиеся в основном реквизите.

Основной реквизит всегда выделяется жирным шрифтом. Чтобы его можно было отличить от других реквизитов. Конструктор формы, когда создаёт новую форму, называет

стандартный реквизит одним и тем же образом: Список, если форма показывает несколько объектов данных, или Объект, если форма показывает данные одного объекта.

Сейчас вы собрались изменять форму документа. В форме у вас единственный реквизит, он же является основным реквизитом. И называется он Объект. Что в этом реквизите? Раскройте его содержимое, нажав мышью на крестик (рисунок 2.249).

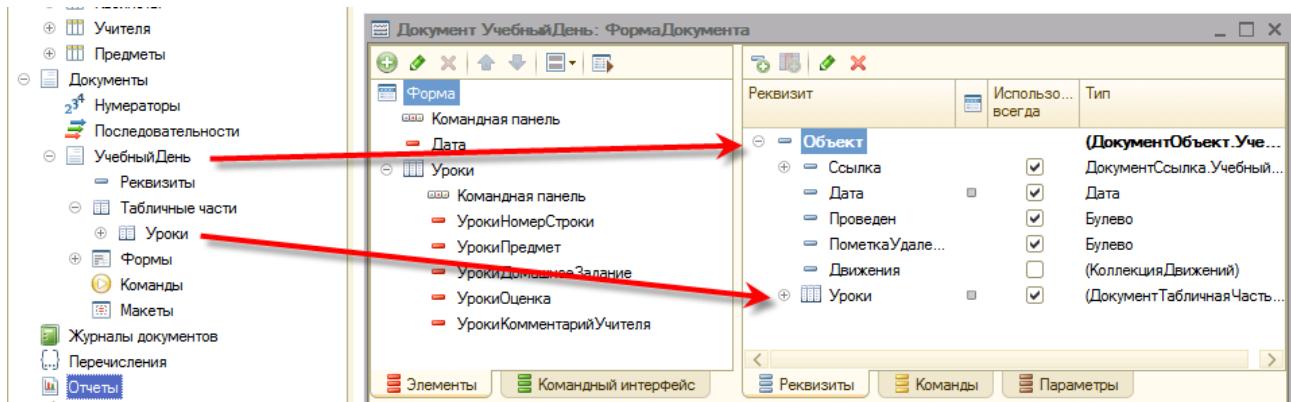


Рисунок 2.249. Состав основного реквизита

В этом реквизите то же самое, что в объекте конфигурации УчебныйДень: Ссылка, Дата, Проведен и т. д. Это стандартные реквизиты. Они не видны в дереве объектов конфигурации. А вот дальше идёт табличная часть Уроки. Та самая, которую добавляли вы. Раскройте её (рисунок 2.250).

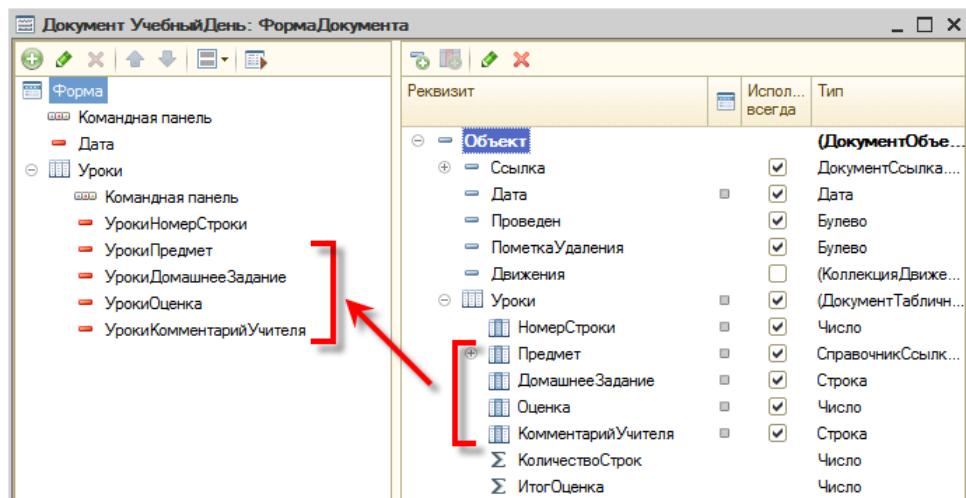


Рисунок 2.250. Реквизиты табличной части в форме

Вы видите, что все реквизиты табличной части уже показаны в форме.

А теперь вспомните: где вы записывали, какой учитель какой предмет преподаёт? Правильно. В справочнике Предметы. И в табличной части как раз есть реквизит, который ссылается на элемент этого справочника. Если вы раскроете реквизит Предмет и посмотрите на его содержимое, то увидите, что в нём есть и кабинет, и учитель. То есть ровно то, что содержится в справочнике Предметы (рисунок 2.251).

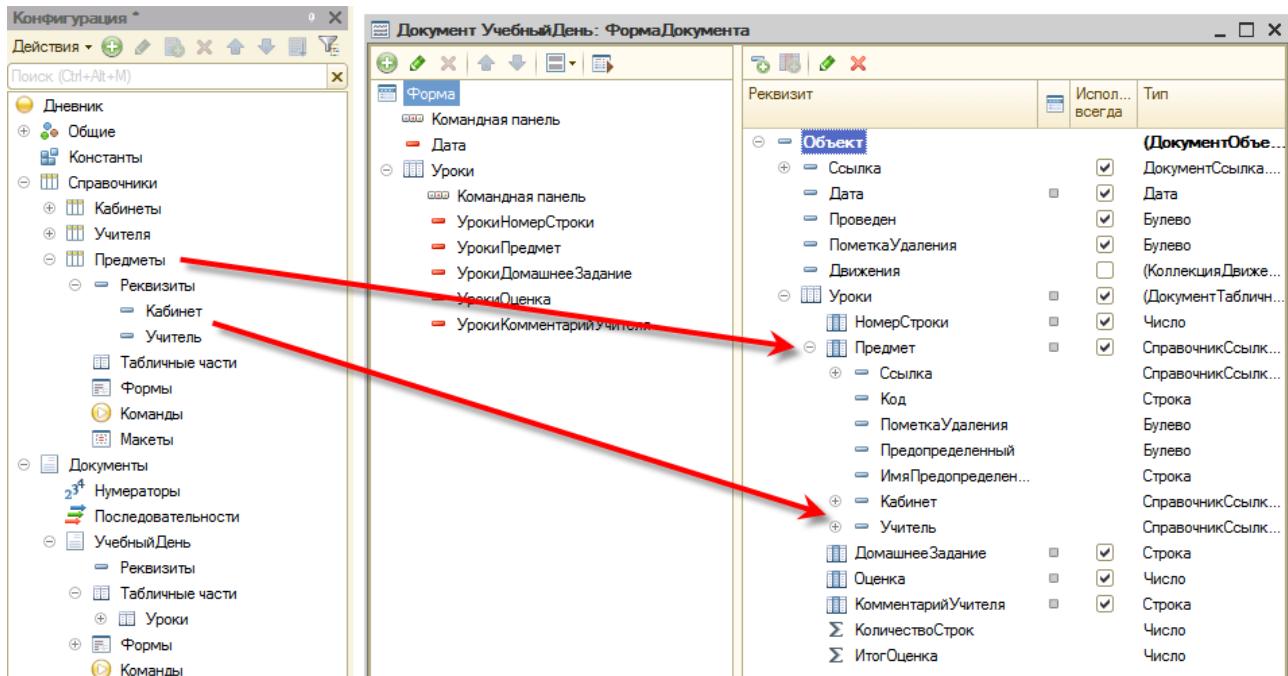


Рисунок 2.251. Состав реквизита «Предмет»

Ещё раз: почему так получилось? Потому, что в этом реквизите у вас хранится ссылка на некоторый элемент справочника *Предметы*. И значит, в том месте, где есть эта ссылка, вы можете узнать не только наименование этого элемента (название предмета), но и все остальные его данные. Например, учителя, который преподаёт, и кабинет, в котором проходят занятия.

Отлично. То, что нужно показать, вы нашли. Теперь вопрос: как это показать? Как сделать, чтобы эти данные появились в форме?

Оказывается, очень просто. Не нужно создавать элементы формы. Не нужно указывать им путь к данным. Всё гораздо проще. Достаточно только потянуть реквизит мышью и перетащить его в то место дерева элементов, где он должен находиться. Всё остальное платформа сделает за вас сама.

Попробуйте. Потащите мышью реквизит *Кабинет* и поместите его после элемента *УрокиПредмет* (рисунок 2.252).

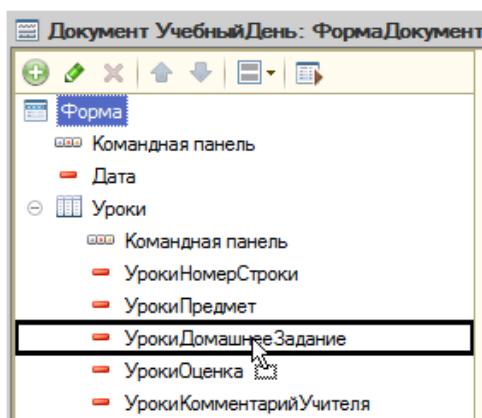


Рисунок 2.252. Перетаскивание реквизита «Кабинет»

Когда вы отпустите мышь, в дереве элементов появится новый элемент. Он автоматически уже будет связан с реквизитом, который вы перетаскивали (рисунок 2.253).

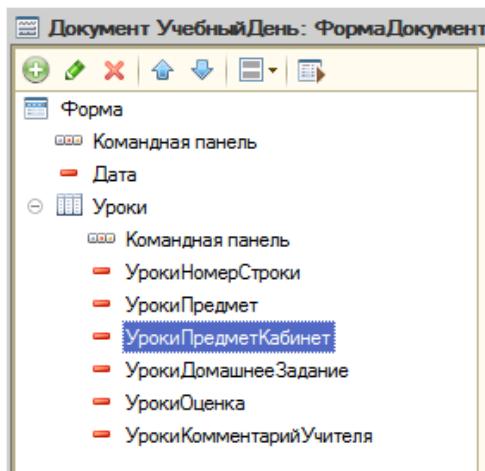


Рисунок 2.253. Новый элемент «УрокиПредметКабинет»

И если теперь вы посмотрите на форму, то увидите, что в таблице после колонки *Предмет* появилась ещё одна колонка (рисунок 2.254).

N	Предмет	К...	Домашнее задание

Рисунок 2.254. Новая колонка в таблице

Точно таким же образом перетащите в дерево реквизит *Учитель*. Поместите его после элемента *УрокиПредметКабинет*. Если вы случайно промахнулись, можете использовать голубые стрелки вверх и вниз, чтобы перемещать элементы в дереве.

Запустите конфигурацию в режиме отладки и посмотрите, что получилось. Откройте, например, учебный день 4 сентября (рисунок 2.255).

N	Предмет	Кабинет	Учитель	Домашнее задание
1	Литература	409	Авдеева В. М.	
2	Математика	311	Рыбакова А. Е.	
3	История	127	Герасимова Е. С.	
4	Литература	409	Авдеева В. М.	
5	Физ. культура	223	Крюков В. В.	
6	Информатика	418	Давыдова А. В.	

Рисунок 2.255. Учебный день 4 сентября

Здорово! Рядом с каждым предметом есть кабинет и есть учитель. Но теперь таблица получилась очень широкой, и ей, может быть, неудобно пользоваться.

Ничего страшного. Эти данные можно расположить более компактным образом.

Закройте 1С:Предприятие и вернитесь в конфигуратор.

Раньше я рассказывал вам, что в форме есть специальный элемент — группа. Он позволяет группировать вместе другие элементы формы. Так вот, этот элемент можно использовать и внутри таблицы.

Вызовите контекстное меню у элемента Уроки (это как раз и есть ваша таблица). И выполните команду *Добавить* (рисунок 2.256).

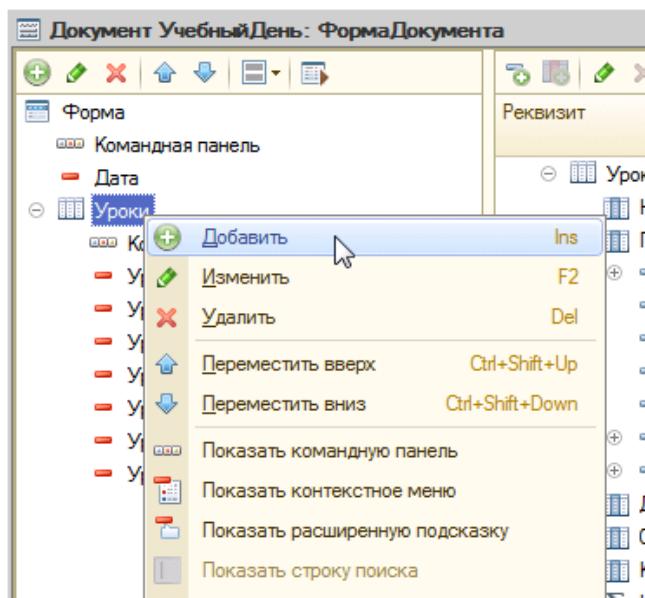


Рисунок 2.256. Добавление элемента в форму

В открывшемся окне выберите элемент *Группа — Группа колонок* и нажмите *OK* (рисунок 2.257).

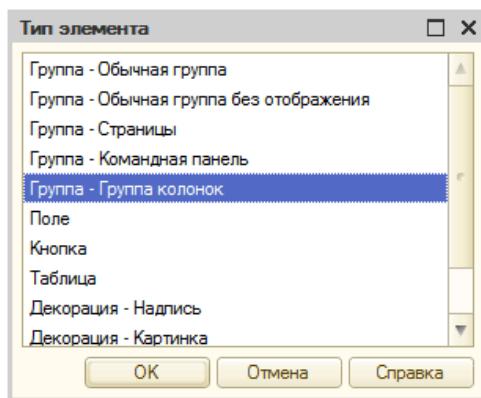


Рисунок 2.257. Добавление группы

В дереве элементов появится группа (рисунок 2.258).

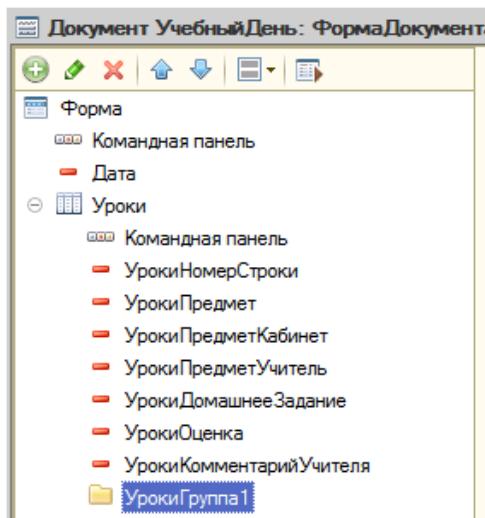


Рисунок 2.258. Новая группа

В палитре свойств назовите её *ГруппаКабинетУчитель*. А затем с помощью стрелок вверх или вниз поставьте её после реквизита *Предмет* (рисунок 2.259).

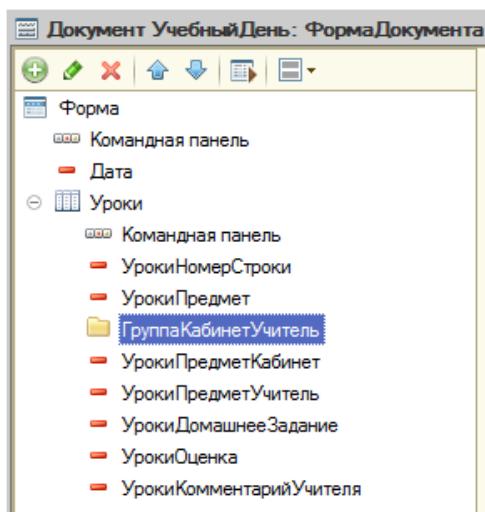


Рисунок 2.259. ГруппаКабинетУчитель

Обратите внимание, что внутри этой группы элементы будут располагаться вертикально. Это указано в её свойстве *Группировка*, которое вы можете посмотреть в палитре свойств.

Если вам тяжело искать среди большого количества свойств, которые есть в палитре, воспользуйтесь поиском. В поле, расположенном вверху палитры, начните вводить имя свойства. Например «гру».

Платформа сразу же покажет вам подходящие свойства (рисунок 2.260).

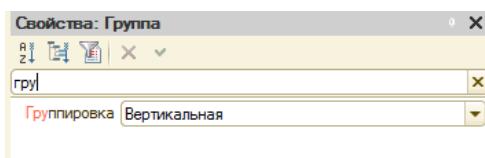


Рисунок 2.260. Поиск в палитре свойств

Совет

После того как вы нашли и посмотрели интересующее вас свойство, не забудьте очистить поиск, нажав на крестик.

Иначе поиск продолжит действовать. И когда вы перейдёте к другому элементу формы, вы можете вообще ничего не увидеть в панели свойств. Например, потому что у другого элемента нет свойств, начинающихся на «гру».

Итак, группу, которую вы добавили, располагает элементы вертикально. Перенесите в неё несколько элементов.

Выделите в дереве (удерживая клавишу **Ctrl**) элементы УрокиПредметКабинет и УрокиПредметУчитель (рисунок 2.261).

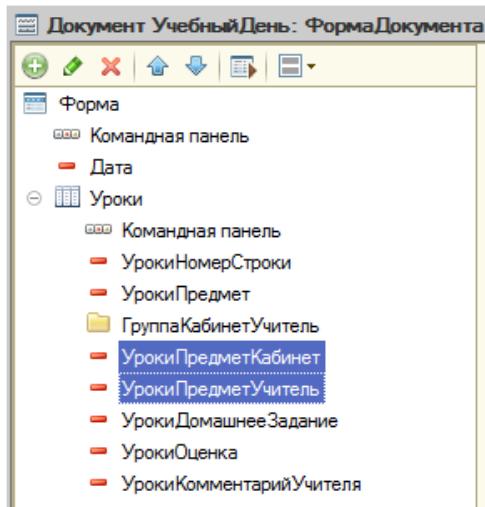


Рисунок 2.261. Выделение нескольких элементов

А затем перетащите их в группу ГруппаКабинетУчитель. У вас получится такая картина (рисунок 2.262).

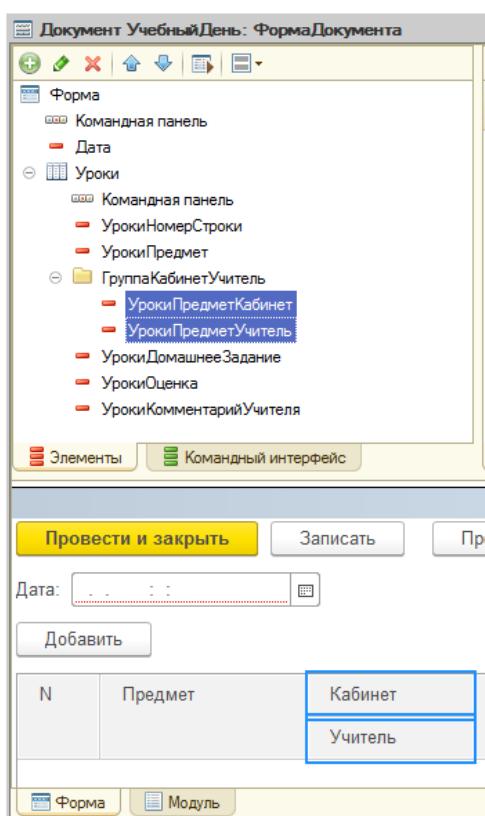


Рисунок 2.262. Колонки, размещённые вертикально

Как это будет выглядеть в форме, вы видите внизу. В одной ячейке таблицы будут находиться сразу и предмет, и кабинет, и учитель. Чтобы стало совсем понятно, можете запустить конфигурацию в режиме отладки и посмотреть на это вживую (рисунок 2.263).

Добавить		↑	↓	Еще ▾	
N	Предмет	Кабинет		Домашнее задание	Оценка
		Учитель			
1	Литература	409			
		Авдеева В. М.			
2	Математика	311			
		Рыбакова А. Е.			
3	История	127			
		Герасимова Е. С.			

Рисунок 2.263. Реквизиты, расположенные вертикально

Теперь строки таблицы стали уж слишком высокими. А названия предметов теряются среди прочей информации, которая есть в этой колонке.

Продолжим работать над формой и исправим эти недостатки. Закройте 1С:Предприятие и вернитесь в конфигуратор.

Чтобы сделать строки уже, расположите кабинет и учителя не вертикально, а друг рядом с другом. И не просто в соседних колонках, а в одной колонке. Потому что и фамилия учителя, и номер кабинета — это справочная информация. Не предполагается, что пользователь будет её редактировать.

Выделите элемент УрокиПредметУчитель и поставьте его перед элементом УрокиПредметКабинет (рисунок 2.264).

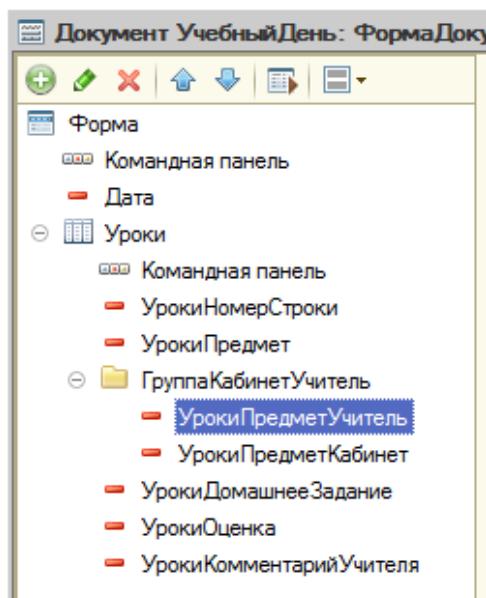


Рисунок 2.264. Элементы «УрокиПредметУчитель» и «УрокиПредметКабинет»

Откройте свойства группы ГруппаКабинетУчитель. Для этой группы задайте группировка — В ячейке (рисунок 2.265).

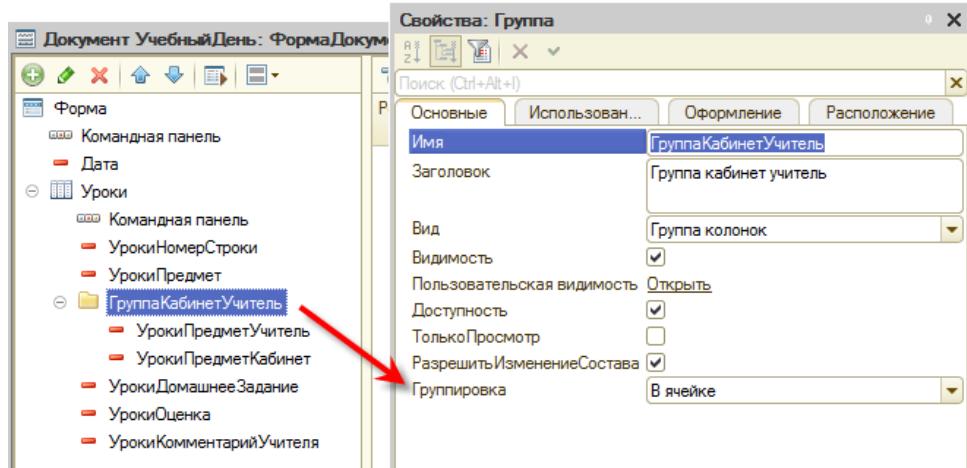


Рисунок 2.265. Группировка «В ячейке»

Внизу вы видите, как это будет выглядеть в форме (рисунок 2.266).

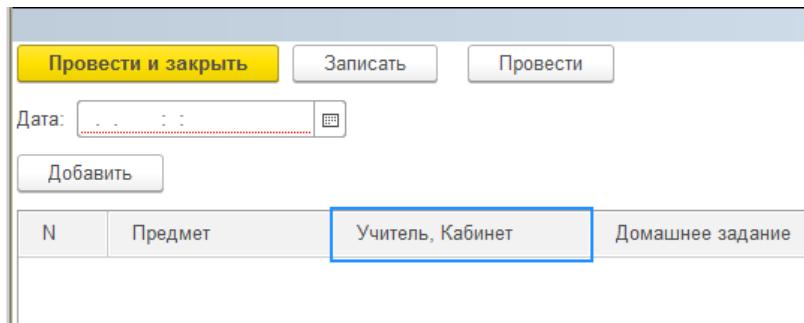


Рисунок 2.266. Эскиз формы

И вот теперь, чтобы учитель и кабинет не мешали вам видеть названия предметов, измените их цвет.

Выделите оба этих элемента, найдите в палитре свойство *ЦветТекста* (рисунок 2.267).

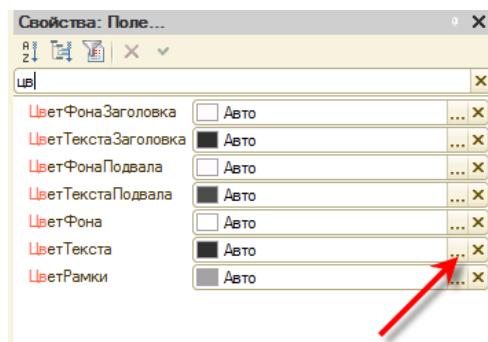


Рисунок 2.267. Цвет текста

Нажмите кнопку выбора и измените его на темно-серый. Если вы не помните, как пользоваться диалогом выбора цвета, посмотрите в разделе 2.9.2 «Режим отладки» на страницах 76–77.

Теперь запустите конфигурацию в режиме отладки и посмотрите, что получилось (рисунок 2.268).

						Добавить	↑	↓	Еще ▾	
N	Предмет	Учитель, Кабинет	Домашнее задание	Оценка	Комментарий учителя					
1	Литература	Авдеева В. М., 409								
2	Математика	Рыбакова А. Е., 311								
3	История	Герасимова Е. С., 127								
4	Литература	Авдеева В. М., 409								
5	Физ. культура	Крюков В. В., 223								
6	Информатика	Давыдова А. В., 418								

Рисунок 2.268. Таблица уроков

Так стало гораздо «веселее».

И теперь сделайте ещё одно изменение. Чтобы понять, зачем оно нужно, попробуйте в ячейку *Домашнее задание* ввести какой-нибудь длинный текст. Любой. Например:

«Домашнее задание, которое нам задали в самый первый день. Хотя обещали, что ничего задавать не будут» (рисунок 2.269).

						Добавить	↑	↓	Еще ▾	
N	Предмет	Учитель, Кабинет	Домашнее задание	Оценка	Комментарий учителя					
1	Литература	Авдеева В. М., 409	что задавать не будут							
2	Математика	Рыбакова А. Е., 311								

Рисунок 2.269. Текст домашнего задания

Обратите внимание, что весь текст вводится в одну строку. И если вы захотите изменить что-то в начале строки, вам придётся перемещать туда курсор стрелкой влево. Это неудобно.

Кроме того, если где-то в середине этой фразы вы захотите перенести её продолжение на другую строку, то вам это не удастся. Нажатие клавиши *Enter* просто приведёт к тому, что редактирование будет закончено (рисунок 2.270).

						Добавить	↑	↓	Еще ▾	
N	Предмет	Учитель, Кабинет	Домашнее задание	Оценка	Комментарий учителя					
1	Литература	Авдеева В. М., 409	Домашнее задание, которое нам задали в самый первый день. Хотя обещали, что ничего задавать не будут							
2	Математика	Рыбакова А. Е., 311								

Рисунок 2.270. Редактирование закончено

В результате введённый вами текст будет расположена в одну строку. Если ширина колонки не позволяет увидеть его полностью, вы можете подвести к нему мышь и подождать немного. Появится всплывающая подсказка, в которой он будет показан полностью (рисунок 2.271).

						Добавить	↑	↓	Еще ▾	
N	Предмет	Учитель, Кабинет	Домашнее задание	Оценка	Комментарий учителя					
1	Литература	Авдеева В. М., 409	Домашнее задание, которое нам задали в самый первый день. Хотя обещали, что ничего задавать не будут							
2	Математика	Рыбакова А. Е., 311								

Рисунок 2.271. Текст во всплывающей подсказке

Таким образом, получается, что просматривать текст можно, но не очень удобно. А вводить и редактировать его в одной строке — совсем неудобно. Хотелось бы редактировать и просматривать его в несколько строк.

Это легко исправить. Закройте 1С:Предприятие, не сохраняйте этот документ. Вернитесь в конфигуратор.

Чтобы сделать то, что вы хотите, понадобится обратиться не к форме, а к дереву объектов конфигурации. К реквизиту *Домашнее Задание*, который находится в табличной части документа *Учебный День*.

Дело в том, что не все свойства, влияющие на отображение данных, задаются в той или иной форме. Есть некоторые свойства, которые очень тесно связаны с самими данными. Например, способ, которым редактируется домашнее задание. Хочется, чтобы домашнее задание, где бы оно ни появилось, в какой бы форме ни было показано, всегда редактировалось именно таким способом.

Поэтому существует возможность не задавать этот способ снова и снова в каждой форме, где появляется домашнее задание. А задать его один раз, для реквизита, в дереве конфигурации. И потом платформа автоматически установит нужный способ редактирования в любой форме, где этот реквизит появится.

Итак, выделите в дереве конфигурации реквизит *Домашнее Задание* и найдите в панелире два свойства: *Многострочный режим* и *Расширенное редактирование*. Они расположены рядом, и поиском можно не пользоваться.

Установите оба этих флажка (рисунок 2.272).

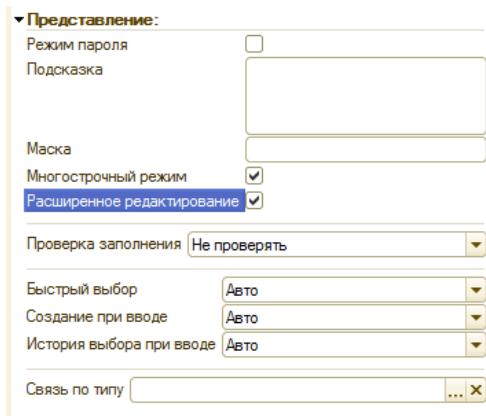


Рисунок 2.272. Многострочный режим и расширенное редактирование

Флажок *Многострочный режим* позволит вам редактировать текст не в одной строке, а в нескольких.

А флажок *Расширенное редактирование* позволит вам переносить текст на новую строку, начинать новые строки, использовать символы табуляции и так далее.

Теперь запустите конфигурацию в режиме отладки и снова попробуйте ввести какой-нибудь текст домашнего задания.

В процессе ввода, чтобы перейти на новую строку, нажмите сочетание клавиш **Shift+Enter** (рисунок 2.273).

N	Предмет	Учитель, Кабинет	Домашнее задание	Оценка	Комментарий учителя
1	Литература	Авдеева В. М., 409	Домашнее задание, которое нам задали в самый		
2	Математика	Рыбакова А. Е., 311			

Рисунок 2.273. Ввод нескольких строк текста

Ну, и теперь, если вы не поленитесь и введёте длинное домашнее задание, вы увидите, что оно показывается не в две строки, как раньше, а в три (рисунок 2.274). Так удобнее.

N	Предмет	Учитель, Кабинет	Домашнее задание	Оценка	Комментарий учителя
1	Литература	Авдеева В. М., 409	Домашнее задание, которое нам задали в самый первый день. Хотя обещали, что ...		
2	Математика	Рыбакова А. Е., 311			

Рисунок 2.274. Показ длинного текста в три строки

А если вы решите его отредактировать, то оно будет показано не в одну строку, как раньше, а в несколько. И это гораздо удобнее (рисунок 2.275).

N	Предмет	Учитель, Кабинет	Домашнее задание	Оценка	Комментарий учителя
1	Литература	Авдеева В. М., 409	Домашнее задание, которое нам задали в самый первый день. Хотя обещали, что ...		
2	Математика	Рыбакова А. Е., 311			

Рисунок 2.275. Редактирование текста в несколько строк

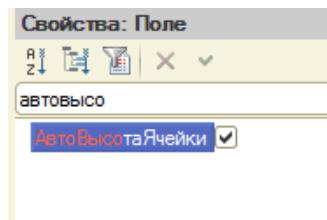
Остался один маленький нюанс. После того как реквизит *Домашнее Задание* вы сделали многострочным, автоматически раздвинулись все строки в табличной части документа. И те, где написан текст, и пустые строки тоже (рисунок 2.276).

Добавить ↑ ↓ Еще ▾					
N	Предмет	Учитель, Кабинет	Домашнее задание	Оценка	Комментарий учителя
1	Литература	Авдеева В. М., 409	Домашнее задание, которое нам задали в самый первый день. Хотя обещали, что ...		
2	Математика	Рыбакова А. Е., 311			
3	История	Герасимова Е. С., 127			
4	Литература	Авдеева В. М., 409			
5	Физ. культура	Крюков В. В., 223			
6	Информатика	Давыдова А. В., 418			

Рисунок 2.276. Высота всех строк увеличилась

Это не очень красиво. Хочется, чтобы раздвигалась только та строка, в которой что-то написано. А пустые строки пусть будут тонкими.

Чтобы это поправить, вернитесь в конфигуратор, откройте свойства элемента формы УрокиДомашнееЗадание. С помощью строки поиска найдите свойство АвтоВысотаЯчейки. Установите флажок (рисунок 2.277).

**Рисунок 2.277.** Свойство «АвтоВысотаЙчейки»

Если теперь вы запустите 1С:Предприятие в режиме отладки и откроете документ, то увидите такую картину (рисунок 2.278).

Добавить ↑ ↓ Еще ▾					
N	Предмет	Учитель, Кабинет	Домашнее задание	Оценка	Комментарий учителя
1	Литература	Авдеева В. М., 409	Домашнее задание, которое нам задали в самый первый день. Хотя обещали, что ...		
2	Математика	Рыбакова А. Е., 311			
3	История	Герасимова Е. С., 127			
4	Литература	Авдеева В. М., 409			
5	Физ. культура	Крюков В. В., 223			
6	Информатика	Давыдова А. В., 418			

Рисунок 2.278. Раздвигаются только заполненные строки

Подробнее

Подробнее вы можете прочитать о редакторе формы в документации «Руководство разработчика. Раздел 27.1. Редактор формы».

Подробнее про сочетания клавиш для работы с полем ввода вы можете прочитать во встроенной справке: Главное меню — Справка — Содержание справки — Сочетания клавиш (Конфигуратор) — Поле ввода.

Итак, вы уже создали целое полезное прикладное решение! Да, оно, конечно, обладает пока не очень большими возможностями. Но в нём уже можно хранить расписание занятий и домашние задания. То есть это полноценное прикладное решение, которым можно пользоваться.

А ведь при этом вы не написали пока ни одной строчки программного кода. Всё потому, что до сих пор вы занимались визуальным конструированием. И все команды, которые нужно выполнять, платформа подставляла за вас сама. Так, что вы этого даже не замечали.

Но теперь вы подошли к тому моменту, когда одного только визуального конструирования мало. Чтобы продвигаться дальше и сделать прикладное решение более удобным и полезным, вам понадобится использовать встроенный язык. То есть вам понадобится ещё глубже проникнуть в 1С:Предприятие. Заняться ещё более тонкой настройкой вашего прикладного решения.

Встроенный язык 1С:Предприятия не сложен для тех, кто уже имеет опыт программирования. Но если вы не имеете такого опыта, то вам понадобится некоторое «вступление» в программирование. Ему и будет посвящена следующая часть.

Информационная база

Прикладное решение, которое должно получиться у вас к этому моменту, содержится в демонстрационной базе «03 Визуальное Конструирование Итог.dt». Как её подключить, написано в разделе А.1 «Как подключить демонстрационную базу» на странице 543.

Глава 3

Встроенный язык

Не читайте эту главу!

Если вы знаете:

- зачем нужны модули и события;
- чем значение отличается от типа и от представления;

вы можете смело перейти к разделу [3.8 «Где писать примеры и чем пользоваться»](#) на странице [194](#).

Если кроме этого вы знаете:

- почему текст программы разноцветный;
- когда нужна переменная, а когда нужен литерал;
- какие инструкции используются чаще всего;
- как написать условие и выполнить цикл;
- в чём разница между функцией и процедурой;
- что такое контекст и область видимости;

вы можете смело перейти к разделу [3.10 «Коллекции значений»](#) на странице [278](#).

Если кроме этого вы знаете:

- чем массив отличается от структуры;
- как знакомиться с незнакомыми объектами;
- как выполнить программу по шагам и посмотреть значения переменных;
- что называют коллекцией значений, а что — объектом встроенного языка;

вы можете смело перейти к разделу [3.11 «Прикладные типы»](#) на странице [309](#).

Если кроме этого вы знаете:

- чем объектные данные отличаются от необъектных;
- что такое контекст клиента и контекст сервера;
- как прочитать или записать документ;

смело переходите к главе [4 «Автоматическое заполнение расписания»](#) на странице [351](#).

Информационная база

Прикладное решение, которое должно получиться у вас к этому моменту, содержится в демонстрационной базе «03 Визуальное Конструирование Итог.dt». Как её подключить, написано в разделе [А.1 «Как подключить демонстрационную базу»](#) на странице [543](#).

Итак наконец-то вы зашли в самую глубь 1С:Предприятия. Глубже уже некуда. Если сравнивать ваши занятия с прогулкой по лесу, то сейчас вы зашли в лес 1С:Предприятие, забрались на одно из деревьев и притянули к себе одну из небольших веток. Сейчас вы будете изучать, как устроены отдельные листья на этой ветке (рисунок [3.1](#)).



Рисунок 3.1. Сейчас вы тут

Зачем это нужно? Это нужно для того, чтобы вы могли сделать своё дерево таким, какое нужно именно вам.

До сих пор вы использовали средства визуального конструирования. С их помощью вы добавили в конфигурацию несколько объектов. Платформа знает, что делать с этими объектами, как их показывать, как изменять и сохранять их данные. Но это только некоторые стандартные действия. Такие действия, которые требуются в любом прикладном решении.

Кроме этого каждое прикладное решение, в том числе и ваше, требует каких-то особых, специфических действий. Таких действий, которые присущи только вашему прикладному решению.

Например, автогенерируемые формы не всегда вас устраивали. И вы заменили их собственными. Чтобы прикладное решение показывало данные так, как хочется вам.

Теперь вы пойдёте ещё дальше и научите прикладное решение выполнять разные действия, которые нужны вам.

Это делается с помощью *встроенного языка*. На нём вы сможете написать свои команды, которые прикладное решение выполнит в определённый момент.

Чтобы понять, как это работает, выполните сначала небольшой пример. А потом мы его обсудим.

3.1 Ваша первая программа — заголовок приложения

Сейчас в заголовке вашего приложения написано *Дневник*. Потому что в самом начале вы дали своей конфигурации такой синоним — *Дневник* (рисунок [3.2](#)).

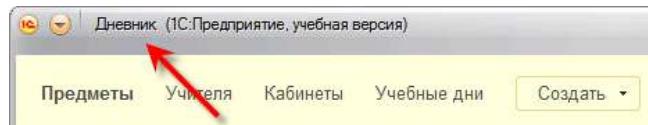


Рисунок 3.2. Заголовок приложения

Теперь вы сделаете так, чтобы при запуске прикладного решения в его заголовке появлялись ваши имя и фамилия. Чтобы было понятно, что это именно ваш дневник.

Откройте своё прикладное решение в конфигураторе.

В корне конфигурации вызовите контекстное меню.

Выполните команду *Открыть модуль управляемого приложения* (рисунок 3.3).

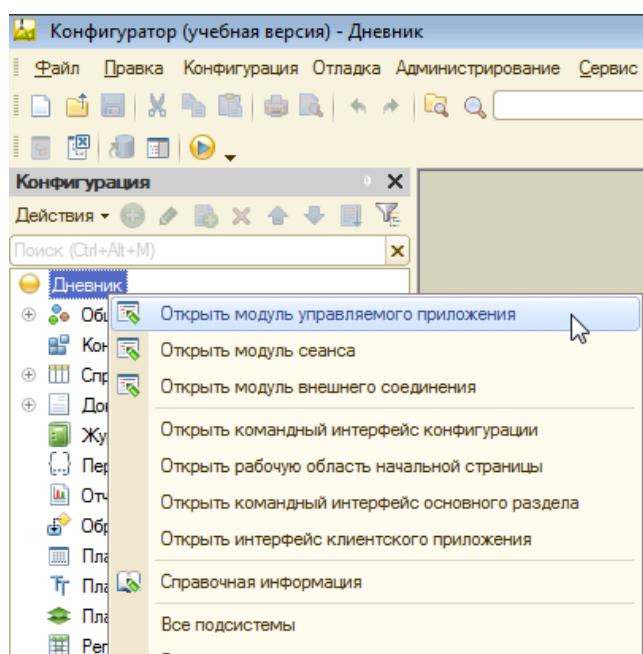


Рисунок 3.3. Открыть модуль управляемого приложения

Конфигуратор откроет вам пустой *модуль управляемого приложения*.

В поле выбора процедуры нажмите кнопку выпадающего списка.

Выберите строку <ПриНачалеРаботыСистемы> (рисунок 3.4).

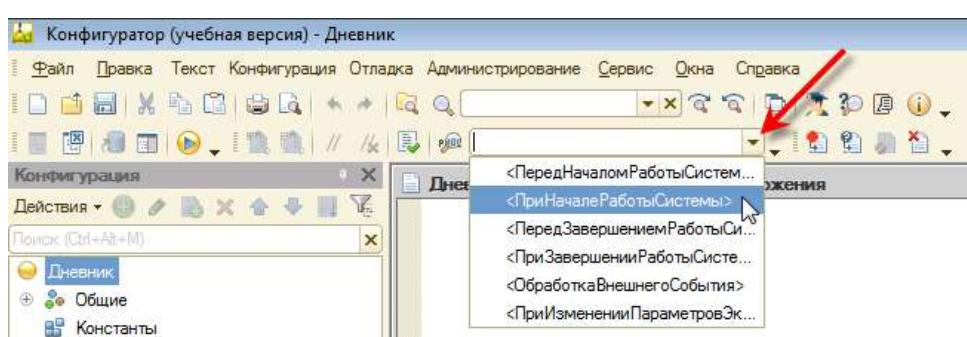


Рисунок 3.4. Выбор процедуры «ПриНачалоРаботыСистемы»

В модуле появится заготовка процедуры (листинг 3.1).

Листинг 3.1. Заготовка процедуры «ПриНачалоРаботыСистемы»

Процедура ПриНачалоРаботыСистемы()

// Вставить содержимое обработчика.

КонецПроцедуры

Вместо текста // Вставить содержимое обработчика напишите следующую команду (листинг 3.2).

Листинг 3.2. Установка заголовка приложения

```
УстановитьКраткийЗаголовокПриложения("Иванов Петя");
```

Здесь вместо *Иванов Петя* можете написать свои фамилию и имя. В результате у вас должна получиться такая программа (листинг 3.3).

Листинг 3.3. Обработчик «ПриНачалРаботыСистемы»

```
Процедура ПриНачалРаботыСистемы()
```

```
УстановитьКраткийЗаголовокПриложения("Иванов Петя");
```

КонецПроцедуры

Теперь запустите свою конфигурацию в режиме отладки и посмотрите на заголовок приложения (рисунок 3.5).

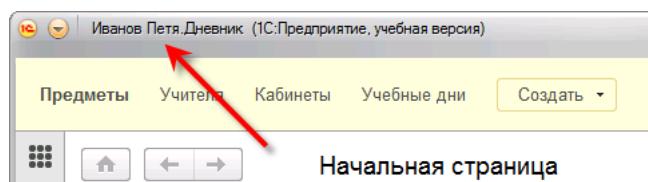


Рисунок 3.5. Новый заголовок приложения

Он изменился.

А теперь я расскажу о том, что вы сейчас сделали.

3.2 События

Любое прикладное решение 1С:Предприятия имеет совершенно определённый жизненный цикл. Прикладное решение запускается. Открывается основной раздел. После этого прикладное решение ждёт действий пользователя.

Пользователь может открыть какую-то форму. Может изменить в ней какие-то данные. Может закрыть форму.

В конце концов пользователь решает, что он сделал всё, что хотел. Тогда он закрывает программу. Работа прикладного решения завершается.

Этот жизненный цикл фиксирован. Он определён платформой, и изменить его нельзя. Но в нём есть отдельные моменты, в которые вы можете вмешаться. Вот, например, в один из таких моментов вы только что вмешались и заставили прикладное решение выполнить вашу команду. В результате этого изменился заголовок приложения.

Такие моменты, когда вы можете вмешаться в «жизнь» прикладного решения, называются *события*. Набор этих событий фиксирован, постоянен. Его определяет платформа. Придумать какое-то своё событие вы не можете. Но вы можете использовать любое событие из тех, которые имеются.

События в прикладном решении очень похожи на события, которые происходят в вашей жизни постоянно, изо дня в день. Например, утром вы просыпаетесь и встаёте с кровати.

Первое событие, которое происходит с вами, можно назвать «перед тем, как встать с кровати».

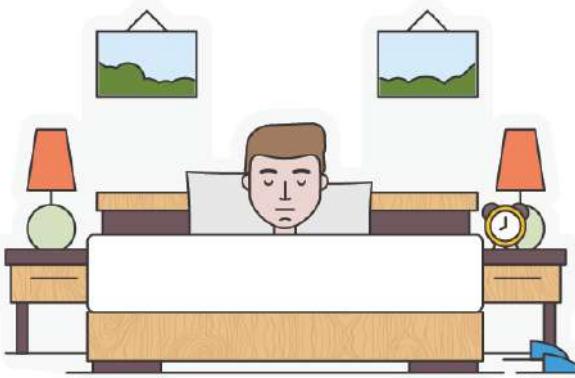


Рисунок 3.6. Перед тем, как встать с кровати

Когда в вашей жизни наступает такое событие, какие возможны варианты? Например, вы посмотрите на часы и решите не вставать, а поспать ещё. Или вы посмотрите не на часы, а на окружающих и решите: «Раз все спят, я тоже посплю». То есть при наступлении события «перед тем, как встать с кровати» вы анализируете доступную вам информацию и принимаете решение, вставать с кровати или нет.

Следующее событие, которое происходит с вами, можно назвать «при вставании с кровати».

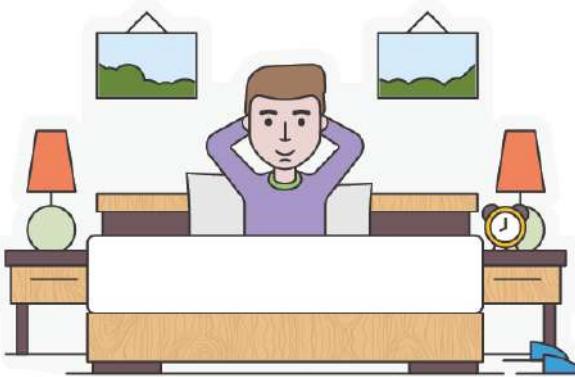


Рисунок 3.7. При вставании с кровати

Какие варианты действий есть у вас при наступлении этого события? Например, потянуться и надеть тапочки. Или разбудить брата. Или разбудить детей. То есть при наступлении события «при вставании с кровати» вы выполняете какой-то набор дополнительных действий.

Всё то же самое происходит и в прикладном решении. Когда прикладное решение запускается и начинает работать, происходят два события, в которые вы можете вмешаться. Первое называется *ПередНачаломРаботыСистемы*, а второе называется *ПриНачалеРаботыСистемы*.

Слова «перед» и «при» встречаются во многих событиях 1С:Предприятия. Слово «перед» обозначает тот момент, когда действие ещё не наступило. То есть когда система ещё не начала работать. Когда вы ещё не начали вставать с кровати.

А слово «при» означает именно тот момент, когда действие уже выполняется. Когда система уже начинает работать. Когда вы точно встаёте с кровати.

Когда вы устанавливали заголовок приложения, вы как раз и вмешались в одно из этих событий. В событие *ПриНачалоРаботыСистемы*. В этом событии вы заставили программу выполнить вашу команду.

1С:Предприятие содержит большое количество событий. Вы не будете изучать их все. В этом нет никакой необходимости. По ходу книги вы познакомитесь лишь с некоторыми из них.

Но я научу вас, как самостоятельно узнать, какие события существуют. И где прочитать о том, когда эти события возникают и для чего они нужны.

3.3 Модули

Каким образом вы вмешались в событие *ПриНачалоРаботыСистемы*? Вы открыли какой-то модуль и что-то в нём написали. *Модуль* — это такое место в конфигурации, куда вы можете написать свои команды. В конфигурации много модулей, они расположены в разных её частях.

Когда в жизни вашего приложения возникает очередное событие, платформа идёт в один из модулей и смотрит, есть ли там команды, которые нужно выполнить. Если есть, то сначала выполняет их, а потом — всё то, что она обычно делает в этом случае.

Как платформа узнаёт, в какой модуль ей нужно идти? Очень просто. Для каждого события заранее известно, в каком модуле нужно искать команды.

Например, когда вы запускаете приложение, возникают два события. Сначала возникает событие *ПередНачалоРаботыСистемы*. Платформа знает, что команды, которые она должна выполнить при наступлении этого события, находятся в *модуле управляемого приложения*. Она идёт в этот модуль и смотрит, есть ли там команды для события *ПередНачалоРаботыСистемы* (листинг 3.4).

Листинг 3.4. Модуль управляемого приложения

Процедура ПриНачалоРаботыСистемы()

 УстановитьКраткийЗаголовокПриложения("Иванов Петя");

КонецПроцедуры

Для этого события там нет команд. Вы их не добавляли. Значит, платформа просто продолжает делать то, что она обычно делает в этой ситуации. Начинает запускать ваше приложение.

Тут возникает второе событие — *ПриНачалоРаботыСистемы*. Платформа снова идёт в модуль управляемого приложения. Потому что она знает, что команды для этого события (если они есть) написаны именно в этом модуле. И они есть!

Тогда она сначала выполняет вашу команду — устанавливает краткий заголовок приложения. А затем уже делает всё то, что она обычно делает в этой ситуации. То есть показывает на экране ваше приложение, показывает начальную страницу и список учебных дней.

После этого она ждёт ваших дальнейших действий (рисунок 3.8).

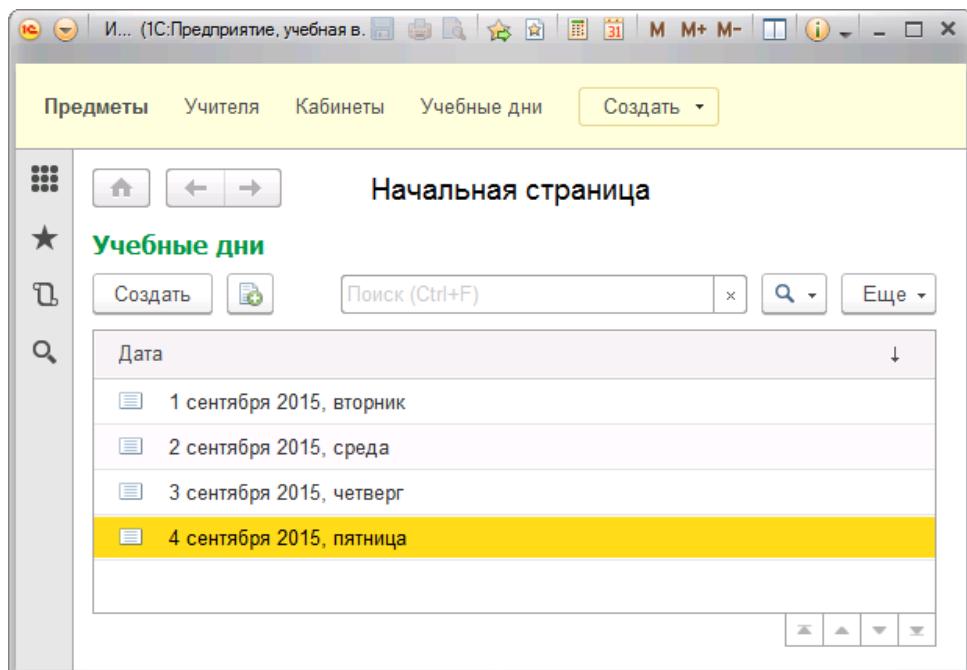


Рисунок 3.8. Приложение запустилось и ждёт ваших действий

Вы, например, решаете посмотреть, какие занятия будут 4 сентября. Дважды щёлкните мышью по этой строке. В этот момент платформа открывает вам форму документа УчебныйДень и показывает её.

Когда форма открывается, тоже происходит несколько событий. Но эти события связаны уже не со всей конфигурацией, а именно с этой формой. Поэтому команды для этих событий платформа будет искать уже в *модуле формы* документа УчебныйДень.

Модуль формы есть у каждой формы, которую вы добавили в конфигурацию. Найти его очень просто. Нужно в конфигураторе открыть нужную форму для редактирования, и внизу вы увидите две закладки. Откройте, например, форму документа УчебныйДень (рисунок 3.9).

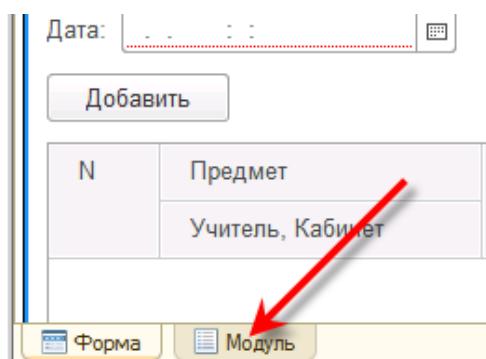


Рисунок 3.9. Закладки «Форма» и «Модуль»

Конструктор формы всегда открывается на первой закладке *Форма*. Но если вы перейдёте на вторую закладку, *Модуль*, то вы увидите модуль этой формы. Сейчас он пустой.

Когда вы решите закончить работу со своим прикладным решением в режиме 1С:Предприятие, также будут вызваны два события, связанные с завершением работы. Команды для этих событий платформа снова будет искать в модуле управляемого приложения.

Примечание

Модули в конфигурации расположены в тех местах, которые логически связаны с происходящими событиями.

Например, если события относятся ко всему приложению в целом, то модуль расположен в корне конфигурации. Если события относятся к какой-то форме, то модуль расположен рядом с этой формой.

Кроме модуля управляемого приложения и модулей форм в конфигурации есть и другие модули. Они расположены в конфигурации точно по такому же принципу. Некоторые из них вы позже рассмотрите.

Подробнее

Подробнее вы можете прочитать про модули в документации «Руководство разработчика 8.3. Раздел 4.2.1. Что такое программный модуль».

3.4 Встроенный язык

Теперь я расскажу о том, что вы написали в модуле. В модуле вы написали небольшую программу на встроенном языке 1С:Предприятия (листинг 3.5).

Листинг 3.5. Программа в модуле

Процедура ПриНачалеРаботыСистемы()

УстановитьКраткийЗаголовокПриложения("Иванов Петя");

КонецПроцедуры

Встроенный язык — это язык программирования, который придумала фирма «1С». Специально для того, чтобы вы могли на нём описать собственный набор действий, который должно выполнить ваше приложение.

С одной стороны, встроенный язык имеет много общих черт с другими языками, такими как Pascal, Java Script, Basic. С другой стороны, встроенный язык специально разрабатывался так, чтобы его могли использовать не только профессиональные программисты.

Например, все ключевые слова и инструкции языка имеют русское написание. Поэтому вам не нужно знать английский язык, для того чтобы написать собственную программу.

Прежде чем вы перейдёте к изучению самого языка, нужно овладеть тремя очень важными понятиями: значение, тип и представление. Без них вам будет сложно разбираться в текстах программ.

Подробнее

Подробнее вы можете прочитать про встроенный язык в документации «Руководство разработчика 8.3. Глава 4. Встроенный язык».

3.5 Значение

Что делает программа? Вы можете сказать, что любая программа выполняет какой-то набор действий. Что она содержит последовательность команд, которые выполняются друг за другом. Да, это правильно. Если смотреть на программу, так сказать, издалека.

Но что если взглянуть на неё более внимательно и пристально? В этом случае вы увидите, что любая программа занята исключительно тем, что выполняет какие-то действия со *значениями*. Причём все эти действия совершенно чётко делятся на три большие группы.

Сначала программа получает какие-то значения. Может быть, эти значения были записаны на диске. Тогда она читает их с диска. Может быть, вы сами ввели эти значения с клавиатуры. Например, когда вы добавляли кабинеты и набрали номер кабинета 101. 101 — это значение, которое вы передали программе. Есть много способов, которыми программа может получить значения. Это не важно. Самое главное, что программа их откуда-то получает.

Это очень похоже на то, что вы делаете на берегу реки или моря, когда вам становится скучно. Вы идёте собирать ракушки. Или красивые камни (рисунок 3.10). Это ваши «значения».



Рисунок 3.10. Вы собираете «значения»

Затем вы что-то делаете с камнями или ракушками. Например, стройте из них пирамидку (рисунок 3.11).



Рисунок 3.11. Вы выполняете различные действия со своими «значениями»

Точно так же и программа что-то делает с теми значениями, которые она получила. Например, вы ввели время начала занятий и время окончания занятий. Тогда программа может из одного значения вычесть другое и получить третье значение — продолжительность ваших занятий. Или, например, вы ввели значение 101. Это номер кабинета, в котором будут проходить занятия. А программа может проверить, есть у вас в списке такой кабинет или ещё нет.

В конце концов любая программа завершает свою работу тем, что выводит значения. То есть показывает их вам на экране. Или записывает их на диск. Или печатает на принтере. Или передаёт другому компьютеру. Конкретный способ тоже не важен. Важно лишь то, что значения являются ещё и результатом деятельности программы.

А что является результатом вашей деятельности на берегу моря? Правильно, красивый замок из камней, водорослей, ракушек (из ваших «значений»), который вы оставите на память другим отдыхающим (рисунок 3.12).

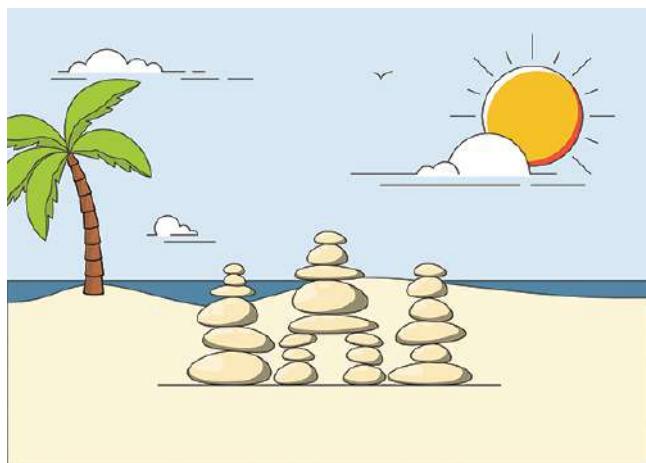


Рисунок 3.12. Результат вашей работы

Примечание

Основная вещь, которой занимается программа, — это значения. Она их получает, выполняет с ними какие-то действия и показывает.

3.6 Тип

Ваши «значения», которые вы собираете на берегу моря, могут быть самыми разными. Это могут быть необычные ракушки или красивые камни (рисунок 3.13). Главное, что они вас чем-то заинтересовали.

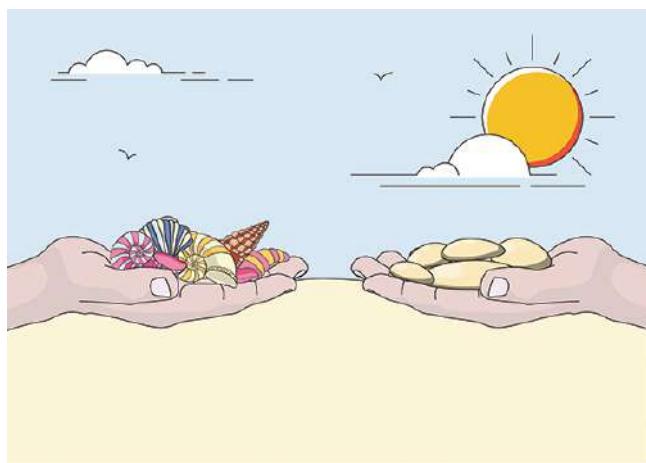


Рисунок 3.13. Значения могут быть самыми разными

Дальше вы начинаете из них что-то мастерить. Например, подходящие камни можно сложить в пирамиду. Потому что они плоские и могут лежать друг на друге (рисунок 3.14).

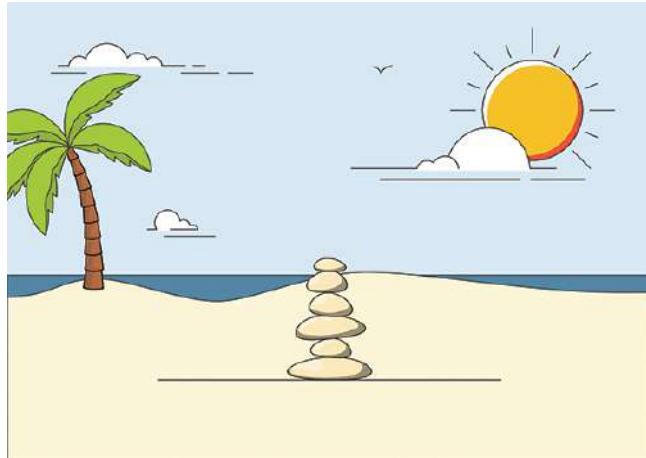


Рисунок 3.14. Башня из камней

А ракушки в такую пирамидку уже не сложишь. Потому что они не будут лежать друг на друге. Зато из ракушек можно сложить горку, которая будет похожа на сказочное морское животное (рисунок 3.15).



Рисунок 3.15. Пирамида из ракушек

То есть не со всеми «значениями», которые вы собрали на берегу моря, можно делать одно и то же. Но с однотипными предметами можно делать одинаковые действия.

Точно так же и в компьютере. В компьютере может существовать много самых разных значений. Чтобы программа могла что-то делать с этими значениями, она должна знать их *тип*. С типами вы уже сталкивались, когда добавляли справочники Кабинеты или Учителя. Например, чтобы записывать номера кабинетов, вы выбрали тип Число. А чтобы записывать фамилии преподавателей, вы выбрали тип Стока.

Тогда это было понятно и естественно. Потому что номер кабинета — это набор цифр. А фамилия преподавателя — это последовательность букв. В тот момент вы не задумывались о том, для чего это может понадобиться. А теперь есть удобная возможность с этим разобраться.

Когда вы вводите номер кабинета — это значение. Когда вы вводите фамилию преподавателя — это другое значение. Программа нужна для того, чтобы что-нибудь сделать с этими значениями.

Как вы думаете, что будет, если программа попытается сложить 101 и «Казаков К. Д.»? Или разделить «Герасимова Е. С.» на «Давыдова А. В.»? Ничего хорошего из этого не получится.

Чтобы вышел какой-то толк, программа прежде всего должна знать, какой тип имеют те значения, с которыми ей предстоит работать. Например, если это числа, то их можно складывать, делить, умножать, вычитать. А если это строки, то умножать и делить их нельзя. Их можно только сложить. На языке компьютера это будет означать, что к одной строке нужно дописать другую строку. А если значения имеют разные типы (например, одно значение имеет тип *Число*, а другое значение имеет тип *Строка*), то с такими значениями вообще никаких действий произвести нельзя.

Для чего ещё программе нужно знать тип значений? Чтобы записать их на диск, например. Потому что числа хранятся одним образом, а строки — другим. А если вы захотите записать на диск картинку, то программа выберет для этого третий способ хранения данных.

Примечание

Каждое значение относится к какому-либо типу. Говорят: «Значение имеет тип *Строка*». Или: «Значение имеет тип *Число*».

Если бы вы спросили компьютер, что он видит на картинке 3.14, он бы ответил, что на ней «значения типа "ПлоскийКамень"». А на картинке 3.15 — «значения типа "Ракушка"».

3.7 Представление

Пока программа записывает значения на диск или передаёт их другому компьютеру, она это делает так, как удобно ей, и в такой форме, которая ей удобна. Но когда эти значения нужно показать на экране или напечатать, программа не может делать это так, как ей хочется. Она должна сделать это так, чтобы нам с вами или любому другому пользователю было понятно, что это за значения.

Как раз для этого и нужно *представление*. Каждый тип имеет собственное представление. То есть способ, которым значения такого типа нужно показывать на экране.

Например, представление типа *Число* — это последовательность цифр. В этой последовательности может встретиться одна запятая. Она отделяет целую часть числа от дробной части. Например, 346,45 или 58,2. В этой последовательности может встретиться пробел. Он отделяет друг от друга группы разрядов. Это помогает легче читать большие числа. Например, 1 475 или 395 987,41.

А у типа *Строка* представление другое. Это просто последовательность любых символов. В этой последовательности тоже может встретиться пробел или запятая. Но никакого особенного смысла они не имеют. Например, «Давыдова А. В.» или «математика, мой любимый предмет».

Примечание

Таким образом, каждый тип имеет представление. Это правило, по которому обозначаются значения этого типа.

Соответственно, и каждое значение тоже имеет своё представление. Это набор символов, которым обозначается это значение на экране или на принтере.

Теперь, когда вы познакомились с основными понятиями, вы можете перейти к упражнениям.

3.8 Где писать примеры и чем пользоваться

Чтобы познакомиться со встроенным языком, вы будете использовать самый простой способ. Примеры вы будете писать в модуле управляемого приложения, после той строчки, которая там уже есть (листинг 3.6).

Листинг 3.6. Модуль управляемого приложения

Процедура ПриНачалеРаботыСистемы()

```
УстановитьКраткийЗаголовокПриложения("Иванов Петя");
```

КонецПроцедуры

При этом вы сразу же будете использовать все инструменты, которые используют разработчики.

Во-первых, это *синтакс-помощник*. В нём описаны все команды, которые есть во встроенном языке. Описаны типы данных, которые вы можете использовать. Также в нём описаны операции, которые можно выполнять со значениями этих типов. То есть это такой большой справочник.

Кроме этого вы сразу же будете использовать основной, наверное, инструмент разработчика — *отладчик*. Он позволяет найти ошибки в программе. Позволяет остановиться в нужном месте программы. Он даже позволяет выполнить программу по отдельным шагам, чтобы вы могли посмотреть, что получается на каждом шаге.

Кроме этого вы научитесь пользоваться *контекстной подсказкой*. Она облегчает написание текста программы. Позволяет вам избежать многих ошибок и описок. Она может подсказать вам, какие слова или команды вы можете использовать в том или ином месте программы.

Теперь откройте в конфигураторе модуль управляемого приложения, и приступим.

3.9 Простые типы

3.9.1 Почему текст разноцветный

Почему слова, которые написаны в модуле, разного цвета? Почему некоторые из них красные, некоторые синие, а некоторые даже чёрные?

Всё очень просто. Они разного цвета для того, чтобы вы не запутались. Чтобы вам было легче читать то, что написано. И дальше вы увидите, что они могут быть даже зелёные и коричневые!

Когда вы что-то пишете в модуле, вы на самом деле объясняете компьютеру, что он должен сделать. Объясняете на встроенном языке, который понятен и ему, и вам. Этот язык очень похож на наш обычный язык. Только он гораздо проще. И в нём есть только такие предложения, которые выражают приказ, просьбу или совет.

Чтобы было понятно, представьте, что вы позвали друга к себе в гости. Вы объясняете ему, как добраться до вашего дома. Только вы очень спешите, поэтому говорите быстро и коротко:

— Садись в автобус № 395. Выйди на 4-й остановке. Перейди на другую сторону улицы. Заходи в дом № 15. Если автобус проедет мимо магазина «Продукты», значит, ты пропустил нужную остановку. Выйди из автобуса на следующей остановке и вернись на одну остановку назад.

В обычной жизни вы слышите и понимаете эти предложения последовательно, одно за другим. Если переставить предложения местами, получится ерунда. В результате ваш друг вообще не сможет никуда уехать или уедет в другой конец города:

— Перейди на другую сторону улицы. Садись в автобус № 395. Выйди из автобуса на следующей остановке и вернись на одну остановку назад. Заходи в дом № 15. Если автобус проедет мимо магазина «Продукты», значит, ты пропустил нужную остановку. Выйди на 4-й остановке.

Точно так же и в компьютере. Текст, который вы будете писать на встроенном языке, состоит из отдельных предложений. Каждое такое предложение называется *инструкция*. Порядок этих предложений (инструкций) важен, потому что компьютер «читает» их одно за другим. В той последовательности, в которой они написаны.

В обычной жизни мы отделяем одно предложение от другого точкой и пробелом. Иногда могут использоваться и другие символы, но чаще всего это точка. Во встроенном языке одна инструкция от другой отделяется символом «точка с запятой» — «;». Кроме того, каждую инструкцию принято писать на новой строке. Поэтому записка для вашего друга, написанная на языке компьютера, была бы оформлена таким образом (листинг 3.7).

Листинг 3.7. Записка для вашего друга на языке компьютера

```
Садись в автобус № 395;  
Выйди на 4-й остановке;  
Перейди на другую сторону улицы;  
Заходи в дом № 15;  
Если автобус проедет мимо магазина «Продукты», значит, ты пропустил нужную остановку;  
Выйди из автобуса на следующей остановке и вернись на одну остановку назад;
```

Сейчас в модуле управляемого приложения вы написали одну инструкцию. Если бы вы писали записку своему другу, то это было бы первое предложение: «Садись в автобус № 395». И платформа раскрасила бы его вот так (листинг 3.8).

Листинг 3.8. Первое предложение из записи вашему другу

```
УстановитьКраткийЗаголовокПриложения("Иванов Петя");
```

```
Садись в автобус № 395;
```

Синим цветом платформа раскрасила бы то, что вашему другу известно и понятно без вашего объяснения. Ваш друг знает, что существуют автобусы. Он знает, что в автобус можно сесть. Поэтому вы не рассказываете вашему другу, что такие автобусы, для чего они нужны. Не рассказываете, что автобусы умеют ездить. Что они останавливаются только на остановках. Не объясняете, что в автобус можно сесть, можно выйти из него. Ваш друг знает это всё и без вас. Вы просто говорите ему «садись в автобус».

Точно так же «УстановитьКраткийЗаголовокПриложения» — это нечто, что платформе известно. Нечто, что она умеет и может делать без дополнительных объяснений. Поэтому она раскрашивает это синим цветом.

Примечание

Итак, синим цветом платформа выделяет то, что ей известно. То, что она знает и умеет выполнять.

А что платформа раскрашивает красным цветом? А красным цветом она раскрашивает те символы и слова, которые являются обязательными. Без которых не получится правильная инструкция, правильное предложение.

Например, красным цветом она раскрашивает символ «точка с запятой», который находится в конце строки. Потому что каждая инструкция обязательно должна заканчиваться этим символом. Так же как в русском языке каждое предложение должно заканчиваться точкой.

Ещё красным цветом она раскрасила бы символ «№» в вашем предложении. Потому что, когда вы говорите «садись в автобус», всегда подразумевается, что дальше вы должны сказать, в какой именно автобус садиться. И чтобы было понятно, что сейчас вы назовёте номер автобуса, в русском языке вы пишете символ «№». То есть это тоже обязательный символ, который должен быть.

По этой же причине платформа раскрашивает красным цветом скобки в первой строке. Платформа знает, что такое «УстановитьКраткийЗаголовокПриложения». Но также она знает, что после этого обязательно должен быть указан сам заголовок, который нужно установить. Этот заголовок она будет искать внутри скобок, которые обязательно должны быть в такой инструкции.

Примечание

Красным цветом платформа выделяет то, что является обязательным. То, что обязательно должно быть в этой инструкции. То, без чего инструкция будет неправильной.

Теперь осталось разобраться с чёрным цветом. А с ним всё просто. Чёрным цветом платформа выделяет значение. Значение, которое вы написали прямо в тексте программы. Такие значения, написанные прямо в тексте программы, называются *литералами*.

Когда вы говорите другу, что ему нужен автобус № 395, вы сообщаете ему конкретное значение — 395.

Когда вы говорите программе, что нужно установить заголовок «Иванов Петя», вы тоже сообщаете ей конкретное значение — *Иванов Петя*.

Примечание

Чёрным цветом платформа выделяет значения, записанные прямо в тексте программы. Такие значения называются литералами.

На этом примере хорошо видно, что не все литералы записываются одинаково. Существуют определённые правила.

Если значение, которое вы хотите написать в тексте программы, имеет тип *Число*, то вы пишете его так, как вы обычно привыкли. Между цифрами не ставите пробелов, а вот дробную часть отделяете от целой части с помощью точки, а не с помощью запятой. Литералы типа *Число* могут выглядеть так (листинг 3.9).

Листинг 3.9. Литералы типа «Число»

395
2016
3.14
29748.0021

А если значение, которое вы хотите написать, имеет тип *Строка*, то эту строку вы обязательно должны заключить в прямые кавычки. Причём на клавиатуре есть две похожих кнопки. Одинарная кавычка — «'» и двойная кавычка — «"». Вам нужно использовать двойную кавычку. Литералы типа *Строка* могут выглядеть так (листинг 3.10).

Листинг 3.10. Литералы типа «Строка»

```
"Иванов Сергей Николаевич"  
"сегодня 1 апреля 2016 года."  
"если я дочитаю до конца, я стану крутым программистом"  
"
```

Совет

Обратите внимание, что в начале и в конце строкового литерала могут находиться пробелы. Это вполне нормальная ситуация. Чуть позже вы увидите, зачем это может понадобиться.

3.9.2 Какие бывают инструкции

В русском языке вы можете составлять самые разные предложения. Они могут быть короткие. Например: «Садись в автобус № 395». Они могут быть совсем короткие, например: «Садись!» Они могут быть длинные и очень длинные, например: «Если ты увидел, что подъехал автобус № 395 или № 18, то можешь сесть в него, хотя и на трамвае тоже можно доехать, но в нём обычно бывает много народа». То есть тут всё зависит только от вашего желания, словарного запаса и привычек.

Во встроенном языке всё гораздо проще. Нельзя писать какие угодно инструкции. Существует всего лишь 5 основных инструкций, которые применяются наиболее часто. Это:

- инструкция присваивания;
- инструкция *Если*;
- инструкция вызова процедуры;
- инструкция *Цикл*;
- инструкция *Попытка*.

Вы изучите первые четыре и, поверьте, этого вам хватит для большинства задач. На самом деле во встроенном языке есть ещё несколько инструкций, но используются они крайне редко. Поэтому я даже не буду упоминать их.

Кстати, инструкция, которую вы написали в модуле управляемого приложения (листинг 3.11), — это инструкция вызова процедуры.

Листинг 3.11. Инструкция вызова процедуры

```
УстановитьКраткийЗаголовокПриложения("Иванов Петя");
```

Подробнее

Подробнее вы можете прочитать про инструкции в документации «Руководство разработчика 8.3. Раздел 4.6. Операторы и синтаксические конструкции».

3.9.3 Инструкция присваивания

Самая простая и самая популярная инструкция во встроенном языке — это *инструкция присваивания*. Выглядит она очень просто (листинг 3.12).

Листинг 3.12. Пример инструкции присваивания

```
КоличествоЗанятий = 6 * 5;
```

Почему количество занятий равно 6 умножить на 5? Потому что, например, каждый день у вас по 6 занятий и вы занимаетесь 5 дней в неделю.

Отличительным признаком инструкции присваивания является знак равенства «=». Это обязательный символ в этой инструкции. Поэтому платформа выделяет его красным цветом.

Подробнее

Полный список специальных символов вы можете посмотреть в документации «Руководство разработчика 8.3. Раздел 4.2.5. Специальные символы, используемые в исходном тексте».

Знак равенства работает как экскаватор (рисунок 3.16).

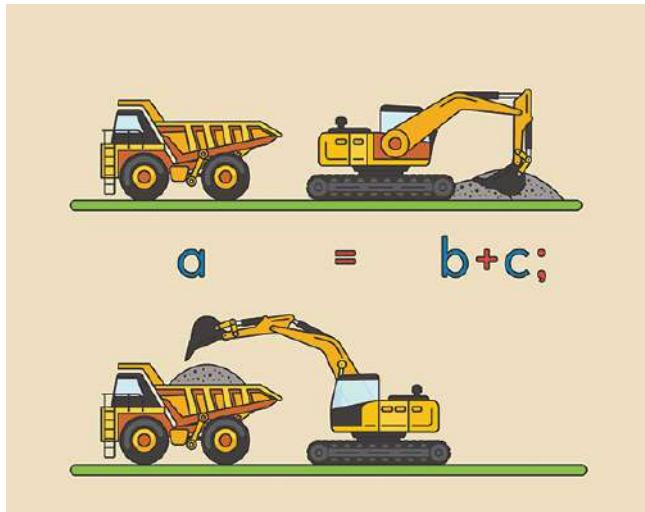


Рисунок 3.16. Знак равенства работает как экскаватор

Он берёт то, что справа, и кладёт это в то место, которое находится слева от знака равенства. Тут важно обратить внимание на следующее. Сначала экскаватор захватывает своим ковшом всё, что ему нужно. И только после этого он переносит всё это в другое место.

Во встроенном языке то же самое. Сначала платформа вычисляет всё, что находится справа от знака равенства. И только после этого она помещает результат в то место, которое обозначено слева от знака равенства.

Зачем экскаватор переносит землю с одного места на другое? Для этого могут быть две причины.

Во-первых, это нужно для того, чтобы отвезти землю куда-то в другое место. И там её использовать.

Во-вторых, это нужно тогда, когда ничего никуда везти не надо. А надо просто засыпать яму, которая рядом с экскаватором (рисунок 3.17).

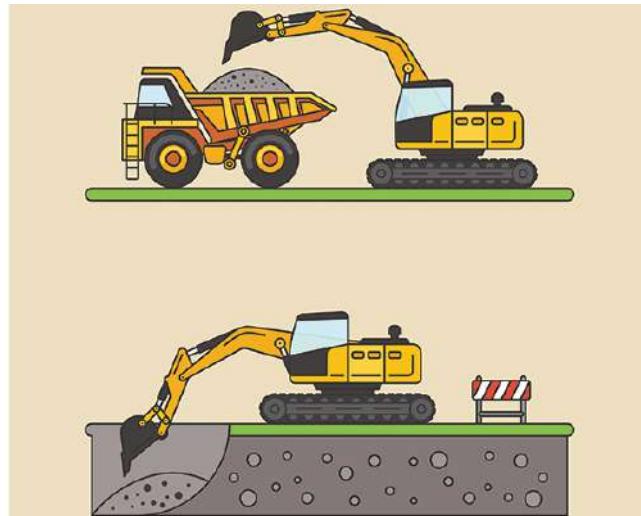


Рисунок 3.17. Зачем экскаватор переносит землю?

Во встроенном языке то же самое. Есть два случая, когда используется инструкция присваивания.

Во-первых, для того чтобы получить (вычислить) какое-то значение и использовать его в другом месте. В этом случае схематически инструкция выглядит так:

Листинг 3.13. Инструкция присваивания

<Имя переменной> = <Выражение>;

Слева находится имя переменной, а справа — некоторое выражение. Это вы сейчас и рассмотрите.

Во-вторых, инструкция присваивания используется тогда, когда значение, которое вы получите или вычислите, нужно вам прямо тут, прямо в этом месте. Тогда схематически инструкция выглядит так:

Листинг 3.14. Инструкция присваивания

<Имя свойства> = <Выражение>;

Слева находится имя свойства, а справа — снова какое-то выражение. Этот случай вы тоже рассмотрите, но позже.

Итак, справа от знака равенства находится непонятное для вас пока «выражение». А слева — тоже непонятная пока для вас «переменная». Сначала вы познакомьтесь с переменными.

Подробнее

Подробнее вы можете прочитать про инструкцию присваивания в документации «[Руководство разработчика 8.3. Раздел 4.4. Оператор присваивания](#)».

3.9.4 Переменная

Вспомните, как вы собирали красивые камни на берегу моря (рисунок 3.18).



Рисунок 3.18. Вы собираете «значения»

Тогда вы говорили, что вы собираете «значения» и каждый камень — это одно значение.

Так вот, *переменная* — это ваша ладонь. Только если в жизни вы можете положить в ладонь несколько камней, в компьютере в переменную можно положить только одно значение, один камень.

Одну переменную от другой компьютер отличает по имени. Имя вы придумываете сами.

Переменных может быть сколько угодно, пока у компьютера не закончится оперативная память.

Переменные существуют не всегда, а только в то время, пока выполняется ваша программа. Поэтому, когда ваша программа перестанет выполняться, все переменные исчезнут.

Примечание

Переменная — это специальное место в памяти компьютера. В этом месте можно сохранить одно значение на то время, пока выполняется ваша программа.

Для того чтобы создать переменную, вам не нужно ничего делать. Нужно только придумать для неё имя и написать это имя слева от знака равенства в инструкции присваивания. Когда компьютер начнёт выполнять вашу программу, он сам создаст переменную и поместит в неё значение, которое вы написали справа от знака равенства.

Попробуйте. Напишите, сколько вам лет (листинг 3.15).

Листинг 3.15. Мой возраст

```
МойВозраст = 25;
```

В компьютере у вас должно получиться примерно то же самое, что в листинге 3.16.

Листинг 3.16. Переменная «МойВозраст»

```
Процедура ПриНачалеРаботыСистемы()
```

```
УстановитьКраткийЗаголовокПриложения("Иванов Петя");
```

```
МойВозраст = 25;
```

```
КонецПроцедуры
```

Совет

Сразу возьмите себе за правило ставить пробелы перед знаком «=» и после него. Тогда текст программы будет читаться легко.

Наверняка вы помните, по каким правилам нужно придумывать имена. Если не помните, то я напомню.

Имя должно начинаться с буквы и не должно содержать пробелов. Если хочется составить имя из нескольких слов, каждое следующее слово принято писать с большой буквы. Буква «ё» в именах не используется, вместо неё надо писать букву «е».

Совет

Придумывайте такие имена, чтобы они были понятны и вам, и другому человеку. Не стесняйтесь использовать несколько слов для составления имени. Два, три слова — это нормально.

Когда имя очень короткое или оно является каким-то сокращением, другому человеку будет трудно понять, что вы имели в виду. Да и вы, если посмотрите на свою программу через несколько месяцев, с трудом вспомните, что вы хотели сказать таким именем. Вот, например, плохие имена: «доб», «стро», «квоТТ».

Очень длинное имя — это тоже нехорошо. Например, «ДеньгиНалиБезНалМеждунамиИКонтрагентами». Использовать такую переменную в каких-нибудь формулах будет очень неудобно.

В общем, старайтесь быть краткими и в то же время понятными.

А теперь познакомьтесь с тем, как работает эта инструкция присваивания. Вы не будете никуда ничего выводить, чтобы где-то в программе можно было увидеть значение этой переменной. Вы сразу поступите как настоящие профессиональные программисты. «Возьмёте в руки» отладчик и с его помощью заберётесь в самую глубь 1С:Предприятия. Чтобы увидеть, как всё происходит на самом деле. Это очень интересно!

3.9.5 Точки останова и просмотр значений

Никаких особых действий, чтобы воспользоваться отладчиком, производить не нужно. Почти всё вам уже известно. Единственное, чего вы пока не знаете, это как сказать программе, чтобы она остановилась в нужном месте.

Для этого используются *точки останова*. Точка останова — это специальная отметка (рисунок 3.19). Она показывает, перед выполнением какой инструкции платформа должна остановиться и ждать дальнейших команд от вас.

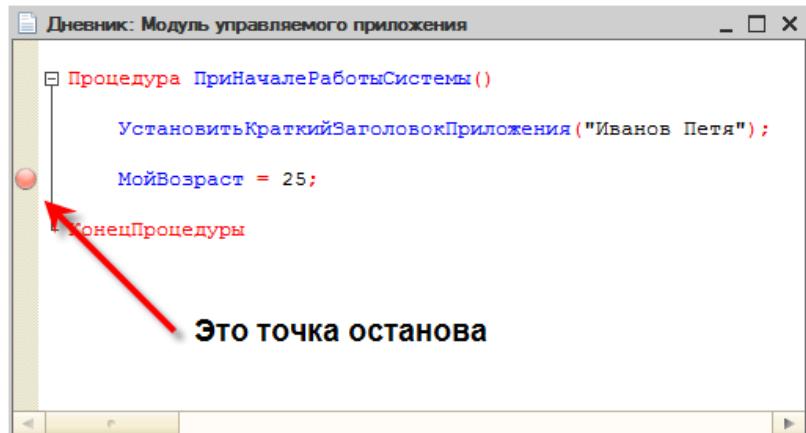


Рисунок 3.19. Точка останова

Чтобы установить точку останова, нужно дважды щёлкнуть мышью на серой полосе слева. В той строке, перед выполнением которой нужно остановиться.

Если на точке останова снова дважды щёлкнуть мышью, то точка останова пропадёт.

Но прежде чем установить точку останова, нужно сохранить те изменения, которые вы сделали в модуле. Для этого в командной панели наверху нажмите кнопку *Обновить конфигурацию базы данных*. Вы уже делали это однажды. Но если вы забыли, где находится эта команда, посмотрите на рисунок 3.20.

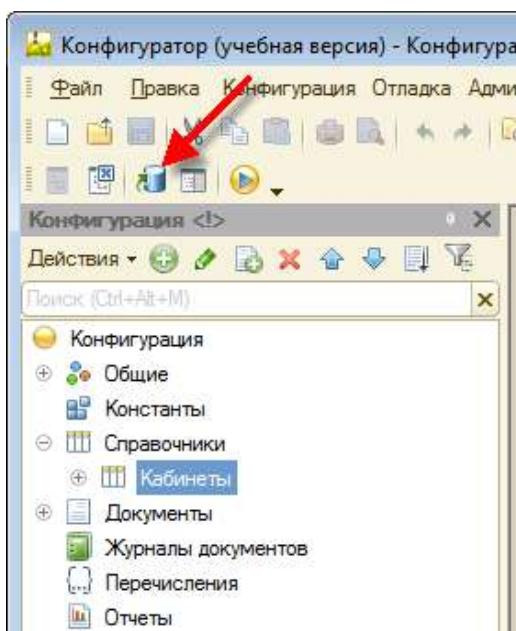


Рисунок 3.20. Обновить конфигурацию базы данных

После этого установите точку останова так, как показано на рисунке 3.19.

Теперь всё, что вам нужно, это запустить 1С:Предприятие в режиме отладки. Вы прекрасно знаете, как это делать.

В результате вы не увидите привычного интерфейса вашей программы. Как только платформа дойдёт до выполнения инструкции, которую вы отметили, она остановится и переключится в конфигуратор. Вы увидите такую картинку (рисунок 3.21).

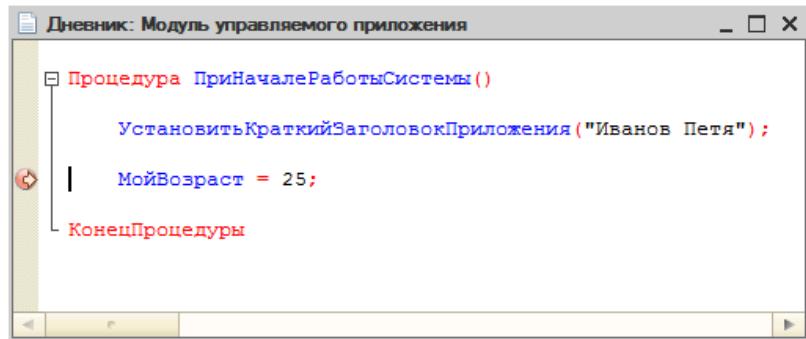


Рисунок 3.21. Работа программы остановлена на перед строкой «МойВозраст = 25;»

Жёлтая стрелка на серой полосе слева показывает, какую инструкцию платформа подготовилась выполнять. То есть сейчас она приготовилась выполнять инструкцию присваивания, которую вы написали.

Как посмотреть значение переменной *МойВозраст*? Для этого есть несколько способов. Первый и самый простой — открыть окно *Локальные переменные*. В этом окне отладчик автоматически покажет все переменные (и их значения), которые доступны в данный момент.

Чтобы открыть это окно, в командной панели наверху нажмите кнопку *Локальные переменные* (рисунок 3.22).

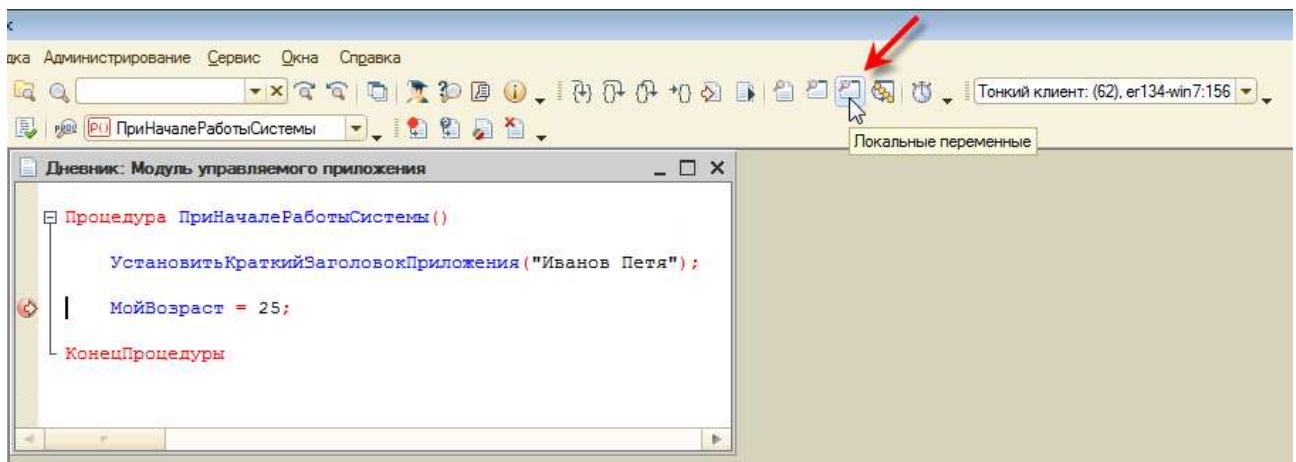


Рисунок 3.22. Кнопка «Локальные переменные»

Окно *Локальные переменные* появится в нижней части экрана (рисунок 3.23). В нём будет единственная переменная *МойВозраст*.

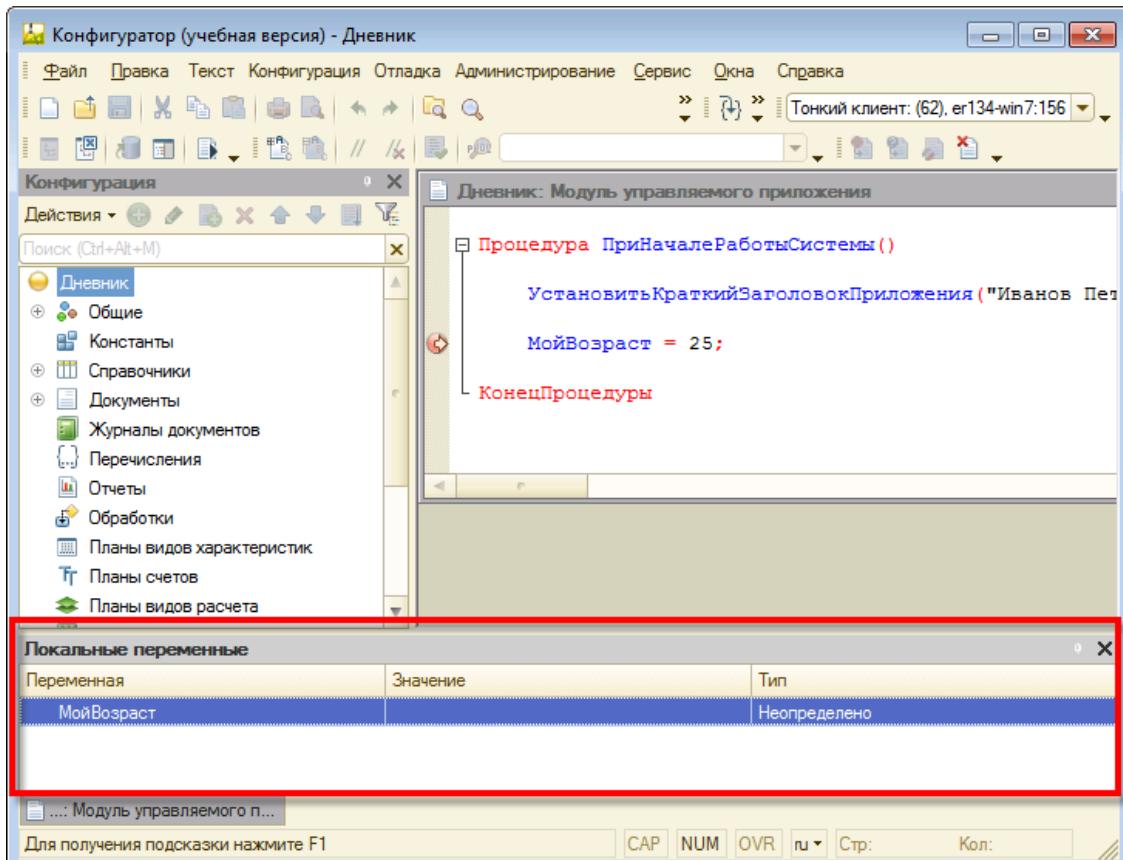


Рисунок 3.23. Окно «Локальные переменные»

Это окно устроено очень просто. Каждая переменная в нём занимает одну строку. В первой колонке показано имя переменной.

Во второй колонке — её значение. Пока у переменной *МойВозраст* нет никакого значения, поэтому во второй колонке пусто.

В третьей колонке — тип этой переменной. Вы пока ничего не помещали в переменную *МойВозраст*, поэтому платформа не знает, какой тип у этой переменной. В таких случаях она пишет *Неопределено*.

Теперь нужно сказать платформе, чтобы она выполнила вашу инструкцию. Ту, перед которой она остановилась.

Для выполнения программы по одному шагу (или по нескольким шагам сразу) есть ряд кнопок. Они расположены в командной панели наверху (рисунок 3.24).

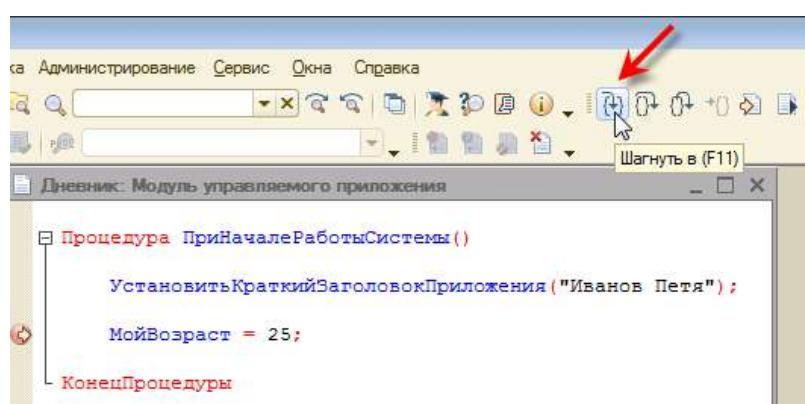


Рисунок 3.24. Кнопка «Шагнуть в»

Первая из них — *Шагнуть в* — именно та, что вам нужна. Она используется чаще всего. Если подвести к ней указатель мыши и подержать, то появится подсказка. В подсказке написано, какой клавишей можно сделать то же самое действие, — F11.

Нажмите F11 на клавиатуре. Это самый простой и удобный способ, которым пользуются программисты. Платформа выполнит одну инструкцию и остановится перед выполнением следующей (рисунок 3.25).

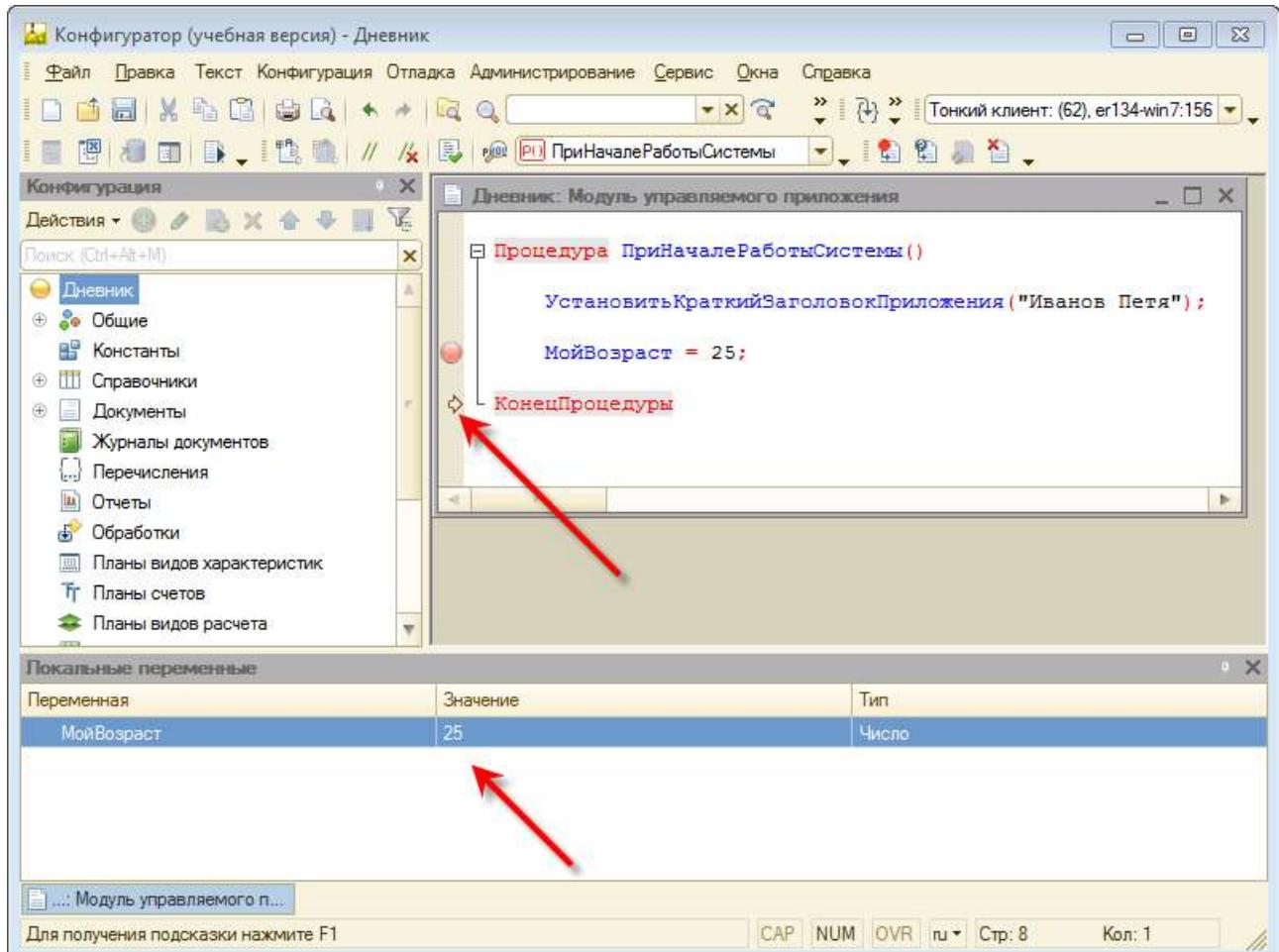


Рисунок 3.25. Значение переменной в окне «Локальные переменные»

Строка, перед выполнением которой платформа остановилась, опять отмечена жёлтой стрелкой. То есть ваша инструкция присвоения выполнилась.

Это действительно так, потому что в окне *Локальные переменные* у вашей переменной появилось значение 25 и тип *Число*.

Есть и другой способ посмотреть значение переменной. Вы можете просто подвести курсор мыши к тому месту программы, где написано имя переменной. И подержать немножко. Появится подсказка, в которой будет написано значение переменной (рисунок 3.26).

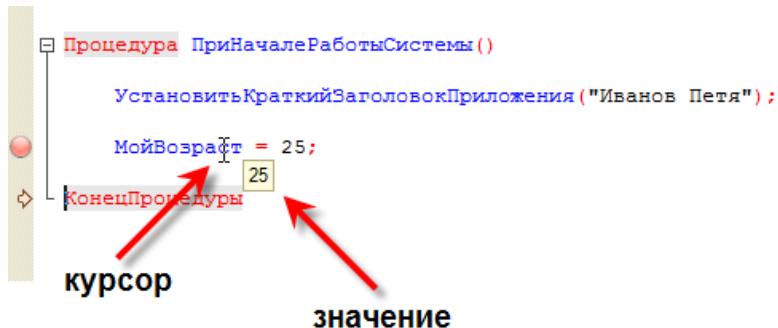


Рисунок 3.26. Значение переменной в тексте программы

После того как вы посмотрели всё, что вам нужно в конфигураторе, вы можете сказать платформе, чтобы она завершила отладку. Для этого выполните команду *Отладка — Завершить* (рисунок 3.27).

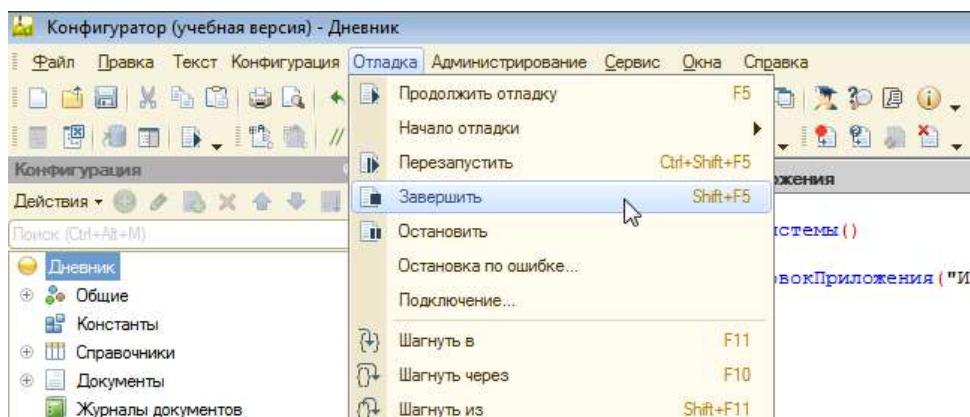


Рисунок 3.27. Завершить отладку

Выполняя примеры, которые расположены дальше, придерживайтесь этой последовательности действий:

- Внесите изменения в модуль (напишите новую команду или измените старую).
- Если вы хотите поставить точку останова на той строке, которую вы только что ввели:
 - то обновите конфигурацию базы данных;
 - установите точку останова.
- Если точка останова уже стоит в той строке, которая вам нужна, ничего делать не надо.
- Запустите 1С:Предприятие в режиме отладки.
- После того как вы посмотрели все интересующие вас значения, завершите отладку.

Совет

Когда вы в очередной раз решите посмотреть значение какой-нибудь переменной, не забывайте, что запускать 1С:Предприятие нужно обязательно в режиме отладки. Если вы запустите его как обычный пользователь, без отладки, платформа не будет обращать внимания на точки останова.

Заметка

Даже если вы запустили 1С:Предприятие в режиме отладки, платформа не всегда будет останавливаться на точках останова. Есть особенные режимы работы программы (они в этой книге не используются), когда требуется дополнительная настройка. Подробнее вы можете прочитать о настройке разных режимов отладки в документации «[Руководство разработчика 8.3. Раздел 28.2.2.1.2. Настройка приложения для работы в отладочном режиме](#)».

Подробнее

Подробнее вы можете прочитать про точки останова в документации «[Руководство разработчика 8.3. Раздел 28.2.3. Точка останова](#)».

Про пошаговое выполнение: «[Руководство разработчика 8.3. Раздел 28.2.4. Пошаговое выполнение](#)».

Подробнее вы можете прочитать про сочетания клавиш для работы с отладчиком во встроенной справке (командная панель сверху): *Справка — Содержание справки — Сочетания клавиш (Конфигуратор) — Отладчик*.

3.9.6 Изменение значений переменных

Теперь вы умеете создавать переменные и помещать в них значения. Познакомьтесь с тем, как можно изменять значения переменных.

Редко бывает так, что программист создал переменную, поместил в ней какое-то значение, а дальше только пользуется этим значением. Чаще бывает по-другому.

Программист создал переменную, поместил в неё значение, использовал тут, там. Потом поместил в неё другое значение. И тоже использовал эту переменную где-то. Потом опять изменил значение переменной. И так далее.

Изменить значение переменной очень просто. Нужно снова написать инструкцию присваивания.

Например, в переменной *МойВозраст* находится значение 25. Чтобы изменить значение этой переменной, напишите ещё одну инструкцию (рисунок 3.28) и поставьте на ней точку останова.

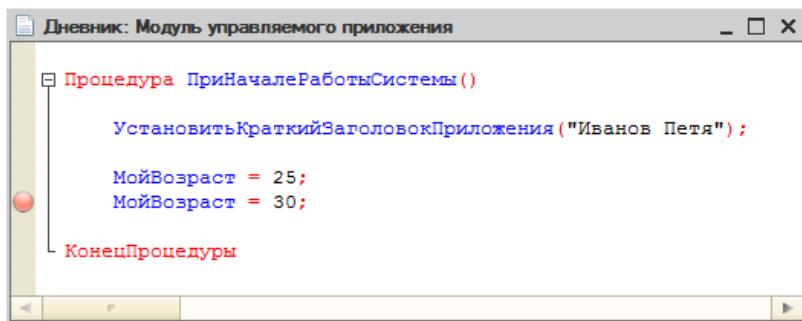


Рисунок 3.28. Вторая инструкция присваивания

Запустите 1С:Предприятие в режиме отладки и посмотрите, как будет изменяться значение переменной *МойВозраст*.

Сначала оно будет равно 25. Когда вы сделаете один шаг (F11), вторая инструкция выполнится, и значение переменной станет 30 (рисунок 3.29).

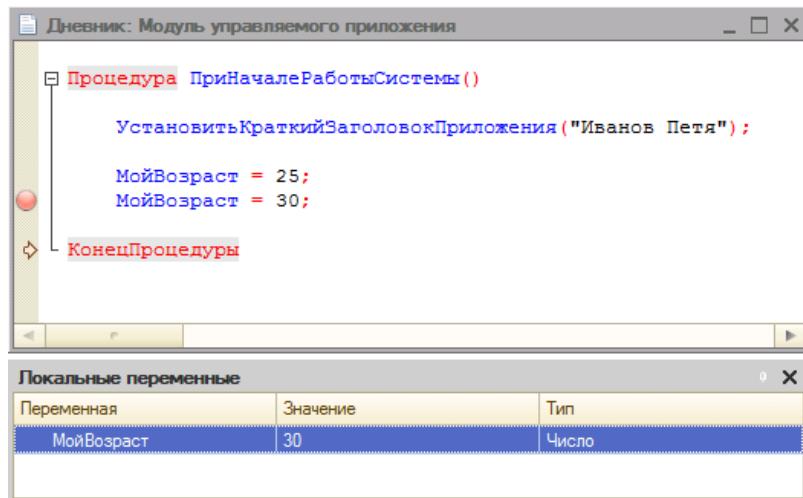


Рисунок 3.29. Результат выполнения второй инструкции присваивания

На этой картинке хорошо видно, что одинаковые на первый взгляд инструкции имеют совершенно разный смысл.

Первая инструкция присваивания создаёт переменную *МойВозраст* и помещает туда значение 25.

Вторая инструкция присваивания ничего не создаёт. В ту переменную, которая уже существует, она помещает значение 30.

3.9.7 Контекстная подсказка

Сейчас подходящее время для того, чтобы познакомиться с ещё одним важным инструментом разработчика — *контекстной подсказкой*. Она значительно упрощает жизнь программиста и позволяет избежать большого количества нелепых ошибок. Особенно на первом этапе, когда многие слова во встроенном языке являются для вас новыми и непривычными.

Когда вы писали вторую инструкцию присваивания, очень важно было не сделать ошибку в имени переменной *МойВозраст*. Если бы вы написали «МойВозратс» (случайно переставили местами две последние буквы), платформа подумала бы, что вы хотите создать новую переменную. И вместо того, чтобы поместить значение 30 в переменную *МойВозраст*, создала бы новую переменную, и значение 30 поместила бы в неё.

Попробуйте написать ещё одну инструкцию присваивания, но уже используя контекстную подсказку. В переменную *МойВозраст* нужно поместить значение 12.

В новой строке напишите начало имени вашей переменной «Мо» и вызовите контекстную подсказку. Для этого нужно нажать сочетание клавиш **Ctrl+Пробел** (рисунок 3.30).

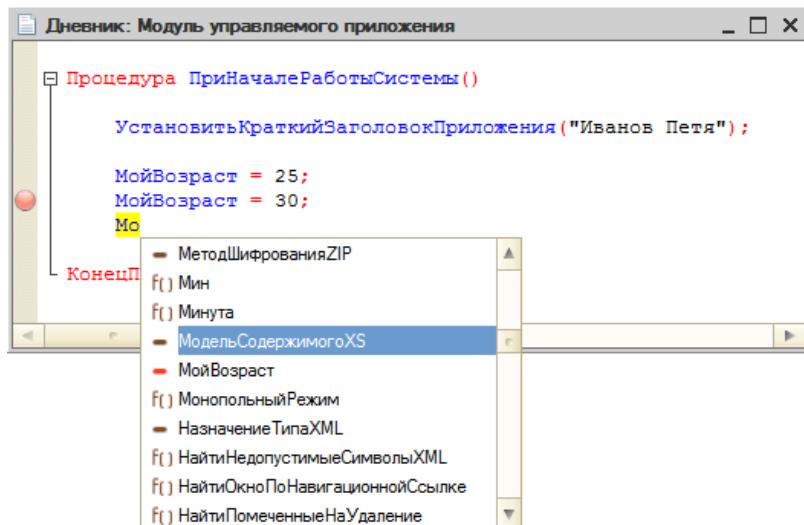


Рисунок 3.30. Контекстная подсказка

Список контекстной подсказки будет содержать всё, что платформа «знает». Всё, что вы можете использовать в этом месте без каких-либо дополнительных «объяснений». Причём этот список отсортирован по алфавиту, и в нём выделена первая строка, которая соответствует тем символам, которые вы начали набирать, — «Мо».

Строчкой ниже находится ваша переменная *МойВозраст*. Платформа тоже «знает» её, потому что эту переменную вы создали парой строк выше.

Вы можете нажать мышью на строку *МойВозраст*, и она подставится в текст программы. Или вы можете перейти к ней с помощью курсора и затем нажать **Enter** на клавиатуре. Попробуйте.

Также вы можете не искать «глазами» нужную строчку в этом списке. А уже после того, как окно контекстной подсказки открыто, вы можете продолжать набирать имя переменной на клавиатуре. Набрать символ «й». И тогда будет выделена строка *МойВозраст* (рисунок 3.31), потому что других подходящих строк больше нет.

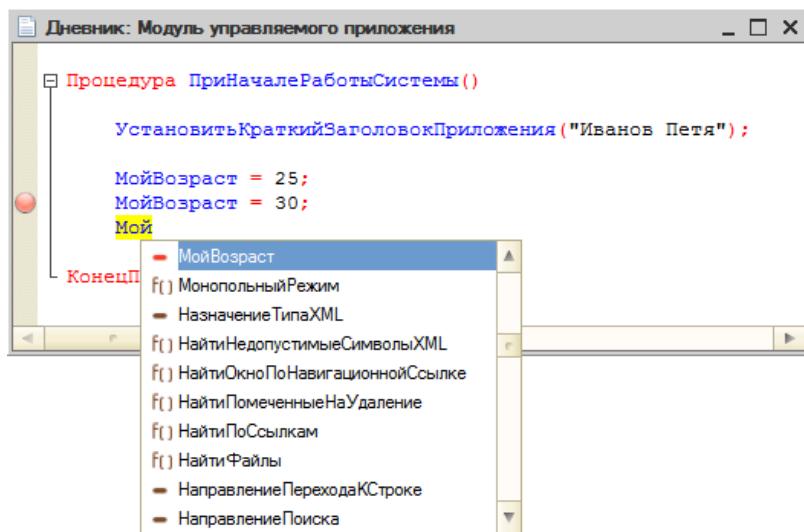


Рисунок 3.31. Контекстная подсказка подбирает нужную переменную

Вам останется только нажать клавишу **Enter**, и *МойВозраст* подставится в текст программы. Попробуйте.

Есть ещё более интересный способ. Например, вы точно знаете, а со временем вы будете это знать, что с букв «Мой» начинается только нужная вам переменная — и больше ничего. Тогда вы можете написать эти буквы (рисунок 3.32) и вызвать контекстную подсказку.

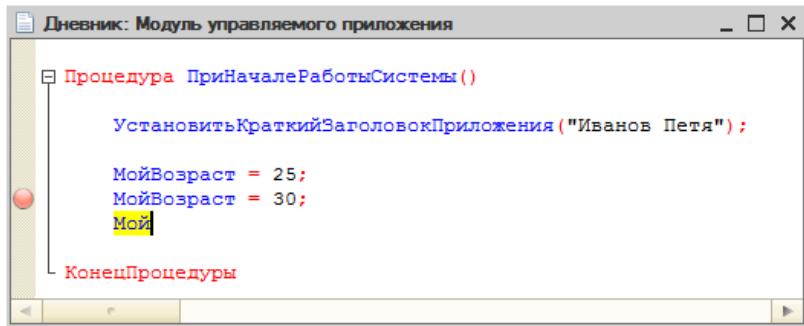


Рисунок 3.32. Момент, в который нужно вызвать контекстную подсказку

В этом случае у платформы выбора не будет. Поскольку в этом месте есть единственная известная ей переменная, которая начинается на «мой». Тогда платформа не будет предлагать вам ничего выбрать, а сразу же подставит имя переменной в текст программы (рисунок 3.33). Попробуйте.

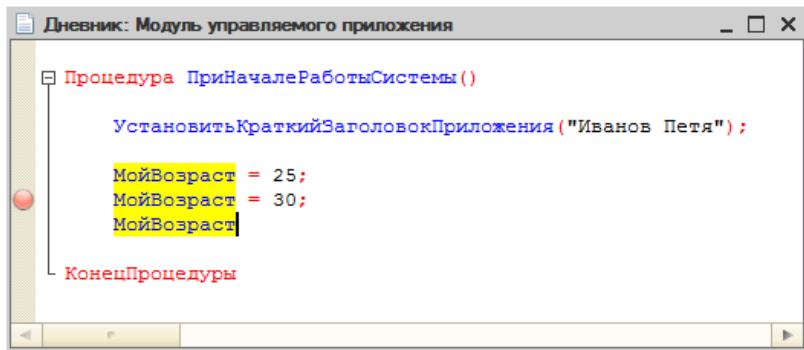


Рисунок 3.33. Контекстная подсказка сразу подставит имя переменной

Примечание

Использовать контекстную подсказку удобно, полезно и нужно. Во-первых, это поможет вам избежать многих орфографических ошибок. Во-вторых, значительно сэкономит ваше время, потому что большинство слов вам не нужно будет дописывать до конца. Платформа допишет за вас.

На рисунке 3.32 видно и ещё один инструмент, который поможет вам не допускать ошибок. Это выделение текущих идентификаторов. Эту возможность, если вы помните, вы включили в настройках конфигуратора в самом начале нашей книги.

Сейчас у вас текущие идентификаторы выделяются жёлтым цветом. Текущий идентификатор — это имя, которое находится в данный момент под курсором.

Например, вы не пользуетесь контекстной подсказкой. Или вы исправляете имя какого-то переменной. В этих случаях от ошибок вас подстрахует подсветка текущих идентификаторов.

Использовать её надо следующим образом. Как только вы закончите набирать имя переменной, платформа тут же выделит вашу переменную везде, где она встречается в тексте модуля. Таким образом, если вы написали имя правильно, вы увидите то, что на рисунке 3.33.

А если вы ошиблись в имени переменной, вы увидите вот такую картинку (рисунок 3.34).

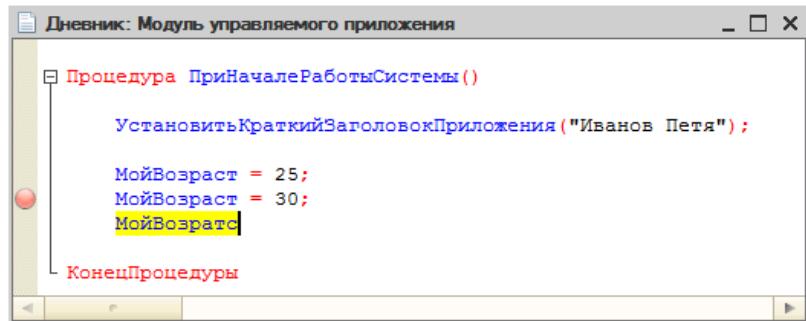


Рисунок 3.34. Ошибка в имени переменной

Примечание

Когда вы пишете имена переменных, обращайте внимание на то, что подсвечивается в модуле. Если вы хотите использовать одну из переменных, которые уже есть в модуле, и после написания её имени она не подсветилась в остальных местах, значит, вы допустили ошибку. Проверьте, что вы написали.

И теперь последний интересный эксперимент, который вы тут выполните. Он ещё раз продемонстрирует вам, что в первой инструкции вы создаёте переменную, а во второй инструкции присваиваете значение той же переменной, которая уже существует.

Чтобы было удобнее, отредактируйте текст так, как показано на рисунке 3.35. Удалите последнюю строку и добавьте пустую строку между инструкциями.

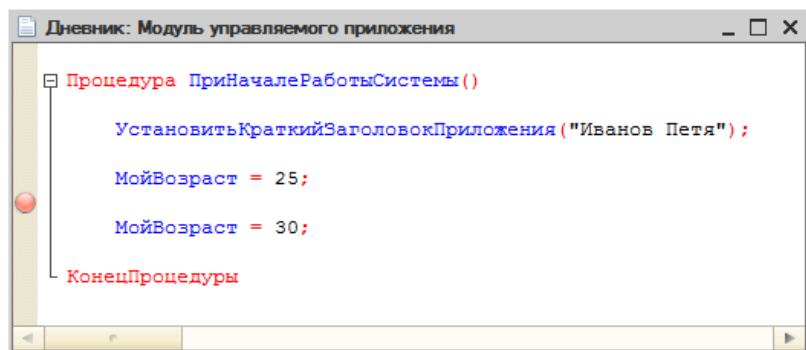


Рисунок 3.35. Как отредактировать модуль

Теперь перед первой инструкцией присваивания напишите «мо» и вызовите контекстную подсказку (рисунок 3.36).

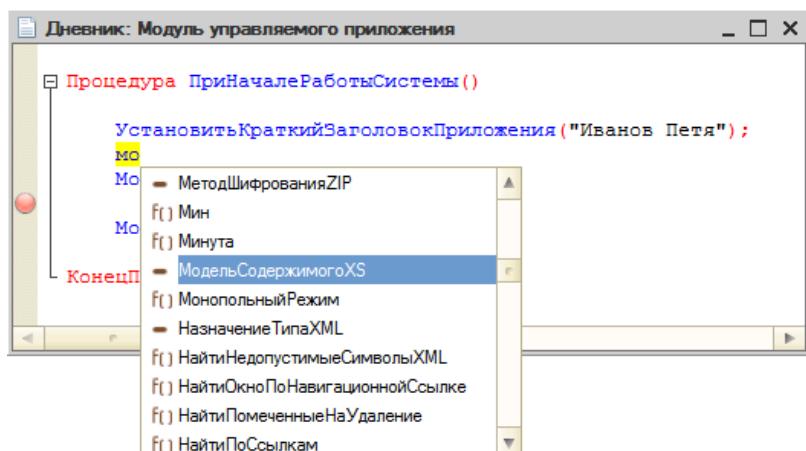


Рисунок 3.36. Контекстная подсказка до описания переменной

Посмотрите, в списке контекстной подсказки нет вашей переменной *МойВозраст*. Поэтому что в этом месте она ещё не описана. Только в следующей строке платформа узнает, что вы хотите использовать такую переменную.

А теперь напишите «мо» между инструкциями присваивания и вызовите контекстную подсказку (рисунок 3.37).

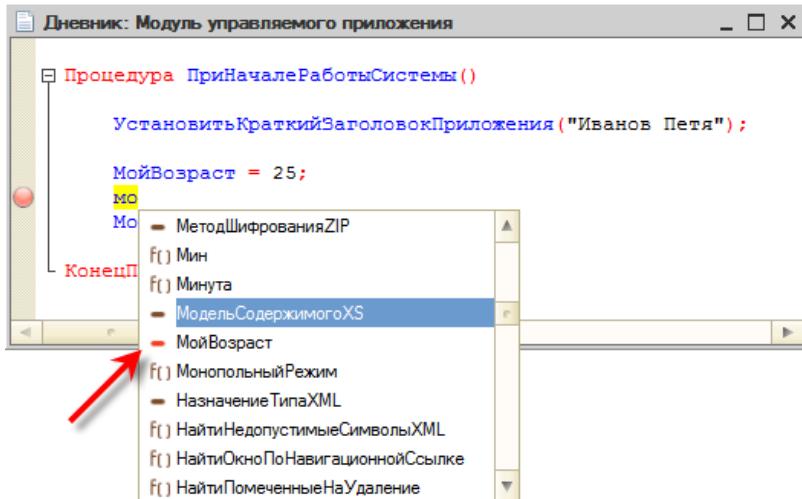


Рисунок 3.37. Контекстная подсказка после определения переменной

А в этом месте платформа уже знает, что есть такая переменная *МойВозраст*. Поэтому что она описана строчкой выше. И значит, инструкции присваивания, которые идут дальше, могут изменять её значение.

Примечание

Подробнее вы можете прочитать про сочетания клавиш для работы с текстом программ во встроенной справке (командная панель сверху): *Справка — Содержание справки — Сочетания клавиши (Конфигуратор) — Редактор текстовых документов и модулей*.

3.9.8 Выбор имени для переменной

Важно, чтобы имя переменной, которую вы хотите создать, не совпало с тем, что платформа уже знает в этом месте модуля. А как узнать, что знает платформа? Правильно! Вызвать контекстную подсказку.

Попробуйте сделать такой пример (листинг 3.17).

Листинг 3.17. Неправильный выбор имени переменной

Процедура ПриНачалеРаботыСистемы()

УстановитьКраткийЗаголовокПриложения("Иванов Петя");

РазмерКартинки = 30;

КонецПроцедуры

Запустите 1С:Предприятие в режиме отладки. Вы увидите сообщение об ошибке (рисунок 3.38).

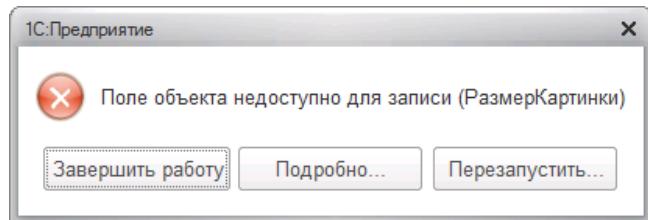


Рисунок 3.38. Сообщение об ошибке

Так получилось потому, что имя *РазмерКартинки* платформа уже использует в этом месте. Своей инструкцией присваивания вы попытались изменить его значение. Но сделать это невозможно. О чём платформа вам и сказала.

Чтобы обезопасить себя от подобных ошибок, поступайте следующим образом. Когда вы пишете инструкцию присваивания, которая создаёт новую переменную, пользуйтесь контекстной подсказкой (рис. 3.39).

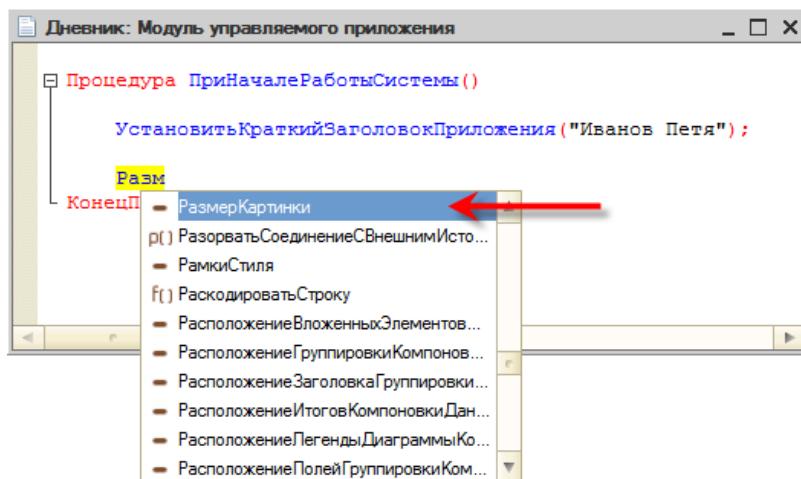


Рисунок 3.39. Контекстная подсказка при вводе имени переменной

Наберите первые буквы вашей переменной, откройте контекстную подсказку и продолжайте вводить имя переменной. Если оно случайно совпадёт с чем-то, что платформе уже известно, вы тут же об этом узнаете.

Совет

Если в процессе выполнения самостоятельных примеров у вас возникнет ошибка, посмотрите в разделе А.2 «Как прочитать сообщение об ошибке» на странице 544, как прочитать сообщение об ошибке и как исправить ошибку.

Задание 3.1

Все задания выполняйте в модуле управляемого приложения, в процедуре *ПриНачалеРаботыСистемы()*. После того как закончите, удалите их из модуля, чтобы они не мешали вам в дальнейшем.

Создайте переменную *МойРост*. Запишите в неё свой рост. Затем измените значение этой переменной на тот рост, который будет у вас, например, через год.

Запустите 1C:Предприятие в режиме отладки и посмотрите, как будет изменяться значение этой переменной.

Задание 3.2

Создайте переменную, в которой будет храниться ваше имя. Посмотрите значение этой переменной в режиме отладки.

Задание 3.3

Создайте переменную, в которой будет храниться домашний адрес. Ваш домашний адрес или адрес другого человека — это не важно. Запишите в неё свой адрес.

Задание 3.4

Выберите любой учебный день. Создайте две переменные. В одной будет храниться название первого урока в этот день, в другой — название второго урока.

Задание 3.5

Создайте переменную, в которой будет храниться телефонный номер. Ваш номер или любого другого человека — это не важно. Запишите в неё свой телефонный номер.

Задание 3.6

Создайте переменную, в которой будет храниться оценка, полученная на уроке. На одном уроке вы получили пятёрку, на другом уроке — четыре с плюсом. Запишите это в программе. Посмотрите в режиме отладки, как будет изменяться значение этой переменной.

3.9.9 Выражение

Теперь вы знаете, как создавать переменные, как придумывать им имена, как присваивать значения переменным. Ещё раз посмотрите, как схематически выглядит инструкция присваивания, которой вы сейчас занимаетесь:

Листинг 3.18. Инструкция присваивания

<Имя переменной> = <Выражение>;

Теперь наша задача — понять, что это за «выражение», которое находится справа от знака равенства. В этом вам поможет инструкция присваивания, которую вы уже видели (листинг 3.19):

Листинг 3.19. Пример инструкции присваивания

КоличествоЗанятий = 6 * 5;

Выражение — это математическая, или логическая, или строковая формула, по которой вычисляется значение. В примере это $6 * 5$.

Выражение обычно состоит из одной или нескольких *операций*. В примере одна операция — это операция умножения, которая обозначается знаком звёздочки — «*».

Самое простое выражение может не содержать ни одной операции, а только значение. Такой пример вы уже писали (листинг 3.20).

Листинг 3.20. Пример самого простого выражения

```
МойВозраст = 25;
```

В этом примере значение — это 25, то есть литерал типа Число.

В выражениях можно использовать не только литералы, но и переменные. Например, вы можете взять пример в листинге 3.19 и изменить его так, чтобы умножались не два числа, а две переменные (рисунок 3.40).

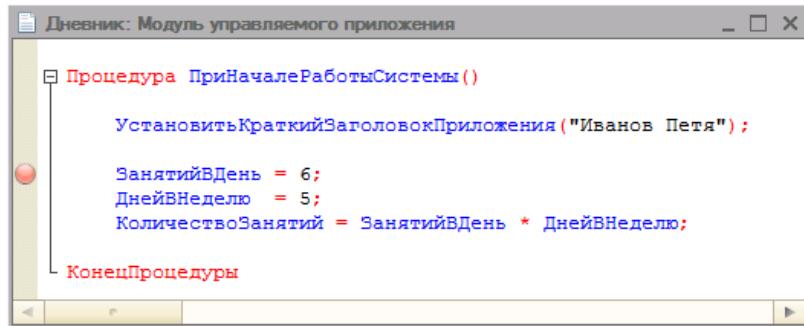


Рисунок 3.40. Выражение с двумя переменными

Попробуйте. При вводе имён переменных в третьей строке не забывайте пользоваться контекстной подсказкой.

Совет

Сразу возьмите себе за правило ставить пробелы перед знаками операций (+, -, *, /) и после них. Тогда текст программы будет читаться легко.

Установите точку останова в первой строке примера и пройдите его по шагам в режиме отладки. Посмотрите, как меняются значения переменных.

Теперь вспомните картинку с экскаватором (рисунок 3.41).

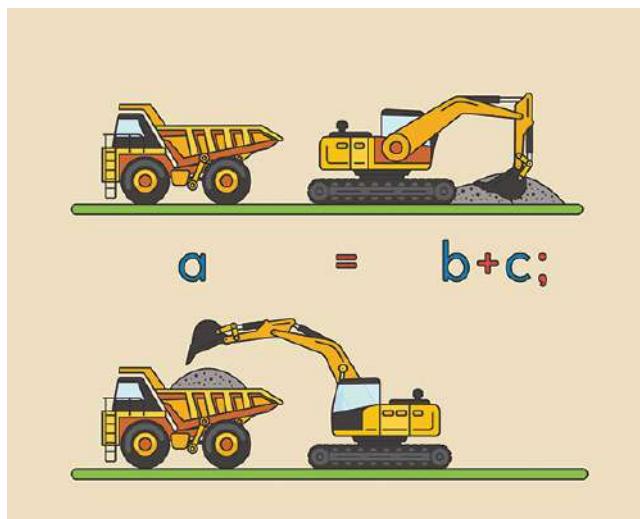


Рисунок 3.41. Знак равенства работает как экскаватор

Сначала он собирает своим ковшом то, что справа, и только после этого кладёт это в переменную, которая написана слева.

В конфигураторе есть специальная команда, с помощью которой вы можете посмотреть, что именно экскаватор собрал в свой ковш. Ещё до того, как он положит это куда-то. Познакомьтесь с этой командой.

В последнем примере поставьте точку останова на третьей строке и запустите отладку (рисунок 3.42).

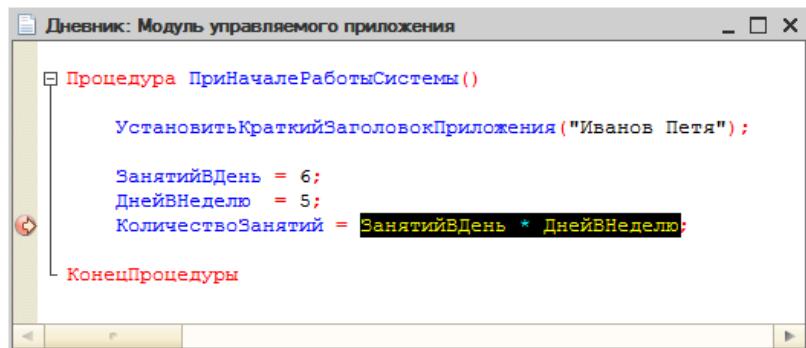


Рисунок 3.42. Выделите выражение

Обратите внимание, что значение переменной *КоличествоЗанятий* пока ещё не определено. Потому что эта инструкция ещё не выполнялась. Но значения переменных *ЗанятийВДень* и *ДнейВНеделю* уже известны.

Теперь с помощью мыши выделите всё выражение, которое находится справа от знака равенства. И нажмите сочетание клавиш Shift+F9. Откроется окно вычисления выражений (рисунок 3.43).

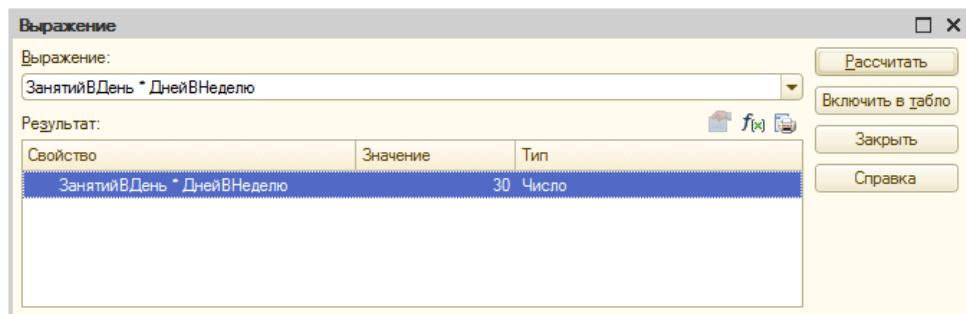


Рисунок 3.43. Окно вычисления выражений

В этом окне вы увидите и своё выражение, и значение, которое получится в результате его вычисления. В вашем примере выражение простое. Поэтому кажется, что такая функция не очень-то и нужна. Но в реальных программах выражения могут быть сложными. С помощью этой функции вы легко можете выделить всё выражение (или только его часть) и посмотреть, какое значение получится.

Платформа, так же как и вы, сначала вычисляет выражение, а потом помещает его в переменную слева. Поэтому переменную, которая находится слева от знака равенства, можно использовать в выражении в этой же инструкции присваивания.

Попробуйте сделать такой пример. Сначала в переменную запишите свой возраст. А теперь представьте, что прошёл один год и в следующей строке вам нужно записать в эту переменную свой возраст через год. Как вы это сделаете?

Ответ находится на следующем рисунке (рисунок 3.44). Если у вас получилось то же самое — замечательно! Если нет — не расстраивайтесь, скопируйте пример к себе в конфигурацию.

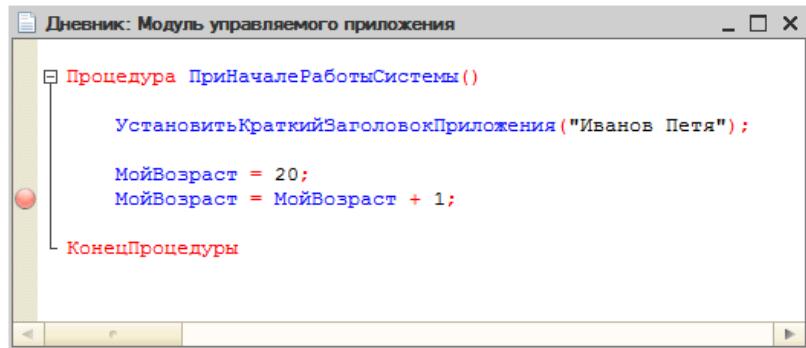


Рисунок 3.44. Вы стали на год взросле

На первый взгляд такая «конструкция» кажется странной. Но только не для компьютера.

Установите точку останова на второй строке примера и в режиме отладки посмотрите, как изменится значение переменной *МойВозраст*. Сначала в ней будет значение 20. А после выполнения второй строки значение изменится на 21.

Подробнее

Подробнее вы можете прочитать про сочетания клавиш для работы с отладчиком во встроенной справке (командная панель сверху): *Справка – Содержание справки – Сочетания клавиш (Конфигуратор) – Отладчик*.

Задание 3.7

В одной переменной сохраните вашу среднюю скорость — 5 км/ч. В другой переменной сохраните расстояние до школы — 6 км. В третьей переменной посчитайте количество минут, которое вам понадобится, чтобы дойти до школы.

Задание 3.8

Используйте среднюю скорость и расстояние до школы из задания 3.7 В третьей переменной посчитайте, сколько раз за сутки вы сможете дойти до школы и вернуться обратно, если не будете останавливаться и спать. После этого с помощью вычисления выражения посмотрите, сколько километров вы можете пройти за сутки.

Задание 3.9

В этом же примере (3.8) посчитайте, сколько раз вы сможете сделать то же самое, если будете ехать на велосипеде только в светлое время суток. Средняя скорость велосипедиста — 15 км/ч. Светлое время суток длится в среднем 13 часов.

Задание 3.10

В вашем портфеле были только учебники. В понедельник, чтобы перекусить в школе, вы взяли из дома 2 яблока. Но не стали их есть, и они остались в портфеле. Во вторник и в среду вы тоже брали яблоки из дома и оставляли их в портфеле. Сколько яблок будет в вашем портфеле в среду, если каждый день вы брали из дома на два яблока больше, чем в предыдущий? Для решения этой задачи используйте две переменные: *ВПортфеле* (количество яблок в портфеле) и *ВзялИзДома* (количество яблок, которое вы взяли из дома).

3.9.10 Арифметические операции

Как вы теперь знаете, в выражении можно использовать разные операции. Также вы знаете, я говорил об этом, что не со всеми значениями можно выполнять одни и те же операции.

Например, если ваши значения имеют тип `Число`, то с ними можно выполнять арифметические операции: сложение, вычитание, умножение и деление. Эти операции вам хорошо известны. Единственное, что может вызвать трудность, это каким знаком обозначать умножение и деление.

Но и это вы уже знаете. Умножение обозначается знаком звёздочки «*», а деление — знаком косая черта «/».

Подробнее

Есть и другие арифметические операции. Подробнее вы можете прочитать про них в документации [«Руководство разработчика 8.3. Раздел 4.5.1. Арифметические операции»](#).

Одновременно в выражении может участвовать любое количество арифметических операций. Но выполняются они не в той последовательности, как они написаны. Вам это известно из математики. В математике существует определённый порядок выполнения операций. А кроме этого используются скобки, чтобы указать именно тот порядок, который вам нужен.

Во встроенном языке то же самое:

- если в выражении есть скобки, то сначала вычисляется то, что в скобках;
- если скобок нет, то в первую очередь выполняются операции умножения и деления в том порядке, в котором они написаны;
- в самую последнюю очередь выполняются операции сложения и вычитания (в том порядке, в котором они написаны).

Задание 3.11

Создайте четыре переменные:

- `ДлинаУрока` — 45 минут;
- `ДлинаПеремены` — 15 минут;
- `ДлинаБольшойПеремены` — 25 минут;
- `ВсегоУроков` — 6.

Посчитайте, сколько минут вы проводите в школе в течение дня, если одна из перемен между уроками всегда большая.

Задание 3.12

Лестницы в многоэтажных домах обычно устроены следующим образом (рисунок 3.45).

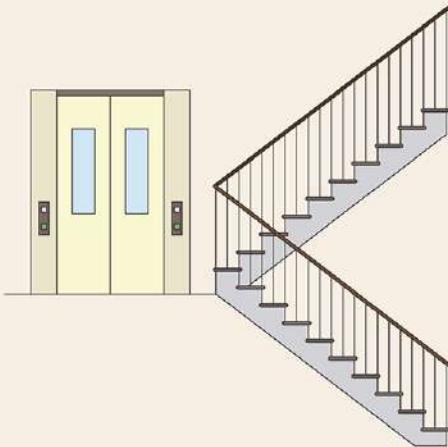


Рисунок 3.45. Лестница в многоэтажном доме

Чтобы подняться с этажа на этаж, нужно пройти два лестничных марша. Ещё несколько ступеней есть перед входом в подъезд.

Сколько ступеней нужно пройти, чтобы подняться на ваш этаж?

Предусмотрите, что у разных людей в разных домах следующие величины могут быть разными. Перед вычислением сохраните эти значения в отдельных переменных:

- количество ступеней перед входом в подъезд;
- количество ступеней в марше;
- этаж.

3.9.11 Операции со строками

Если ваши значения имеют тип *Строка*, то они тоже могут участвовать в выражении. Но операций, которые часто используются при работе со строками, всего лишь две. Одну из них вы сейчас и рассмотрите.

Эта операция заключается в том, чтобы к одной строке дописать другую строку. Называется она сложным словом *конкатенация*, а обозначается знаком «+».

Сделайте такой пример. В одной переменной сохраните название вашего города. В другой переменной — название улицы, на которой вы живёте. А в третью переменную с помощью выражения запишите ваш адрес, который будет состоять из названия города и улицы.

Чтобы проверить себя, можете посмотреть на рисунок 3.46.

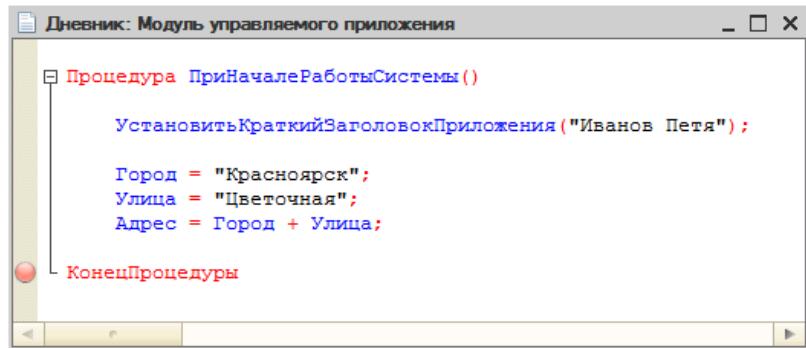


Рисунок 3.46. Пример конкатенации строк

Установите точку останова на строке *КонецПроцедуры* и в режиме отладки посмотрите, какое значение получилось в переменной *Адрес*.

Вы увидите, что две строки действительно «склеились» и получилась строка *КрасноярскЦветочная*.

С одной стороны, это хорошо, потому что конкатенация работает. С другой стороны, не очень хорошо, потому что результат выглядит «не очень».

Поэтому при конкатенации строк очень часто используют литералы типа *Строка*, чтобы отделить одно строковое значение от другого. И для того, чтобы это понятно читалось и хорошо выглядело.

В вашем примере хотелось бы вместо *КрасноярскЦветочная* видеть *Красноярск, Цветочная*. Чтобы добиться такого результата, измените последнюю строку так, как показано в листинге 3.21.

Листинг 3.21. Использование литерала для разделения строк

```
Адрес = Город + ", " + Улица;
```

Запустите этот пример в режиме отладки и посмотрите, какой получается результат.

Подробнее

Подробнее вы можете прочитать про операцию конкатенации в документации «Руководство разработчика 8.3. Раздел 4.5.2. Операция конкатенации».

Задание 3.13

Используя переменную *Возраст* типа *Строка*, запишите фразу «Мой возраст 21 год».

Задание 3.14

При отправке почтовых сообщений существует определённый порядок перечисления реквизитов адреса:

1. Название улицы, номер дома, номер квартиры.
2. Название населённого пункта (города, посёлка и т. п.).
3. Название района.
4. Название республики, края, области, автономного округа (области).

Используя переменные *Улица*, *НомерДома*, *НомерКвартиры*, *Город* и *Область*, запишите адрес, выделенный на рисунке 3.47.

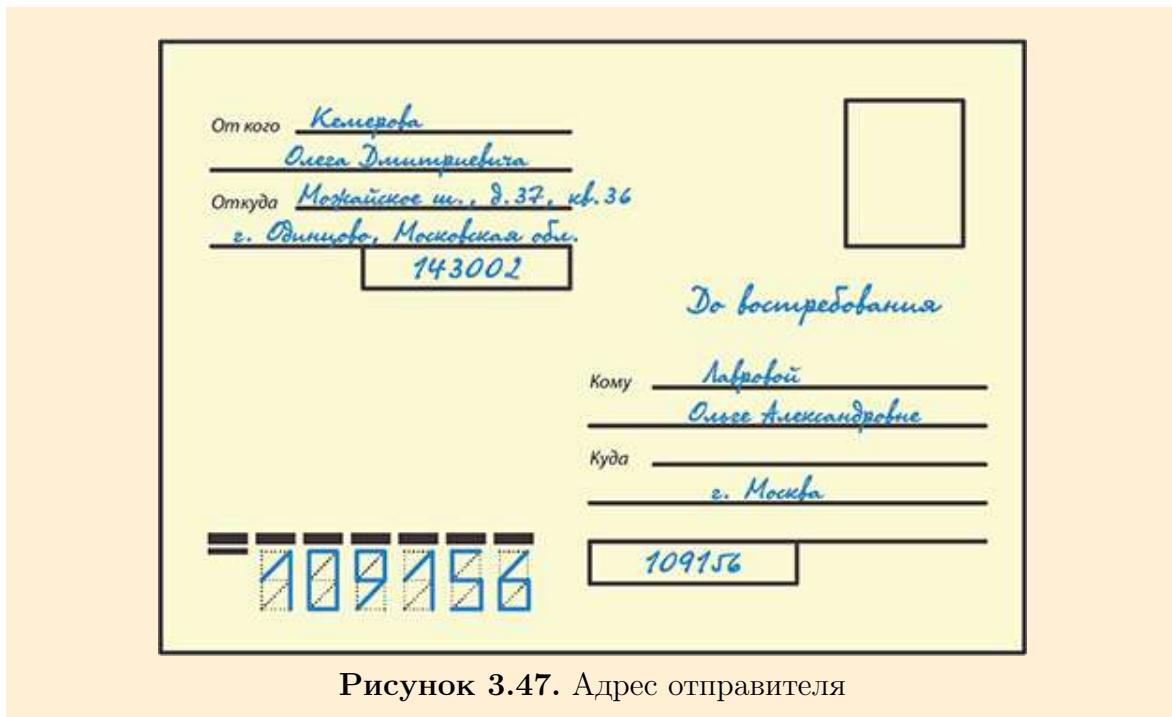


Рисунок 3.47. Адрес отправителя

3.9.12 Тип Дата и операции с датами

Дата — это первый необычный тип данных, с которым вам придётся столкнуться. В повседневной жизни вы используете даты очень часто и часто делаете с ними какие-то операции. Но в жизни вы привыкли воспринимать даты отдельными частями.

Например, когда вас спрашивают: «Сколько вам лет?», вы из одного года вычитаете другой год. То есть из одного числа вычитаете другое. Когда вас спрашивают, через сколько минут начнётся фильм, вы тоже из одного числа (количество минут) вычитаете другое число (количество минут).

В 1С:Предприятии не так. В 1С:Предприятии значение типа *Дата* — это всегда календарная дата вместе со временем с точностью до секунд. Например, «31.03.2016 12:25:59». Это значит «двенадцать часов двадцать пять минут пятьдесят девять секунд тридцать первого марта две тысячи шестнадцатого года».

Чтобы убедиться в этом, вы можете выполнить простой пример. Напишите в модуле инструкцию, которая показана на рисунке 3.48. Когда будете писать *ТекущаяДата()*, не забывайте пользоваться контекстной подсказкой.

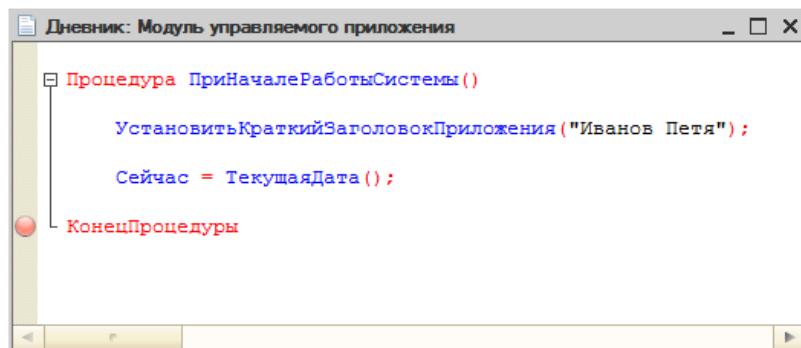


Рисунок 3.48. Текущая дата

Установите точку останова на строке *КонецПроцедуры*, запустите 1С:Предприятие в

режиме отладки и посмотрите значение переменной *Сейчас*. В ней будет текущее время, установленное на вашем компьютере.

Значения типа *Дата*, так же как числа и строки, можно записывать прямо в тексте программы. Это используется не очень часто, но используется. Поэтому посмотрите, как выглядит литерал типа *Дата* (рисунок 3.49).

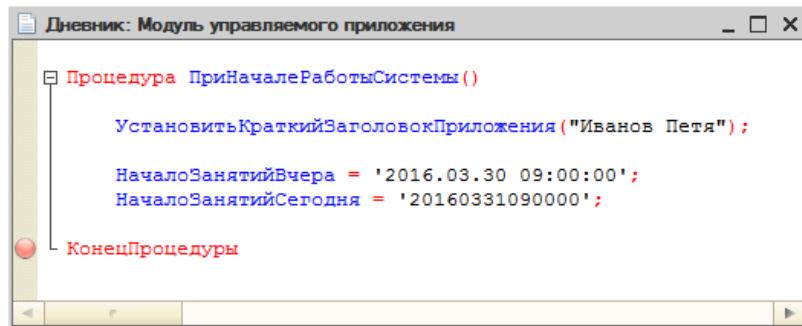


Рисунок 3.49. Литерал типа «Дата»

Он обязательно обрамляется символами одинарная кавычка — «'». А внутри разные части даты указываются в такой последовательности: год, месяц, день, час, минуты, секунды. При этом вы можете отделять их друг от друга какими-нибудь символами, чтобы легче читалось, или не отделять. И так, и так будет правильно. Лишь бы соблюдалась последовательность, в которой они указаны. Попробуйте.

Как я уже говорил, литералы типа *Дата* используют не часто. Потому что выглядят они довольно сложно, а читать их не очень удобно. Гораздо чаще для работы с датами используют *функции встроенного языка*. Просто запомните это название. Позже я объясню, что это такое.

Посмотреть, какие есть функции для работы с датами, очень просто. Для этого вам понадобится *синтакс-помощник*, который я уже упоминал раньше. Чтобы его открыть, нужно нажать на кнопку *Синтакс-помощник* в командной панели наверху (рисунок 3.50).

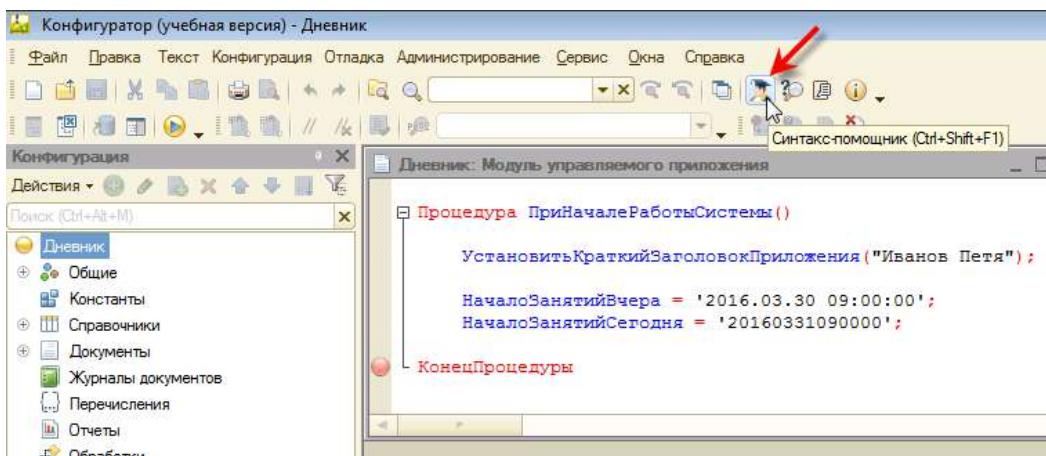


Рисунок 3.50. Открыть синтакс-помощник

Синтакс-помощник открывается в правой стороне экрана, на том же месте, что и палитра свойств. Чтобы переключаться между ними, используйте закладки, расположенные в нижней части экрана (рисунок 3.51).

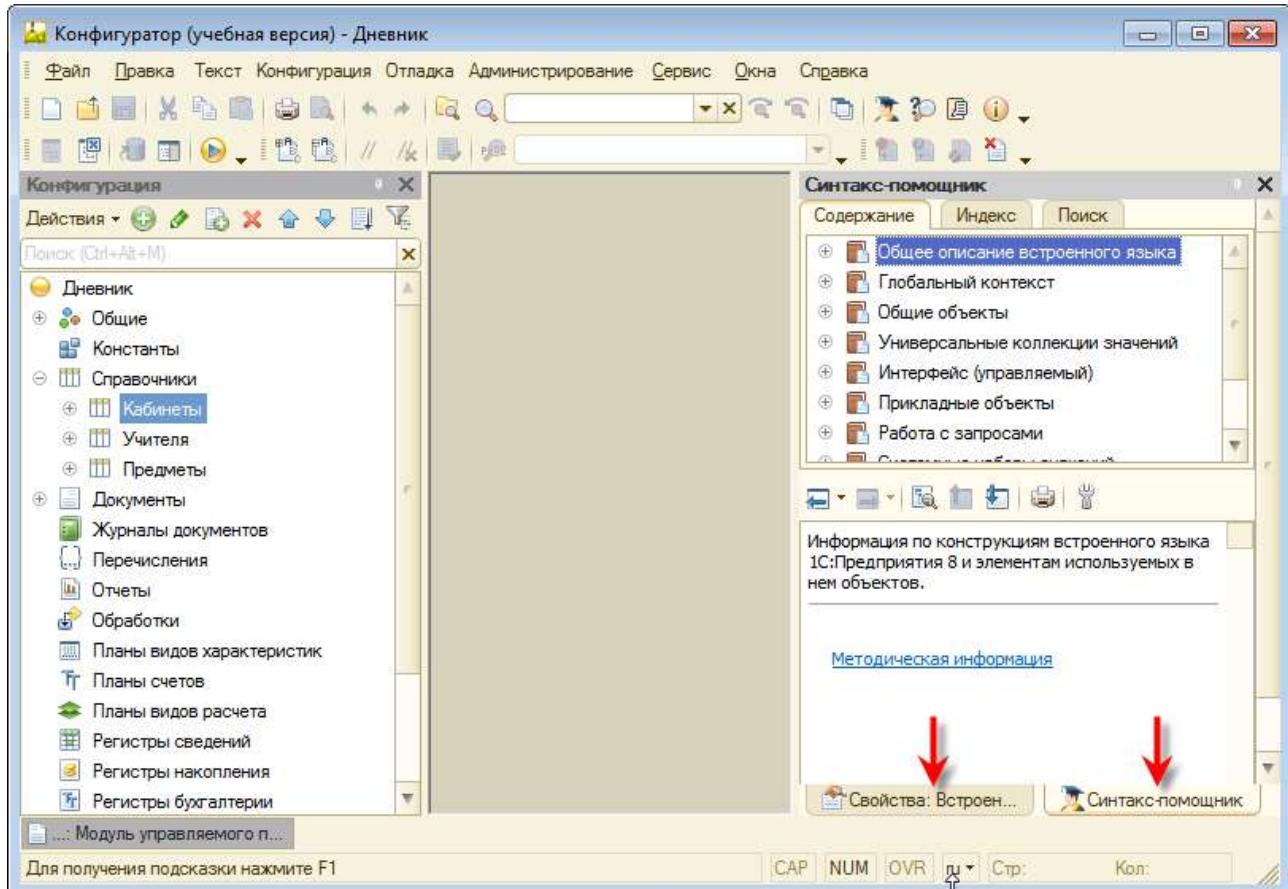


Рисунок 3.51. Переключение между синтакс-помощником и палитрой свойств

Синтакс-помощник — это большой справочник по встроенному языку. В верхнем окне находится его «оглавление». В нём вы можете выбрать интересующий вас раздел и дважды щёлкнуть на нём мышью. Тогда в нижнем окне откроется содержимое этого раздела (рисунок 3.52).

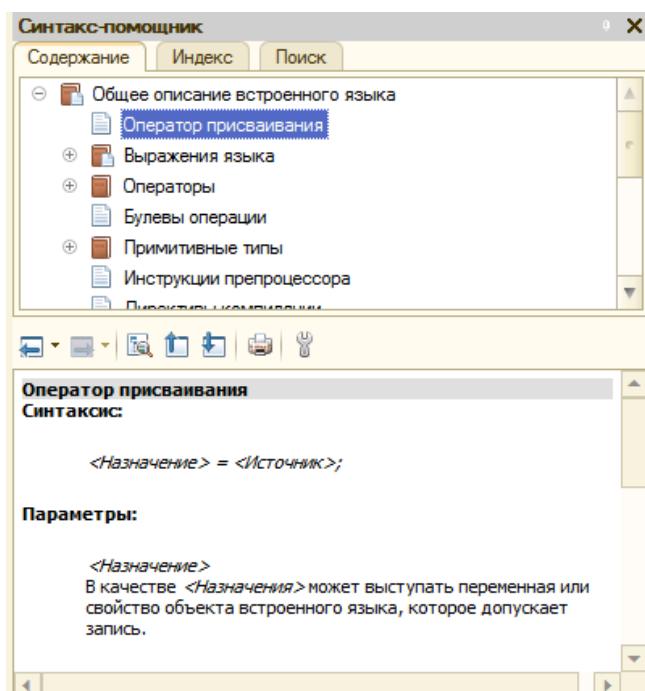


Рисунок 3.52. Содержимое раздела в синтакс-помощнике

Если вам не хватает места в верхнем или нижнем окне, вы можете потянуть мышью за левую границу синтакс-помощника, и он станет шире. Также вы можете изменять высоту верхнего и нижнего окна. Для этого нужно подвести курсор к границе верхнего окна, чтобы он изменил свой вид на стрелку. А затем потянуть мышью (рисунок 3.53).

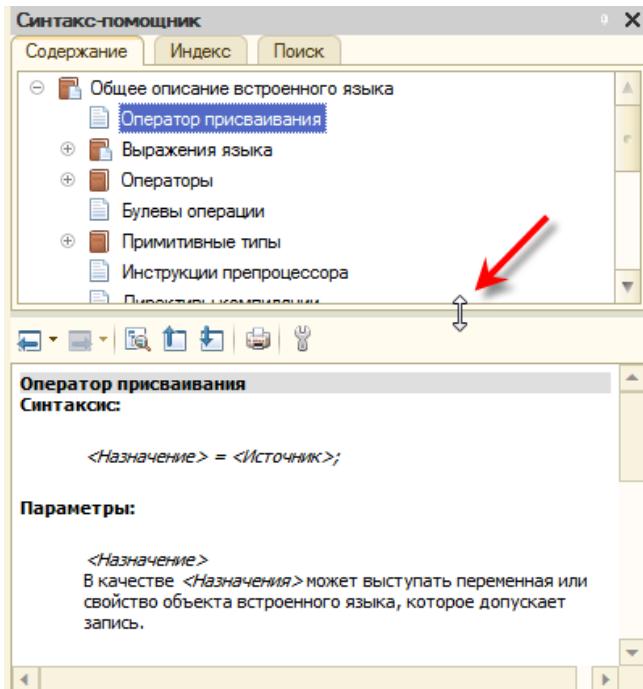


Рисунок 3.53. Изменение вертикального размера окон

Встроенные функции, предназначенные для работы с датами, находятся в разделе *Глобальный контекст — Функции работы со значениями типа Дата* (рисунок 3.54).

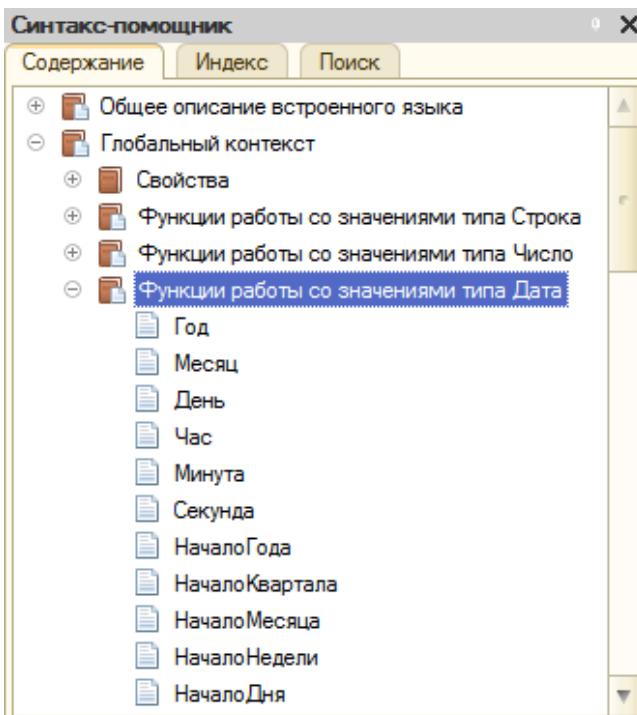


Рисунок 3.54. Функции для работы со значениями типа Дата

А кроме них в разделе *Функции преобразования значений* находится ещё одна полезная функция — *Дата()* (рисунок 3.55). С ней вы познакомитесь прямо сейчас.

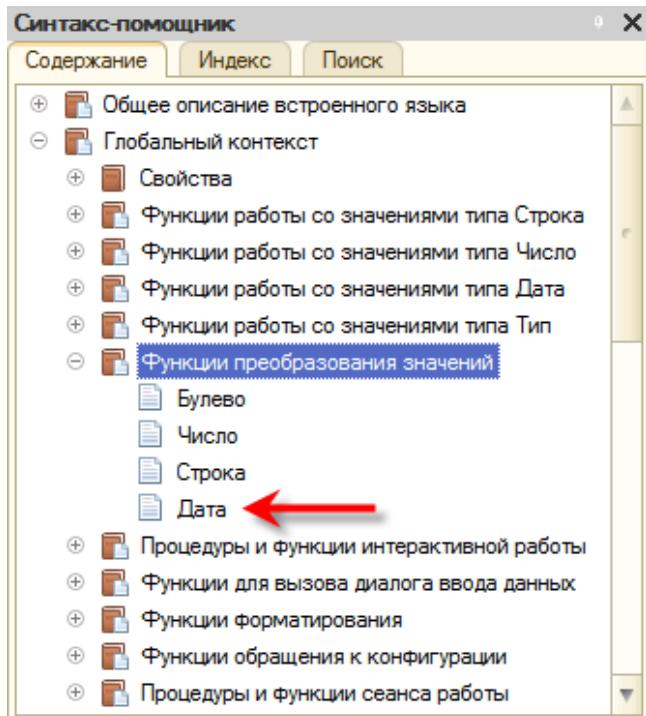


Рисунок 3.55. Функция «Дата()»

Она создаёт значение типа *Дата* из года, месяца, дня и т. д., которые вы перечислите в скобках через запятую. Попробуйте.

После первой строки примера вставьте пустую строку и напишите в ней *НачалоЗанятийВчера* = (рисунок 3.56). Не забывайте пользоваться контекстной подсказкой!

```

Дневник: Модуль управляемого приложения

Процедура ПриНачалеРаботыСистемы()
    УстановитьКраткийЗаголовокПриложения ("Иванов Петя");
    НачалоЗанятийВчера = '2016.03.30 09:00:00';
    НачалоЗанятийВчера =
    НачалоЗанятийСегодня = '20160331090000';
КонецПроцедуры

```

Рисунок 3.56. Начало строки

А теперь вы поступите очень интересно. Синтакс-помощник не только рассказывает, из чего состоит встроенный язык, но и помогает писать программы. Поэтому вы не будете писать слово «Дата», а просто перетащите его мышью из окна синтакс-помощника в вашу программу (рисунок 3.57).

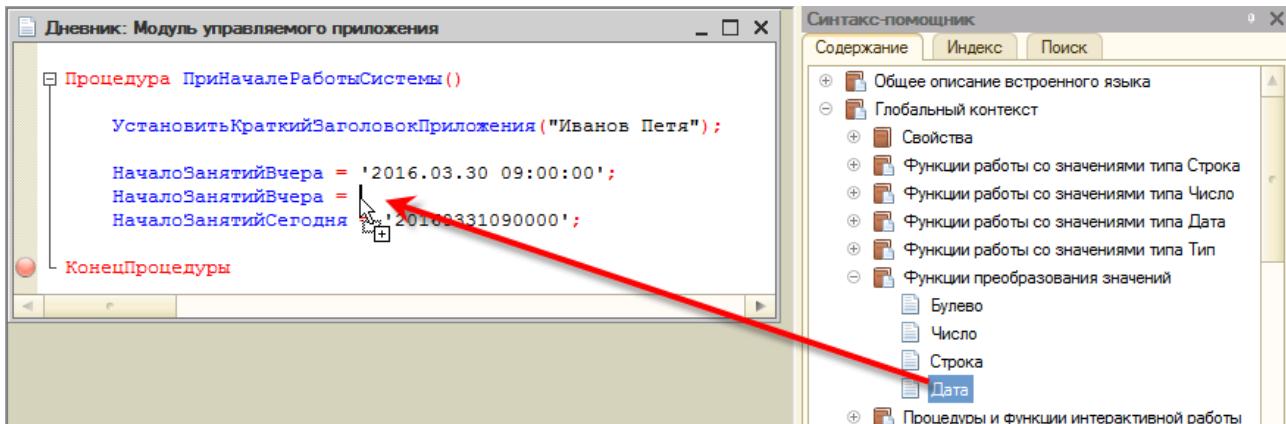


Рисунок 3.57. Перетащите функцию в текст программы

Платформа вставит в программу «заготовку» этой функции. Вам останется только дописать её содержимое (рисунок 3.58).

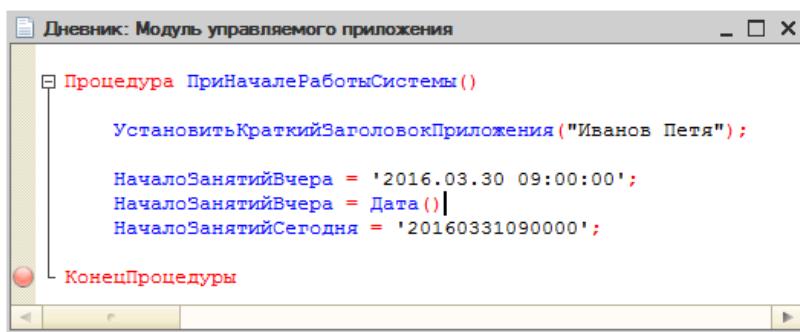


Рисунок 3.58. «Заготовка» функции

Поставьте курсор внутрь скобок, наберите 2016 и поставьте запятую.

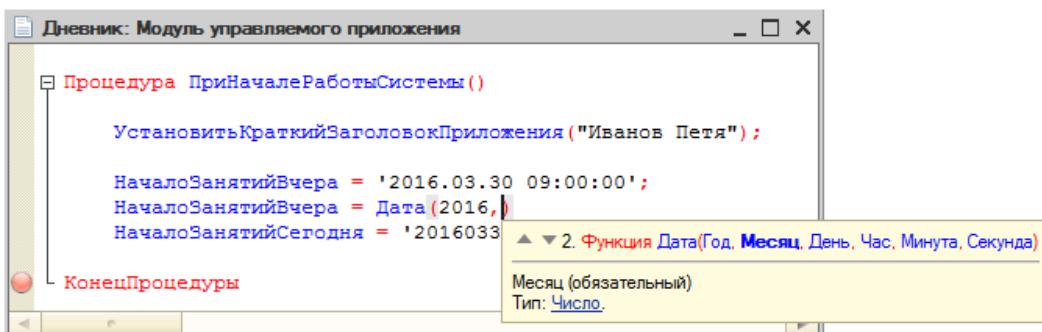


Рисунок 3.59. Контекстная подсказка функции

Появится контекстная подсказка. Она подскажет вам, что нужно написать в скобках. Год вы уже написали. Теперь нужно написать месяц. Поэтому он и выделен жирным шрифтом.

А тип у этого «месяца» — Число. Значит, вы можете написать не 03, как было выше, а просто 3. Как только вы поставите запятую после тройки, подсказка изменится. Теперь она будет говорить вам, что нужно ввести день (рисунок 3.60).

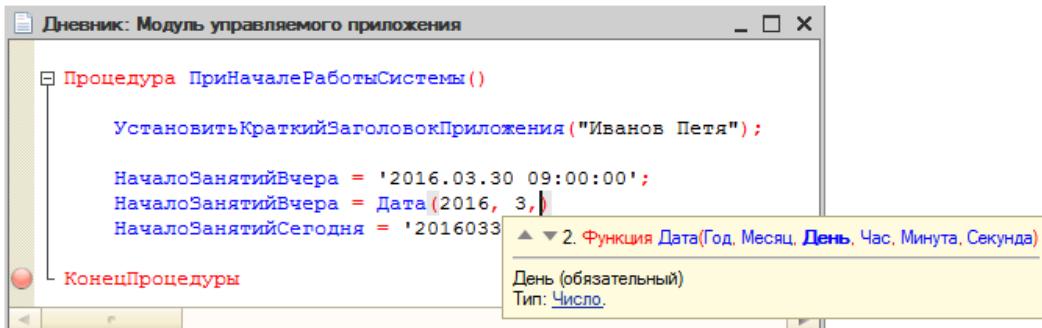


Рисунок 3.60. Подсказка следующего параметра

Совет

Сразу возьмите себе за правило ставить пробелы после запятой. Тогда текст программы будет читаться легко.

Вот так, пользуясь контекстной подсказкой, допишите инструкцию до конца. Не забудьте закончить её точкой с запятой.

После этого запустите 1С:Предприятие в режиме отладки и убедитесь, что в переменной *НачалоЗанятийВчера* то же значение, которое было и в предыдущем примере. Однако с функцией *Дата()* текст выглядит и читается лучше (рисунок 3.61).

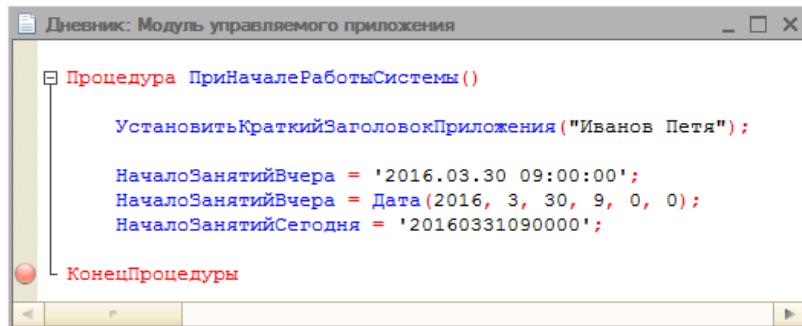


Рисунок 3.61. Использование функции «Дата()»

Совет

Если окно контекстной подсказки параметров закрылось, вы в любой момент можете снова открыть его. Для этого используйте комбинацию клавиш **Ctrl+Shift+Пробел**.

Подробнее

Подробнее вы можете прочитать про сочетания клавиш для работы с текстом программ во встроенной справке (командная панель сверху): *Справка — Содержание справки — Сочетания клавиш (Конфигуратор) — Редактор текстовых документов и модулей*.

Теперь вернёмся к функциям для работы с датами. Одну из них вы уже знаете и использовали. Это функция *ТекущаяДата()* (рисунок 3.48).

Очень часто в 1С:Предприятии вам могут понадобиться такие даты, как начало некоторого дня и конец некоторого дня. Для их получения существуют две функции с аналогичными именами: *НачалоДня()* и *КонецДня()*. Познакомьтесь с тем, где в 1С:Предприятии день начинается и где он заканчивается.

В одну переменную поместите текущую дату, а в другую переменную — результат функции *НачалоДня()*, применённой к первой переменной (рисунок 3.62).

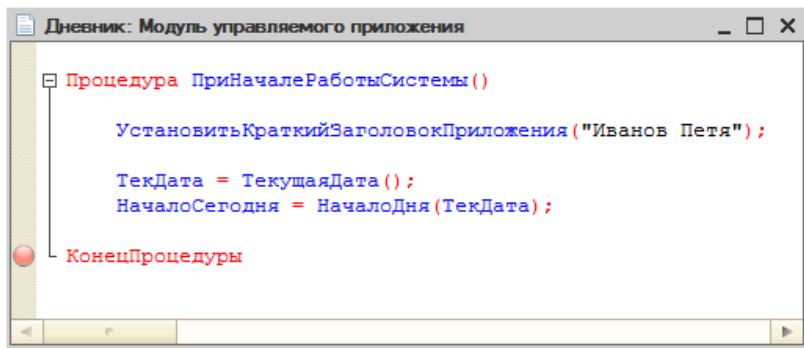


Рисунок 3.62. Начало дня

Запустите 1С:Предприятие в режиме отладки и посмотрите, чему равно *НачалоСегодня*. Это будет сегодняшняя дата и нулевое время. Например, 31.03.2016 0:00:00.

Совет

В 1С:Предприятии довольно часто встречаются прикладные задачи, когда нужно знать дату, но точное время не важно.

Например, вам нужно отмечать в календаре, ходили вы в этот день в школу (на работу) или не ходили. Вам не важно, в какое именно время проставляется такая отметка. Важно лишь то, что она есть в этот день. Или её нет.

В таких случаях в качестве даты такой отметки обычно используют начало дня.

Теперь посмотрите, чему равен конец дня. Создайте другую переменную, в которую будет помешён результат функции *КонецДня()*. Но внутри скобок у этой функции напишите не имя переменной *ТекДата*, а саму функцию *ТекущаяДата()* (рисунок 3.63).

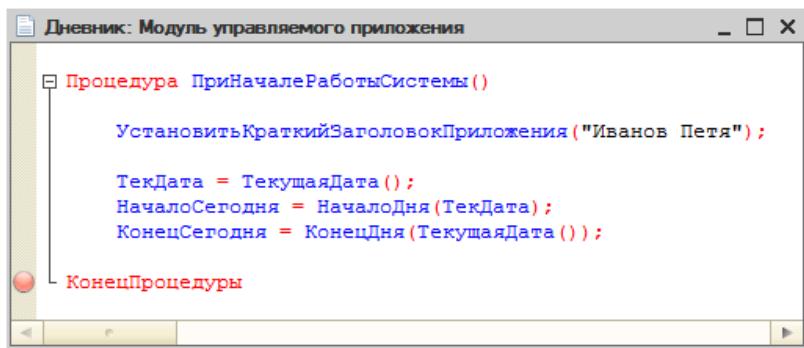


Рисунок 3.63. Использование результата функции в качестве параметра

Так тоже можно делать. Потому что внутри скобок может быть не только литерал или переменная, но и выражение. То есть некоторая формула, результатом которой может быть значение. Например, функция, возвращающая значение.

Запустите 1С:Предприятие в режиме отладки и посмотрите, чему равно *НачалоСегодня*. Это будет сегодняшняя дата и время 23:59:59. Например, 31.03.2016 23:59:59.

Теперь познакомьтесь с тем, какие операции можно выполнять с датами. Для значений типа *Дата* и типа *Число* во встроенном языке определены операции сложения и вычитания. То есть к дате можно прибавить некоторое количество секунд. Это используется для того, чтобы прибавить или убавить какой-то фиксированный промежуток времени.

Например, в одной переменной у вас находится текущее время. Как в другой переменной получить время на один час больше? Попробуйте выполнить это самостоятельно.

Для сравнения посмотрите на рисунок 3.64.

```

Дневник: Модуль управляемого приложения

Процедура ПриНачалоРаботыСистемы()
    УстановитьКраткийЗаголовокПриложения ("Иванов Петя");
    ТекДата = ТекущаяДата();
    НачасБольше = ТекДата + 60 * 60;
КонецПроцедуры

```

Рисунок 3.64. Дата на час больше

Чтобы получить дату на час больше, нужно к дате прибавить 60 раз по 60 секунд. То есть то количество секунд, которое содержится в одном часе.

Операция сложения (вычитания) даты с числом часто используется тогда, когда нужно получить начало следующего дня или конец предыдущего дня. Поскольку они отличаются всего на одну секунду, нужно эту секунду прибавить или отнять.

Попробуйте самостоятельно получить конец предыдущего дня. А потом сравните свой вариант с тем, что показано на рисунке 3.65.

```

Дневник: Модуль управляемого приложения

Процедура ПриНачалоРаботыСистемы()
    УстановитьКраткийЗаголовокПриложения ("Иванов Петя");
    ТекДата = ТекущаяДата();
    НачалоСледующегоДня = КонецДня(ТекДата) + 1;
КонецПроцедуры

```

Рисунок 3.65. Конец предыдущего дня

Ещё одна операция, которая определена для значений типа *Дата*, — это вычитание. Из одной даты можно вычесть другую дату. Что получится в этом случае?

Правильно. В результате получится некоторое количество секунд. К сожалению, во встроенным языке нет никакой функции, которая могла бы преобразовать количество секунд к виду, привычному для нас. Например: от одной даты до другой «прошло 2 месяца, 11 дней, 5 часов и 48 минут». И в рамках этой книги решать такие задачи вам не понадобится.

Но если вам интересно, вы можете попробовать самостоятельно написать программу из нескольких инструкций, которая будет это делать. Для этого вам потребуются операция деления, функция получения целой части *Цел()* и операция получения остатка от деления *%*.

Задание 3.15

Запишите начало дня 2 сентября 2016 года с помощью литерала даты.

Задание 3.16

Запишите начало дня 2 сентября 2016 года с помощью функции *Дата()*.

Задание 3.17

В одной переменной сохраните произвольную дату. В другой переменной вычислите начало следующего понедельника для произвольной даты.

Задание 3.18

В одной переменной сохраните произвольную дату. В другой переменной вычислите девять утра для произвольной даты.

Задание 3.19

Разность двух дат представьте в виде количества часов, минут и секунд. Например, «3 ч. 29 мин. 40 с».

3.9.13 Тип Булево и логические операции

Настало время познакомиться с логическими операциями и новым типом данных — Булево.

Самая простая логическая операция — это операция *Равно*. Она выясняет, равны между собой два значения или нет.

Эта операция определена для значений любых типов: для чисел, для строк, для дат и так далее. Важно лишь, чтобы оба значения, которые вы сравниваете, были одного и того же типа. Например, два числа или две строки.

Логическая операция *Равно* обозначается знаком «=». В подавляющем большинстве случаев логические операции используются в инструкциях *Если* и *Цикл*. Но эти инструкции вам пока незнакомы.

Поэтому знакомиться с операцией *Равно* вы будете с помощью инструкции присваивания. В ней эта операция будет выглядеть немного странно, поэтому для пущей важности вы заключите её в скобки (рисунок 3.66).

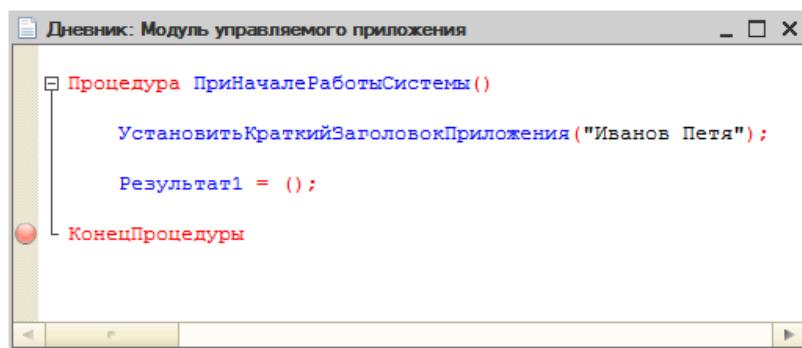


Рисунок 3.66. Заготовка для операции «Равно»

Теперь внутри скобок напишите операцию *Равно*. Например, $5 = 5$.

Затем напишите ещё одну инструкцию присваивания, но так, чтобы в операции *Равно* участвовали разные числа. Например, 5 и 2 .

Запустите 1C:Предприятие в режиме отладки и посмотрите, чему равны переменные в этих инструкциях (рисунок 3.67).

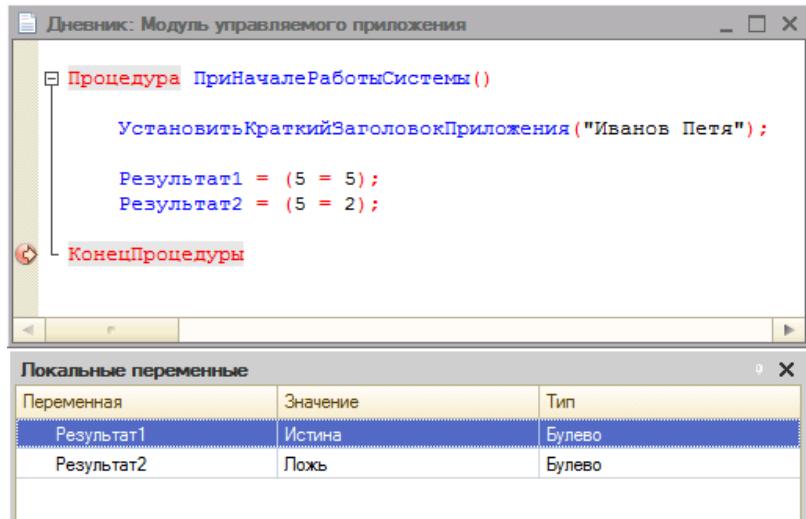


Рисунок 3.67. Значение «Истина» и значение «Ложь»

Вы увидите, что там, где сравниваются два одинаковых числа, результатом является значение *Истина*. А там, где сравниваются разные числа, результатом является значение *Ложь*. К тому же оба этих значения имеют тип *Булево* (ударение на букву «у»).

Примечание

Значения типа *Булево* — это значения, получаемые в результате логических операций. Значений типа Булево не бесконечное множество, как чисел или строк, а всего лишь два: *Истина* и *Ложь*.

Логических операций во встроенном языке, как и в жизни, довольно много. Все они интуитивно понятны, и нет особой надобности в том, чтобы тренироваться в их использовании. Единственное, что может вызвать трудность, — это то, какими символами они обозначаются. Но в этом вам поможет таблица 3.1.

Таблица 3.1. Операции сравнения

Название	Символ	Пример	Результат
Равно	=	5 = 5	Истина
		2 = 5	Ложь
Не равно	<>	2 <> 5	Истина
		5 <> 5	Ложь
Больше	>	5 > 2	Истина
		2 > 5	Ложь
Больше или равно	>=	2 >= 2	Истина
		2 >= 5	Ложь
Меньше	<	2 < 5	Истина
		5 < 2	Ложь
Меньше или равно	<=	2 <= 2	Истина
		5 <= 2	Ложь

Все перечисленные в этой таблице операции называются операциями сравнения. Поэтому что они сравнивают два значения. Причём операции *Равно* и *Не равно* можно применять к значениям любых типов. Главное, чтобы типы были одинаковыми с одной и с другой стороны операции.

А вот оставшиеся четыре операции (больше/меньше) можно применять только к двум числам, двум строкам или к двум датам.

В этом разделе вы не будете выполнять примеры с операциями сравнения. Но когда вы познакомитесь с инструкцией *Если*, у вас будет возможность как следует поупражняться в написании разных операций сравнения.

3.9.14 Булевые операции

А сейчас можно познакомиться с самыми интересными логическими операциями. Кроме операций сравнения есть ещё группа логических операций, которые называются *булевы операции*.

Если операции сравнения можно было выполнять (в основном) над числами, строками и датами, то булевые операции выполняются только со значениями типа *Булево*. То есть со значениями *Истина* и *Ложь*.

Пока что вам трудно представить, зачем это может понадобиться. Но сейчас на нескольких примерах вы сразу всё поймёте.

Для этих примеров вам понадобится записывать булевые значения прямо в тексте программы, то есть использовать литералы. Так вот, литералы типа *Булево* выглядят очень просто. Значение *Истина* обозначается *Истина*, а значение *Ложь* обозначается *Ложь*.

Например, если вы хотите создать переменную, которая будет описывать погоду за окном, вы можете написать так (рисунок 3.68).

```
Дневник: Модуль управляемого приложения
Процедура ПриНачалоРаботыСистемы()
УстановитьКраткийЗаголовокПриложения ("Иванов Петя");
СегодняСолнечно = Истина;
КонецПроцедуры
```

Рисунок 3.68. Литерал значения «Истина»

А если в другой переменной вы хотите уточнить, есть на улице дождь или нет, вы можете написать так (рисунок 3.69). Попробуйте и посмотрите, чему равны переменные.

```
Дневник: Модуль управляемого приложения
Процедура ПриНачалоРаботыСистемы()
УстановитьКраткийЗаголовокПриложения ("Иванов Петя");
СегодняСолнечно = Истина;
ИдетДождь = Ложь;
КонецПроцедуры
```

Рисунок 3.69. Литерал значения «Ложь»

Обратите внимание, что платформа раскрашивает слова *Истина* и *Ложь* красным цветом. Потому что это специальные зарезервированные слова. И это значит, что переменную с таким именем, если вам вдруг захочется, создать нельзя.

Подробнее

Полный список зарезервированных слов вы можете посмотреть в документации «Руководство разработчика 8.3. Раздел 4.2.4.6. Зарезервированные слова».

Чтобы познакомиться с первой булевой операцией, создайте две переменные: **ЯУмеюПлавать** и **РядомЕстьМоре**. Задайте им те значения, которые есть на самом деле. А в книге, для примера, я присвою им значение **Истина** (рисунок 3.70).

```

Дневник: Модуль управляемого приложения
└ Процедура ПриНачалоРаботыСистемы()
    УстановитьКраткийЗаголовокПриложения ("Иванов Петя");
    ЯУмеюПлавать = Истина;
    РядомЕстьМоре = Истина;
    КонецПроцедуры

```

Рисунок 3.70. Переменные «ЯУмеюПлавать» и «РядомЕстьМоре»

Теперь вам нужно с помощью логической операции узнать, будете вы плавать в море или нет.

Сначала попробуйте сказать это обычными словами. Наверное, получится что-то вроде «если я умею плавать и если рядом есть море, то тогда я буду плавать в море».

А теперь посмотрите (рисунок 3.71), как эта же фраза выглядит на встроенном языке.

```

Дневник: Модуль управляемого приложения
└ Процедура ПриНачалоРаботыСистемы()
    УстановитьКраткийЗаголовокПриложения ("Иванов Петя");
    ЯУмеюПлавать = Истина;
    РядомЕстьМоре = Истина;
    ЯБудуПлаватьВМоре = ЯУмеюПлавать И РядомЕстьМоре;
    КонецПроцедуры

```

Рисунок 3.71. Операция «И»

Чтобы сказать то же самое на встроенном языке, используется операция *И*. Как она работает, вы можете проверить сами на этом примере, изменив значения переменных.

Плавать в море вы будете только в том случае, когда оно есть рядом и вы умеете плавать. Если моря рядом нет, то плавать просто негде. А если вы не умеете плавать, то у вас тоже ничего не получится. Это очень опасно, вы можете утонуть. Ну, а если и моря нет, и плавать вы не умеете, то тем более поплавать в море вам не удастся.

Примечание

Итак, операция *И* возвращает значение *Истина* только в том случае, когда оба операнда имеют значение *Истина*. Во всех остальных случаях она возвращает значение *Ложь*.

Теперь решите другую задачу. Создайте две переменные **УМеняЕстьЛодка** и **УМеняЕстьПлот**. Можете задать им любые значения. А в книге, для примера, я присвою им значение **Истина** (рисунок 3.72).

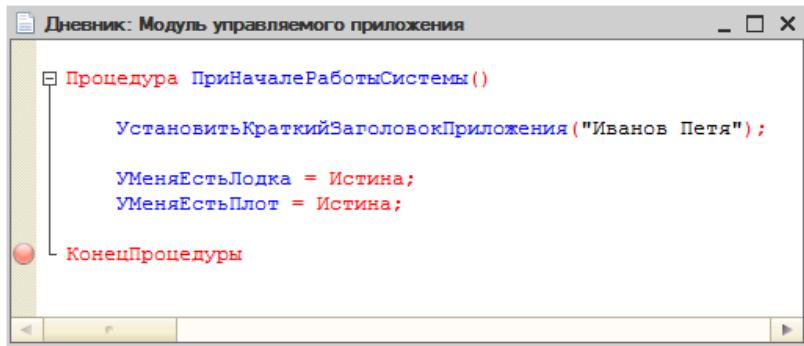


Рисунок 3.72. Переменные «УМеняЕстьЛодка» и «УМеняЕстьПлот»

Теперь вам нужно с помощью логической операции узнать, сможете вы переплыть через озеро или нет.

Сначала попробуйте сказать это обычными словами. Наверное, получится что-то вроде «если у меня есть плот или лодка, то я могу переплыть через озеро».

А теперь посмотрите (рисунок 3.73), как эта же фраза выглядит на встроенном языке.

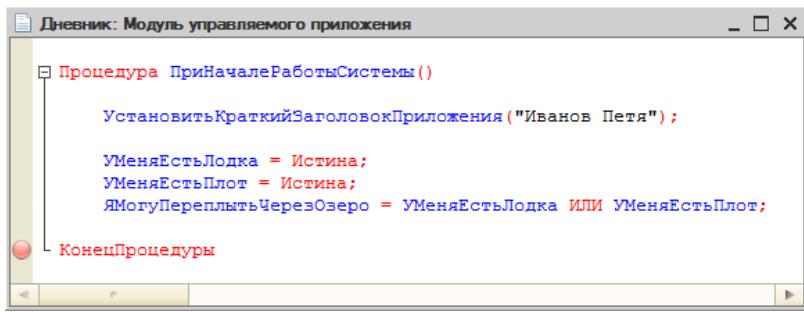


Рисунок 3.73. Операция «ИЛИ»

Чтобы сказать то же самое на встроенном языке, используется операция *ИЛИ*. Как она работает, вы можете проверить сами на этом примере, изменив значения переменных.

Если у вас есть лодка, вы можете переплыть через озеро? Да.

А если у вас не лодка, а плот — тогда сможете? Да, сможете.

А если у вас есть и лодка, и плот? Конечно! Даже два раза! Один раз на лодке, а другой — на плоту (шутка)!

А в каком случае вам не удастся переплыть через озеро? Правильно. Когда у вас нет ни лодки, ни плота.

Примечание

Итак, операция *ИЛИ* работает аналогично операции *I*, только в отношении значения *Ложь*. Она возвращает значение *Ложь* только в том случае, когда оба операнда имеют значение *Ложь*. Во всех остальных случаях она возвращает значение *Истина*.

Теперь усложните свой пример и познакомьтесь с тем, что в одном выражении могут использоваться сразу несколько булевых операций.

Создайте три переменные: *ЯУмеюПлавать*, *РядомЕстьМоре* и *РядомЕстьБассейн*. Задайте им те значения, которые есть на самом деле. А в книге я снова не буду усложнять и присвою им значение *Истина* (рисунок 3.74).

```

Дневник: Модуль управляемого приложения

Процедура ПриНачалеРаботыСистемы()
    УстановитьКраткийЗаголовокПриложения("Иванов Петя");
    ЯУмеюПлавать = Истина;
    РядомЕстьМоре = Истина;
    РядомЕстьБассейн = Истина;
КонецПроцедуры

```

Рисунок 3.74. Переменные «ЯУмеюПлавать», «РядомЕстьМоре» и «РядомЕстьБассейн»

А вопрос, решением которого вы займётесь, будет практически тем же самым. Чему будет равна переменная **ЯБудуПлавать**?

Если вы будете отвечать словами, то скажете: «Я буду плавать, если я умею плавать и рядом есть море или бассейн». Попробуйте записать это на встроенном языке (рисунок 3.75).

```

Дневник: Модуль управляемого приложения

Процедура ПриНачалеРаботыСистемы()
    УстановитьКраткийЗаголовокПриложения("Иванов Петя");
    ЯУмеюПлавать = Истина;
    РядомЕстьМоре = Истина;
    РядомЕстьБассейн = Истина;
    ЯБудуПлавать = ЯУмеюПлавать И РядомЕстьМоре ИЛИ РядомЕстьБассейн;
КонецПроцедуры

```

Рисунок 3.75. Переменная «ЯБудуПлавать»

На первый взгляд кажется, что всё хорошо. Но нужно проверить.

Например, вы не умеете плавать. В этом случае не важно, есть рядом бассейн или нет: плавать вы не должны. Но что говорит вам программа? Попробуйте (рисунок 3.76).

Переменная	Значение	Тип
ЯУмеюПлавать	Ложь	Булево
РядомЕстьМоре	Истина	Булево
РядомЕстьБассейн	Истина	Булево
ЯБудуПлавать	Истина	Булево

Рисунок 3.76. ЯБудуПлавать = Истина

Программа говорит вам, что плавать вы будете. Получается какое-то «опасное» программирование. С таким программированием и утонуть недолго!

В чём же дело? Может быть, вы написали неправильные операции? Нет, правильные.

Просто вы не учли, что если несколько булевых операций встречаются в одном выражении, у них есть определённый порядок выполнения. Точно так же, как было с арифметическими операциями $+$, $-$, $*$ и $/$.

Сначала выполняется операция *И*, а затем уже выполняется операция *ИЛИ*. А если подряд встречаются несколько одинаковых операций, то они выполняются в той последовательности, в которой написаны.

Поэтому в примере выражение было посчитано таким образом (рисунок 3.77).

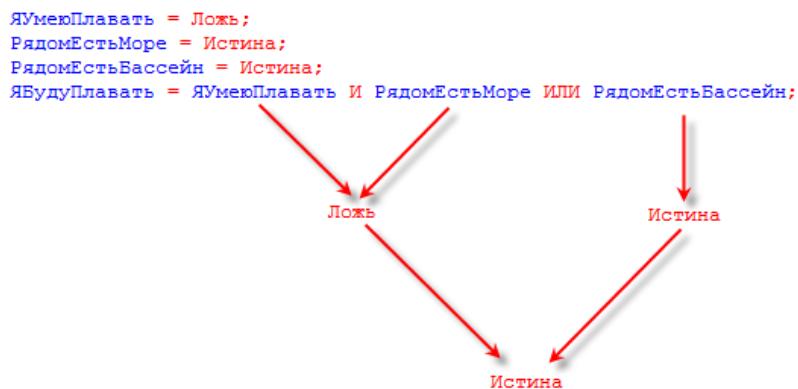


Рисунок 3.77. Порядок вычисления выражения

Сначала была выполнена операция *И*, которая дала *Ложь*. А затем операция *ИЛИ*, результатом которой явилась *Истина*.

На самом же деле для вас в этом выражении важны две вещи. Что вы умеете плавать и что есть, где плавать. А море это или бассейн — не так важно.

Поэтому правильным решением будет *РядомЕстьМоре ИЛИ РядомЕстьБассейн* заключить в скобки (рисунок 3.78). Попробуйте.

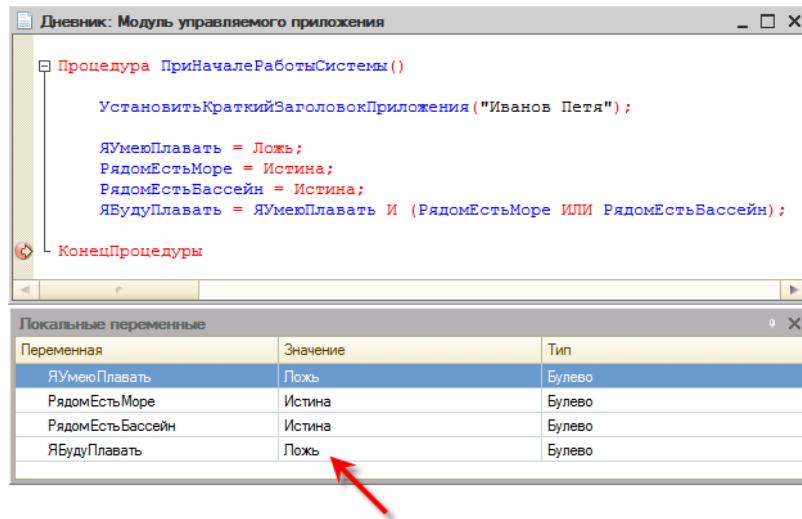


Рисунок 3.78. ЯБудуПлавать = Ложь

Тогда платформа сначала вычислит то выражение, которое в скобках. То есть узнает, есть, где плавать, или плавать негде. А потом уже поинтересуется вашим умением плавать (рисунок 3.79).

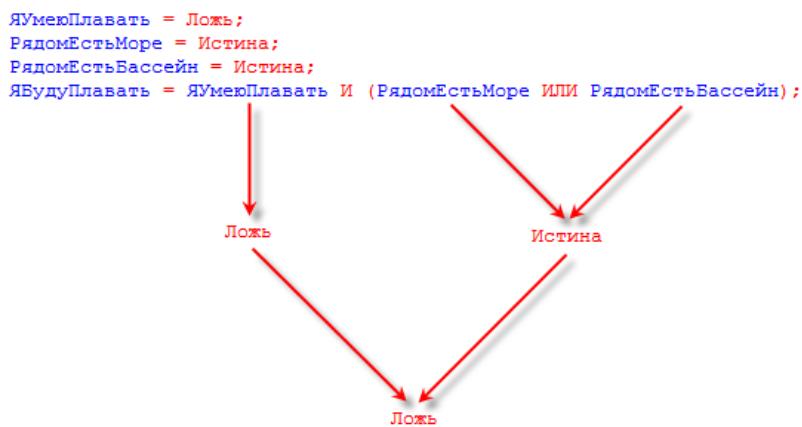


Рисунок 3.79. Порядок вычисления выражения

Чтобы закрепить этот пример, немного измените его условие. Создайте три переменные: *ЯУмеюПлавать*, *РядомЕстьМоре* и *РядомЕстьВанна*. Задайте им те значения, которые были в последнем примере: *Ложь*, *Истина* и *Истина* (рисунок 3.80).

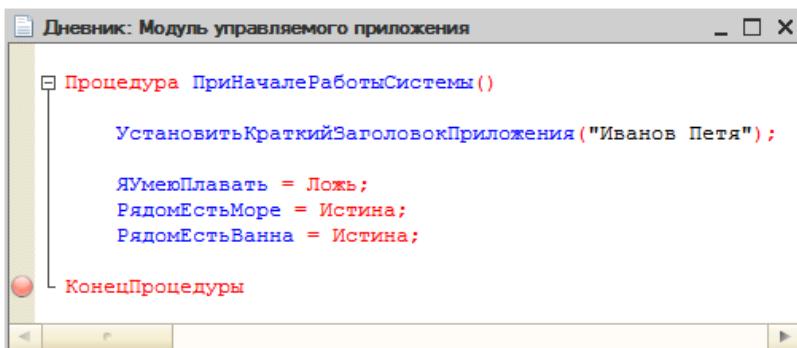


Рисунок 3.80. Переменные «ЯУмеюПлавать», «РядомЕстьМоре» и «РядомЕстьВанна»

А вычислить нужно будет переменную, которая называется *ЯБудуКупаться*. Будьте внимательны: имя переменной изменилось не просто так.

Попробуйте записать выражение на встроенном языке и потом сравните с тем, что на рисунке 3.81.

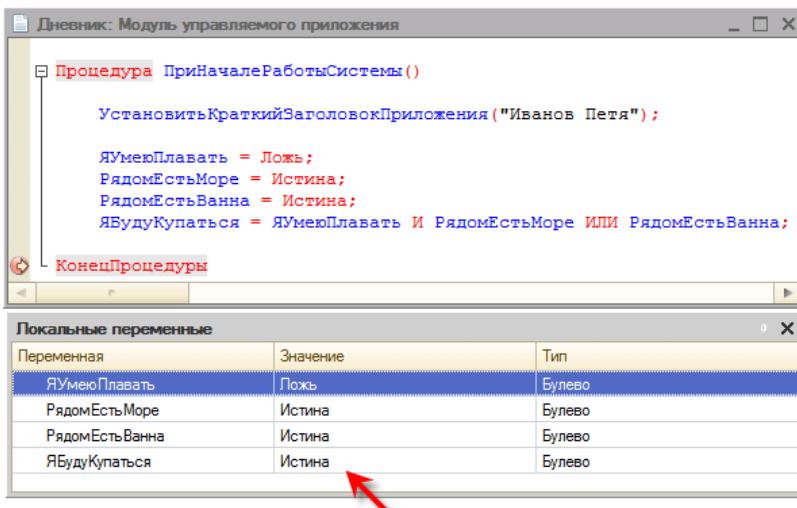


Рисунок 3.81. ЯБудуКупаться = Истина

Запустите 1С:Предприятие в режиме отладки и посмотрите, какой получается результат, если изменить значения переменных.

Почему в этом случае скобки не понадобились, хотя пример внешне очень похож на предыдущий? Скобки не понадобились потому, что «естественный» порядок выполнения логических операций в этом выражении вполне вас устраивает (рисунок 3.82).

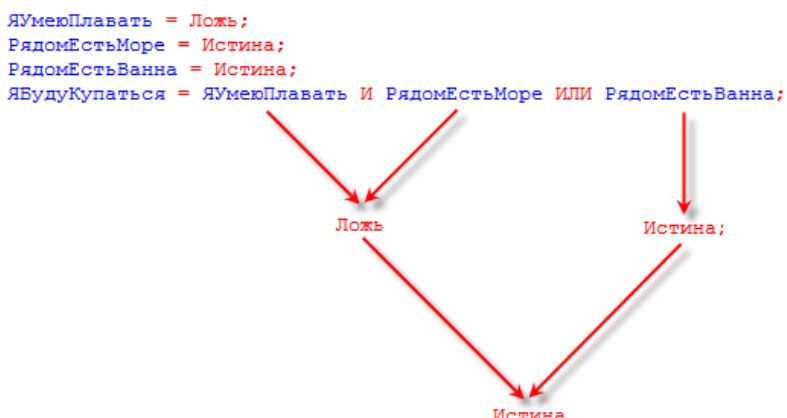


Рисунок 3.82. Порядок вычисления выражения

Действительно, умение плавать важно только тогда, когда вы собираетесь зайти в море. Если вы хотите искупаться в ванне, то умение плавать вам совершенно не нужно.

Поэтому совершенно правильно, что сначала выполняется операция *И* и выясняется, можете ли вы купаться в море. А затем уже проверяется наличие ванны, для использования которой способность плавать не требуется.

В заключение нужно сказать ещё об одной булевой операции — операции *НЕ*. Она очень простая. Она возвращает булево значение, противоположное тому, которое имеется.

Чтобы посмотреть, как она работает, вы можете использовать пример, показанный на рисунке 3.83.

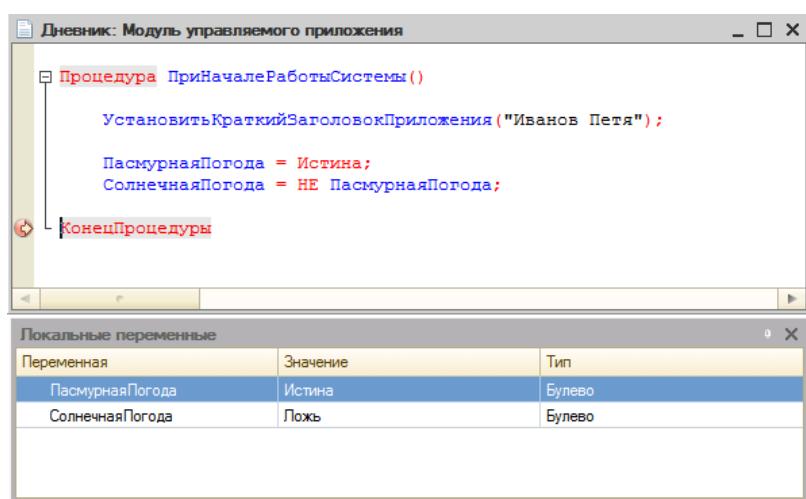


Рисунок 3.83. Операция «НЕ»

Обычно с использованием этой операции никаких трудностей не возникает. Единственное, что может вам понадобиться, это порядок выполнения булевых операций. Полностью он выглядит так:

- сначала выполняется то, что в скобках;
- затем операция *НЕ*;
- затем операция *И*;

- затем операция *ИЛИ*.

Задание 3.20

Чтобы по пути на работу защититься от дождя, вы берёте с собой зонт. С помощью переменных *ЯИдуНаРаботу* и *ИдетДождь* вычислите значение переменной *НадоВзятьЗонт*.

Проверьте, что ваша инструкция правильно работает при любых значениях исходных переменных.

Задание 3.21

В каких случаях вы не ходите в школу? Когда выходной (праздники, каникулы) и когда вы болеете. С помощью переменных *СегодняВыходной* и *ЯБолею* вычислите значение переменной *ЯНеИдуВШколу*.

Проверьте, что ваша инструкция правильно работает при любых значениях исходных переменных.

Задание 3.22

В хорошую погоду, когда у вас нет занятий, вы всегда идёте гулять. С помощью переменных *ХорошаяПогода*, *СегодняВыходной* и *СегодняПраздник* вычислите значение переменной *ЯИдуГулять*.

Проверьте, что ваша инструкция правильно работает при любых значениях исходных переменных.

Задание 3.23

Вы работаете только по будним дням. С помощью переменных *СегодняСуббота* и *СегодняВоскресенье* вычислите значение переменной *ЯИдуНаРаботу*.

Проверьте, что ваша инструкция правильно работает при любых значениях исходных переменных.

3.9.15 Инструкция Если

Наконец настало время познакомиться с ещё одной инструкцией, которая используется во встроенном языке.

Если бы вы использовали только инструкцию присваивания, то ваша программа выглядела бы как прямая дорога из пункта «А» в пункт «Б». От начала к завершению (рисунок 3.84).

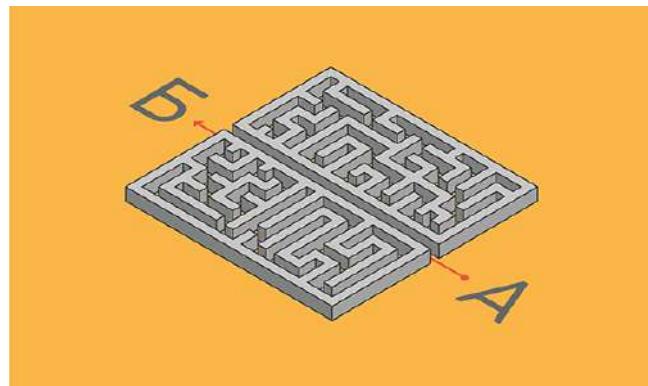


Рисунок 3.84. Прямая дорога из пункта «А» в пункт «Б»

Потому что инструкции присваивания выполняются одна за другой в том порядке, в котором они написаны.

Но на самом деле большинство программ выглядят по-другому. Между началом программы и её завершением существует много разных путей, по которым может пойти исполнение программы (рисунок 3.85).

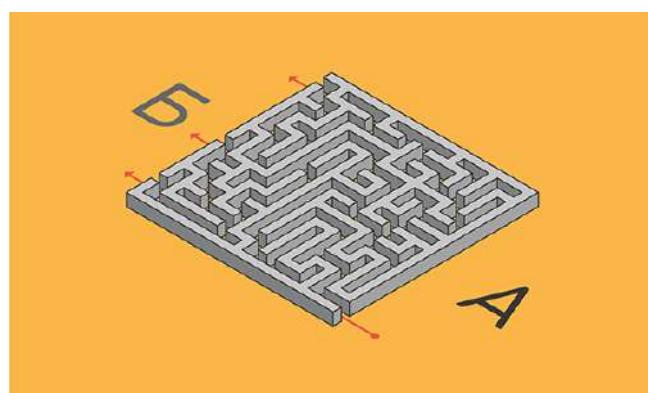


Рисунок 3.85. Много путей из пункта «А» в пункт «Б»

Для этого есть много разных причин. Например, это зависит от того, какие данные уже есть в программе. Если есть все необходимые данные, то исполнение пойдёт по одному пути. Если каких-то данных не хватает, то по другому пути.

Также это зависит от того, что вы хотите получить в результате. Если из своего электронного дневника вы хотите узнать, будет завтра урок математики или нет, то это будет один путь, простой. А если вы хотите получить расписание уроков на всю неделю с указанием кабинетов, преподавателей и того, что задано, то это будет другой путь, сложный. Потому что программе нужно будет обойти много разных мест и собрать для вас всю информацию, которую вы просите.

Именно для того, чтобы направить исполнение программы по одному или по другому пути, существует инструкция *Если*. Выглядит она очень просто (рисунок 3.86).

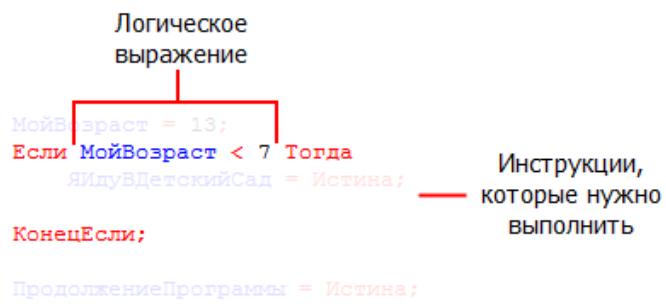


Рисунок 3.86. Инструкция «Если»

Она позволяет выделить несколько инструкций, которые будут выполняться не всегда, а только тогда, когда в результате вычисления логического выражения получается *Истина*.

Слова *Если*, *Тогда* и *КонецЕсли* являются обязательными в этой инструкции. Логическое выражение пишется между словами *Если* и *Тогда*. А *КонецЕсли* показывает, где заканчиваются инструкции, выполнение которых зависит от условия.

Чтобы посмотреть, как это работает, сделайте небольшой пример. Создайте переменную и запишите в неё свой возраст. А затем создайте другую переменную, *ЯИдуВДетскийСад*, и присвойте ей значение *Истина*. Но эту переменную создайте только в том случае, если вы ещё не достигли школьного возраста. Чтобы не запутаться, будем считать, что школьный возраст начинается с 7 лет.

Когда вы сделаете свой пример, сравните его с тем, что на рисунке 3.87.

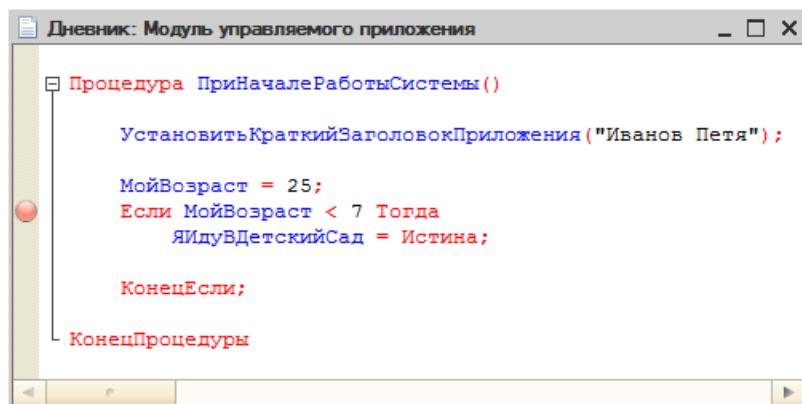


Рисунок 3.87. Пример

Установите точку останова на строке *Если*, запустите 1С:Предприятие в режиме отладки и посмотрите значение выражения *МойВозраст < 7* (*Shift+F9*) (рисунок 3.88).

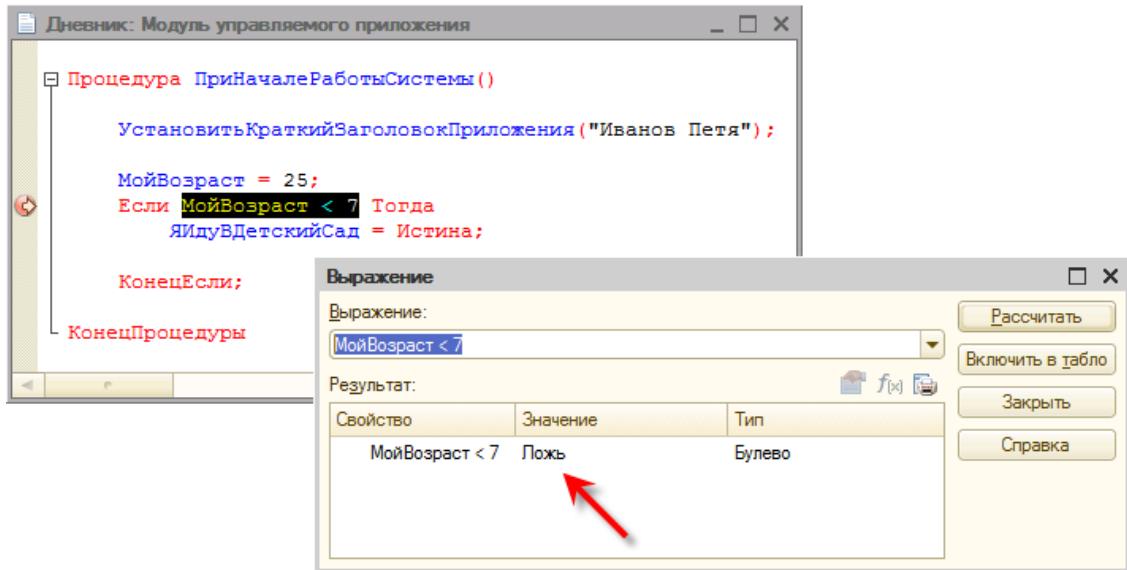


Рисунок 3.88. Значение выражения «МойВозраст < 7»

Вы увидите, что оно равно *Ложь*. И это правильно. Ваш возраст больше, чем 7 лет.

Раз выражение ложно, значит, платформа должна пропустить все инструкции, которые написаны внутри инструкции *Если*. Проверьте это с помощью пошагового исполнения (F11). Сделайте один шаг (рисунок 3.89).

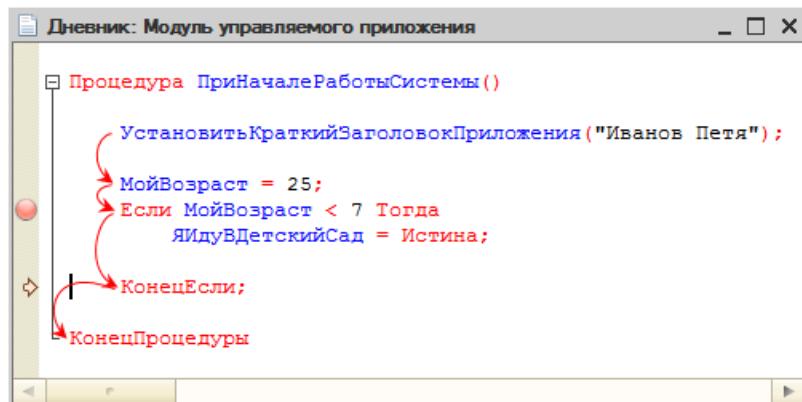


Рисунок 3.89. Один шаг исполнения

Действительно, платформа перейдёт на строку *КонецЕсли*. А если вы шагнёте ещё раз, то она остановится на строке *КонецПроцедуры*. При этом значение переменной *ЯИдуВДетскийСад* будет *Неопределено*, потому что эта инструкция не выполнялась и такая переменная даже не создавалась.

А теперь познакомьтесь с тем, как поведёт себя платформа в том случае, когда выражение истинно. Для этого вы не будете изменять текст программы. Вы воспользуетесь возможностью *изменения значений переменных* прямо в процессе отладки.

Чтобы это сделать, вам потребуется сначала *перезапустить отладку*. Для этого выполните команду *Отладка — Перезапустить* (рисунок 3.90).

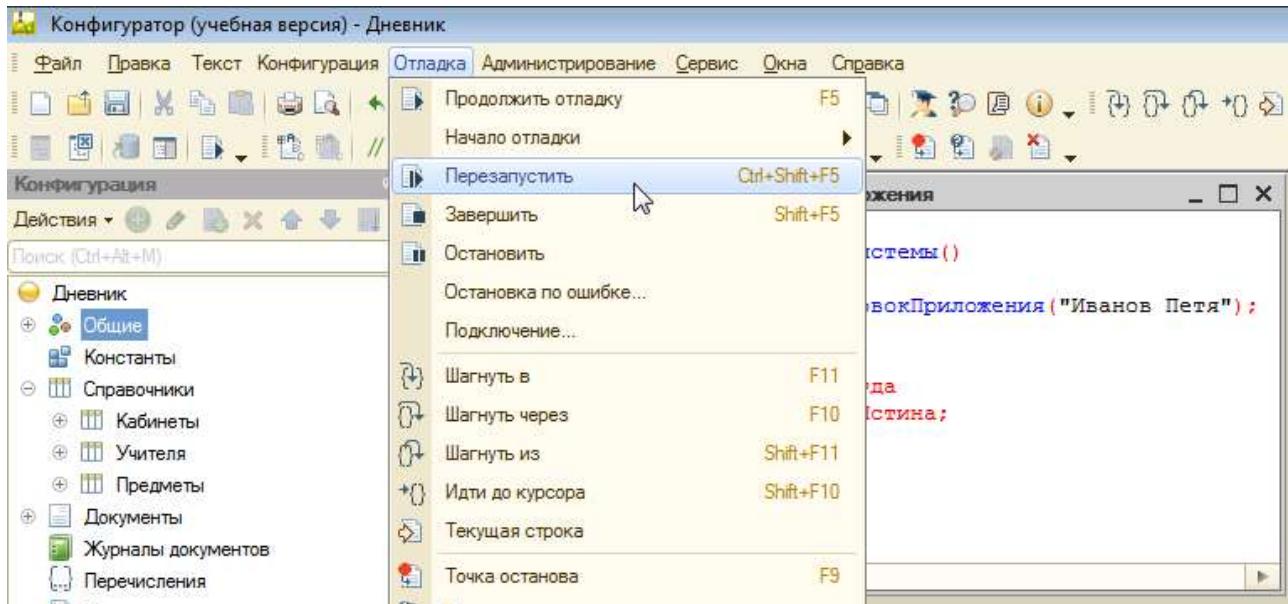


Рисунок 3.90. Перезапустить отладку

После того как отладка перезапустится, исполнение снова остановится на строке *Если*. Откройте локальные переменные (если это окно у вас закрыто).

Дважды щёлкните мышью по ячейке со значением 25 (рисунок 3.91).

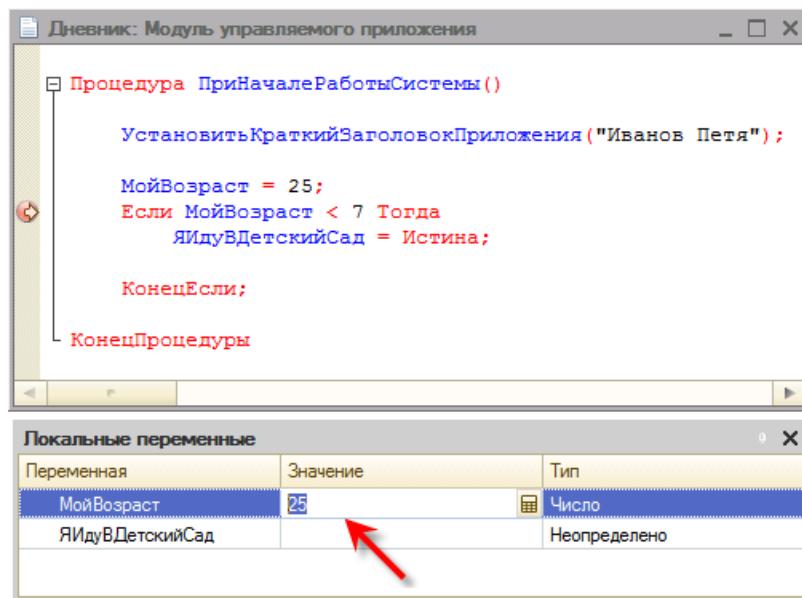


Рисунок 3.91. Изменение значения переменной

Она перейдёт в режим редактирования. Введите 6 и нажмите Enter.

С этого момента значение переменной *МойВозраст* будет равно 6. А вы можете продолжать отладку так же, как и раньше.

Посмотрите значение выражения *МойВозраст < 7*. Теперь оно будет равно *Истина* (рисунок 3.92).

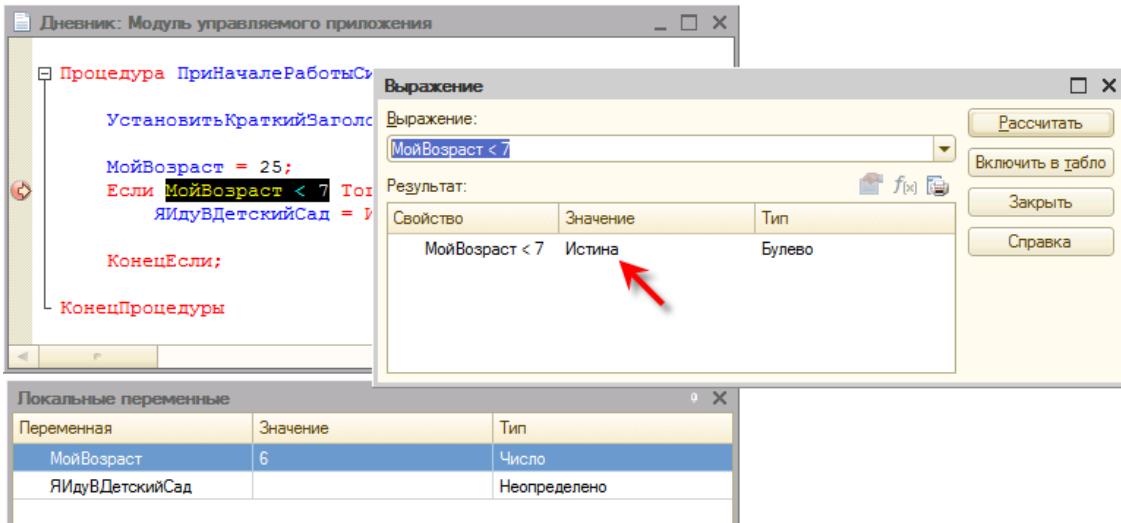


Рисунок 3.92. Значение выражения «МойВозраст < 7»

Сделайте один шаг. Исполнение зайдёт «внутрь» инструкции *Если* (рисунок 3.93).

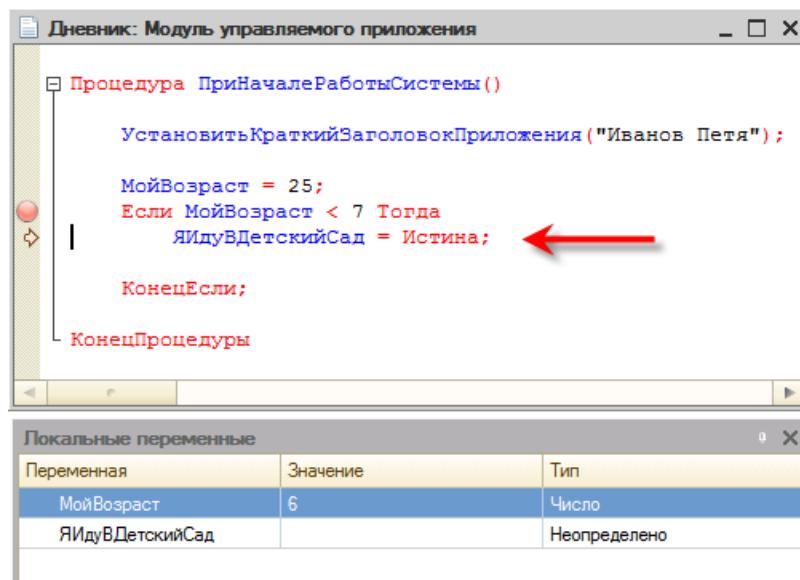


Рисунок 3.93. Один шаг исполнения

Теперь, если вы сделаете ещё пару шагов, вы дойдёте до конца процедуры. А значение переменной *ЯИдуВДетскийСад* будет равно *Истина*.

Подробнее

Подробнее вы можете прочитать про изменение значений переменных при отладке в документации «[Руководство разработчика 8.3. Раздел 28.2.10. Изменение значения переменной](#)».

До сих пор вы рассматривали самую простую форму инструкции *Если*. На самом деле она может быть более сложной. Смотрите.

В вашем примере «особенные» действия выполняются только в том случае, когда ваш возраст меньше 7 лет. Но представьте, что вам нужно принять решение из двух вариантов. Если меньше 7 лет, вы идёте в детский сад. А во всех других случаях вы идёте в школу. Как это записать на встроенном языке?

Оказывается, очень просто. Для этого используется ключевое слово *Иначе* (листинг 3.22).

Листинг 3.22. Ключевое слово «Иначе»

```

МойВозраст = 25;
Если МойВозраст < 7 Тогда
    ЯИдуВДетскийСад = Истина;
Иначе
    // Инструкции, которые будут выполнены во всех остальных случаях
КонецЕсли;

```

Доработайте пример так, чтобы в результате его работы у вас создавались две переменные: *ЯИдуВДетскийСад* и *ЯИдуВШколу*. И чтобы они принимали правильные значения (*Истина* или *Ложь*) в зависимости от возраста, указанного в переменной *МойВозраст*.

Сравните свой результат с тем, что показано на рисунке 3.94.

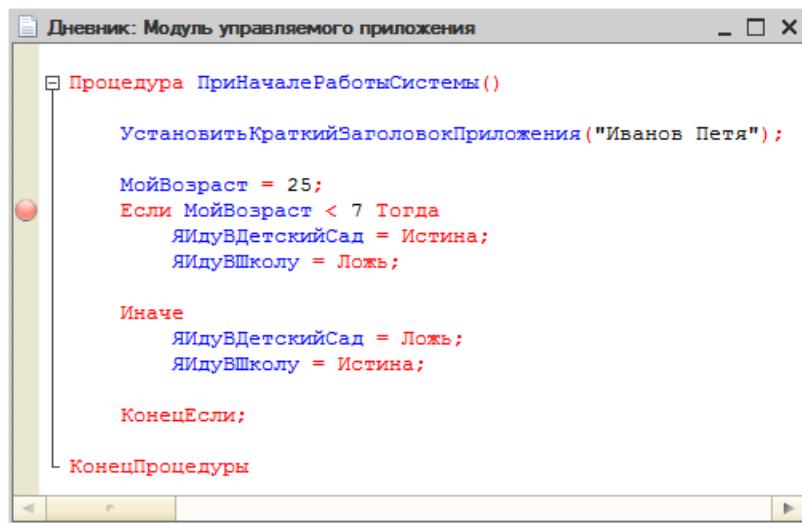


Рисунок 3.94. Доработанный пример

Запустите 1С:Предприятие в режиме отладки и проверьте по шагам правильность работы примера для разных значений возраста. Изменяйте возраст прямо в процессе отладки.

Очень часто в инструкции *Если* анализируется не одно, а несколько условий. Например, в одно прекрасное утро вы проснулись и вспомнили, что вам нужно идти учиться. Но вы не помните, куда именно: в детский сад, в школу или в институт.

Зато вам известны правила. До 7 лет нужно идти в детский сад. После детского сада нужно идти в школу. Учиться в школе заканчивают в 18 лет. А в 19 лет поступают в институт.

Чтобы записать этот алгоритм на встроенном языке, вам понадобится ещё одно ключевое слово — *ИначеЕсли* (листинг 3.23).

Листинг 3.23. Ключевое слово «ИначЕсли»

```

МойВозраст = 25;
Если МойВозраст < 7 Тогда
    ЯИдуВДетскийСад = Истина;
    ЯИдуВШколу = Ложь;

ИначЕсли      Тогда      // Условие, при котором нужно идти в школу
    ЯИдуВДетскийСад = Ложь;
    ЯИдуВШколу = Истина;

Иначе

КонецЕсли;

```

В вашем примере его нужно будет написать вместо ключевого слова *Иначе*. А затем написать условие, при котором вам нужно идти в школу.

А в конце примера вы снова добавите слово *Иначе* и укажете, что во всех остальных случаях вы идёте в институт. Попробуйте сделать пример самостоятельно.

Чтобы проверить свой результат, сравните его с рисунком 3.95.

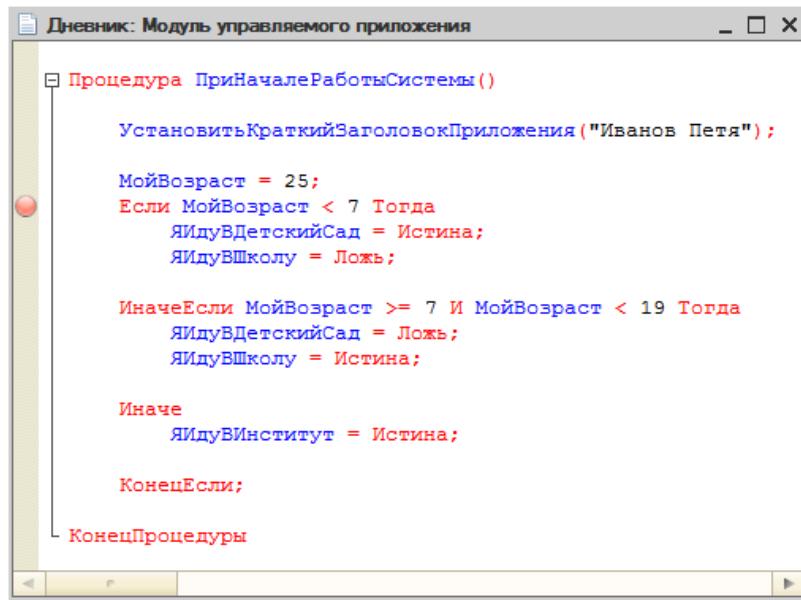


Рисунок 3.95. Пример «Я иду в институт»

В режиме отладки пройдите по разным веткам инструкции *Если* и посмотрите, как она работает.

Вы увидите, что выражения, которые указаны в инструкции *Если*, вычисляются и анализируются по очереди. В том порядке, в котором они написаны. Как только попадается истинное выражение, выполняется соответствующая ветка инструкции *Если*. После этого исполнение переходит на *КонецЕсли*. То есть инструкции, расположенные в тех ветках, где выражение было ложно, или в тех ветках, до которых проверка выражений не дошла, просто не выполняются.

В этом заключается особенность инструкции *Если*, о которой нужно помнить и которую нужно учитывать. Если вы откроете локальные переменные, то увидите, что после выполнения примера переменная *ЯИдуВИнститут* может оказаться не определена. Например, если ваш возраст равен 15 (рисунок 3.96).

Локальные переменные		
Переменная	Значение	Тип
МойВозраст	15	Число
ЯИдуВДетскийСад	Ложь	Булево
ЯИдуВШколу	Истина	Булево
ЯИдуВИнститут		Неопределено

Рисунок 3.96. Переменная «ЯИдуВИнститут» не определена

А если возраст равен 20, то не определены окажутся переменные ЯИдуВДетскийСад и ЯИдуВШколу (рисунок 3.97).

Локальные переменные		
Переменная	Значение	Тип
МойВозраст	20	Число
ЯИдуВДетскийСад		Неопределено
ЯИдуВШколу		Неопределено
ЯИдуВИнститут	Истина	Булево

Рисунок 3.97. Не определены переменные «ЯИдуВДетскийСад» и «ЯИдуВШколу»

А ведь в «живой» программе вы наверняка будете создавать подобные переменные не просто так, а для того чтобы использовать их в дальнейшем. И если какой-то из этих переменных не окажется, ваша программа работать не сможет.

Какой выход из этой ситуации? Нудно и дотошно в каждой ветке инструкции *Если* прописывать создание всех переменных, которые вам могут понадобиться? Нет.

Есть гораздо более простое и удобное решение. Прямо перед инструкцией *Если* вы создаёте все переменные, которые вам понадобятся. И присваиваете им некоторое стандартное значение — значение «по умолчанию». Например, *Ложь*.

А дальше, в ветках инструкции *Если*, вы меняете значения только тех переменных, которым это действительно нужно. Посмотрите пример на рисунке 3.98.

```

Дневник: Модуль управляемого приложения
Процедура ПриНачалеРаботыСистемы()
    УстановитьКраткийЗаголовокПриложения("Иванов Петя");
    МойВозраст = 25;
    ЯИдуВДетскийСад = Ложь;
    ЯИдуВШколу = Ложь;
    ЯИдуВИнститут = Ложь;
    Если МойВозраст < 7 Тогда
        ЯИдуВДетскийСад = Истина;

    ИначеЕсли МойВозраст >= 7 И МойВозраст < 19 Тогда
        ЯИдуВШколу = Истина;

    Иначе
        ЯИдуВИнститут = Истина;

    КонецЕсли;
КонецПроцедуры

```

Рисунок 3.98. Другой вариант примера

Сделайте этот пример в своей конфигурации и посмотрите, как он работает.

При любых значениях возраста будут существовать все переменные. И они будут иметь правильные значения (рисунок 3.99).

Локальные переменные		
Переменная	Значение	Тип
МойВозраст	25	Число
ЯИдуВДетскийСад	Ложь	Булево
ЯИдуВШколу	Ложь	Булево
ЯИдуВИнститут	Истина	Булево

Рисунок 3.99. Все переменные определены

Задание 3.24

Вы с другом каждый день играете в игру через Интернет. В будний день родители разрешают вам играть только один час. В выходные дни вы можете играть по 4 часа. Используя переменную *НомерДняНедели*, вычислите, сколько часов вы можете играть с другом в выбранный день. Результат поместите в переменную *ВремяДляИгры*.

Задание 3.25

В задании 3.24 нужно учесть, что номер дня недели может быть задан с ошибкой. В случае ошибки ваша программа должна вернуть результат 0.

Задание 3.26

Супермаркет работает с 9 часов утра до 8 часов вечера. Вам нужно сходить в супермаркет и купить кефир. Если кефира не будет, то ряженку. Если не будет ни того, ни другого, тогда нужно зайти в круглосуточный магазин и купить молоко, если оно там есть.

Используйте переменные *ЕстьКефир*, *ЕстьРяженка*, *ЕстьМолоко* и *ТекущийЧас*. Название своей покупки в виде строки поместите в переменную *МояПокупка*.

3.9.16 Красивая программа

Я уже говорил раньше, что ваша конфигурация должна быть красивой. Говорил о том, что имена переменных должны быть удобными, понятными. Говорил, что выражения должны быть записаны понятно и «читабельно». Теперь настал удобный случай рассказать о том, как должен выглядеть текст вашей программы.

Вы, наверное, уже привыкли к тому, что платформа сама раскрашивает слова в вашей программе. И это удобно. Ещё, может быть, вы обратили внимание на то, что платформа сама делает отступ, когда вы пишете инструкцию *Если*.

Если вы этого не заметили, то поставьте маленький эксперимент. В модуле напишите *Если a = 2 Тогда*. И нажмите клавишу *Enter* (листинг 3.24).

Листинг 3.24. Синтаксический отступ

```
Если a = 2 Тогда
// Синтаксический отступ
```

Вы увидите, что на новой строке курсор встанет не под буквой «Е», а с некоторым сдвигом вправо. Этот сдвиг называется *синтаксический отступ*. Он помогает вам лучше

и легче читать текст программы. Потому что «подчинённые» инструкции, расположенные внутри одной ветки, оказываются выделены визуально (листинг 3.25).

Листинг 3.25. Есть синтаксический отступ

```
Если МойВозраст < 7 Тогда
    ЯИдуВДетскийСад = Истина;
    ЯИдуВШколу = Ложь;
ИначеЕсли МойВозраст >= 7 И МойВозраст < 19 Тогда
    ЯИдуВДетскийСад = Ложь;
    ЯИдуВШколу = Истина;
Иначе
    ЯИдуВИнститут = Истина;
КонецЕсли;
```

Посмотрите, какая «каша» получилась бы, если бы синтаксического отступа не было (листинг 3.26).

Листинг 3.26. Нет синтаксического отступа

```
Если МойВозраст < 7 Тогда
ЯИдуВДетскийСад = Истина;
ЯИдуВШколу = Ложь;
ИначеЕсли МойВозраст >= 7 И МойВозраст < 19 Тогда
ЯИдуВДетскийСад = Ложь;
ЯИдуВШколу = Истина;
Иначе
ЯИдуВИнститут = Истина;
КонецЕсли;
```

Прочитать такую «кашу» и разобраться в ней очень сложно.

Поэтому синтаксический отступ — это важная, обязательная часть программы. Именно поэтому платформа делает его автоматически, когда вы набираете текст инструкции последовательно.

Но далеко не всегда вы работаете именно так. Даже при выполнении ваших простых примеров вы видите, что приходится изменять то, что уже написано. Вставлять новые строки. Копировать из одного места в другое. В этих случаях платформа не может автоматически сделать синтаксический отступ. В результате у вас вполне может получиться такой текст (листинг 3.27).

Листинг 3.27. Текст с нарушенным форматированием

```
Если МойВозраст < 7 Тогда
    ЯИдуВДетскийСад = Истина;
    ЯИдуВШколу = Ложь;
ИначеЕсли МойВозраст >= 7 И МойВозраст < 19 Тогда
ЯИдуВДетскийСад = Ложь;
ЯИдуВШколу = Истина;
Иначе
    ЯИдуВИнститут = Истина;
КонецЕсли;
```

Как с ним быть? Двигать каждую строку вручную, чтобы было «красиво»? Это скучно, долго и неинтересно.

Поэтому в платформе есть команда, которая может сделать это автоматически. Но сейчас вы её не видите. Панель Текст, в которой расположена эта команда, при стандартных настройках конфигуратора не показывается. Но вы всегда можете её включить, чтобы команда форматирования была у вас под рукой.

Для этого нажмите правой кнопкой мыши на пустом месте справа в верхней командной панели (рисунок 3.100).

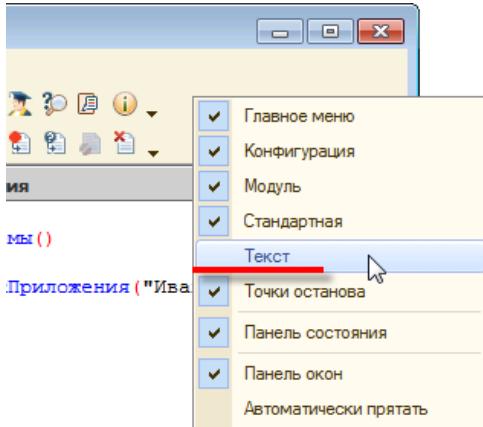


Рисунок 3.100. Настройка панелей конфигуратора

В появившемся меню щёлкните по строке Текст. В нижней части окна появится новая командная панель (рисунок 3.101).

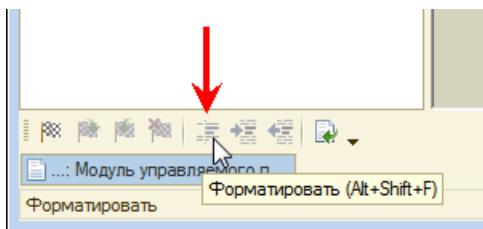


Рисунок 3.101. Командная панель «Текст» и команда «Форматировать»

В этой панели находится команда Форматировать, которая вам и нужна.

Пользоваться этой командой очень просто. Сначала вы выделяете текст, который нужно отформатировать. Для этого, например, нажимаете мышью на серой полосе в первой строке текста (рисунок 3.102).

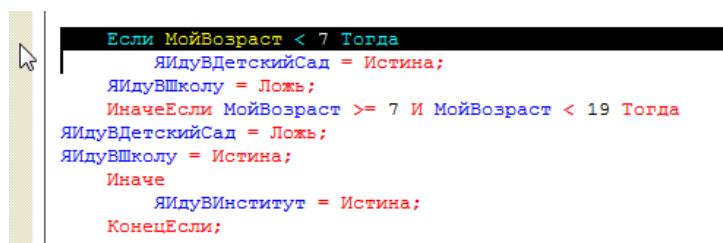


Рисунок 3.102. Начало выделения текста

Затем, не отпуская кнопку мыши, ведёте её до последней строки, которая вам нужна (рисунок 3.103).

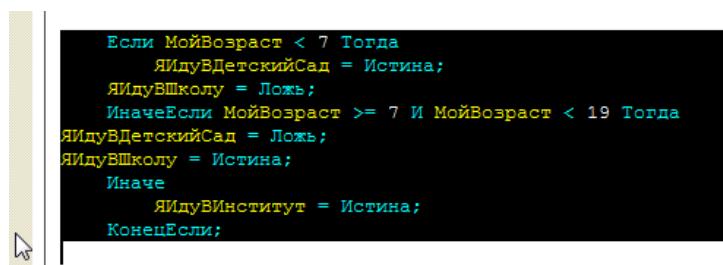


Рисунок 3.103. Выделен текст, который нужно форматировать

И нажимаете кнопку *Форматировать*. Текст сразу становится красивым и отформатированным (рисунок 3.104).

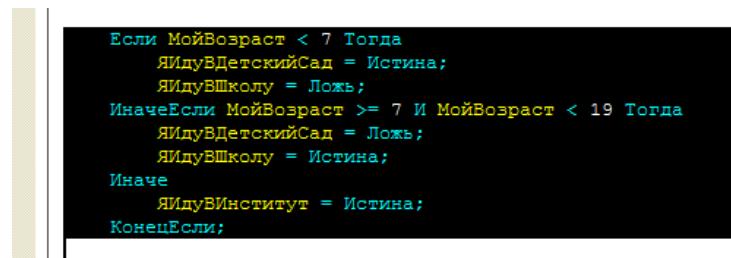


Рисунок 3.104. Отформатированный текст

Подробнее

Подробнее вы можете прочитать про форматирование текста программы в документации «[Руководство разработчика 8.3. Раздел 27.2.1.4.1. Форматирование синтаксических конструкций](#)».

Помимо синтаксического отступа есть и другие способы сделать текст более понятным. Например, один из хороших способов — это добавление пустых строк в текст программы. Они позволяют разделить текст на несколько отдельных смысловых блоков.

Если взять пример с инструкцией *Если*, то такие смысловые блоки напрашиваются сами собой. Каждая ветка этой инструкции является отдельным смысловым блоком. Поэтому, если перед началом ветки вы добавите пустую строку, это только улучшит читаемость вашей программы (листинг 3.28).

Листинг 3.28. Использование пустых строк

```

Если МойВозраст < 7 Тогда
    ЯИдуВДетскийСад = Истина;
    ЯИдуВШколу = Ложь;

ИначеЕсли МойВозраст >= 7 И МойВозраст < 19 Тогда
    ЯИдуВДетскийСад = Ложь;
    ЯИдуВШколу = Истина;

Иначе
    ЯИдуВИнститут = Истина;

КонецЕсли;

```

Другой хороший и очень часто используемый приём — создание *комментариев*. Комментарий — это пояснение к программе, которое находится прямо в тексте программы (листинг 3.29).

Листинг 3.29. Комментарий

```

МойВозраст = 25;

// Присвоить переменным начальные значения.
ЯИдуВДетскийСад = Ложь;
ЯИдуВШколу      = Ложь;
ЯИдуВИнститут   = Ложь;

Если МойВозраст < 7 Тогда
    ЯИдуВДетскийСад = Истина;

```

Комментарии всегда начинаются с пары косых черт — `//`. Платформа выделяет комментарии зелёным цветом.

Комментарии, конечно, могут быть разными. Сформулировать правило: «Комментарии должны быть вот такими...» — невозможно. Ситуации бывают разные.

Но в большинстве случаев нужно стремиться к тому, чтобы описать не **что** делается, а чтобы пояснить, **зачем** это делается.

Главная сложность тут заключается в том, что в тот момент, когда вы пишете программу, вы прекрасно понимаете, зачем вы делаете то или это. И вам кажется, что писать по этому поводу комментарии совершенно не нужно. Ведь и так всё понятно!

Но проходит неделя, месяц, год... Вы открываете свою старую программу и вдруг с удивлением обнаруживаете, что в ней вообще ничего не понятно. Иногда даже бывает такое ощущение, что всё это писали не вы, а кто-то другой.

Поэтому всегда нужно самого себя немного «заставлять» писать комментарии. В этот момент можно представить, что вы пишете программу не «для себя», а для другого программиста. Которому нужно будет разобраться с ней без вашей помощи.

Если вы себе это представите, то сразу поймёте, например, что такой комментарий будет бесполезен (листинг 3.30).

Листинг 3.30. Бесполезный комментарий

```
// Присвоить переменным Ложь.  
ЯИдуВДетскийСад = Ложь;  
ЯИдуВШколу      = Ложь;  
ЯИдуВИнститут  = Ложь;
```

А если вы напишете комментарий так, то он, наоборот, очень поможет и вам, и другим (листинг 3.31).

Листинг 3.31. Полезный комментарий

```
// Присвоить переменным начальные значения,  
// чтобы потом установить в Истина только ту,  
// которая соответствует возрасту.  
ЯИдуВДетскийСад = Ложь;  
ЯИдуВШколу      = Ложь;  
ЯИдуВИнститут  = Ложь;
```

```
Если МойВозраст < 7 Тогда  
    ЯИдуВДетскийСад = Истина;
```

Умение писать нужные и полезные комментарии можно приобрести только с опытом. Поэтому пробуйте, пишите, не ленитесь.

Чтобы комментарии выглядели понятно и красиво, придерживайтесь нескольких простых правил.

Большие комментарии и комментарии к блоку инструкций пишите с начала строки. Начинайте их с большой буквы. Между `//` и комментарием оставляйте пробел. Предложение завершайте точкой. Перед комментарием вставляйте пустую строку.

Например, так (листинг 3.32).

Листинг 3.32. Однострочный комментарий

```
МойВозраст = 25;  
  
// Присвоить переменным начальные значения.  
ЯИдуВДетскийСад = Ложь;  
ЯИдуВШколу      = Ложь;  
ЯИдуВИнститут   = Ложь;
```

Или так (листинг 3.33).

Листинг 3.33. Многострочный комментарий

```
МойВозраст = 25;  
  
// Присвоить переменным начальные значения,  
// чтобы потом установить в Истина только ту,  
// которая соответствует возрасту.  
ЯИдуВДетскийСад = Ложь;  
ЯИдуВШколу      = Ложь;  
ЯИдуВИнститут   = Ложь;
```

Комментарии к отдельным строкам программы пишите в этих же строках. С маленькой буквы и без точки в конце (листинг 3.34).

Листинг 3.34. Небольшие комментарии в строках

```
Если МойВозраст < 7 Тогда    // ясли не учитываем  
    ЯИдуВДетскийСад = Истина;  
  
ИначеЕсли МойВозраст >= 7 И МойВозраст < 19 Тогда  
    ЯИдуВШколу = Истина;  
  
Иначе    // считаем, что в институт можно пойти в любом возрасте  
    ЯИдуВИнститут = Истина;  
  
КонецЕсли;
```

Подробнее

Подробнее вы можете прочитать про комментарии в документации «[Руководство разработчика 8.3. Раздел 4.2.4.1. Комментарии](#)».

3.9.17 Инструкция Цикл

Представьте себе такую задачу. Вам нужно напечатать список. В этом списке должно быть указано, сколько занятий у вас в понедельник, во вторник и так далее:

- в 1-й день — 6 занятий;
- во 2-й день — 8 занятий;
- в 3-й день — 7 занятий и т. д.

Чтобы это сделать, у вас есть две переменные. Одна — *ДеньНедели*. В ней вы можете хранить порядковый номер дня недели. Другая — *Занятий*. В ней вы можете хранить количество занятий.

Печатать вы пока не умеете. Поэтому упростим задачу. Вместо того чтобы напечатать, вам будет достаточно выполнить такую инструкцию:

Листинг 3.35. Пример инструкции

```
Сказать = "В " + Стока(ДеньНедели) + "-й день " + Стока(Занятий) + " занятий.";
```

То есть в переменную *Сказать* вы просто поместите ту строку, которую надо было бы напечатать.

Эта строка «склеивается» из нескольких частей, из нескольких строк. В ней используются переменные *ДеньНедели* и *Занятий*. Но поскольку эти переменные содержат числа, то в «голом» виде их в этой инструкции использовать нехорошо. Сначала нужно сделать из них строки. Для этого используется два раза функция встроенного языка *Стока()*. То значение, которое указано у неё в скобках, она преобразует к значению типа *Строка*.

Но это я отвлёкся от задачи. А она у вас такая. Есть две переменные, и есть инструкция. Нужно с их помощью «напечатать» список.

Если бы вы знали только инструкцию присваивания, вы бы стали делать эту задачу так (листинг 3.36).

Листинг 3.36. Последовательный способ решения задачи

```
// По-порядку "печатаем" для каждого дня недели.
ДеньНедели = 1;
Занятий = 6;
Сказать = "В " + Стока(ДеньНедели) + "-й день " + Стока(Занятий) + " занятий.";

ДеньНедели = 2;
Занятий = 8;
Сказать = "В " + Стока(ДеньНедели) + "-й день " + Стока(Занятий) + " занятий.";

ДеньНедели = 3;
Занятий = 7;
Сказать = "В " + Стока(ДеньНедели) + "-й день " + Стока(Занятий) + " занятий.";
```

Не поленитесь, сделайте в своей конфигурации этот пример. Чтобы три раза не писать одно и то же, напишите только для первого дня. А затем скопируйте эти строки два раза. В скопированных строках измените день недели и количество занятий.

В режиме отладки пройдите по шагам и посмотрите, какие строки получаются в переменной *Сказать* (рисунок 3.105).

Сказать	"В 1-й день 6 занятий."	Строка
Сказать	"В 2-й день 8 занятий."	Строка
Сказать	"В 3-й день 7 занятий."	Строка

Рисунок 3.105. Результат

Почему вы не стали доделывать этот пример до конца? Потому что каждый день делается одно и то же. Зная порядковый день недели, вы выясняете, сколько занятий в этот день. А для этого очень хорошо подходит инструкция *Если* (листинг 3.37).

Листинг 3.37. Пример использования инструкции «Если»

```
Если ДеньНедели = 1 Тогда
    Занятий = 6;

ИначеЕсли ДеньНедели = 2 Тогда
    Занятий = 8;

// и так далее
```

Расписание у вас таково, что в понедельник, четверг и пятницу у вас 6 занятий, во вторник 8, в среду 7, а в выходные занятий нет. Напишите это с помощью инструкции *Если*.

У вас получится такой пример (листинг 3.38).

Листинг 3.38. Инструкция «Если»

```
Если ДеньНедели = 1 ИЛИ ДеньНедели = 4 ИЛИ ДеньНедели = 5 Тогда
    Занятий = 6;

ИначеЕсли ДеньНедели = 2 Тогда
    Занятий = 8;

ИначеЕсли ДеньНедели = 3 Тогда
    Занятий = 7;

ИначеЕсли ДеньНедели = 6 ИЛИ ДеньНедели = 7 Тогда
    Занятий = 0;

КонецЕсли;
```

Если теперь после инструкции *Если* добавить вашу «печать», то получится именно то, что нужно сделать для каждого дня (листинг 3.39). Вы узнаёте количество занятий и «печатаете».

Листинг 3.39. Инструкция «Если» и «печать»

```
Если ДеньНедели = 1 ИЛИ ДеньНедели = 4 ИЛИ ДеньНедели = 5 Тогда
    Занятий = 6;

ИначеЕсли ДеньНедели = 2 Тогда
    Занятий = 8;

ИначеЕсли ДеньНедели = 3 Тогда
    Занятий = 7;

ИначеЕсли ДеньНедели = 6 ИЛИ ДеньНедели = 7 Тогда
    Занятий = 0;

КонецЕсли;
```

```
Сказать = "В " + Стока(ДеньНедели) + "-й день " + Стока(Занятий) + " занятий.";
```

Весь вопрос теперь в том, как выполнить эти строки программы для каждого дня недели. А для этого как раз и нужна инструкция *Цикл*. Она «снаружи» обрамляет тот фрагмент, который нужно выполнить несколько раз, и выглядит следующим образом (листинг 3.40). Сделайте такой же пример в своей конфигурации.

Листинг 3.40. Инструкция «Цикл»

```
Для ДеньНедели = 1 По 7 Цикл
```

```
Если ДеньНедели = 1 ИЛИ ДеньНедели = 4 ИЛИ ДеньНедели = 5 Тогда  
    Занятий = 6;
```

```
Иначе Если ДеньНедели = 2 Тогда  
    Занятий = 8;
```

```
Иначе Если ДеньНедели = 3 Тогда  
    Занятий = 7;
```

```
Иначе Если ДеньНедели = 6 ИЛИ ДеньНедели = 7 Тогда  
    Занятий = 0;
```

```
КонецЕсли;
```

```
Сказать = "В " + Стока(ДеньНедели) + "-й день " + Стока(Занятий) + " занятий.";
```

```
КонецЦикла;
```

Работает она следующим образом. Когда исполнение программы доходит до цикла, переменной цикла присваивается начальное значение. В вашем случае переменная цикла — *ДеньНедели*, а начальное значение для неё — 1.

Затем начинают выполняться те инструкции, которые расположены внутри цикла.

Когда исполнение доходит до строки *КонецЦикла*, переменная цикла автоматически увеличивается на единицу и всё возвращается в начало цикла.

Если переменная цикла не превысила своё конечное значение (в вашем случае 7), инструкции внутри цикла выполняются ещё раз. Если переменная цикла превысила своё конечное значение, исполнение переходит дальше, к тем строкам, которые расположены после инструкции *КонецЦикла*.

Запустите пример в режиме отладки и, двигаясь по шагам, посмотрите, как это работает. День недели будет последовательно изменяться от 1 до 7. И в переменной *Сказать* у вас будут получаться строки с количеством занятий в каждый из дней.

Это не единственная форма инструкции *Цикл*. Есть ещё две другие формы этой инструкции. С ними вы тоже познакомитесь, но позже.

Задание 3.27

Выведите количество дней в каждом месяце из второй декады в виде «4-й месяц 30 дней». Используйте переменную *НомерМесяца*. Результат помещайте в переменную *Сказать*. С помощью пошагового выполнения проверьте все значения, которые окажутся в переменной *Сказать*.

Задание 3.28

Перечислите школьные оценки от самой лучшей к самой худшей. Результат помещайте в переменную *Сказать*.

3.9.18 Функции

Продолжим исследовать и улучшать ваш пример с количеством занятий в каждый из дней. Посмотрите на ту его часть, которая, зная день недели, узнаёт вам количество

занятий (листинг 3.41).

Листинг 3.41. Часть примера, которая определяет день недели

Для ДеньНедели = 1 По 7 Цикл

```

Если ДеньНедели = 1 ИЛИ ДеньНедели = 4 ИЛИ ДеньНедели = 5 Тогда
    Занятий = 6;

ИначеЕсли ДеньНедели = 2 Тогда
    Занятий = 8;

ИначеЕсли ДеньНедели = 3 Тогда
    Занятий = 7;

ИначеЕсли ДеньНедели = 6 ИЛИ ДеньНедели = 7 Тогда
    Занятий = 0;

КонецЕсли;

Сказать = "В " + Стока(ДеньНедели) + "-й день " + Стока(Занятий) + " занятий./";

КонецЦикла;

```

В вашем примере она выглядит довольно просто, но в реальной программе это может быть большой и сложный алгоритм. Он по сложным правилам, в зависимости от многих условий решает, сколько будет занятий.

Представьте себе, что этот алгоритм вам нужно использовать не только здесь, где он написан, но и в каком-то другом месте вашей конфигурации. Только там, например, вам нужно просто вычислить, сколько занятий будет в какой-то один определённый день. Который вы заранее не знаете.

Что делать? В том, другом, месте заново ещё раз писать весь этот алгоритм? Это нехорошо.

А если таких мест не одно, а несколько? А если вы потом захотите изменить этот алгоритм? Придётся изменять его во всех местах, где он написан? Тогда легко допустить ошибку и что-то пропустить.

Но выход есть! Нужно просто оформить этот алгоритм в виде *функции*. Тогда он будет написан у вас в одном единственном месте, а использовать его вы сможете в разных местах своей конфигурации.

Функция — это не инструкция. Это просто специальная форма записи нескольких инструкций вместе.

Чтобы не было скучно, воспользуйтесь синтакс-помощником. Откройте в нём ветку *Общее описание встроенного языка — Операторы*. Нажмите мышью на *Функция* и перетащите её в свой модуль парой строк ниже строки *КонецПроцедуры* (рисунок 3.106).

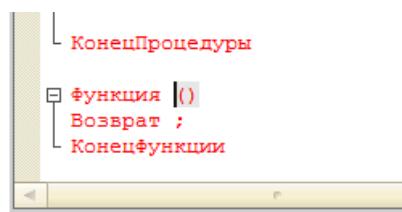


Рисунок 3.106. Заготовка функции

Платформа вставит в ваш модуль заготовку функции.

Теперь после строки *Функция ()* добавьте пустую строку и перетащите в неё всю инструкцию *Если*, которая выделена в листинге 3.41. Используйте форматирование, чтобы платформа сама расставила синтаксические отступы. У вас получится такая конструкция (листинг 3.42).

Листинг 3.42. Тело функции

```
Функция ()  
    Если ДеньНедели = 1 ИЛИ ДеньНедели = 4 ИЛИ ДеньНедели = 5 Тогда  
        Занятий = 6;  
  
    ИначеЕсли ДеньНедели = 2 Тогда  
        Занятий = 8;  
  
    ИначеЕсли ДеньНедели = 3 Тогда  
        Занятий = 7;  
  
    ИначеЕсли ДеньНедели = 6 ИЛИ ДеньНедели = 7 Тогда  
        Занятий = 0;  
  
    КонецЕсли;  
  
    Возврат ;  
КонецФункции
```

То, что находится между строкой *Функция ()* и строкой *КонецФункции*, называется *тело функции*. Это те инструкции, которые будут выполнены при вызове функции.

Первая строка — *Функция ()* — это *определение функции*. Им вы сейчас и займётесь. Потому что пока у вас есть только заготовка.

Определение функции всегда начинается с ключевого слова *Функция*. За ним, через пробел, нужно указать имя этой функции. Имя вы придумываете сами. Оно нужно для того, чтобы вызвать эту функцию из другого места. Назовите функцию *СколькоЗанятий* (листинг 3.43).

Листинг 3.43. Имя функции — «СколькоЗанятий»

```
Функция СколькоЗанятий ()  
    Если ДеньНедели = 1 ИЛИ ДеньНедели = 4 ИЛИ ДеньНедели = 5 Тогда  
        Занятий = 6;
```

После имени, без пробела, всегда следуют круглые скобки — *()*. В них может быть что-то написано, а может и не быть. Это зависит от того, нужны вашей функции специальные, особенные данные для работы или нет.

В вашем случае функция должна сказать, сколько занятий в тот день недели, который вас интересует. Поэтому такие специальные данные ей нужны. Ей нужно сказать, какой именно день недели вас интересует.

Посмотрите на тело функции. Функция ожидает, что этот день недели будет находиться в переменной *ДеньНедели*. То есть значение, которое будет в этой переменной, функция должна получить откуда-то «снаружи».

Чтобы платформа это поняла, напишите имя переменной *ДеньНедели* внутри круглых скобок (листинг 3.44).

Листинг 3.44. Параметр функции

```
Функция СколькоЗанятий( ДеньНедели )
    Если ДеньНедели = 1 ИЛИ ДеньНедели = 4 ИЛИ ДеньНедели = 5 Тогда
        Занятий = 6;
```

Теперь платформа знает, что значение для переменной *ДеньНедели* будет получено в тот момент, когда вы вызовете эту функцию. Если это сейчас непонятно, не расстраивайтесь. Чуть позже вы увидите, как это происходит.

Такие переменные, написанные в определении в скобках после имени функции, называются *формальными параметрами* функции.

В данном случае слово «формальный» нужно понимать в значении «ожидаемый», «планируемый». Например, формально вы должны просыпаться в 7 часов утра, чтобы успеть в школу или на работу. Но фактически вы можете проспать, или, наоборот, проснуться раньше. Формально спектакль должен начаться в 7 часов вечера. Но фактически он может начаться позже, потому что не все зрители успели зайти в зал.

То есть формальные параметры — это то, что функция ожидает получить в тот момент, когда вы её вызовете.

У вашей функции всего один формальный параметр. Другую переменную, *Занятий*, она создаст сама в процессе работы. Поэтому описание функции готово.

После описания функции принято вставлять пустую строку. Так текст программы читается лучше. Вставьте её и вы (листинг 3.45).

Листинг 3.45. Пустая строка после описания функции

```
Функция СколькоЗанятий(ДеньНедели)
```

```
    Если ДеньНедели = 1 ИЛИ ДеньНедели = 4 ИЛИ ДеньНедели = 5 Тогда
        Занятий = 6;
```

Теперь переместитесь в конец функции. Там есть строка *Возврат ;*. *Возврат* — это специальное ключевое слово. С его помощью вы можете указать, что именно должна вернуть функция в то место, откуда её вызвали.

В вашем случае функция должна вернуть количество занятий. Оно получится в переменной *Занятий*. Поэтому после слова *Возврат* через пробел укажите имя этой переменной (листинг 3.46).

Листинг 3.46. Возвращаемое значение

```
КонецЕсли;
```

```
    Возврат Занятий ;
```

```
КонецФункции
```

Последняя строка — это *КонецФункции*. Перед ней тоже принято вставлять пустую строку, чтобы программа читалась лучше. Вставьте её и вы (листинг 3.47).

Листинг 3.47. Пустая строка перед концом функции

```
КонецЕсли;
```

```
    Возврат Занятий ;
```

```
КонецФункции
```

Обратите внимание, что после слова КонецФункции нет точки с запятой. Это ещё раз говорит о том, что функция не является инструкцией, которая будет выполняться. Функция — это просто специальная форма записи нескольких инструкций вместе.

В стандартных настройках конфигуратора указано, что нужно группировать и сворачивать функции. Поэтому платформа автоматически дорисовала слева от вашей функции вертикальную линию со знаком «-» наверху (рисунок 3.107).

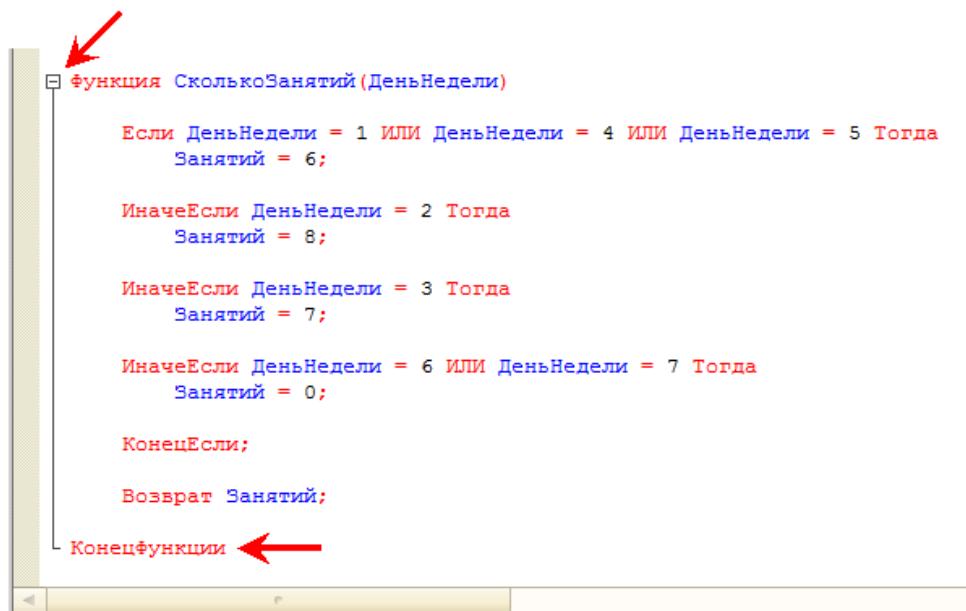


Рисунок 3.107. Группировка и сворачивание функций

Если вы нажмёте мышью на «-», текст процедуры свернётся, останется только её определение. А «-» изменится на «+» (рисунок 3.108).

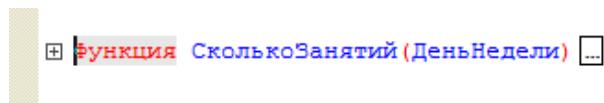


Рисунок 3.108. Свёрнутая функция

Это удобно в тех случаях, когда в одном модуле находится много функций. Тогда их можно быстро просмотреть и открыть ту, которая нужна.

Итак, теперь ваша функция полностью готова. Если хотите, можете раскрыть её обратно. Если не хотите, можете не раскрывать.

Теперь вам нужно вызвать эту функцию в том месте, в котором нужно. А в котором нужно? Наверное, вы уже догадались. В том самом месте, откуда вы «утащили» инструкцию *Если*.

Чтобы вызвать функцию, нужно написать её имя, а в скобках — выражение или переменную. Значение этого выражения или переменной будет передано в функцию.

А поскольку функция вернёт вам количество занятий, то вызов функции нужно написать в инструкции присваивания. И присвоить это значение той переменной, которая вам нужна. То есть переменной *Занятий* (листинг 3.48).

Листинг 3.48. Вызов функции

```

Для ДеньНедели = 1 По 7 Цикл
    Занятий = СколькоЗанятий(ДеньНедели);

Сказать = "В " + Стока(ДеньНедели) + "-й день " + Стока(Занятий) + " занятий.;

КонецЦикла;

```

Как будет работать эта инструкция? Сначала, как вы знаете, платформа вычислит выражение, которое находится справа от знака равенства. То есть она вызовет функцию `СколькоЗанятий()`. В эту функцию она передаст значение переменной `ДеньНедели`.

В этом месте — там, где функция вызывается, — переменные, указанные в скобках, называются *фактическими параметрами*. То есть это то, что на самом деле будет передано в функцию.

Результатом вычисления выражения справа от знака равенства будет значение, которое вернёт ваша функция. И это значение будет присвоено переменной `Занятий`. После чего вы используете эту переменную, чтобы сформировать «печатаемую» строку.

Чтобы убедиться, что вы не допустили ошибки, посмотрите на рисунок 3.109. Ваш модуль должен выглядеть так. Заодно ещё раз обратите внимание на то, что называется фактическим параметром, а что — формальным.

```

Дневник: Модуль управляемого приложения

Процедура ПриНачалеРаботыСистемы()
    УстановитьКраткийЗаголовокПриложения("Иванов Петя");

    Для ДеньНедели = 1 По 7 Цикл
        Занятий = СколькоЗанятий(ДеньНедели); Фактический параметр

        Сказать = "В " + Стока(ДеньНедели) + "-й день " + Стока(Занятий) + " занятий.;

    КонецЦикла;

КонецПроцедуры

Функция СколькоЗанятий(ДеньНедели)
    Если ДеньНедели = 1 ИЛИ ДеньНедели = 4 ИЛИ ДеньНедели = 5 Тогда
        Занятий = 6;

    ИначеЕсли ДеньНедели = 2 Тогда
        Занятий = 8;

    ИначеЕсли ДеньНедели = 3 Тогда
        Занятий = 7;

    ИначеЕсли ДеньНедели = 6 ИЛИ ДеньНедели = 7 Тогда
        Занятий = 0;

    КонецЕсли;

    Возврат Занятий;

КонецФункции

```

Рисунок 3.109. Модуль с процедурой

Теперь самое интересное! Установите точку останова в начале цикла и запустите 1С: Предприятие в режиме отладки. Посмотрите, как это работает.

Сделайте один шаг. Платформа остановится на инструкции присваивания. Она приготовится её исполнять (рисунок 3.110).

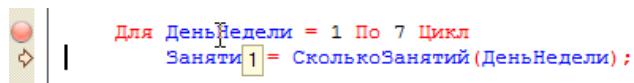


Рисунок 3.110. Перед вызовом функции

Это первый «проход» цикла, поэтому день недели равен 1.

Шагните ещё раз. Платформа «провалился» в функцию (рисунок 3.111).

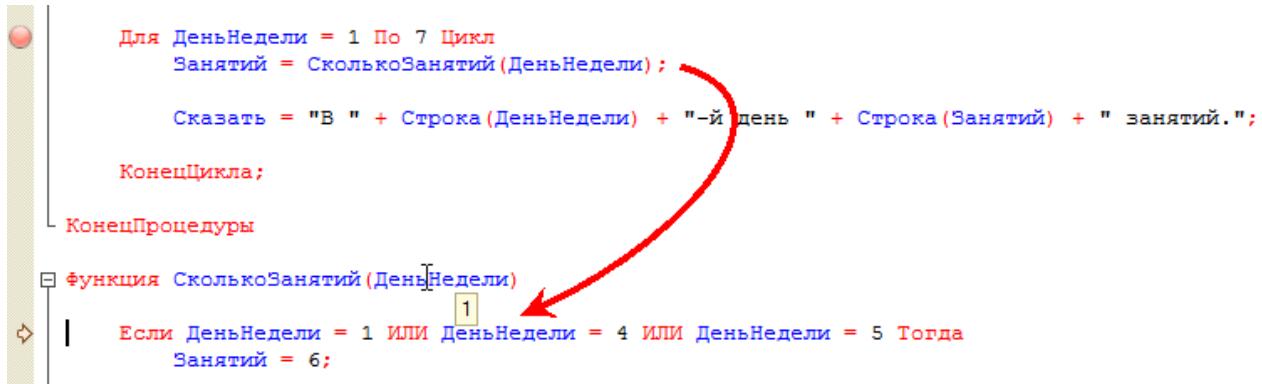


Рисунок 3.111. Исполнение «провалилось» в функцию

Обратите внимание, что платформа передала сюда значение дня недели 1. И теперь есть все необходимые данные для того, чтобы выполнить инструкцию *Если*.

Сделайте ещё три шага. Платформа остановится перед выполнением инструкции *Возврат* (рисунок 3.112).

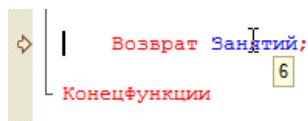


Рисунок 3.112. Инструкция «Возврат»

Вы можете посмотреть, какое значение функция собирается возвращать. Это значение 6.

Сделав ещё шаг, вы перейдёте на строку *КонецФункции*. Это значит, что исполнение функции закончилось.

А шагнув ещё раз, вы вернётесь обратно к инструкции присваивания (рисунок 3.113).

Для ДеньНедели = 1 По 7 Цикл
 Занятий = СколькоЗанятий(ДеньНедели);
 Сказать = "В " + Стока(ДеньНедели) + "-й день " + Стока(Занятий) + " занятий."
 КонецЦикла;
 КонецПроцедуры

Функция СколькоЗанятий(ДеньНедели)
 Если ДеньНедели = 1 ИЛИ ДеньНедели = 4 ИЛИ ДеньНедели = 5 Тогда
 Занятий = 6;
 ИначеЕсли ДеньНедели = 2 Тогда
 Занятий = 8;
 ИначеЕсли ДеньНедели = 3 Тогда
 Занятий = 7;
 ИначеЕсли ДеньНедели = 6 ИЛИ ДеньНедели = 7 Тогда
 Занятий = 0;
 КонецЕсли;
 Возврат Занятий;
 Конецфункции

A red arrow points from the assignment statement `Занятий = 6;` in the `СколькоЗанятий` function down to the variable `Занятий` in the `Сказать` string of the main loop.

Рисунок 3.113. Возврат в место вызова функции

Сейчас у вас картинка очень похожа на то, что было на рисунке 3.110. Платформа подготовилась выполнить инструкцию присваивания. Но если раньше платформа собиралась вычислить выражение, находящееся справа, то теперь она его уже вычислила. И сейчас она готова поместить получившееся значение в переменную.

Сделайте один шаг. Исполнение перейдёт к следующей строке. А в переменной `Занятий` появится значение 6. Это то значение, которое вернула функция (рисунок 3.114).

Для ДеньНедели = 1 По 7 Цикл
 Занятий = СколькоЗанятий(ДеньНедели);
 Сказать = "В " + Стока(ДеньНедели) + "-й день " + Стока(Занятий) + " занятий."
 6

The value 6 is highlighted in a yellow box above the assignment statement `Занятий = 6;`

Рисунок 3.114. Инструкция присваивания выполнена

И если вы шагнёте ещё раз, то сформируется строка, предназначенная «для печати» (рисунок 3.115).

Для ДеньНедели = 1 По 7 Цикл
 Занятий = СколькоЗанятий(ДеньНедели);
 Сказать = "В " + Стока(ДеньНедели) + "-й день " + Стока(Занятий) + " занятий."
 "В 1-й день 6 занятий."
 КонецЦикла;

The resulting string `"В 1-й день 6 занятий."` is highlighted in a yellow box.

Рисунок 3.115. Сформирован текст «для печати»

После этого исполнение цикла продолжится, но день недели будет равен 2. И так далее. Можете пройти несколько итераций цикла и посмотреть ещё раз, как работает этот пример.

Итак, чтобы закрепить свои знания, посмотрите на рисунке 3.116. На нём стрелками показана последовательность исполнения встроенного языка при вызове функции.



Рисунок 3.116. Последовательность исполнения при вызове функции

В тот момент, когда платформа встречает вызов функции, исполнение «проваливается» в функцию (2). После того как функция выполнит инструкции, написанные в её теле, исполнение возвращается в ту точку, из которой была вызвана функция. И только после этого исполнение встроенного языка переходит к следующей инструкции (3).

3.9.19 Контекст и область видимости

Вы, наверное, обратили внимание на то, что в вашем примере имена фактического и формального параметра совпали — *ДеньНедели* (рисунок 3.117).

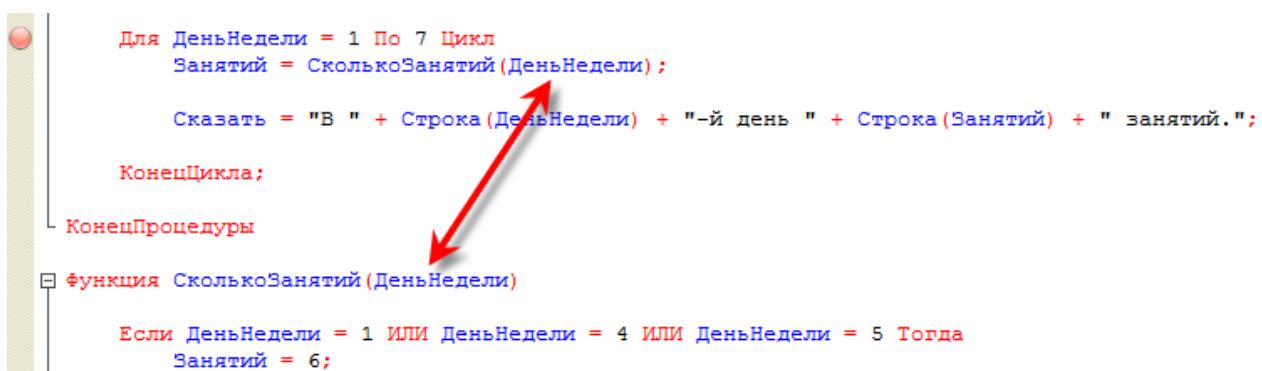


Рисунок 3.117. Имена фактического и формального параметров

У вас это получилось случайно. Просто потому, что вы перетаскивали фрагмент программы из одного места в другое.

На самом деле имена формальных и фактических параметров не совпадают. В этом нет необходимости.

Чтобы разобраться с этим, упростите пример. В процедуре создайте переменную и напишите вызов функции. А функцию тоже упростите, чтобы она возвращала только два значения: 5 или 6. И обратите внимание, что в этом примере специально изменены имена переменных так, чтобы они были разными внутри функции и снаружи неё (рисунок 3.118).

```

НомерДняНедели = 1;
Занятий = СколькоЗанятий(НомерДняНедели);

КонецПроцедуры

Функция СколькоЗанятий(ДеньНедели)
|
|   Если ДеньНедели = 1 Тогда
|       КоличествоЗанятий = 5;
|
|   Иначе
|       КоличествоЗанятий = 6;
|
|   КонецЕсли;
|
|   Возврат КоличествоЗанятий;
|
Конецфункции

```

Рисунок 3.118. Упрощённый пример

Установите точку останова на первой инструкции присваивания, запустите 1С:Предприятие в режиме отладки и откройте локальные переменные. Они вам сейчас очень помогут разобраться с тем, что будет происходить (рисунок 3.119).

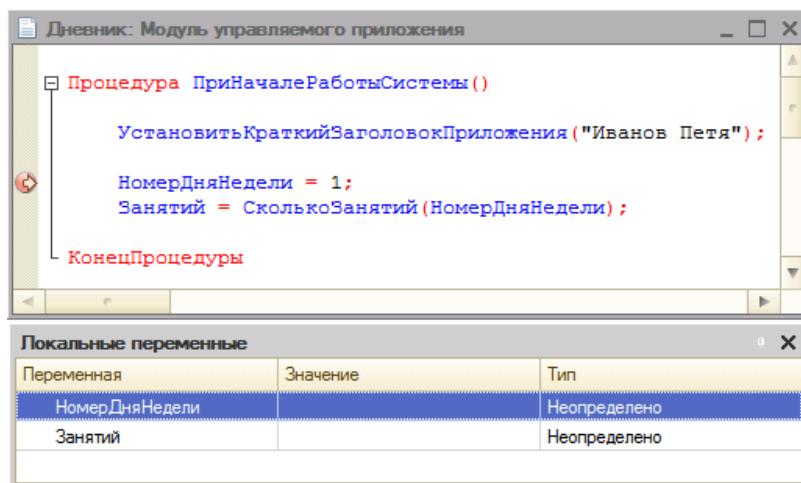


Рисунок 3.119. Локальный контекст процедуры

Сейчас вы находитесь в процедуре *ПриНачалеРаботыСистемы()*. Вы пока ещё не знакомились с тем, что такое процедуры. Но это не страшно. Для этого примера будет достаточно представить, что процедура — это то же самое, что и функция.

Так вот. Вы находитесь внутри процедуры. И здесь вам доступны переменные *НомерДняНедели* и *Занятий*. Они показаны в окне локальных переменных. Это те переменные, которые создаются и определяются в этой процедуре.

Теперь сделайте пару шагов, чтобы перейти в функцию (рисунок 3.120).

The screenshot shows the 'Diary' application window with the title 'Дневник: Модуль управляемого приложения'. The main pane displays the following VBA-like pseudocode:

```
Процедура ПриНачалеРаботыСистемы()
    УстановитьКраткийЗаголовокПриложения("Иванов Петя");
    НомерДняНедели = 1;
    Занятий = СколькоЗанятий(НомерДняНедели);
    КонецПроцедуры

Функция СколькоЗанятий(ДеньНедели)
    Если ДеньНедели = 1 Тогда
        КоличествоЗанятий = 5;
    Иначе
        КоличествоЗанятий = 6;
    КонецЕсли;
    Возврат КоличествоЗанятий;
КонецФункции
```

Below the code, a 'Локальные переменные' (Local Variables) window is open, showing the following table:

Переменная	Значение	Тип
ДеньНедели	1	Число
КоличествоЗанятий		Неопределено

Рисунок 3.120. Локальный контекст функции

Обратите внимание на то, что вы видите в окне локальных переменных. *НомерДняНедели* и *Занятий* исчезли. Зато вместо них появились *ДеньНедели* и *КоличествоЗанятий*, которые определены внутри этой функции.

Теперь, если вы пройдёте всю функцию и вернётесь обратно в процедуру, состав локальных переменных снова изменится (рисунок 3.121).

The screenshot shows the 1C Development Environment. The main window displays a procedure named 'ПриНачалеРаботыСистемы()' which sets a short application title and initializes variables. It then calls a function 'СколькоЗанятий(ДеньНедели)'. This function uses an if-statement to set the number of lessons based on the day of the week. A return statement is present, and the function ends. Below the procedure, a table titled 'Локальные переменные' (Local Variables) lists two variables: 'НомерДняНедели' with value 1 and 'Занятий' with value 5.

Переменная	Значение	Тип
НомерДняНедели	1	Число
Занятий	5	Число

Рисунок 3.121. Локальный контекст процедуры

Он станет таким, каким он был в самом начале. На что это похоже?

Это похоже на то, что вы ходите из одной комнаты в другую. Каждая функция (процедура) — как отдельная комната. Например, гостиная и кухня.

Когда вы находитесь в гостиной, вы видите только то, что находится в гостиной. Диван, кресло, журнальный столик (рисунок 3.122). Вы не видите то, что находится на кухне.



Рисунок 3.122. Гостиная

Если вы выйдете из гостиной и зайдёте на кухню, вы увидите то, что находится на кухне. Обеденный стол, стулья, холодильник, духовку (рисунок 3.123).



Рисунок 3.123. Кухня

И в это же время вы перестанете видеть, что находитесь в гостиной.

Для обозначения этой ситуации во встроенном языке используют слово *контекст*. *Контекст* — это то, что доступно вам в том месте программы, в котором вы находитесь в данный момент.

То есть, на языке программистов, когда вы в гостиной, «вы находитесь в контексте гостиной». А если вы на кухне, то «вы находитесь в контексте кухни».

Находясь в контексте гостиной, вы можете сесть в кресло и взять книгу с журнального столика. Когда вы находитесь в контексте кухни, кресло и журнальный столик вам недоступны. Но зато вы можете достать что-нибудь вкусное из холодильника или разогреть обед в духовке.

Если переменная определена в какой-то функции, то она доступна только в ней. И недоступна в других функциях. Поэтому в разных функциях вы можете создавать переменные с разными именами или с одинаковыми именами. В любом случае это будут разные переменные.

Это правило относится и к параметрам функций. Переменные, которые указаны в качестве фактического и формального параметра, могут иметь одинаковые имена. Или разные имена. Это не важно. Потому что каждая из этих переменных существует только в контексте «своей» процедуры или функции.

Если опять вернуться к гостиной и кухне: обратите внимание, что и там, и там есть телевизор. Эти телевизоры разные. Кухонный телевизор не видно из гостиной. А с кухни не видно тот телевизор, который находится в гостиной (рисунок 3.122 и 3.123). Но показывают они одну и ту же программу.

Этим они очень похожи на фактический и формальный параметры. Телевизор в гостиной — это переменная, указанная в качестве фактического параметра. Телевизор на кухне — это переменная, указанная в качестве формального параметра. Когда вы уходите из гостиной и переходите на кухню, вы оказываетесь в другом контексте. Но с помощью формального параметра (кухонный телевизор), вы можете смотреть ту же передачу, что и в гостиной. Но уже в другом контексте.

Заметка

На самом деле понятие контекста во встроенном языке гораздо шире. В этом примере речь идёт о локальном контексте. Локальный контекст — это только одна из частей, входящих в понятие «контекст». Но об этом я расскажу позже.

Теперь познакомьтесь с другим термином, который часто используется при написании программ. Это *область видимости*.

Примечание

Область видимости чего-то — это область конфигурации, откуда это что-то видно и доступно. Если вернуться на бытовой уровень, то область видимости холодильника — это кухня, а область видимости дивана — гостиная.

Про переменные я уже рассказывал, что их область видимости ограничивается той функцией или процедурой, внутри которой они созданы.

Если быть совсем точным, то область видимости переменной начинается с инструкции присваивания, в которой она создается, и заканчивается концом функции или процедуры (рисунок 3.124).

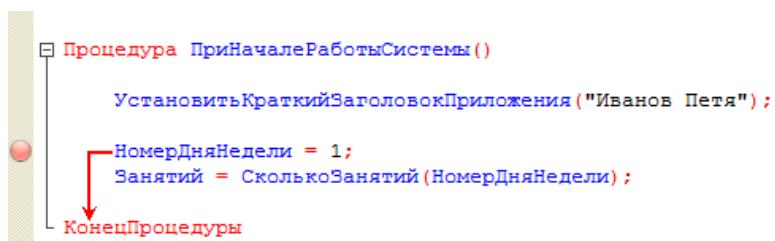


Рисунок 3.124. Область видимости переменной «НомерДняНедели»

Ради развлечения попробуйте использовать переменную *НомерДняНедели* в самом начале процедуры. Например, так, как на рисунке 3.125.

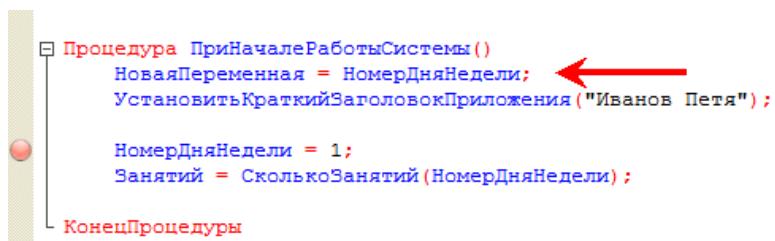


Рисунок 3.125. Попытка использовать переменную до её определения

Если вы попробуете запустить такую программу, то получите вполне ожидаемую ошибку (рисунок 3.126).

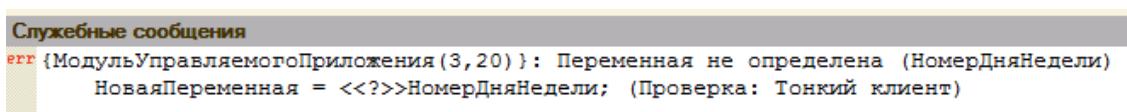


Рисунок 3.126. Сообщение об ошибке

Действительно, в третьей строке переменная *НомерДняНедели* ещё не определена. И платформа о ней ничего не знает. Эта переменная появится только двумя строками ниже.

Заметка

Область видимости переменных может быть и шире. Но эти возможности находятся за рамками этой книги, поэтому о них я говорить не буду.

У процедур и функций тоже есть своя область видимости. Это весь модуль, в котором они определены. Именно поэтому и существует возможность вызывать функцию или процедуру из любого места модуля. Например, как в вашем примере, из другой процедуры.

Если вспомнить, что каждую функцию вы сравнивали с комнатой, то можно сказать тогда, что модуль целиком — это квартира. Когда вы заходите в какую-то комнату (функцию) вы видите мебель (переменные). А на стене висит план всей квартиры. На котором

написано, какие ещё комнаты (процедуры и функции) есть в квартире (модуле) (рисунок 3.127).

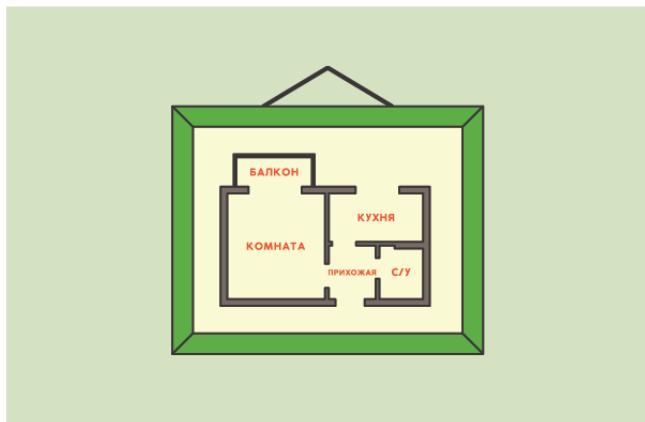


Рисунок 3.127. План квартиры

Заметка

Область видимости процедур и функций может быть и шире. С этим вы познакомитесь в следующих разделах книги.

Задание 3.29

Создайте функцию, которая получает значение типа *Дата*. А возвращает эта функция текстовое представление месяца и года этой даты.

Для формирования текстового представления используйте функцию *ПредставлениеПериода()*. Её описание вы найдёте в синтакс-помощнике в ветке *Глобальный контекст — Функции форматирования*.

Задание 3.30

Возьмите программу из задания 3.19 на странице 230. Преобразуйте её в функцию, которая получает две даты, а возвращает их разность в виде количества часов, минут и секунд.

3.9.20 Процедуры

Теперь, когда вы умеете создавать и использовать функции, можно заняться изучением процедур.

Процедуры очень похожи на функции. Есть только одно отличие. Процедура ничего не возвращает. Она просто выполняет инструкции, которые находятся в её теле.

Во всём остальном процедуры выглядят точно так же, как функции, которые вам уже хорошо известны. Вместо слова *Функция* используется *Процедура*, а вместо *КонецФункции* используется *КонецПроцедуры* (листинг 3.49).

Листинг 3.49. Пример процедуры

Процедура ПриНачалеРаботыСистемы()

УстановитьКраткийЗаголовокПриложения("Иванов Петя");

ПредупредитьОПонедельнике();

КонецПроцедуры

Процедура ПредупредитьОПонедельнике()

Если ДеньНедели(ТекущаяДата()) = 7 Тогда

ПоказатьОповещениеПользователя("Завтра в школу!");

КонецЕсли;

КонецПроцедуры

В этом примере у процедуры нет параметров, но если нужно, вы можете их указать так же, как и в функции.

Напишите в своей конфигурации этот пример. Процедура в этом примере проверяет, какой сейчас день недели. Если номер дня недели равен 7 (воскресенье), она выводит на экран напоминание о том, что завтра нужно идти в школу.

Запустите этот пример и посмотрите, как выглядит это напоминание. Но не забывайте, что оно появится только в воскресенье. Если вы читаете книгу не в воскресенье, то вместо 7 напишите номер вашего текущего дня недели.

Оповещение появится в небольшом окне в правой нижней части экрана, а затем постепенно пропадёт (рисунок 3.128).

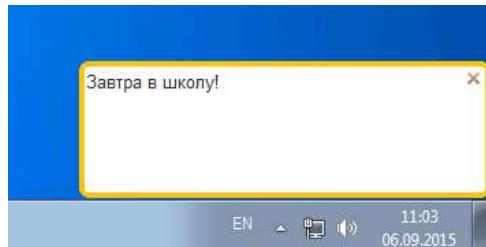


Рисунок 3.128. Оповещение пользователя

Для вывода такого оповещения вы использовали встроенную процедуру *ПоказатьОповещениеПользователя()*. Вы о ней ничего не знаете, просто написали так, как было на рисунке. Но наверняка вы хотите узнать, для чего нужна эта процедура и как ею пользоваться.

В этом вам поможет синтакс-помощник. Причём найти в нём эту процедуру очень просто.

Когда в тексте программы вам попадается незнакомая процедура или функция встроенного языка, вы можете просто установить на неё курсор и нажать сочетание клавиш **Ctrl+F1**. Откроется синтакс-помощник, и в его нижнем окне будет показано описание процедуры или функции (рисунок 3.129).

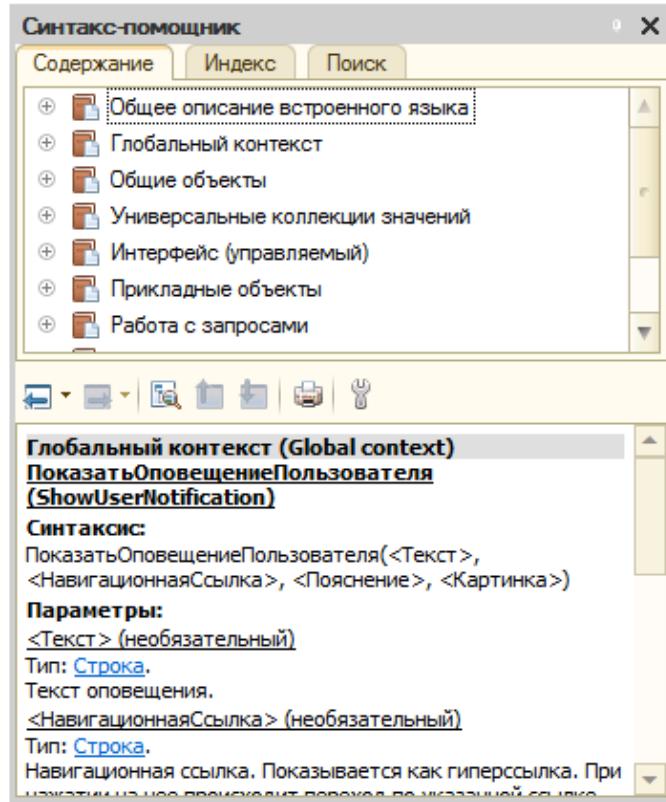


Рисунок 3.129. Описание процедуры в синтакс-помощнике

Если вы захотите узнать, какие ещё есть похожие процедуры и функции, нажмите кнопку *Найти текущий элемент* в дереве на командной панели (рисунок 3.130).

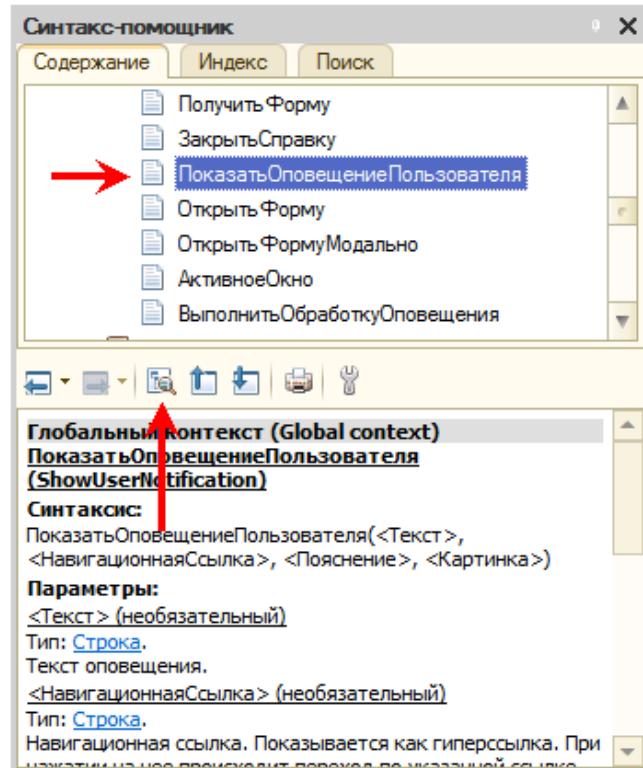


Рисунок 3.130. Кнопка «Найти текущий элемент в дереве»

В верхнем окне синтакс-помощник раскроет дерево и отметит в нём вашу процедуру.

Если вы пролистаете дерево вверх, то увидите, что есть ещё большое количество процедур и функций, предназначенных для интерактивной работы. То есть для «общения» с пользователем (рисунок 3.131).

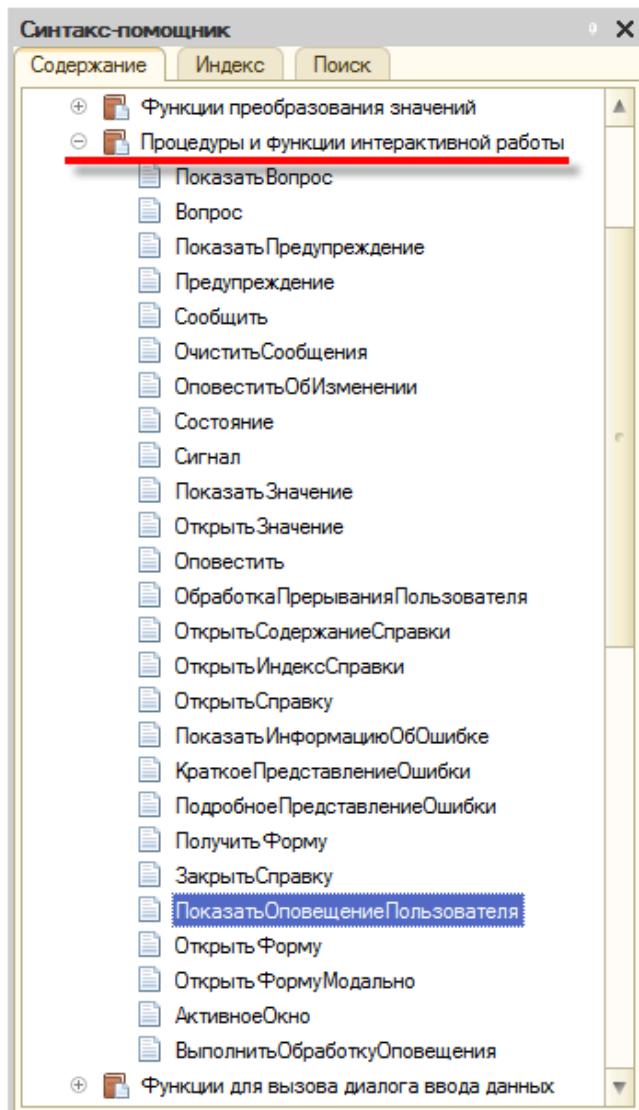


Рисунок 3.131. Процедуры и функции интерактивной работы

Задание 3.31

Пример из задания 3.29 на странице 270 преобразуйте в процедуру. Текстовое представление месяца и года переданной даты показывайте пользователю с помощью функции *ПоказатьОповещениеПользователя()*.

Задание 3.32

Пример из задания 3.30 на странице 270 преобразуйте в процедуру. Представление периода показывайте пользователю с помощью функции *ПоказатьОповещениеПользователя()*.

3.9.21 Чтение и отладка процедур и функций

Процедуры и функции могут быть расположены в самых разных частях конфигурации. Совсем не обязательно, что определение функции будет находиться где-то рядом с тем местом, откуда она вызывается. Это так только в вашем примере.

Чаще всего вызов функции и её определение находятся далеко друг от друга. Может быть, даже в разных модулях конфигурации. Как в этом случае быстро найти определение функции, чтобы посмотреть, какие действия она выполняет?

Для этого есть очень удобный способ. Нужно установить курсор на имя процедуры или функции (в том месте, где она вызывается) и нажать клавишу F12.

Попробуйте на своём примере, в том месте, где вы вызываете функцию `ПредупредитьОПонедельнике()`. Платформа перейдёт к определению процедуры и выделит его в тексте (рисунок 3.132).

```

Дневник: Модуль управляемого приложения

Процедура ПриНачалеРаботыСистемы()
    УстановитьКраткийЗаголовокПриложения("Иванов Петя");
    ПредупредитьОПонедельнике();
    КонецПроцедуры

Процедура ПредупредитьОПонедельнике()
    Если ДеньНедели(ТекущаяДата()) = 7 Тогда
        ПоказатьОповещениеПользователя("Завтра в школу!");
    КонецЕсли;
    КонецПроцедуры

```

Рисунок 3.132. Переход к определению процедуры

Чтобы вернуться назад, нажмите сочетание клавиш `Ctrl+-` в основной части клавиатуры. Курсор вернётся на ту строку, где находится вызов функции (рисунок 3.133).

```

Дневник: Модуль управляемого приложения

Процедура ПриНачалеРаботыСистемы()
    УстановитьКраткийЗаголовокПриложения("Иванов Петя");
    ПредупредитьОПонедельнике(); ←
    КонецПроцедуры

Процедура ПредупредитьОПонедельнике()
    Если ДеньНедели(ТекущаяДата()) = 7 Тогда
        ПоказатьОповещениеПользователя("Завтра в школу!");
    КонецЕсли;
    КонецПроцедуры

```

Рисунок 3.133. Возврат к вызову функции

Ещё несколько интересных приёмов связаны с отладкой процедур и функций. Вы прекрасно умеете выполнять отладку по шагам и используете для этого клавишу F11. Она позволяет вам останавливаться на каждой инструкции, которая исполняется.

Но это не всегда удобно. Бывают случаи, когда хочется выйти из функции раньше, чем закончатся все инструкции, которые в ней выполняются. Бывают случаи, когда не нужно заходить внутрь процедуры, а нужно, чтобы она просто выполнилась, без остановки на каждой инструкции.

Для этого есть ещё две команды пошаговой отладки. Чтобы познакомиться с ними, немного модифицируйте пример. Допишите две инструкции присваивания — до и после вызова процедуры. Например, как на рисунке 3.134.

Рисунок 3.134. Доработанный пример

Установите точку останова на первой инструкции присваивания. Запустите 1С:Предприятие в режиме отладки и по одному шагу дойдите до строки *Если*.

Теперь представьте: вы отлаживаете программу, чтобы найти ошибку. Вы посмотрели на текст функции и поняли, что ошибка не в ней, а где-то дальше. Поэтому нет смысла проходить всю функцию по шагам. Нужно вернуться к тому месту, откуда функция вызывалась.

В этом вам поможет команда *Шагнуть из* (рисунок 3.135).

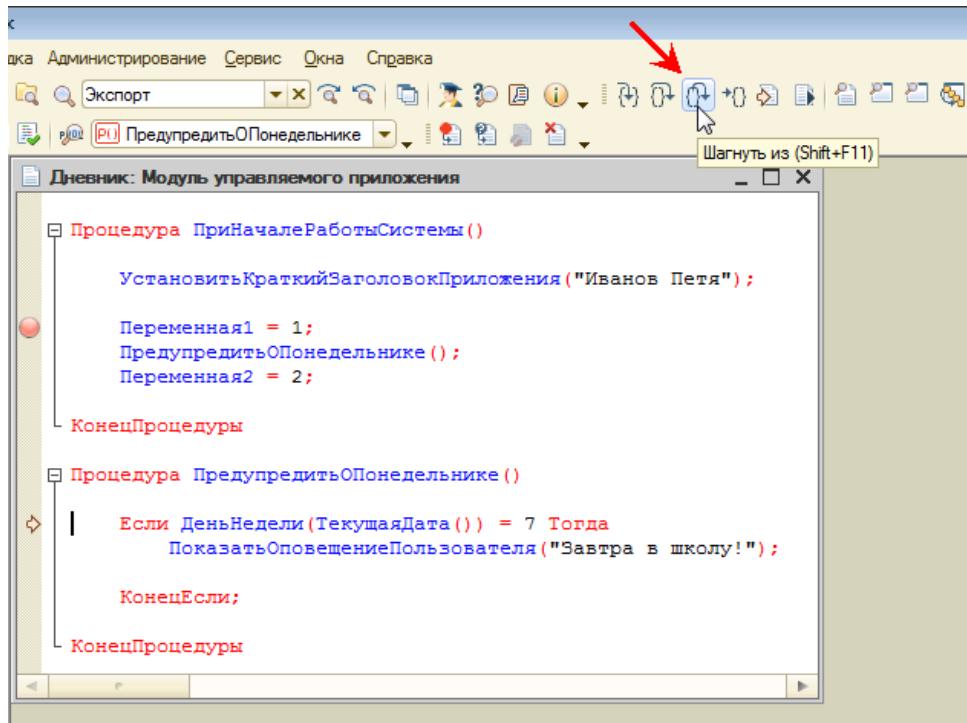


Рисунок 3.135. Команда «Шагнуть из»

В результате выполнения этой команды инструкции, содержащиеся в процедуре, будут исполнены без остановки. А остановка произойдёт тогда, когда исполнение вернётся к той строке, в которой процедура была вызвана (рисунок 3.136). Попробуйте.

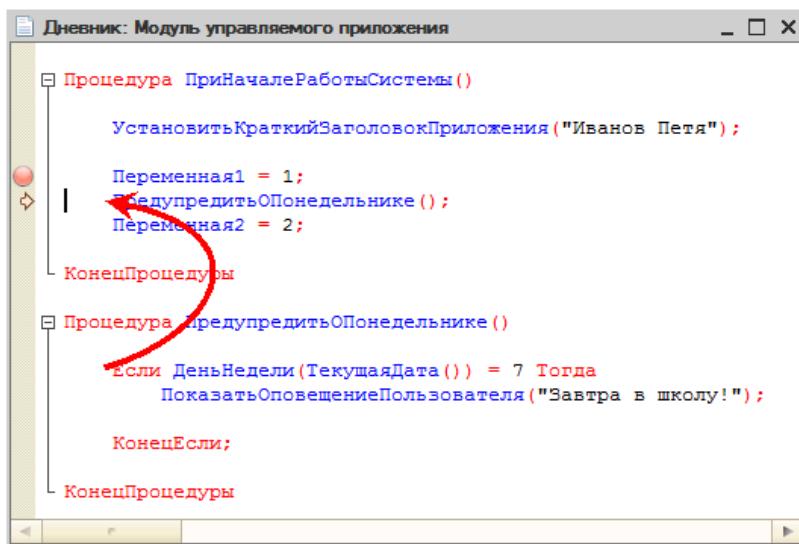


Рисунок 3.136. Переход к вызову функции

Теперь рассмотрим второй случай. Перезапустите отладку.

Например, вы отлаживаете свою программу и заранее точно знаете, что внутри процедуры *ПредупредитьОПонедельнике()* всё работает правильно и нет никакой необходимости заходить внутрь неё.

Тогда вы можете использовать команду *Шагнуть через* (рисунок 3.137).

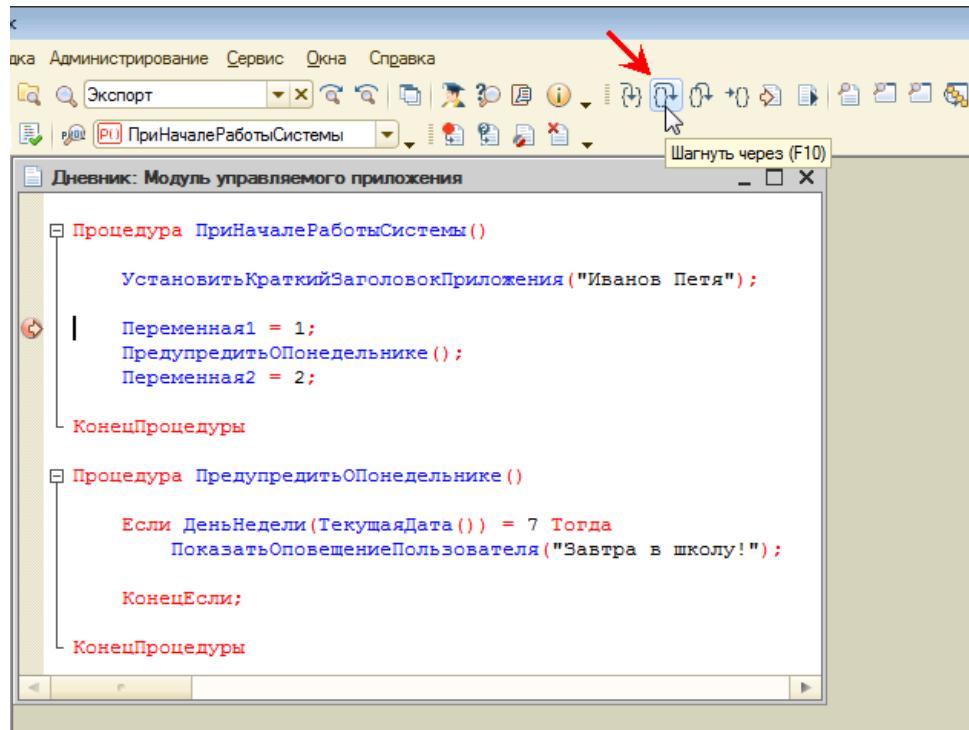


Рисунок 3.137. Команда «Шагнуть через»

Она позволит вам не заходить внутрь процедуры, а переходить от строки к строке (рисунок 3.138). Попробуйте.

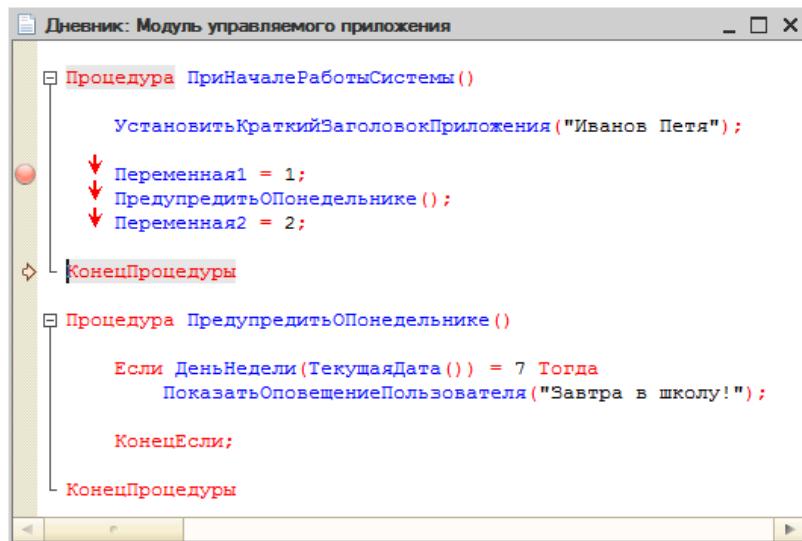


Рисунок 3.138. Переход через вызов процедуры

Подробнее

Подробнее вы можете прочитать про пошаговое выполнение в документации «Руководство разработчика. Раздел 28.2.4. Пошаговое выполнение».

3.10 Коллекции значений

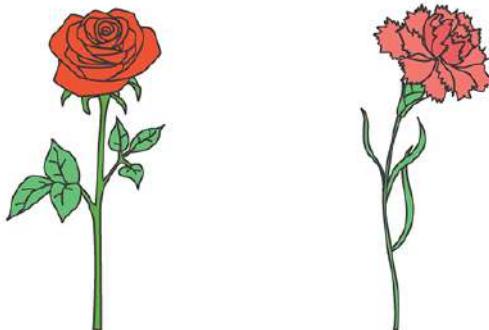
3.10.1 Объекты встроенного языка

Если вы помните, эта книга начиналась с одного интересного упражнения. Вы учились проникать вглубь общих понятий (лес) и разбираться с их деталями (деревья, ветки, листья). А потом вы делали обратное упражнение. Вы удалялись от мелких деталей (они пропадали из вашего зрения) и снова начинали воспринимать множество мелких деталей как единое целое.

Сейчас вам как раз понадобится вторая часть этого упражнения.

До сих пор вы имели дело со значениями типа *Число*, *Строка*, *Дата*, *Булево*. Такие значения являются простыми и неделимыми. В них нельзя выделить отдельные составляющие.

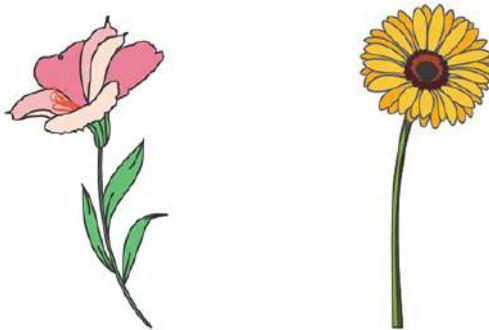
Такие значения можно сравнить с цветками. Например, одно значение типа *Строка* — это роза. А одно значение типа *Число* — это гвоздика (рисунок 3.139).



Значение типа «Строка» Значение типа «Число»

Рисунок 3.139. Роза — значение типа «Строка», гвоздика — значение типа «Число»

Ну, а «непривычные» значения типа *Дата* и *Булево* можно сравнить с альстремерией и зверобоем (рисунок 3.140).



Значение типа «Дата» Значение типа «Булево»

Рисунок 3.140. Альстремерия — значение типа «Дата», зверобой — значение типа «Булево»

С этими цветками вы можете делать какие-то действия. Например, прийти в гости и подарить цветок.

А теперь представьте, что вы пришли в гости и дарите не один цветок, а букет (рисунок 3.141).



Рисунок 3.141. Сложное значение — букет

С практической точки зрения он ничем не отличается от одного цветка. Его так же можно подарить, можно поставить его в вазу. То есть все действия с ним выполняются как с одним целым. Но фактически он является совокупностью отдельных цветков, которые собраны вместе.

Таким образом сейчас вы «удаляетесь» от мелких примитивных значений (цветков) и начинаете мыслить более крупными понятиями — букетами.

Во встроенном языке такие «букеты» называются *коллекциями значений*.

В коллекцию могут входить значения разных типов. То есть это может быть букет из разных цветов, как на рисунке 3.141. Но чаще всего в коллекцию входят значения одинакового типа. То есть получается букет из роз или букет из гвоздик (рисунок 3.142).



Рисунок 3.142. Коллекции значений одного типа

Собрать цветки вместе можно разными способами. Например, можно сделать из них букет. А можно корзину (рисунок 3.143).

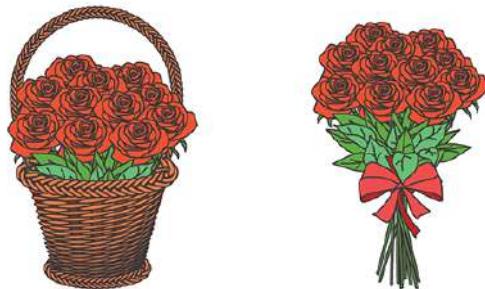


Рисунок 3.143. Разные способы собрать коллекцию цветов

То есть, говоря языком программистов, есть тип «Корзина», который является коллекцией значений, и есть тип «Букет», который тоже является коллекцией значений.

Тип «Корзина» может иметь много разных значений (рисунок 3.144).



Рисунок 3.144. Значения типа «Корзина»

Тип «Букет» тоже может иметь много разных значений (рисунок 3.145).

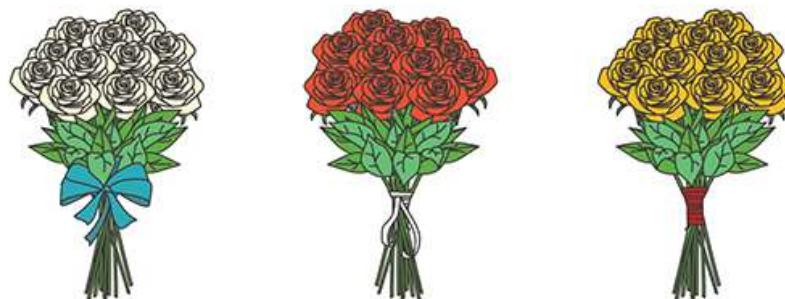


Рисунок 3.145. Значения типа «Букет»

Точно так же во встроенном языке есть разные типы, которые являются коллекциями значений. И таких типов гораздо больше, чем два.

Точно так же каждый из этих типов имеет свои значения. Такие значения сильно отличаются от значений примитивных типов (число, строка). И поэтому все они имеют

собственное название. Такие значения называются *объектами встроенного языка*. Или просто *объектами*.

Объект встроенного языка — это сложная конструкция. Поэтому у неё есть характерные особенности, связанные именно с тем, что объект содержит в себе множество примитивных значений. С этими особенностями вы будете знакомиться постепенно, не сразу.

3.10.2 Методы, конструкторы

Первая особенность связана с тем, что объект «из чего-то состоит». А это значит, что вам могут понадобиться какие-то действия, чтобы изменить его состав.

Например, добавить в букет ещё один цветок. Или убрать из букета пару цветков. Или те цветки, которые есть в букете, расположить в другом порядке. Чтобы букет выглядел красивее.

Такие действия, которые можно выполнять над объектом, называются *методы*.

У каждого сложного типа есть собственный набор методов. Например, у типа «Букет» может быть метод «перевязать букет ленточкой». А у типа «Корзина», естественно, такого метода быть не может.

Вторая особенность объектов тоже связана с тем, что они «из чего-то состоят». И это «что-то» не может в один миг собраться вместе, как по мановению волшебной палочки.

Например, как собирают букет? Сначала на столе расчищают свободное место. Затем начинают собирать по одному цветку. А как собирают корзину? Сначала берут пустую корзину. Потом помещают в неё специальную флористическую губку (питательную среду для цветов). И уже после этого начинают создавать композицию из цветов.

То есть всегда есть какие-то подготовительные действия перед тем, как начать собирать композицию. Сначала нужно сделать какую-то «заготовку».

Точно так же и во встроенном языке. Практически у всех объектов есть такое специальное действие, которое создаёт «заготовку» объекта. И это действие, в отличие от других методов, имеет собственное название — *конструктор*.

Заметка

Конструкторы многих объектов позволяют не только создать «заготовку», но и заполнить объект какими-то значениями. С этим вы познакомитесь позднее. В этом разделе я рассказываю про коллекции значений. А для них более типичны конструкторы, создающие пустые объекты, без данных.

Возможно, прямо сейчас вам сложно привыкнуть ко всем этим новым словам: объекты, методы, конструкторы, коллекции значений. Но не расстраивайтесь. Дальше вы познакомитесь с двумя универсальными коллекциями значений, выполните ряд упражнений, и всё сразу прояснится.

3.10.3 Массив

Универсальные коллекции значений — это несколько типов, которые можно использовать для самых разных задач. Поэтому они и называются «универсальные».

Значения этих типов, объекты, существуют только в то время, пока выполняется ваша программа. Поэтому, когда ваша программа перестанет выполняться, все эти объекты универсальных коллекций исчезнут. То есть они существуют только в оперативной памяти компьютера и в вашем воображении.

Все эти типы, универсальные коллекции значений, описаны в синтакс-помощнике, и вы можете их посмотреть. Откройте синтакс-помощник, раскройте ветку *Универсальные коллекции значений*. Вот они (рисунок 3.146).

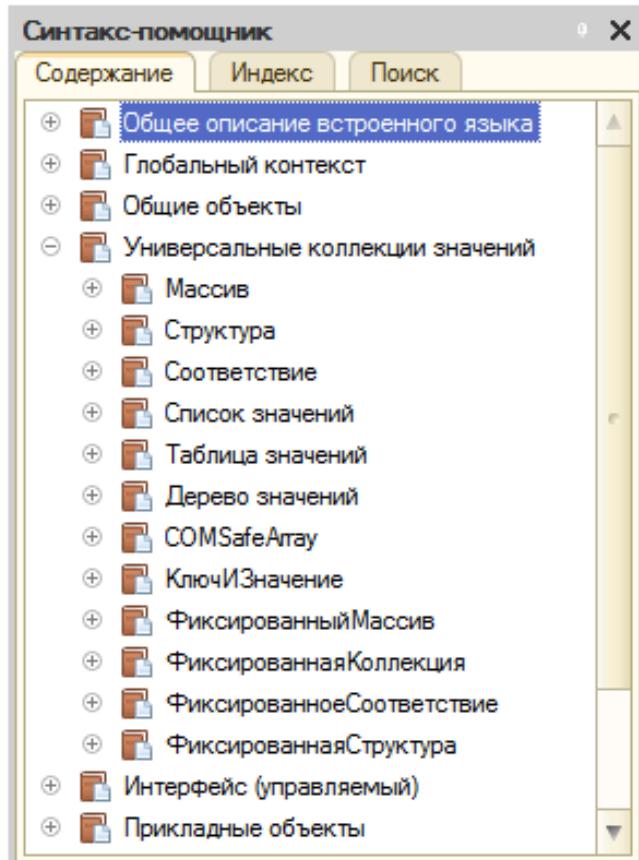


Рисунок 3.146. Универсальные коллекции значений

Вы познакомитесь только с двумя из них: *Массив* и *Структура*. Массив пригодится вам, чтобы улучшить электронный дневник, который вы делаете. Структура тоже пригодится, но ближе к концу книги. Остальные коллекции устроены аналогичным образом. Поэтому, если вы разберётесь с массивами и структурами, остальное сможете освоить самостоятельно.

Начните с массива. Представьте, что вам нужно где-то хранить мячики. Каждый мячик — это одно значение. Тогда массив — это стеллаж, в котором на каждую полку можно положить только один мячик (рисунок 3.147).



Рисунок 3.147. Массив — мячики на стеллаже

Когда вы захотите взять один из мячиков, вы скажете, например: «Дайте мне мячик со второй полки».

Вот так устроен массив, всё очень просто.

Раскройте ветку *Массив*. Вот то, о чём я говорил раньше: *Методы* и *Конструкторы* (рисунок 3.148).

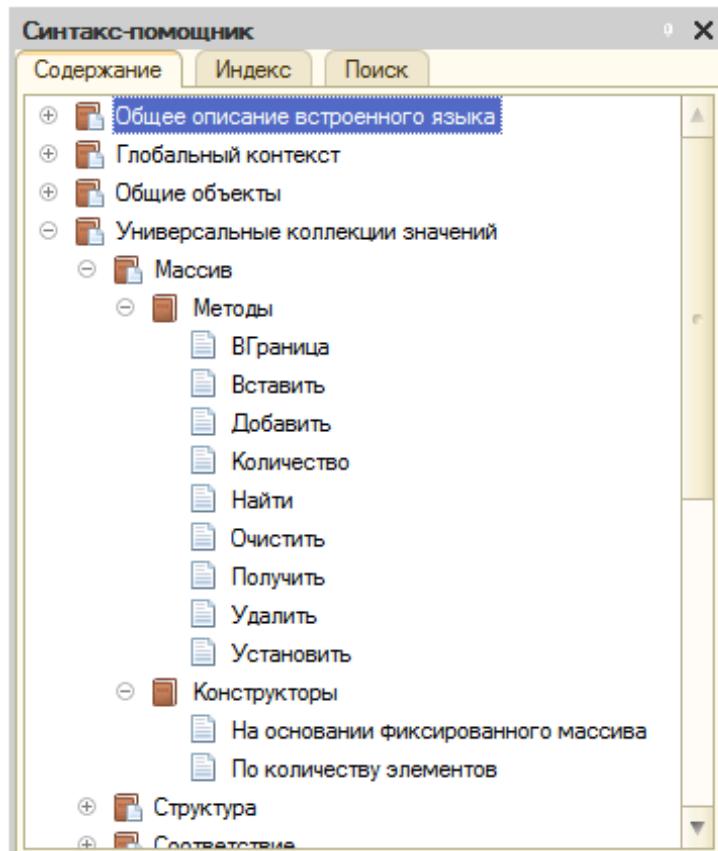


Рисунок 3.148. Методы и конструкторы массива

Зачем нужен тот или иной метод, вы наверняка поймёте прямо сейчас и даже без объяснений. Не про все методы, но про многие. Например, *Добавить()* наверняка добавляет один мячик на свободную полку. А *Удалить()* — удаляет. *Количество()* — это наверняка для того, чтобы посчитать, сколько мячиков на полках. А *Очистить()* — освобождает все полки. Так что ничего сложного тут нет.

Конструкторов два. Но это не должно вас смущать. Один конструктор (*По количеству элементов*) создаёт пустой массив, пустой стеллаж. Им вы будете пользоваться. Чаще всего используется именно он.

А второй конструктор (*На основании фиксированного массива*) делает буквально следующее: «сделайте мне такой же стеллаж с мячиками, как на этой картинке». Им вы пользоваться не будете. Да и в реальных конфигурациях используется он нечасто.

Теперь приступим к делу. Создайте массив, в котором будут содержаться названия дней недели. Каждое название дня недели — это «мячик».

Значение типа *Массив* (объект встроенного языка) будет храниться у вас в переменной *ДниНедели*. Поэтому напишите *ДниНедели* и поставьте знак равенства (рисунок 3.149).

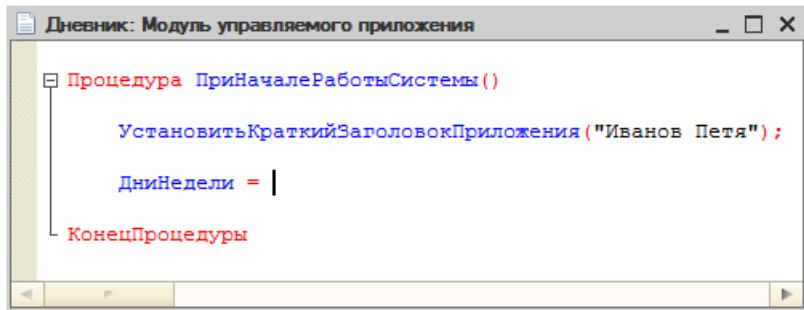


Рисунок 3.149. Переменная «ДниНедели»

Теперь, чтобы не было скучно, возьмите мышью конструктор *По количеству элементов* в синтакс-помощнике и перетащите его в то место, где стоит курсор. Платформа вставит туда заготовку конструктора (рисунок 3.150).

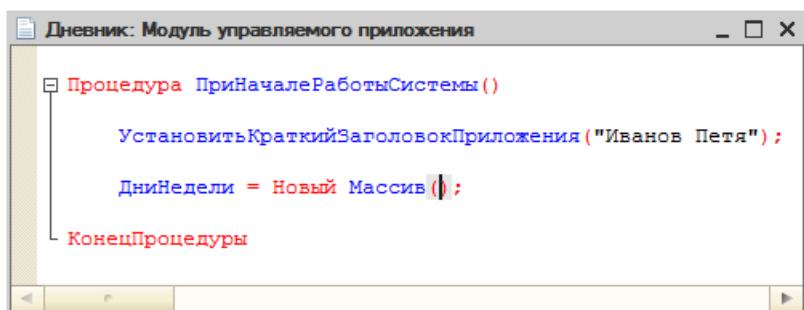


Рисунок 3.150. Заготовка конструктора массива

Конструкторы всех объектов выглядят одинаково. Они начинаются с обязательного слова *Новый*, а затем пишется имя типа. Вы хотите создать объект типа *Массив*, поэтому после *Новый* пишете *Mассив*.

После имени типа в скобках могут указываться, а могут и не указываться дополнительные параметры. Что нужно указывать в случае с массивом, вы можете сами посмотреть с синтакс-помощнике. Дважды щёлкните мышью на конструкторе *По количеству элементов*.

В простом случае вы можете сразу указать там количество элементов, которое будет содержаться в массиве. То есть сколько полок должно быть в стеллаже. Это одномерный массив.

Вы будете создавать одномерный массив, но количество заранее указывать не будете. То есть у вас будет «безразмерный» массив.

Бывают и многомерные массивы. Их вы рассматривать не будете. Но если у вас хорошее воображение и вы хотите представить себе двухмерный массив 3×2 , то это стеллаж из трёх полок (первое измерение), на каждой полке которого стоит ещё один стеллаж из двух полок (второе измерение).

Итак, у вас будет безразмерный массив, поэтому к тому, что вставила платформа, ничего больше дописывать не нужно.

Теперь заполните массив названиями дней недели. Это делается просто (рисунок 3.151). Пока не пишите.

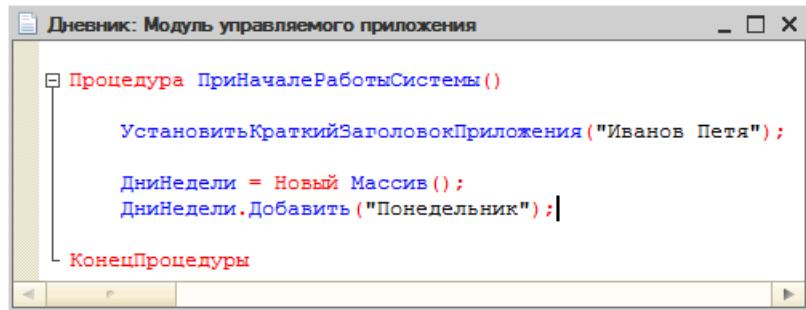


Рисунок 3.151. Добавить значение в массив

В переменной *ДниНедели* у вас находится объект типа *Массив*. Чтобы заставить этот объект выполнить какое-то действие, нужно через точку написать имя метода.

Когда вы напишете *ДниНедели* и поставите точку, произойдёт следующее. Платформа уже знает, что в этой переменной находится не «что-то там», а объект совершенно конкретного типа. Типа *Массив*. Поэтому она сразу откроет контекстную подсказку и предложит вам все методы, которые есть у этого объекта. Попробуйте (рисунок 3.152).

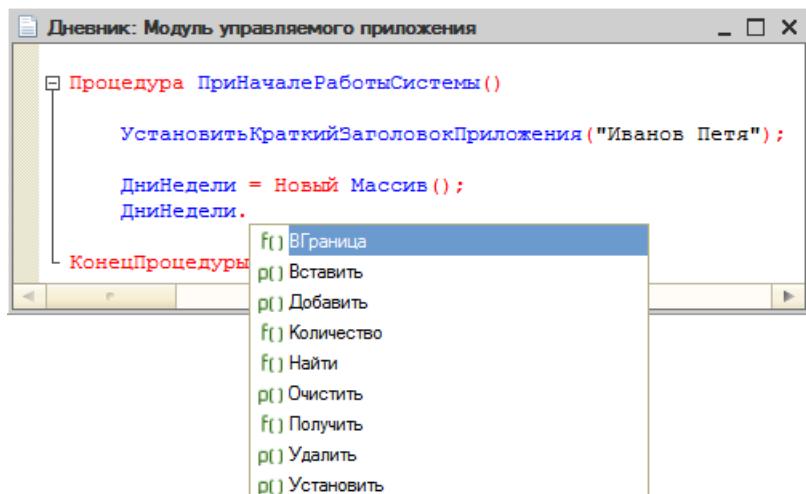


Рисунок 3.152. Контекстная подсказка методов объекта

Заметка

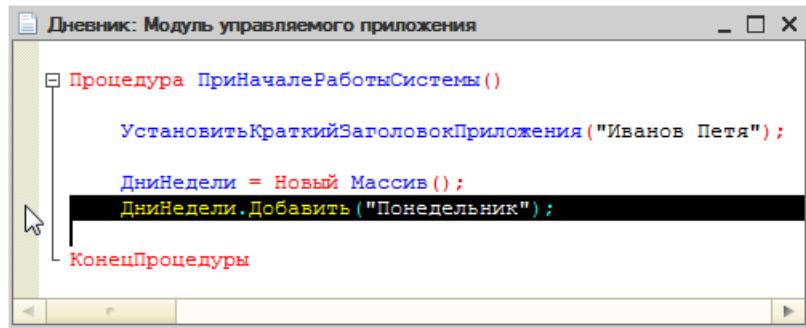
Символом *f()*, от английского слова *function*, платформа обозначает те методы, которые возвращают какое-то значение. То есть методы, которые внутри платформы реализованы как функции. А символом *p()*, от английского слова *procedure*, — методы, которые ничего не возвращают. Такие методы реализованы как процедуры.

Вам останется только выбрать из них тот метод, который вам нужен. Просто, правда? Допишите инструкцию до конца.

Что делает метод *Добавить()* и как его нужно записывать, вы можете посмотреть сами в синтакс-помощнике. В нём нет ничего сложного, просто в скобках нужно указать то значение, которое вы хотите поместить в массив.

А теперь маленькая программистская хитрость. По условиям задачи вам нужно добавить в массив ещё шесть элементов. То есть написать такие же инструкции, только с другими днями недели. Писать одно и то же скучно. Поэтому сейчас вы просто шесть раз скопируете эту строку, а потом только поменяете названия дней недели. Делается это так.

Выделяете строку, нажав мышью на серой линии слева (рисунок 3.153).



```

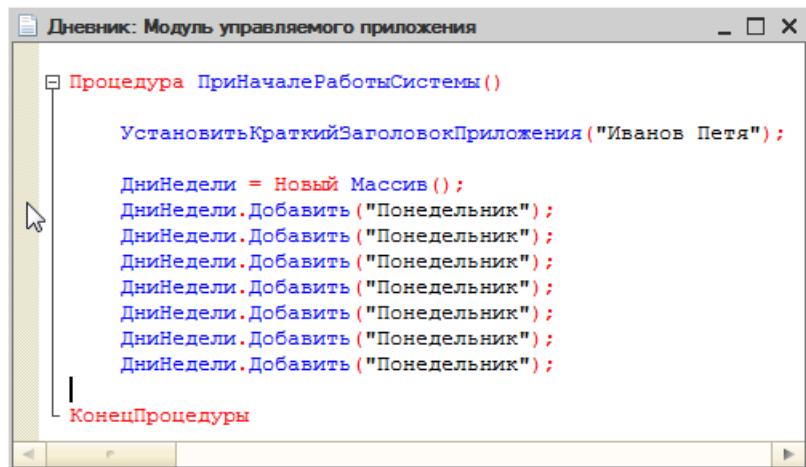
Дневник: Модуль управляемого приложения

Процедура ПриНачалеРаботыСистемы()
    УстановитьКраткийЗаголовокПриложения("Иванов Петя");
    ДниНедели = Новый Массив();
    ДниНедели.Добавить("Понедельник");
КонецПроцедуры

```

Рисунок 3.153. Выделить строку

Нажимаете сочетание клавиш **Ctrl+Insert**, чтобы скопировать выделенное в буфер обмена. А теперь сразу же нажимаете семь раз сочетание клавиш **Shift+Insert**. Чтобы вставить содержимое буфера обмена. В результате у вас получается семь одинаковых строк (рисунок 3.154).



```

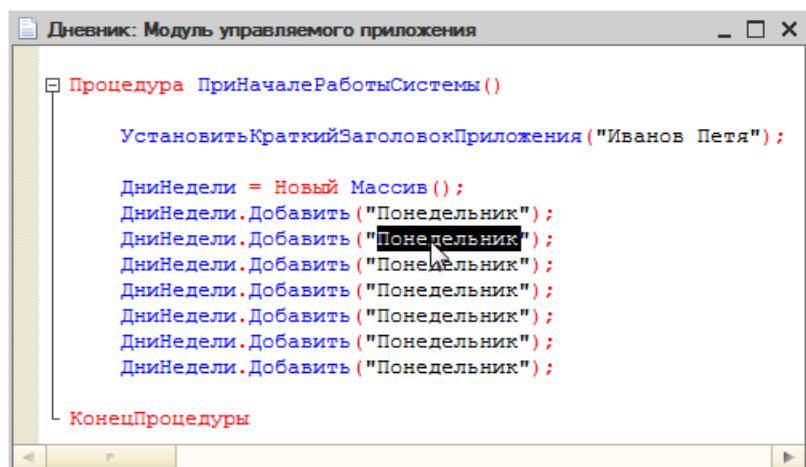
Дневник: Модуль управляемого приложения

Процедура ПриНачалеРаботыСистемы()
    УстановитьКраткийЗаголовокПриложения("Иванов Петя");
    ДниНедели = Новый Массив();
    ДниНедели.Добавить("Понедельник");
    ДниНедели.Добавить("Понедельник");
    ДниНедели.Добавить("Понедельник");
    ДниНедели.Добавить("Понедельник");
    ДниНедели.Добавить("Понедельник");
    ДниНедели.Добавить("Понедельник");
    ДниНедели.Добавить("Понедельник");
КонецПроцедуры

```

Рисунок 3.154. Вставить строку семь раз

Теперь во второй строке дважды щёлкаете мышью на слове *Понедельник* и меняете его на *Вторник* (рисунок 3.155).



```

Дневник: Модуль управляемого приложения

Процедура ПриНачалеРаботыСистемы()
    УстановитьКраткийЗаголовокПриложения("Иванов Петя");
    ДниНедели = Новый Массив();
    ДниНедели.Добавить("Понедельник");
    ДниНедели.Добавить("Вторник");
    ДниНедели.Добавить("Понедельник");
    ДниНедели.Добавить("Понедельник");
    ДниНедели.Добавить("Понедельник");
    ДниНедели.Добавить("Понедельник");
    ДниНедели.Добавить("Понедельник");
КонецПроцедуры

```

Рисунок 3.155. Выделить слово «Понедельник»

И так шесть раз (рисунок 3.156).

```

Дневник: Модуль управляемого приложения

Процедура ПриНачалеРаботыСистемы()
{
    УстановитьКраткийЗаголовокПриложения("Иванов Петя");

    ДниНедели = Новый Массив();
    ДниНедели.Добавить("Понедельник");
    ДниНедели.Добавить("Вторник");
    ДниНедели.Добавить("Среда");
    ДниНедели.Добавить("Четверг");
    ДниНедели.Добавить("Пятница");
    ДниНедели.Добавить("Суббота");
    ДниНедели.Добавить("Воскресенье");

    КонецПроцедуры
}

```

Рисунок 3.156. Массив заполнен названиями дней недели

Всё! Пример готов. Можете установить точку останова на конец процедуры и запустить 1С:Предприятие в режиме отладки.

Но теперь возникает вопрос. Как посмотреть, что находится в массиве?

Способы подвести курсор или открыть локальные переменные в случае с объектами вам не помогут. Тут надо использовать *Вычислить выражение*.

Дважды щёлкните на слове *ДниНедели* и нажмите (если вы забыли) сочетание клавиш Shift+F9. Откроется окно вычисления выражений (рисунок 3.157).

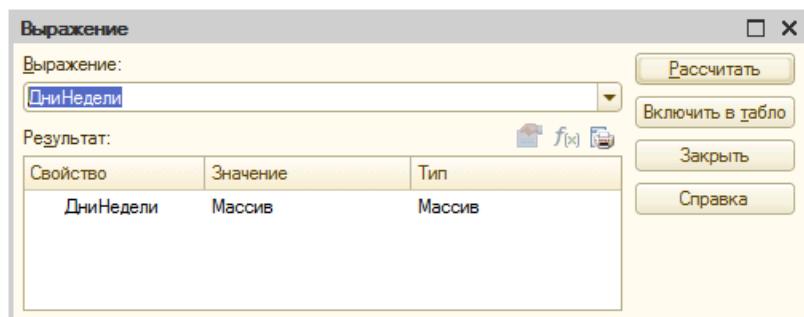


Рисунок 3.157. Выражение

Но тут по-прежнему очень мало информации.

Тогда выделите единственную строку, которая есть в окне Результат, и нажмите F2. И, о чудо! Вот он, ваш массив (рисунок 3.158).

Индекс	Значение элемента	Тип элемента
0	"Понедельник"	Строка
1	"Вторник"	Строка
2	"Среда"	Строка
3	"Четверг"	Строка
4	"Пятница"	Строка
5	"Суббота"	Строка
6	"Воскресенье"	Строка

Рисунок 3.158. Содержимое массива

С помощью этой «волшебной» кнопки F2 вы можете просматривать содержимое любых объектов, которые являются коллекциями. Если вы вдруг её забудете, она всегда под рукой в командной панели и называется *Показать значение в отдельном окне* (рисунок 3.159).

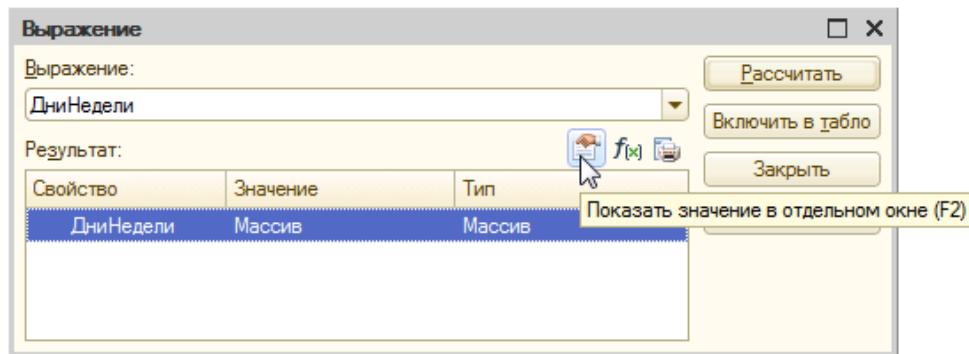


Рисунок 3.159. Показать значение в отдельном окне

Итак, на рисунке 3.158 ваш массив. Он даже тут похож на стеллаж, только перевёрнутый вверх ногами.

Массив — это *нумерованная коллекция*. То есть её элементы расположены не просто так, а в определённом порядке друг за другом. Для этого у каждого элемента массива есть *индекс*. Индекс — это как номер по порядку, только начинается он не с единицы, а с нуля.

Примечание

Везде во встроенным языке, где вам попадётся слово «индекс», знайте: индекс первого элемента всегда равен нулю, а индекс последнего элемента вычисляется по формуле: «количество элементов минус единица».

Итак, у каждого элемента массива есть индекс, а значение элемента массива — это самое значение, которое вы туда поместили. Если вспомнить стеллаж, то в данном случае он будет выглядеть так (рисунок 3.160).

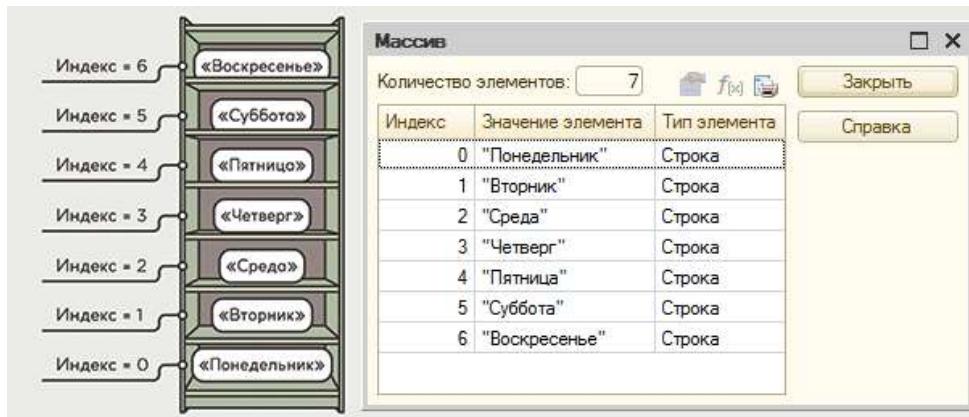


Рисунок 3.160. Массив — стеллаж

Теперь о методах, которые есть у массива. То есть о тех действиях, которые можно выполнять с объектом типа *Массив*.

Метод *Добавить()* вы уже знаете. Он добавляет новое значение в конец массива.

Методы *Вставить()* и *Установить()* тоже добавляют одно значение. Но не в конец, а в ту позицию, которую вы укажете. При этом *Вставить()* раздвигает элементы, а *Установить()* замещает тот элемент, который уже есть в этой позиции.

Например, если вы захотите вставить или установить число 2 во вторую позицию массива, то результат будет такой (рисунок 3.161).

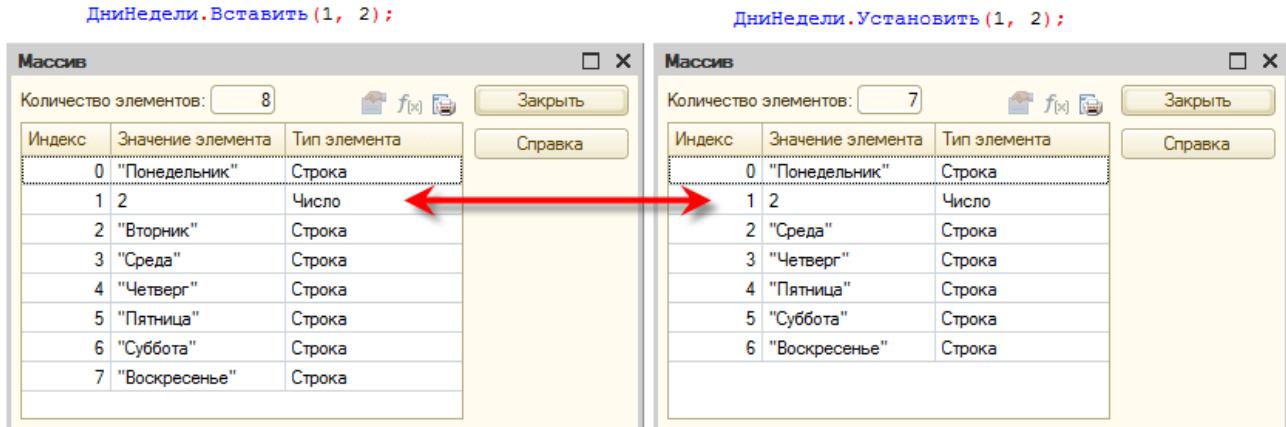


Рисунок 3.161. «Вставить()» и «Установить()»

Метод *Количество()* сообщает, сколько элементов в массиве. А метод *ВГраница()* нужен только для того, чтобы быстро узнать индекс последнего элемента и не писать вместо этого *Количество() - 1*.

Метод *Очистить()* удаляет все элементы, а метод *Удалить()* удаляет только один элемент.

Метод *Получить()* возвращает значение, которое находится по указанному индексу. А метод *Найти()* возвращает индекс того значения, которое вы ищете.

То есть *ДниНедели.Найти("Вторник")* вернёт вам число 1, а *ДниНедели.Получить(1)* вернёт вам строку "Вторник". Можете попробовать.

Задание 3.33

Создайте массив и запишите в него названия всех месяцев по порядку.

Задание 3.34

Посмотрите содержимое массива из предыдущего задания в конфигураторе.

Задание 3.35

В массив из задания 3.33 добавьте два элемента.

Один элемент, «--- Начало лета», добавьте перед месяцем, который называется «Июнь». Этот месяц нужно найти по названию.

Второй элемент, «--- Конец лета», добавьте после месяца, который называется «Август». Этот месяц тоже нужно найти по названию.

Посмотрите в конфигураторе, правильно ли вы выделили летние месяцы.

Задание 3.36

С помощью цикла обойдите все элементы массива и добавьте в конец каждого названия текущий год. Чтобы, например, вместо «Январь» стало «Январь 2016 Г.».

Посмотрите в конфигураторе, правильно ли выглядит результат.

3.10.4 Обрабатывайте ошибочные ситуации

Сейчас я расскажу вам несколько программистских «секретов». Все они связаны с коллекциями значений. Вот первый из них.

Некоторые методы объектов не всегда могут выполняться успешно. Пример этого вы можете увидеть уже в массиве. Например, если в синтакс-помощнике вы посмотрите описание метода *Найти()*, то увидите, что он возвращает *Число* или *Неопределено*. Число он возвращает тогда, когда выполняется успешно и в массиве есть то, что вы ищете. Но вполне может быть такая ситуация, что в массиве не окажется того, что вы ищете. И тогда он вернёт значение *Неопределено*.

Начинающие программисты часто забывают об этой особенности. И это естественно: ведь когда вы пишете программу, вы совершенно точно знаете, что у вас будет в массиве, а чего не будет. Но дело в том, что когда пользователи (да и вы сами) начинают работать с вашей программой, они могут выполнить такую последовательность действий, которую вы не предусматривали в своей программе. И в результате в массиве не окажется того значения, которое, по вашему мнению, там обязательно должно быть.

Примечание

Если вы думаете, что такого не может быть, то знайте, что такое может быть!
Потому что пользователи думают совсем не так, как думаете вы!

А раз так, то правилом хорошего тона в программировании является обработка не только успешных действий, но и ошибочных ситуаций, которые могут возникнуть.

В данном случае в том месте, где вы бы написали просто *Найти()* (листинг 3.50), лучше обработать неудачную ситуацию, которая может возникнуть (листинг 3.51).

Листинг 3.50. Метод «Найти()»

```
ВторойДеньНедели = ДниНедели.Найти("Вторник");
ПоказатьОповещениеПользователя("Сегодня: " + Стока(ВторойДеньНедели + 1) +
    "-й день недели.");
```

Листинг 3.51. Метод «Найти()» и обработка ошибочной ситуации

```
ВторойДеньНедели = ДниНедели.Найти("Вторник");

// Обработать ошибку.
Если ВторойДеньНедели = Неопределено Тогда
    ПоказатьПредупреждение(, "Вторник не найден. Расчёт не выполнен!");
    Возврат;

КонецЕсли;

ПоказатьОповещениеПользователя("Сегодня: " + Стока(ВторойДеньНедели + 1) +
    "-й день недели.");
```

То есть для того случая, когда метод возвращает значение *Неопределено*, написать условие *Если*. В нём обработать ошибочную ситуацию. Например, вывести какое-то информационное сообщение. А если в этой ситуации дальнейшая работа вашего алгоритма невозможна, то и завершить его с помощью инструкции *Возврат*.

Совет

Можете самостоятельно посмотреть в синтакс-помощнике, что делает процедура `ПоказатьПредупреждение()`. У неё несколько параметров.

Обязательным параметром является только один — `<ТекстПредупреждения>`. Его вы и указали. Необязательные параметры, которые находятся после него, можно не указывать и не ставить запятые.

А вот «место» необязательного параметра, который находится перед ним, нужно обозначить запятой. Чтобы платформа понимала, что вашу строку `"Вторник не найден... "` нужно передать именно во второй параметр, а не в первый.

Тогда получится, что если «всё хорошо», исполнение пройдёт мимо условия `Если`. А если «всё плохо», выполнится тело условия и вы выйдите из процедуры, не выполняя остальные инструкции.

Чтобы посмотреть, как это работает, можете смоделировать ошибочную ситуацию. После заполнения массива значениями установите элементу с индексом 1 (`Вторник`) значение 2 (листинг 3.52).

Листинг 3.52. Замените значение «Вторник» значением «2»

```
ДниНедели = Новый Массив();
ДниНедели.Добавить("Понедельник");
ДниНедели.Добавить("Вторник");
ДниНедели.Добавить("Среда");
ДниНедели.Добавить("Четверг");
ДниНедели.Добавить("Пятница");
ДниНедели.Добавить("Суббота");
ДниНедели.Добавить("Воскресенье");

ДниНедели.Установить(1, 2);
```

Если теперь вы запустите 1С:Предприятие, то увидите сообщение `Вторник не найден...`.

Чтобы посмотреть, как этот пример будет работать без условия `Если`, можете временно закомментировать это условие и тем самым исключить из исполнения. Это легко сделать с помощью команды `Добавить комментарий`, которая находится в верхней командной панели (рисунок 3.162). Выделите все строки, которые вам временно не нужны, и закомментируйте их.

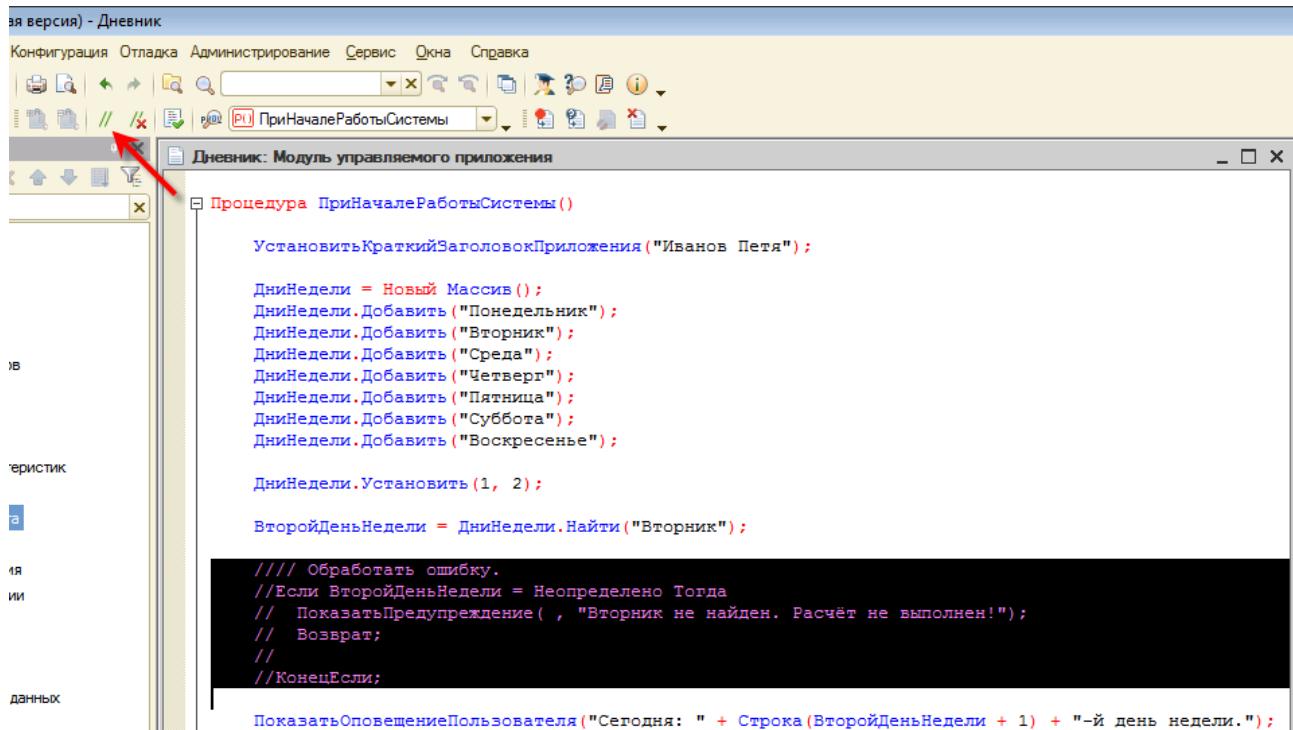


Рисунок 3.162. Добавить комментарий

Если после этого вы запустите 1С:Предприятие, то увидите, что было бы, если бы вы не обрабатывали ошибочную ситуацию (рисунок 3.163).

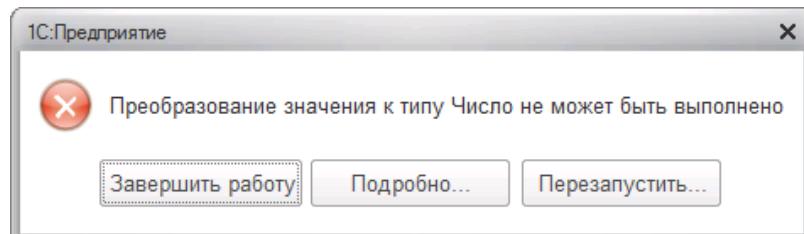


Рисунок 3.163. Сообщение об ошибке

Смысл этого сообщения в том, что платформа не может выполнить операцию *Второй-ДеньНедели* + 1. Потому что в переменной *ВторойДеньНедели* у вас оказалось значение *Неопределено*. А для него не существует операции сложения с числом.

Самое неприятное в этом сообщении то, что платформа не может продолжить дальнейшую работу, и у вас есть единственный выход — нажать кнопку **Завершить работу**.

Поэтому не спешите, не ленитесь, обращайте внимание на то, какие значения возвращают методы. Если возможна ситуация, когда выполнение метода заканчивается «неудачей», обрабатывайте такие ситуации. Во-первых, ваше собственное сообщение поможет быстро понять, в чём проблема. А во-вторых, даже если ваш небольшой алгоритм закончился неудачей, пользователь сможет продолжить работу с вашей программой. Сможет воспользоваться другими её возможностями.

Задание 3.37

Вы ведёте список учеников, которые поедут на экскурсию. Список постоянно меняется, кто-то добавляется, кто-то вычёркивается. Текущий состав списка вы не знаете.

Вам сообщили, что ученик Захаров заболел, и его нужно вычеркнуть из списка.

Создайте массив и заполните его фамилиями Сергеев, Дмитриев, Захаров, Максимов. Напишите программу, которая удаляет из списка ученика по фамилии Захаров.

Теперь в инструкциях, которые заполняют массив фамилиями, замените фамилию Захаров на фамилию Александров. Проверьте, что ваша программа по-прежнему работает без ошибок.

3.10.5 Используйте операцию [...]

Вот второй «секрет». Вы уже видели (а может быть, даже и пробовали), как получить значение по его индексу. Для этого можно использовать метод *Получить()* (листинг 3.53).

Листинг 3.53. Метод «Получить()»

```
ВторойДень = ДниНедели.Получить(1);
```

Но на самом деле есть более простой и короткий способ сделать то же самое. Чаще всего именно этот способ используют при работе не только с массивами, но и с любыми коллекциями значений.

Во встроенном языке существует операция «квадратные скобки». Выглядит она так (листинг 3.54).

Листинг 3.54. Операция «квадратные скобки»

```
ВторойДень = ДниНедели.Получить(1);  
ВторойДень = ДниНедели[1];
```

Две этих инструкции абсолютно идентичны. И в первом, и во втором случае в переменной *ВторойДень* окажется одно и то же значение — *Вторник*. Но вторая инструкция короче и читается легче. Сразу после имени переменной, в которой находится коллекция, ставятся квадратные скобки. А в них указывается индекс того элемента, который нужно получить.

Попробуйте самостоятельно получить, например, третий и пятый элементы массива с помощью квадратных скобок.

Задание 3.38

Создайте массив и заполните его названиями дней недели по порядку. С помощью операции [...] получите названия третьего и пятого дней недели.

Задание 3.39

Создайте массив и заполните его названиями дней недели по порядку. С помощью операции [...] к выходным дням допишите «вых.». Чтобы, например, вместо «Суббота» получилось «Суббота вых.».

3.10.6 Используйте инструкцию Для Каждого Цикл

Вот третий «секрет». Раньше вы уже познакомились с инструкцией *Цикл*. Она позволяет многократно выполнять некоторый набор инструкций. Цикл часто используется при работе с коллекциями. Например, для того чтобы перебрать все элементы коллекции и выполнить с ними какие-то действия.

Чтобы не усложнять, напишите самый простой пример. У вас есть массив с названиями дней недели. Вам нужно выбрать из него только будние дни и скопировать их в новый массив.

С помощью цикла это можно выполнить следующим образом (листинг 3.55).

Листинг 3.55. Перебор элементов коллекции в цикле

```
БудниДни = Новый Массив();  
Для Индекс = 0 По ДниНедели.ВГраница() Цикл  
    ТекущийЭлемент = ДниНедели[Индекс];  
    Если ТекущийЭлемент = "Суббота" ИЛИ ТекущийЭлемент = "Воскресенье" Тогда  
        Продолжить;  
  
    Иначе  
        БудниДни.Добавить(ТекущийЭлемент);  
  
    КонецЕсли;  
  
КонецЦикла;
```

Можете дописать этот пример к своему массиву и посмотреть с помощью отладки, как он работает.

Сначала вы создаёте новый массив, в который будете помещать будние дни. После этого последовательно, от нуля до последнего индекса, перебираете все элементы массива.

Значение текущего элемента вы получаете по его индексу с помощью операции «квадратные скобки». Если это значение *Суббота* или *Воскресенье*, то вы ничего не делаете и переходите к следующей итерации цикла. То есть к его началу. Для этого используется новая для вас инструкция *Продолжить*.

А во всех других случаях вы добавляете название буднего дня в свой заранее созданный массив.

Сделайте этот пример в своей конфигурации и в режиме отладки посмотрите по шагам, как он работает. В том числе посмотрите, как работает инструкция *Продолжить*.

Такие задачи, когда нужно перебрать все элементы коллекции, встречаются очень часто. Поэтому во встроенным языке существует специальная форма инструкции *Цикл*. Она обозначается *Для Каждого Цикл*. И тот же самый пример с её использованием будет выглядеть так (листинг 3.56).

Листинг 3.56. Инструкция «Для Каждого Цикл»

```

БудниДни = Новый Массив();
Для Каждого ТекущийЭлемент Из ДниНедели Цикл
    Если ТекущийЭлемент = "Суббота" ИЛИ ТекущийЭлемент = "Воскресенье" Тогда
        Продолжить;

Иначе
    БудниДни.Добавить(ТекущийЭлемент);

КонецЕсли;

КонецЦикла;

```

Вы видите, что выглядит этот пример проще и читается легче.

Если сравнить оба примера, то видно, что по сути инструкция *Для Каждого Цикл* заменила собой сразу две строки, которые были в вашем старом примере (листинг 3.57).

Листинг 3.57. Инструкции «Для Каждого Цикл» и «Для Цикл»

```

БудниДни = Новый Массив();
Для Каждого ТекущийЭлемент Из ДниНедели Цикл
    Если ТекущийЭлемент = "Суббота" ИЛИ ТекущийЭлемент = "Воскресенье" Тогда
        Продолжить;

Иначе
    БудниДни.Добавить(ТекущийЭлемент);

КонецЕсли;

КонецЦикла;

БудниДни = Новый Массив();
Для Индекс = 0 По ДниНедели.ВГраница() Цикл
    ТекущийЭлемент = ДниНедели[Индекс];
    Если ТекущийЭлемент = "Суббота" ИЛИ ТекущийЭлемент = "Воскресенье" Тогда
        Продолжить;

Иначе
    БудниДни.Добавить(ТекущийЭлемент);

КонецЕсли;

КонецЦикла;

```

Инструкцию *Для Каждого Цикл* используют тогда, когда порядок обхода коллекции не важен. А важно лишь перебрать все элементы, которые есть в коллекции.

Попробуйте в своей конфигурации, как работает второй вариант этого примера.

В инструкции *Продолжить*, с которой вы познакомились в этом примере, нет ничего сложного. Она используется только внутри циклов и только для такой задачи, которая показана в этом примере. Когда нужно перейти к очередной итерации цикла, не выполняя операторы, следующие далее.

Кроме *Продолжить* внутри цикла может использоваться и другая инструкция — *Прервать*. Она тоже очень простая. Она без всяких условий прекращает исполнение цикла и переходит к инструкции, которая расположена после *КонецЦикла*.

Инструкция *Прервать* может понадобиться вам тогда, когда вы точно знаете, что перебирать коллекцию дальше нет смысла. Например, в вашем массиве дни недели всегда

расположены в правильном порядке. Поэтому, если вы перебираете их с начала и дошли до субботы, то понятно, что дальше их перебирать нет смысла. Потому что следующий за субботой день тоже будет выходной. Значит, в условии *Если* вы можете написать *Если ТекущийЭлемент = "Суббота" Тогда*. А затем — *Прервать*.

Небольшое пояснение. Фрагмент, показанный в листинге 3.56, вы вряд ли встретите в реальной программе. Здесь я показал его только в учебных целях. В реальной программе такой фрагмент может появиться разве что в процессе разработки. Когда вы по очереди пишете алгоритмы для каждой ветки условия. И, чтобы была возможность проверить написанное, в той ветке, которая ещё не готова, вы просто пропускаете весь цикл.

На самом деле этот фрагмент обычно выглядит иначе. Как именно, вы можете попробовать сами, выполнив задания к этому разделу.

Другое небольшое пояснение касается инструкции *Прервать*. Тут объяснение этой инструкции пришлось к слову и к месту. Но именно в цикле *Для Каждого* эта инструкция используется редко. Потому что порядок обхода элементов в таком цикле не определён. Гораздо чаще эта инструкция используется в циклах, которые имеют определённый порядок обхода. Такой пример вы тоже можете попробовать в одном из заданий к этому разделу.

Задание 3.40

Используйте массив *ДниНедели*, показанный на рисунке 3.162. Добавьте к нему листинг 3.56.

Теперь представьте, что для будних дней вам нужно не просто выполнить одну строку *БудниДниДобавить(ТекущийЭлемент)*, а реализовать в этом месте большой и сложный алгоритм. Прятать такой алгоритм внутрь условия *Если* нехорошо и неудобно.

В этом случае обычно поступают следующим образом. В начале цикла сразу же определяют, подходит очередной день для выполнения алгоритма или нет. Если не подходит (выходной), то просто переходят к следующей итерации цикла. Больше никаких проверок не делают. В результате получается, что инструкции, написанные после *Если*, будут выполняться только для подходящих элементов.

Измените листинг 3.56 так, чтобы он соответствовал этой стратегии. С помощью пошаговой отладки посмотрите, как работает цикл.

Задание 3.41

Используйте массив *ДниНедели*, показанный на рисунке 3.162. Добавьте к нему листинг 3.56.

Обратная ситуация. Алгоритм, который нужно выполнить для будних дней, небольшой и несложный.

В этом случае с помощью инструкции *Если* отбирают только те элементы, которые подходят для этого алгоритма. И выполняют алгоритм. Получается, что после *Если* нет никаких инструкций, и для неподходящих элементов ничего не выполняется.

Измените листинг 3.56 так, чтобы он соответствовал этой стратегии. С помощью пошаговой отладки посмотрите, как работает цикл.

Задание 3.42

Используйте массив *ДниНедели*, показанный на рисунке 3.162. Добавьте к нему листинг 3.55, в котором используется инструкция *Для По Цикл*.

Дни недели расположены в массиве в правильном порядке. Вы обходите их в цикле в том же правильном порядке. Вы точно знаете, что после субботы будет воскресенье, которое вам не нужно.

Измените листинг так, чтобы не анализировались те дни, которые вам заведомо не нужны. С помощью пошаговой отладки посмотрите, как работает цикл.

Задание 3.43

Создайте массив и в цикле заполните его годами с 2000 по 2020. Затем из этого массива отберите только високосные годы и поместите их в другой массив. Високосным считается год, который делится на 4 без остатка.

Объясните, почему вы использовали ту или иную инструкцию цикла. Посмотрите в конфигураторе, сколько високосных лет у вас получилось.

3.10.7 Удаляйте элементы с конца

И вот последний, четвёртый, «секрет». Инструкция *Для Каждого Цикл* очень удобная и простая. Так и хочется ею пользоваться! Но тут главное — не переусердствовать. Потому что бывают задачи, когда элементы коллекции обязательно нужно перебирать в определённом порядке. Хотя, на первый взгляд, в этом нет никакой необходимости.

Чтобы всё увидеть своими глазами, сделайте простой пример (листинг 3.58).

Листинг 3.58. Удалить все числа, которые меньше 10. Неправильный вариант

```
Числа = Новый Массив();
Числа.Добавить(10);
Числа.Добавить(8);
Числа.Добавить(4);
Числа.Добавить(15);

Для Каждого ТекущееЧисло Из Числа Цикл
  Если ТекущееЧисло < 10 Тогда
    Числа.Удалить(Числа.Найти(ТекущееЧисло));

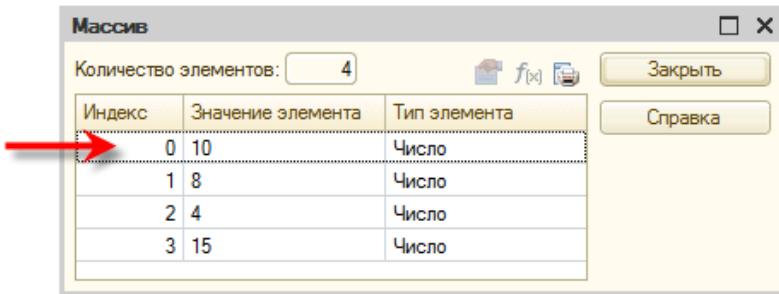
  КонецЕсли;

КонецЦикла;
```

Задача очень простая. У вас есть массив из четырёх чисел. Вам нужно удалить из этого массива все числа, которые меньше 10.

Поставьте точку останова на инструкции *Если* и запустите 1С:Предприятие в режиме отладки.

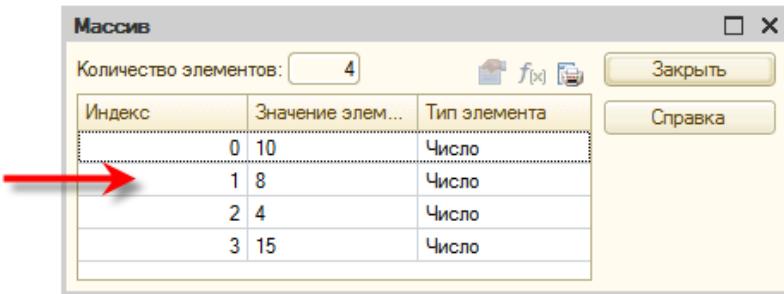
Не ожидая подвоха, вы написали *Для Каждого Цикл*, и дальше происходит вот что. На первой итерации цикла *ТекущееЧисло* у вас — 10 (рисунок 3.164).



Индекс	Значение элемента	Тип элемента
0	10	Число
1	8	Число
2	4	Число
3	15	Число

Рисунок 3.164. Первая итерация цикла

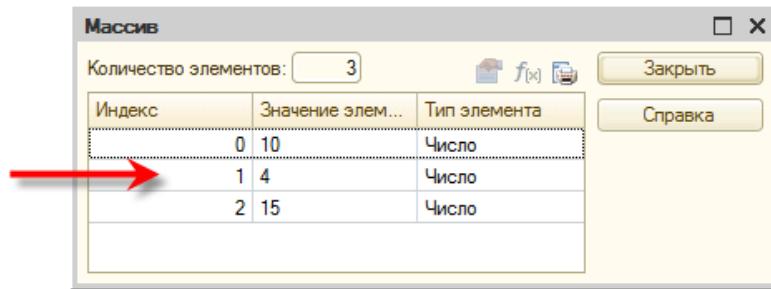
На второй итерации цикла Текущее Число у вас будет 8 (рисунок 3.165).



Индекс	Значение элемента	Тип элемента
0	10	Число
1	8	Число
2	4	Число
3	15	Число

Рисунок 3.165. Вторая итерация цикла

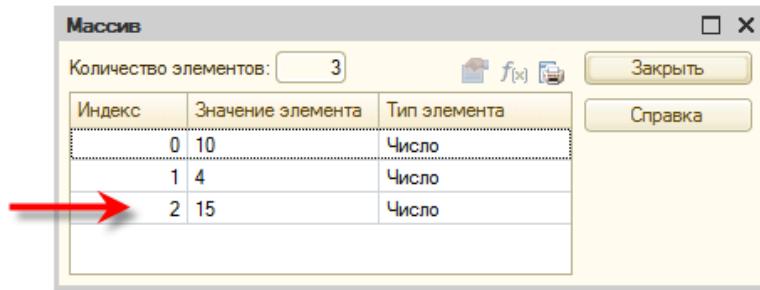
И тут вы обнаруживаете, что этот элемент нужно удалить. Коллекции не терпят пустых мест. Как только вы его удаляете, остальные элементы коллекции перемещаются, чтобы занять освободившееся место (рисунок 3.166).



Индекс	Значение элемента	Тип элемента
0	10	Число
1	4	Число
2	15	Число

Рисунок 3.166. После удаления элемента 8

Получается, что на втором месте оказалось число 4, которое вы ещё не обрабатывали. Но цикл продолжается, и вы, естественно, переходите к следующему элементу коллекции (рисунок 3.167).



Индекс	Значение элемента	Тип элемента
0	10	Число
1	4	Число
2	15	Число

Рисунок 3.167. Третья итерация цикла

Теперь ваше текущее число 15. А 4 «проскочила» мимо вас. И осталась в массиве.

Если вспомнить, что вы представляли массив как стеллаж, то когда вы удаляете один из мячиков, все остальные, которые находятся выше него, «проваливаются» вниз (рисунок 3.168).

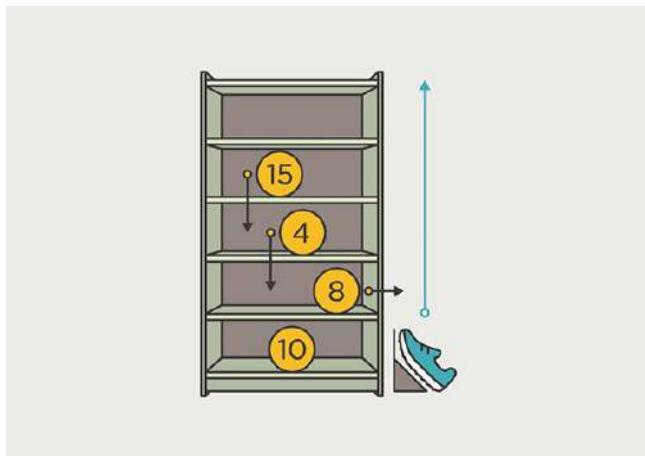


Рисунок 3.168. Остальные мячики проваливаются вниз

И мячик, который вы должны были бы обработать на следующем шаге, проскакивает мимо вас.

Чтобы такого не случалось, нужно двигаться сверху вниз, то есть от конца к началу (рисунок 3.169).

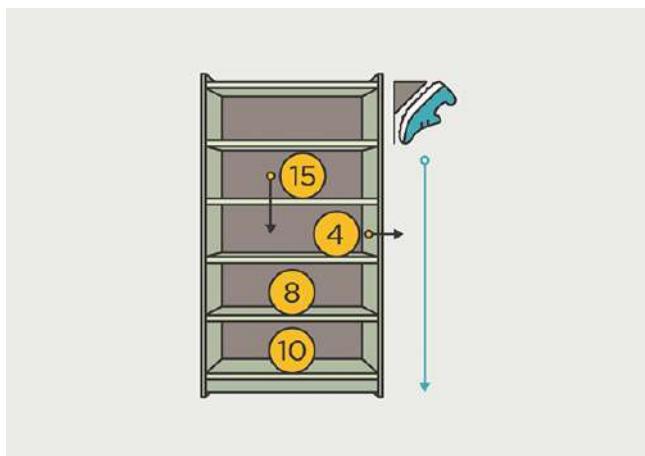


Рисунок 3.169. Обход массива с конца

Тогда, если вы удалите элемент 4, вниз «провалится» элемент с той полки, которую вы уже обрабатывали, — 15. И это не страшно.

Как всё то же самое сделать на встроенном языке? Довольно просто. Но для этого нужно будет использовать ещё один вид инструкции цикла — *Пока Цикл* (листинг 3.59).

Листинг 3.59. Удалить все числа, которые меньше 10. Правильный вариант

```
ТекущийИндекс = Числа.ВГраница();  
Пока ТекущийИндекс >= 0 Цикл  
    Если Числа[ТекущийИндекс] < 10 Тогда  
        Числа.Удалить(ТекущийИндекс);  
  
    КонецЕсли;  
  
    ТекущийИндекс = ТекущийИндекс - 1;  
  
КонецЦикла;
```

В такой форме записи цикл выполняется до тех пор, пока выражение, записанное между *Пока* и *Цикл*, равно *Истина*. Как только это выражение становится равно *Ложь*, выполнение цикла прекращается.

Чтобы обходить цикл, вы создаёте переменную *ТекущийИндекс*. В ней будет находиться индекс того элемента массива, который нужно обработать. Сначала вы помещаете в эту переменную индекс последнего элемента (листинг 3.60).

Листинг 3.60. Помещаете индекс самого последнего элемента

```
ТекущийИндекс = Числа.ВГраница();
```

В конце тела цикла вы каждый раз уменьшаете этот индекс на единицу. Это позволяет вам двигаться от конца к началу (листинг 3.61).

Листинг 3.61. Уменьшаете индекс на единицу

```
ТекущийИндекс = ТекущийИндекс - 1;
```

А в условии цикла вы проверяете, не стал ли текущий индекс отрицательным. Как только он становится меньше нуля, исполнение цикла завершается (листинг 3.62).

Листинг 3.62. Проверяете, не стал ли текущий индекс отрицательным

```
Пока ТекущийИндекс >= 0 Цикл
```

Таким образом вы обходите все индексы от последнего к нулевому. Сделайте этот пример в своей конфигурации. Пройдите его в режиме отладки по шагам, посмотрите, как он работает. Убедитесь, что в результате в массиве останутся только два значения: 10 и 15.

3.10.8 Структура

Ещё одна коллекция, с которой вы познакомитесь, называется *структурой*. Она, как и массив, может содержать в себе набор некоторых значений. Но только если массив это стеллаж, на котором значения-мячики находятся в порядке, друг за другом, то структура похожа на обычную коробку, в которой эти мячики валяются в полном беспорядке (рисунок 3.170).

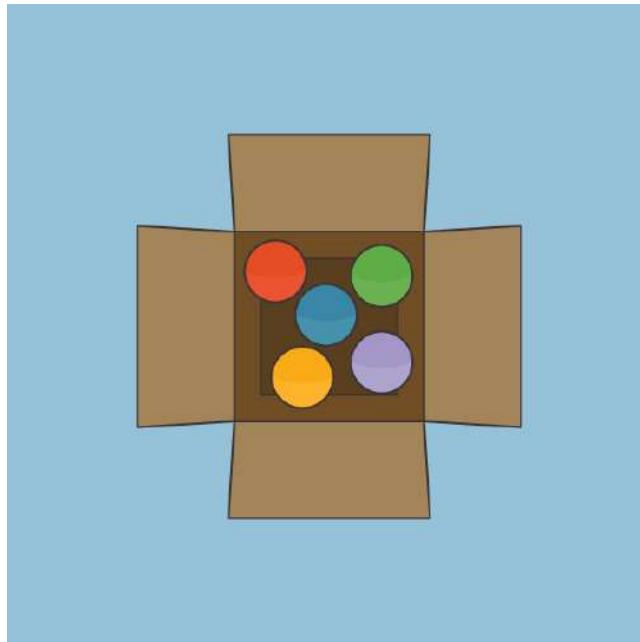


Рисунок 3.170. Структура — мячики в коробке

Как же тогда отличить один мячик от другого, если у них нет никакого порядка? В структуре для этого используется имя. Например, «жёлтый», «красный», «синий» и так далее. Отсюда следует очень простой вывод, что в структуре все имена мячиков-элементов должны быть разными. Повторяться они не могут.

Описание типа *Структура* вы можете найти в синтакс-помощнике рядом с массивом (рисунок 3.171).

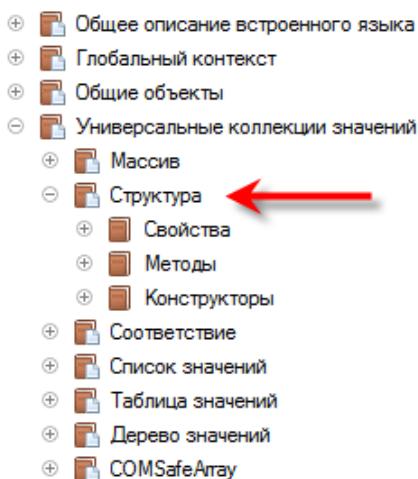


Рисунок 3.171. Тип «Структура»

Значение типа *Структура* — это тоже объект встроенного языка, как и значение типа *Массив*. Поэтому у структуры тоже есть методы и конструкторы. Но, обратите внимание, у структуры есть ещё одна незнакомая вам «вещь», которая называется *Свойства*.

В чём отличие методов от свойств? В том, что, когда вы вызываете метод, объект обязательно выполняет какое-то действие. И в результате, возможно, вернёт вам какое-то значение. А когда вы обращаетесь к свойству, объект не выполняет никаких действий. Но обязательно сообщает вам какую-то информацию о себе, то есть обязательно возвращает некоторое значение.

Если сравнить объект с человеком, то «налей, пожалуйста, чай» — это метод. Человек выполнит действие и даст вам кружку с чаем. Или ответит, что чай закончился. А

«сколько тебе лет?» — это свойство. Человек просто сообщит некоторую информацию о себе.

У структуры есть единственное свойство — это имя ключа (рисунок 3.172).

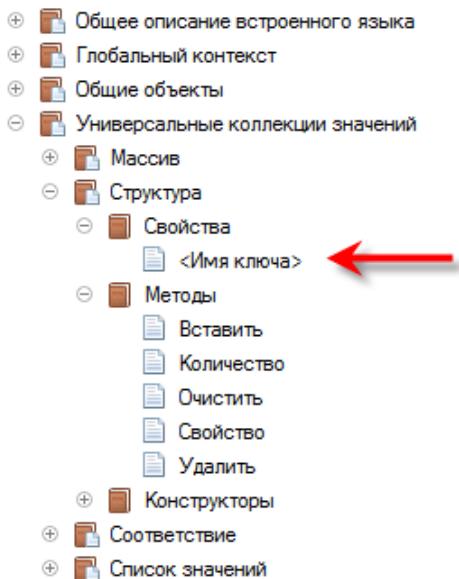


Рисунок 3.172. Свойство «<Имя ключа>»

Почему оно написано в угловых скобках? Ведь имена методов написаны без всяких скобок.

Всё дело в том, что имена методов — это именно те самые имена, которые вы должны использовать в программе. А *<Имя ключа>* — это просто обозначение того, что в программе вы должны будете написать имя того элемента коллекции, к которому хотите обратиться. Ведь заранее неизвестно, какие имена вы придумаете своим элементам.

Чтобы всё прояснилось, сделайте простой пример. Создайте структуру и получите некоторые её элементы (рисунок 3.173).

```

Дневник: Модуль управляемого приложения

Процедура ПриНачалеРаботыСистемы()
    УстановитьКраткийЗаголовокПриложения ("Иванов Петя");

    Урок = Новый Структура();
    Урок.Вставить ("Название", "Математика");
    Урок.Вставить ("НомерКабинета", 311);
    Урок.Вставить ("Начало", Дата(2015, 9, 1, 9, 0, 0));

    НачалоЗанятий = Урок.Начало;
    Предмет = Урок.Название;

КонецПроцедуры

```

Локальные переменные		
Переменная	Значение	Тип
Урок	Структура	Структура
НачалоЗанятий	01.09.2015 9:00:00	Дата
Предмет	"Математика"	Строка

Рисунок 3.173. Структура «Урок»

Запустите 1С:Предприятие в режиме отладки и остановитесь в конце процедуры. Откройте локальные переменные.

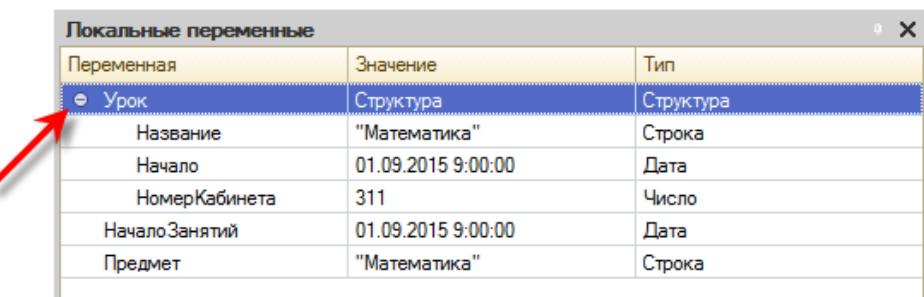
Вы создали структуру и добавили в неё три элемента. Первый элемент имеет имя *Название*. Его значение — это строка с названием предмета. Второй элемент, с именем *НомерКабинета*, имеет значение 311. Это число. Ну, и третий элемент с именем *Начало* — это время начала урока. Функция *Дата()* вам уже знакома.

Теперь вы видите: чтобы узнать начало занятий, нужно обратиться к свойству структуры. Обращение к свойству объекта выглядит точно так же, как к методу, только без круглых скобок в конце. То есть после точки вы пишете имя свойства. В вашем случае это будет имя нужного вам элемента — *Начало*. Ну, а чтобы узнать название предмета, вы обращаетесь к свойству *Название*.

Примечание

Структура — это *именованная коллекция*. То есть её элементы расположены в беспорядке. А обратиться к конкретному элементу можно по имени. И это имя уникально (не повторяется) в пределах одного объекта встроенного языка.

Кстати, обратите внимание на то, что в окне локальных переменных рядом с переменной Урок появился плюсик (рисунок 3.174).



Переменная	Значение	Тип
Урок	Структура	Структура
Название	"Математика"	Строка
Начало	01.09.2015 9:00:00	Дата
НомерКабинета	311	Число
НачалоЗанятий	01.09.2015 9:00:00	Дата
Предмет	"Математика"	Строка

Рисунок 3.174. Просмотр свойств объекта

Это потому, что в этой переменной находится объект, у которого есть свойства. И если вы нажмёте на плюсик, то все свойства вы сможете увидеть прямо в этом окне.

Примечание

Это очень хороший способ знакомства с незнакомыми объектами встроенного языка. Не всегда бывает удобно переходить в синтакс-помощник и читать там о свойствах объекта, который встретился вам в программе.

Зачастую гораздо проще и быстрее прямо в процессе отладки открыть значение этого объекта в окне локальных переменных или в окне вычисления выражения. И сразу же посмотреть, какие свойства есть у этого объекта и чему они равны.

Теперь посмотрите, как выглядит структура «внутри». Выделите переменную Урок и с помощью вычисления выражения посмотрите состав этой коллекции. Чтобы вам было проще понять разницу, рядом я помещу такую же картинку про массив (рисунок 3.175).

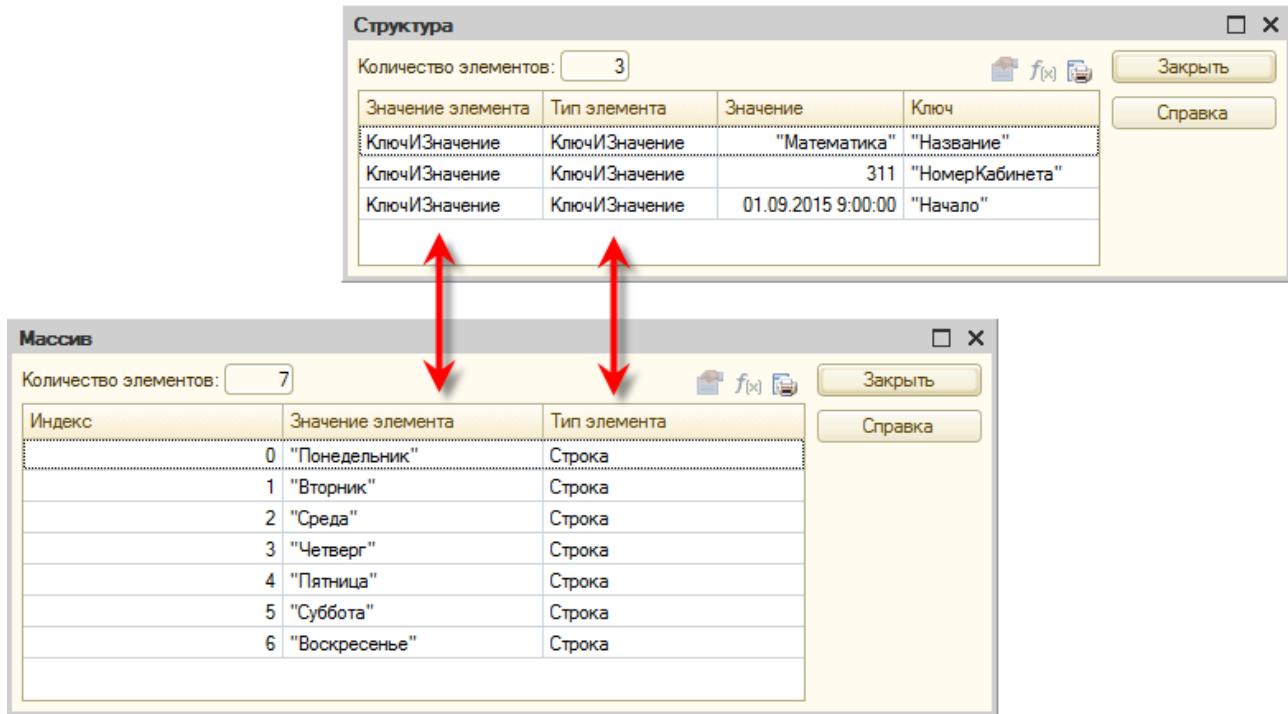


Рисунок 3.175. Содержимое структуры

Структура устроена немного сложнее, чем массив. Элементами массива являлись сами примитивные значения, то есть строки *Понедельник*, *Вторник* и так далее.

Элементами структуры, как вы видите, являются не значения (Математика, 311, 1 сентября), а какие-то *КлючИзначение*. *КлючИзначение* — это тоже объект встроенного языка. Он нужен исключительно для того, чтобы связать вместе некоторое значение и его имя (ключ).

Если вы встанете на первую строку коллекции, где Математика и Название, и нажмёте F2, то платформа покажет вам этот объект *КлючИзначение* в отдельном окне (рисунок 3.176).

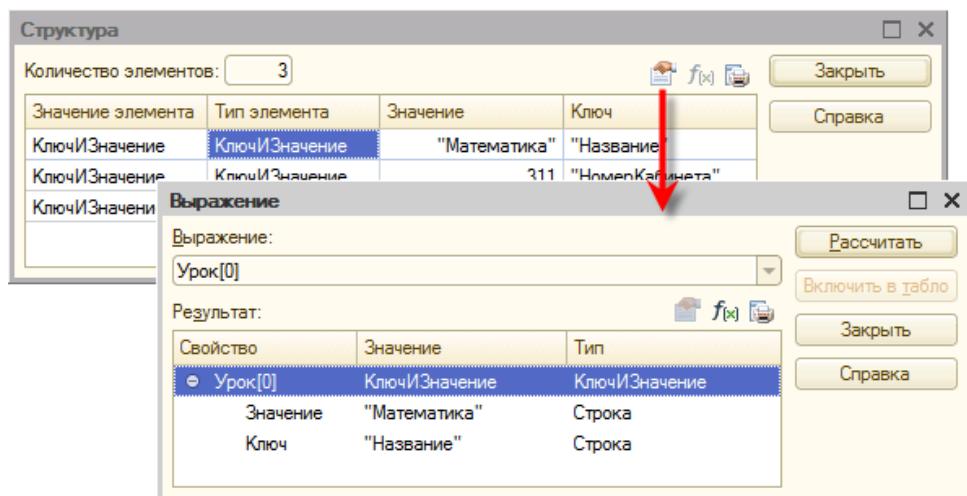


Рисунок 3.176. КлючИзначение

Как видите, у него всего два свойства: значение, которое вы поместили в структуру, и его ключ, то есть имя. По ключу вы можете обратиться к этому значению.

Заметка

Если вы очень внимательны, то, возможно, заметили, что на рисунке 3.176 в строке *Выражение* написано *Урок[0]*. То есть вроде бы к этому элементу структуры можно обратиться по индексу 0. На самом деле это не так. Чтобы обратиться к этому элементу с помощью квадратных скобок, нужно указать его ключ *Урок["Название"]*. Дальше вы с этим познакомитесь.

А здесь, при вычислении выражений, платформа не делает различий между нумерованными и именованными коллекциями. Поэтому она пишет *Урок[0]*, только чтобы обозначить, что в окне вы видите тот элемент коллекции, который в данное время находится на первом месте.

Поэтому если быть совсем точным, то структура выглядит не совсем так, как показано на рисунке 3.170. Скорее, она похожа на большую коробку (*Структура*), в которой лежат маленькие коробочки (*КлючИЗначение*). На каждой маленькой коробочке написано имя, а внутри неё уже находится само значение (рисунок 3.177).

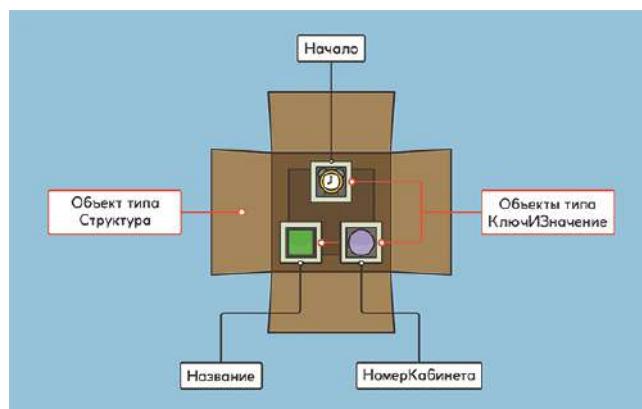


Рисунок 3.177. Элементы структуры — объекты «КлючИЗначение»

Выполните «путешествие внутрь структуры» вы можете не только в режиме отладки, но и в синтакс-помощнике. Закройте все окна вычисления выражения и перейдите в синтакс-помощник. К описанию структуры (рисунок 3.178).

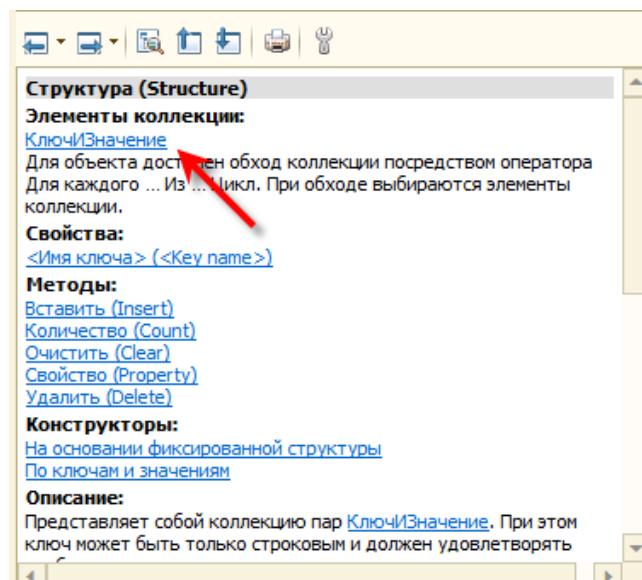


Рисунок 3.178. Описание структуры в синтакс-помощнике

Вы видите, что здесь не просто написано, что элементами коллекции являются объекты *КлючИЗначение*. На этом имени есть гиперссылка. Если вы нажмёте на неё, вы перейдёте к описанию объекта *КлючИЗначение*. А чтобы вернуться обратно, вы можете нажать кнопку в начале командной панели (рисунок 3.179).

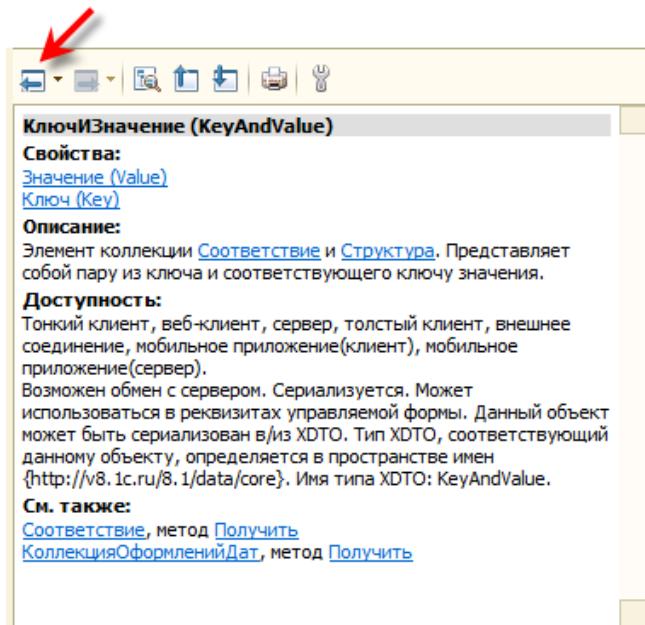


Рисунок 3.179. Описание объекта «КлючИЗначение»

Теперь, когда вы представляете, как устроена структура, познакомьтесь с её методами.

В примере вы создавали пустую структуру, а потом добавляли в неё значения. У структуры, в отличие от массива, есть единственный метод, с помощью которого можно добавить элемент. Это метод *Вставить()*. В нём, как вы уже поняли, нужно указать имя элемента и его значение. Если элемента с таким именем в структуре ещё нет, он появится. А если элемент с таким именем есть, его значение изменится на то, которое указали вы.

Если вы сразу же знаете, какие значения будут в вашей структуре, вы можете создавать не пустую структуру, а структуру со значениями. Их имена и значения нужно указать в конструкторе. Для структуры, которая сейчас в вашем примере, это будет выглядеть так (листинг 3.63). Можете попробовать.

Листинг 3.63. Создание структуры с элементами

```
Урок = Новый Структура("Название, НомерКабинета, Начало", "Математика", 311,
    Дата(2015, 9, 1, 9, 0, 0));
```

В первом параметре перечисляются все имена будущих элементов, а потом через запятую в том же порядке перечисляются значения этих параметров.

Ещё один метод, о котором важно рассказать, называется *Свойство()*. На первый взгляд непонятно, зачем он нужен. Но сделайте простой пример.

Часто бывает так, что структура создаётся в одном месте, а используете вы её совсем в другом. И вы заранее не знаете, будут в ней все элементы, которые вы ожидаете, или нет. В примере с уроком может быть такая ситуация. Не всегда заранее известна фамилия учителя, который будет вести урок. Может быть, он заболел. Может быть, его заменяет другой учитель. Заранее вы этого не знаете.

Но ваша программа ожидает, что в структуре фамилия учителя будет. Поэтому где-то в своей программе вы получаете фамилию учителя из структуры (листинг 3.64).

Листинг 3.64. В структуре нет фамилии учителя

```
// Где-то создается структура урока.
Урок = Новый Структура("Название", НомерКабинета, Начало", "Математика", 311,
    Дата(2015, 9, 1, 9, 0, 0));

// Здесь вы хотите использовать фамилию учителя.
Преподаватель = Урок.ФамилияУчителя;
```

Если в структуре не будет фамилии учителя, вы получите такую ошибку (можете запустить и проверить) (рисунок 3.180).

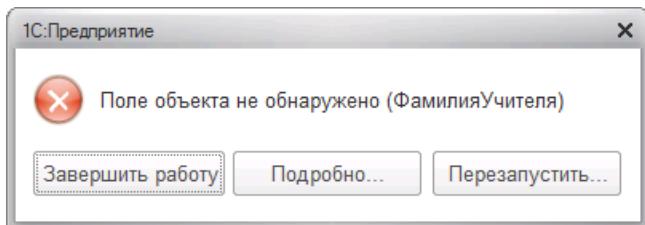


Рисунок 3.180. Ошибка: в структуре нет фамилии учителя

Именно для таких случаев и нужен метод *Свойство()*. Он позволяет избежать ошибки в тех случаях, когда вы не уверены на 100%, что нужный вам элемент есть в структуре (листинг 3.65).

Листинг 3.65. Использование метода «Свойство()»

```
// Здесь вы хотите использовать фамилию учителя.
Преподаватель = Неопределено;
Если НЕ Урок.Свойство("ФамилияУчителя", Преподаватель) Тогда
    ПоказатьПредупреждение(, "Неизвестно, кто ведет урок");
    Возврат;
КонецЕсли;
```

Сначала нужно создать переменную, в которой будет сохранена фамилия учителя. Потом, в условии *Если*, вы вызываете метод *Свойство()*. В параметрах передаёте ему имя элемента, которое надеетесь получить, и переменную, в которую нужно будет поместить значение этого элемента.

Если всё хорошо, то в переменной оказывается значение элемента, *Если* не выполняется и продолжается ваш алгоритм.

А если элемента с таким именем нет, метод вернёт значение *Ложь* и выполнится условие *Если*. В этом условии вы каким-то образом обрабатываете ошибку. Например, выводите сообщение. И если без фамилии учителя продолжение вашей программы невозможно, можете даже написать *Возврат*.

Сделайте этот пример в своей конфигурации и посмотрите, как он работает.

Точно так же, как и любую другую коллекцию, все элементы структуры можно перебрать с помощью инструкции *Для Каждого Цикл*.

Точно так же, как в массиве, в структуре можно обратиться к элементу с помощью операции «квадратные скобки». Только внутри скобок нужно указать не индекс (у элементов структуры нет индексов), а имя.

Например, в вашем примере узнать номер кабинета можно двумя способами (листинг 3.66).

Листинг 3.66. Узнать номер кабинета

```
Комната = Урок.НомерКабинета;  
Комната = Урок["НомерКабинета"];
```

Вторая строка выглядит немного странно, правда? Зачем так усложнять, когда имя можно написать сразу после точки?

Вы абсолютно правы. В подавляющем большинстве случаев именно так и пишут. Точка, а после её имя.

Но есть случаи, когда вы заранее не знаете имя. Например, где-то в форме пользователь указал, что его интересует номер кабинета. И вы запомнили желание пользователя в переменной `ЧтоХочетПользователь`.

Теперь, в другом месте программы, у вас есть только структура `Урок` и только переменная `ЧтоХочетПользователь`. Как получить из структуры нужный пользователю элемент? Посмотрите на листинг 3.67.

Листинг 3.67. Получение элемента структуры с помощью квадратных скобок

```
Урок = Новый Структура();  
Урок.Вставить("Название", "Математика");  
Урок.Вставить("НомерКабинета", 311);  
Урок.Вставить("Начало", Дата(2015, 9, 1, 9, 0, 0));  
  
// Где-то в форме пользователь указал, какая именно информация ему нужна.  
ЧтоХочетПользователь = "НомерКабинета";  
  
// В переменной ЧтоХочетПользователь находится имя того элемента, который ему нужен.  
// Как этот элемент получить из структуры?  
ВернутьПользователю = Урок[ЧтоХочетПользователь];
```

Именно для таких случаев и нужна операция «квадратные скобки» при работе со структурой.

Задание 3.44

Создайте и заполните структуру, в которой будет храниться ваш домашний адрес: улица, номер дома, корпус, квартира, город, район, область, край, республика. Создайте в структуре только те элементы, которые есть в вашем адресе.

Задание 3.45

В конфигураторе посмотрите значения отдельных элементов вашей структуры. Например, улица и город.

Задание 3.46

Используйте структуру из задания 3.44. В одной переменной сформируйте адрес вашего населённого пункта, а в другой — ваш адрес в этом населённом пункте.

Задание 3.47

Создайте структуру, в которой будут содержаться дни рождения двоих ваших самых близких друзей или родственников. Создайте и заполните эту структуру только с помощью конструктора.

Задание 3.48

Структура адреса из примера 3.46 может содержать корпус дома, а может и не содержать его. Сформируйте вторую переменную этого примера (ваш адрес в населённом пункте) так, чтобы программа работала без ошибок как в одном, так и в другом случае.

3.11 Прикладные типы

До сих пор вы учились работать с данными, которые хранятся в оперативной памяти. Переменные, массивы, структуры — все эти данные существуют только в то время, пока выполняется ваша программа, пока работает компьютер.

Но вместе с этим вы видите, что есть и другие данные, которые вы ввели, и они никуда не исчезают. Это списки предметов, учителей, расписание занятий. Эти данные 1С: Предприятие хранит на компьютере специальным образом. И сейчас вы познакомитесь с тем, как «добраться» до этих данных и как с ними работать.

Но прежде я немного расскажу о том, как устроена платформа 1С:Предприятия.

3.11.1 База данных

В большинстве привычных вам компьютеров есть два вида памяти: *оперативная память* и *жёсткий диск* (рисунок 3.181).

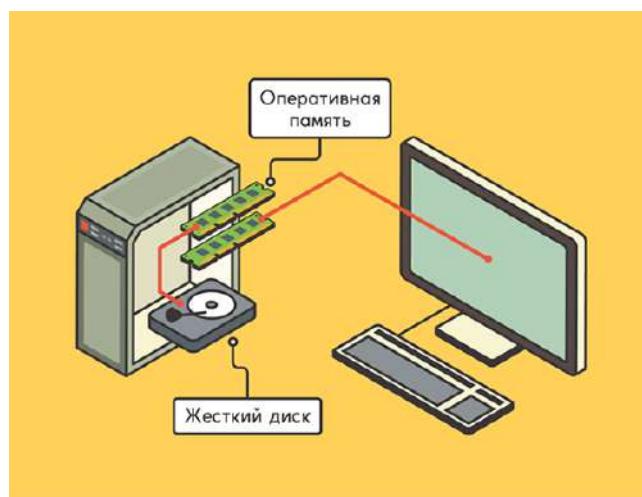


Рисунок 3.181. Оперативная память и жёсткий диск

Вся информация, которая есть на компьютере, хранится на жёстком диске. Он имеет большой объём и может хранить информацию даже тогда, когда компьютер выключен. Но, к сожалению, жёсткий диск работает не очень быстро.

Поэтому любая программа, которую вы запускаете, с жёсткого диска загружается в оперативную память и в ней работает. Оперативная память умеет работать быстро, но, к сожалению, она небольшого объёма. А второй её недостаток заключается в том, что ей обязательно нужно электричество. Если вы выключите компьютер, все данные из оперативной памяти исчезнут.

Те данные, которые нужно хранить долго, 1С:Предприятие записывает на жёсткий диск. Вспомните: во второй части книги, где вы вводили предметы, учителей, расписание

занятий, вы каждый раз нажимали кнопку Записать и закрыть. Именно в этот момент 1С:Предприятие сохраняло ваши данные на жёсткий диск.

Все ваши данные 1С:Предприятие хранит на жёстком диске в специальном виде, который называется *база данных*. База данных представляет собой один файл, который имеет стандартное имя *1Cv8.1CD*. В этом же файле платформа хранит и всю структуру объектов конфигурации, которую вы создали, и все тексты модулей на встроенном языке.

У каждой информационной базы, которую вы создаёте, существует своя собственная база данных. Найти её очень просто.

Во-первых, в списке информационных баз в нижней части окна написан путь к каталогу базы данных (рисунок 3.182).

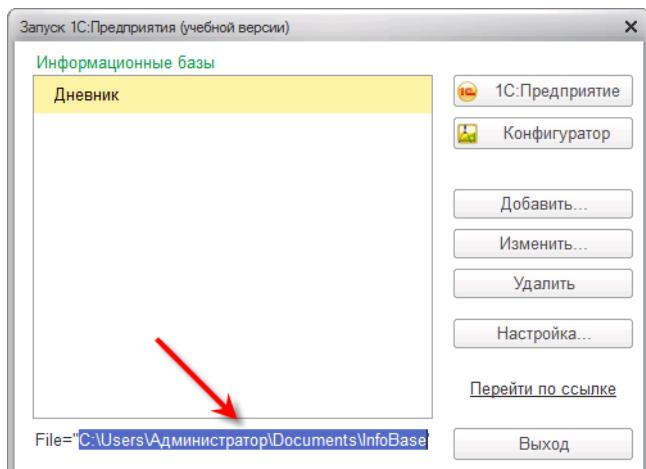


Рисунок 3.182. Путь к каталогу базы данных в списке информационных баз

Вы можете выделить его мышью и скопировать в буфер обмена. А потом просто вставить в проводник и перейти к этому каталогу.

Во-вторых, когда вы находитесь в конфигураторе или работаете в режиме 1С:Предприятие, вы всегда можете открыть информацию о программе. Для этого нужно нажать на кнопку «i» в командной панели (рисунок 3.183).

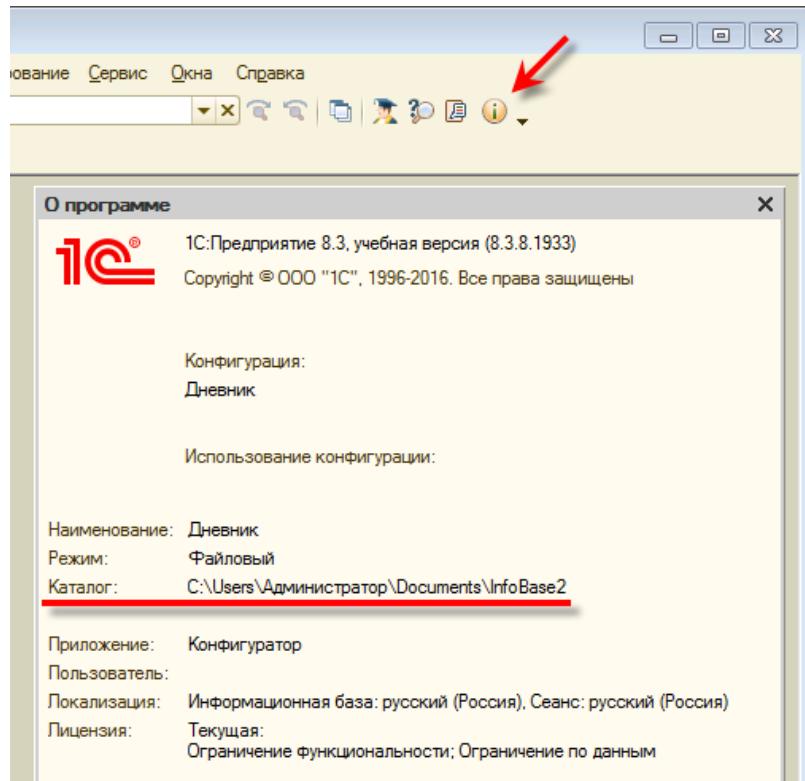


Рисунок 3.183. Путь к каталогу базы данных в окне «О программе»

Путь, указанный в диалоге *О программе*, вы тоже можете скопировать с помощью мыши.

В каталоге, к которому ведёт этот путь, будет находиться не только файл базы данных *1Cv8.1CD*, но и несколько других служебных файлов и каталогов (рисунок 3.184).

Имя	Тип	Размер
1Cv8FTxt	Папка с файлами	
1Cv8JobScheduler	Папка с файлами	
1Cv8Log	Папка с файлами	
1Cv8	Файловая информационная база 1C:Предприятия 8	1 532 КБ
1Cv8.1CD.cfl	Файл "CFL"	0 КБ
1Cv8	Блокировки файловой информационной базы 1C:Предприятия 8	0 КБ
1Cv8.1CL.cfl	Файл "CFL"	0 КБ
1Cv8.cgr	Файл "CGR"	0 КБ
1Cv8.cgr.cfl	Файл "CFL"	0 КБ
1Cv8tmp	Файловая информационная база 1C:Предприятия 8	540 КБ
1Cv8tmp.1CD.cfl	Файл "CFL"	0 КБ
1Cv8tmp	Блокировки файловой информационной базы 1C:Предприятия 8	0 КБ
1Cv8tmp.1CL.cfl	Файл "CFL"	0 КБ

Рисунок 3.184. Каталог информационной базы

Все эти файлы нужны, конечно, во время работы платформы. Но их наличие не является обязательным. Если при запуске прикладного решения платформа вдруг не обнаружит какой-то из этих файлов или каталогов, она создаст его автоматически. Главное, чтобы был файл базы данных *1Cv8.1CD*.

Для чего я так подробно рассказываю вам, где найти файл базы данных?

Во-первых, правилом хорошего тона является периодическое копирование этого файла в «безопасное» место. Например, на другой диск или на флеш-накопитель — «флешку».

Потому что любой компьютер может сломаться, и если у вас не останется копии этого файла, то это будет значить, что все ваши данные потеряны навсегда. Такие копии программисты называют «бэкап», от английского *back up* — выполнять резервное копирование.

Во-вторых, после установки новой, более современной, версии платформы может потребоваться реструктуризация базы данных. Это сложная и ответственная операция. Поэтому платформа, прежде чем выполнять её, всегда предупреждает вас о том, что необходимо сделать резервную копию.

И, в-третьих, копия файла базы данных может понадобиться вам для того, чтобы сделать ещё одну, такую же, информационную базу для каких-нибудь упражнений или тестов. Даже если вы её «испортите», то ничего страшного: основная информационная база у вас не пострадает. А эту, тестовую, вы просто удалите. Кстати, в одном из следующих примеров вам как раз понадобится такая «тестовая» база, чтобы не испортить те данные, которые вы уже занесли в свой электронный дневник.

Заметка

1С:Предприятие поддерживает работу с разными базами данных. Но в рамках этой книги рассматривается только файловая база данных 1С:Предприятия.

3.11.2 Клиент и сервер

До сих пор вы не интересовались тем, что работает в оперативной памяти вашего компьютера, когда вы запускаете 1С:Предприятие. Я просто говорил, что это некоторая программа.

На самом деле платформа 1С:Предприятия — это не одна программа, а совокупность нескольких программ. Эти программы взаимодействуют друг с другом. Сейчас, на вашем компьютере, это незаметно. Потому что вы используете самый простой вариант работы — файловый. На больших предприятиях используется другой режим работы — клиент-серверный.

Особенность платформы заключается в том, что прикладные решения разрабатываются одним и тем же образом независимо от того, в каком режиме они будут работать в дальнейшем. То есть вашу конфигурацию, электронный дневник, можно запустить и в клиент-серверном режиме работы. При этом в ней ничего переделывать или доделывать не придётся.

Это очень хорошая и удобная особенность платформы. Но она приводит к тому, что принципы клиент-серверного взаимодействия нужно понимать всем. И тем, кто делает сложные программы, и тем, кто, как вы, делает самые простые программы для работы в файловом варианте.

Итак, начните знакомиться с этим по порядку.

Как я уже говорил, платформа 1С:Предприятия — это несколько программ. Их можно разделить на три главных блока: *клиентское приложение*, *сервер 1С:Предприятия* и *система управления базой данных* (рисунок 3.185).

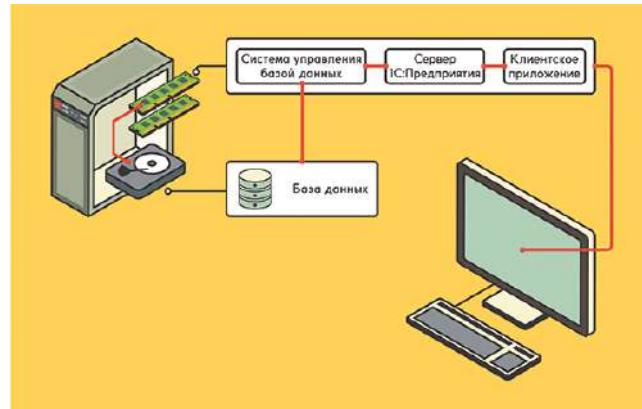


Рисунок 3.185. Клиент-серверная архитектура

Клиентское приложение — это та программу, которую вы запускаете. В результате у вас получается или конфигуратор, или сеанс работы в режиме отладки, или обычный сеанс 1С:Предприятия.

Сервер 1С:Предприятия — это та часть платформы, которая позволяет большому количеству пользователей одновременно работать с одной и той же информационной базой. Сейчас только вы работаете со своим электронным дневником — вносите туда данные, изменяете их. Но на больших предприятиях, в больших учётных системах данные вносят и изменяют одновременно многие сотрудники предприятия.

Система управления базой данных (СУБД) — это та часть платформы, которая записывает данные в базу данных и получает их оттуда.

Цепочка взаимодействия этих частей платформы выглядит следующим образом. Например, вы захотели посмотреть, какие завтра занятия. Для этого вы используете клиентское приложение и, например, дважды щёлкаете мышью на завтрашнем учебном дне. Клиентское приложение обращается с этим запросом к серверу 1С:Предприятия. Сервер 1С:Предприятия передаёт этот запрос СУБД в том виде, в котором он ей понятен. СУБД выбирает из базы данных нужную информацию.

После этого всё «идёт» в обратную сторону. СУБД передаёт данные серверу 1С:Предприятия. Сервер 1С:Предприятия отдаёт их клиентскому приложению в том виде, который будет понятен для вас. И, наконец, клиентское приложение показывает их вам на экране.

Сейчас, на вашем компьютере, в файловом варианте работы, эти три части платформы практически не видны. Потому что сервер 1С:Предприятия и файловая СУБД «спрятаны» внутри вашего клиентского приложения. Но в клиент-серверном варианте все эти три части платформы очень хорошо видны. Чаще всего они располагаются на разных компьютерах. И эти компьютеры могут находиться даже не в разных частях одного здания, а в разных частях света (рисунок 3.186).

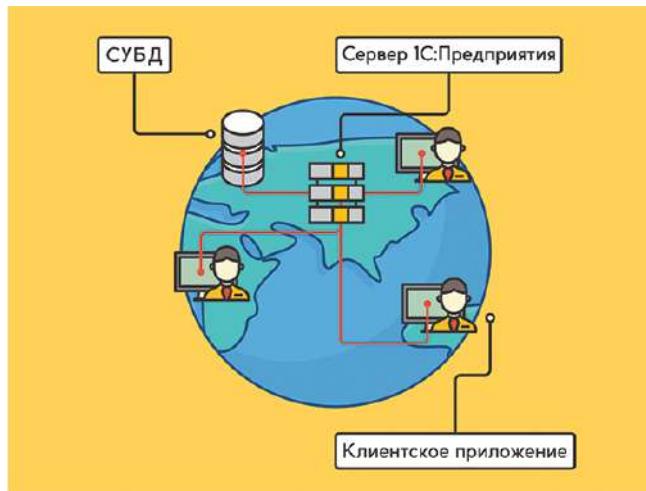


Рисунок 3.186. Клиент-серверный вариант работы

Практически все программы на встроенном языке (за небольшим исключением) всегда начинают работать в клиентском приложении, или «на клиенте». Если им нужно получить какие-то данные из базы данных, они переходят на сервер 1С:Предприятия, или просто «на сервер». После того как они выполнили все операции с данными, они возвращаются на клиент (рисунок 3.187).



Рисунок 3.187. Исполнение встроенного языка на клиенте и на сервере

Глядя на рисунок 3.186, можно хорошо увидеть, что встроенный язык имеет разные возможности на клиенте и на сервере. Например, на клиенте нет базы данных. Поэтому что-то прочитать из неё или записать в неё невозможно. Для этого обязательно нужно перейти на сервер.

С другой стороны, на сервере практически отсутствуют средства взаимодействия с пользователем. Для того чтобы показать что-то пользователю или вывести ему какое-нибудь оповещение, нужно вернуться на клиент.

Поэтому, когда вы пишете программу на встроенном языке, вы всегда должны точно представлять, где будет исполняться та или иная её часть: на клиенте или на сервере. Или, говоря другими словами, в каком контексте она будет исполняться: в *контексте клиента* или в *контексте сервера*.

Некоторые модули, например модуль управляемого приложения, в котором вы выполняли все предыдущие задания, существуют только в единственном контексте. Модуль управляемого приложения исполняется только в контексте клиента. На сервере он никогда не бывает.

Но другие модули могут существовать в разных контекстах, и вам нужно в явном виде указывать, где должен исполняться тот или иной модуль.

Более того, некоторые модули устроены так, что могут исполняться и на клиенте, и на сервере. В таких модулях вы должны будете для каждой процедуры или функции отдельно указывать место её исполнения.

Сейчас это может показаться вам сложным, но не пугайтесь. На самом деле всё очень просто, и скоро вы в этом убедитесь.

В зависимости от того, на клиенте или на сервере исполняется ваш фрагмент программы, вы можете использовать те или иные типы, методы, функции. Или не можете их использовать. Разобраться в этом поможет синтакс-помощник.

Во-первых, для каждого типа, метода, свойства в нём указано, в каком контексте его можно использовать. Например, если вы откроете хорошо знакомый вам тип **Массив**, то в разделе *Доступность* увидите следующее (рисунок 3.188).

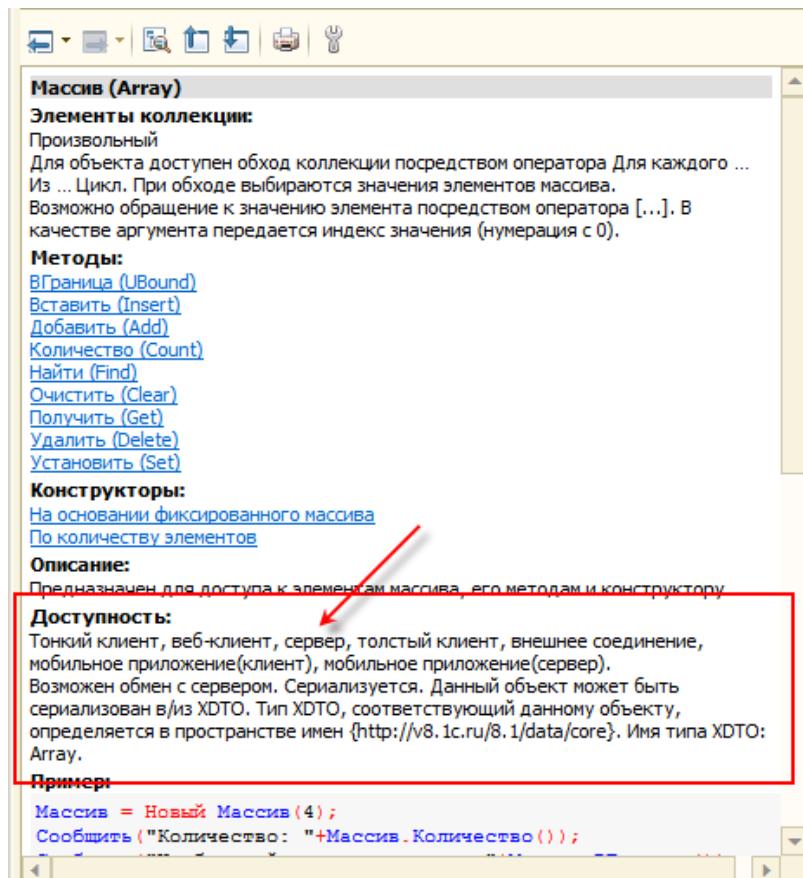


Рисунок 3.188. Доступность типа «Массив»

Слово «сервер» означает, что этот тип можно использовать на сервере. «Тонкий клиент», «веб-клиент», «внешнее соединение» — это разные виды клиентских приложений. Мобильное приложение я в рамках этой книги рассматривать не буду, поэтому не обращайте на него внимания.

Таким образом, как только вы решите использовать тот или иной тип в своей программе, вы всегда можете посмотреть, доступен ли он в том контексте, в котором вы сейчас находитесь.

С помощью синтакс-помощника можно ответить и на другой вопрос: «Какие типы можно использовать на сервере?» Для этого нужно открыть настройки синтакс-помощника и установить нужный контекст. Чтобы открыть настройки, нажмите на кнопку «гаечный ключ» в командной панели синтакс-помощника (рисунок 3.189).

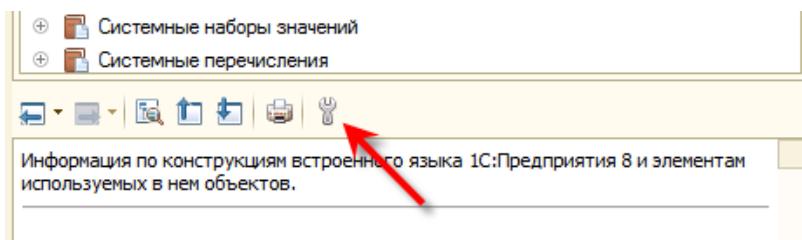


Рисунок 3.189. Открыть режим настройки параметров

Установите только флагок Сервер и нажмите OK (рисунок 3.190).

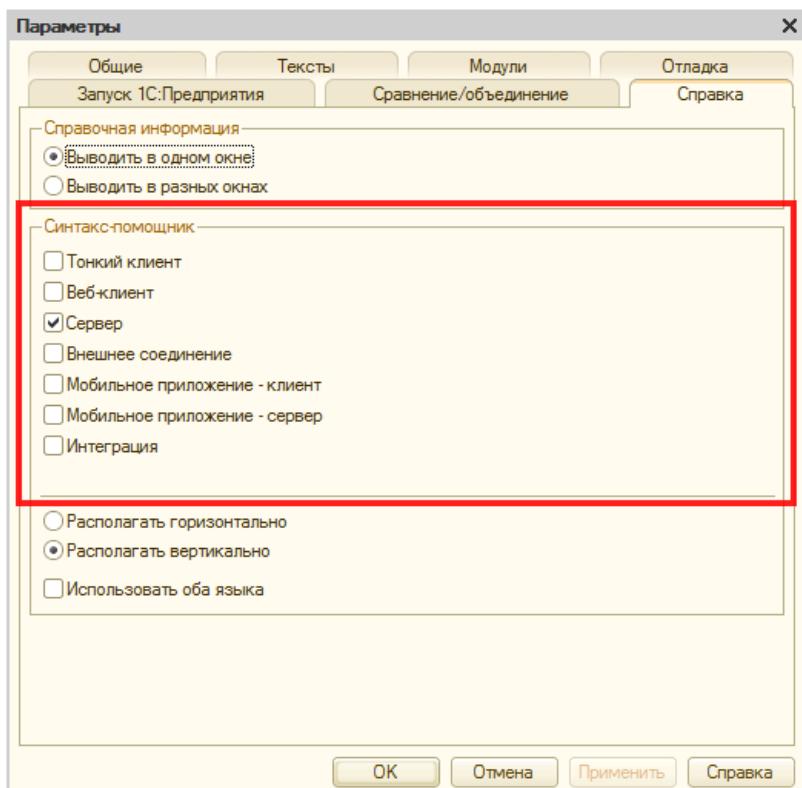


Рисунок 3.190. Установка контекста, отображаемого синтакс-помощником

Если теперь вы раскроете ветку *Прикладные объекты* и посмотрите, какие типы доступны для работы с документами, вы увидите там 7 типов. Все эти типы позволяют читать, выбирать документы из базы данных, изменять их и записывать обратно. Естественно, всё это возможно только на сервере.

А процедур и функций для интерактивной работы будет всего три (рисунок 3.191).

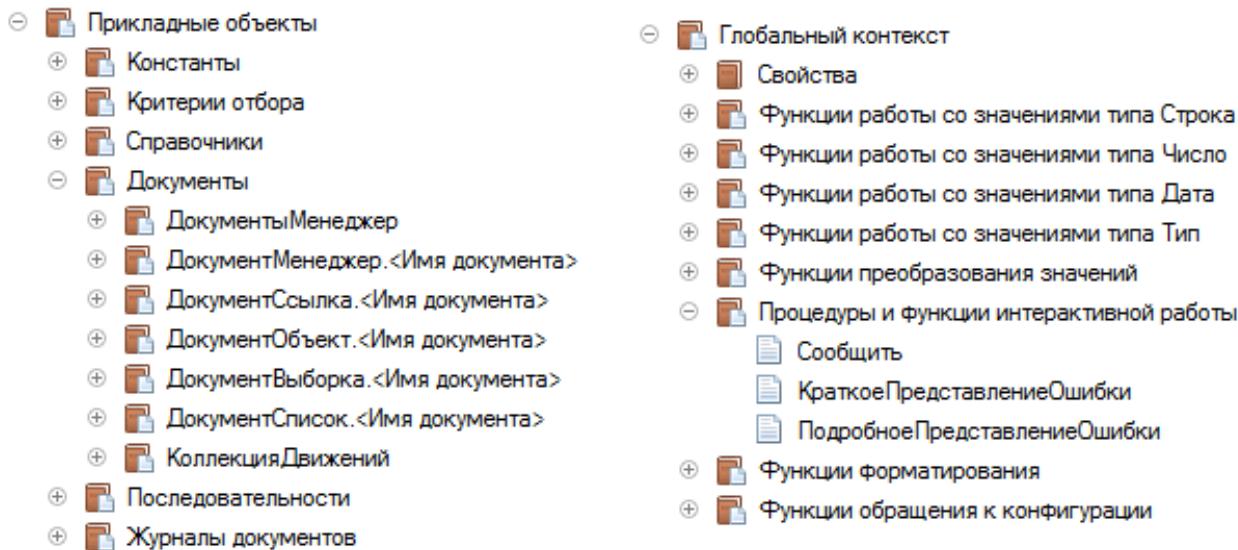


Рисунок 3.191. Контекст сервера

Снова откройте настройки синтакс-помощника и теперь установите только флажок **Тонкий клиент**. Тип для работы с документами останется всего один — **ДокументСсылка**. На клиенте он используется только для того, чтобы показать список документов или выбрать документ в поле ввода.

Но зато интерактивных процедур и функций заметно прибавится (рисунок 3.192).

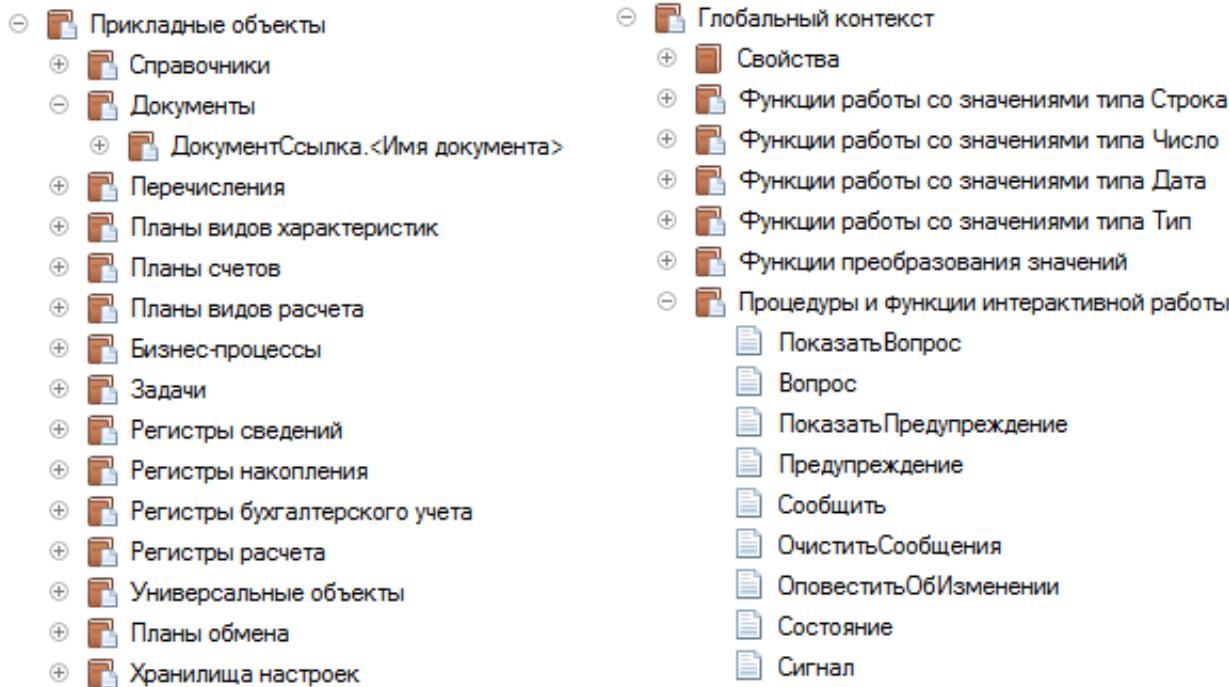


Рисунок 3.192. Контекст тонкого клиента

Таким образом, вы видите, что клиентский и серверный контексты значительно отличаются друг от друга.

Для выполнения примеров из этой книги вам будет удобнее видеть в синтакс-помощнике оба контекста: и клиентский, и серверный. Поэтому ещё раз вызовите настройки синтакс-помощника и установите все флажки, кроме двух, относящихся к мобильному приложению. Нажмите **OK**.

Подробнее

Подробнее вы можете прочитать про обозначение доступности типов в документации «[Руководство разработчика 8.3. Раздел 4.8.2. Особенности использования объектов, их свойств и методов](#)».

3.11.3 Прикладные типы

Для работы с данными, хранимыми в базе данных, существует большой набор типов. Все их вы можете найти в синтакс-помощнике в ветке *Прикладные объекты* (рисунок 3.193).

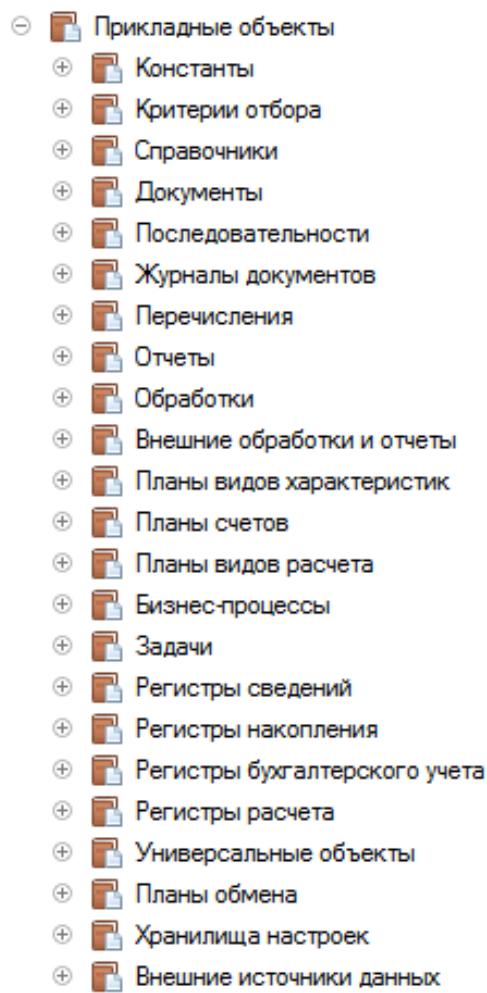


Рисунок 3.193. Прикладные типы

Как вы можете заметить, названия веток в этом разделе соответствуют тем объектам конфигурации, которые вы можете добавлять: *Справочники*, *Документы* и так далее. Именно поэтому все эти типы называются «прикладными».

Если вы раскроете, например, ветку *Справочники*, то увидите, что большинство типов имеют очень странное обозначение с использованием косых скобок (рисунок 3.194).

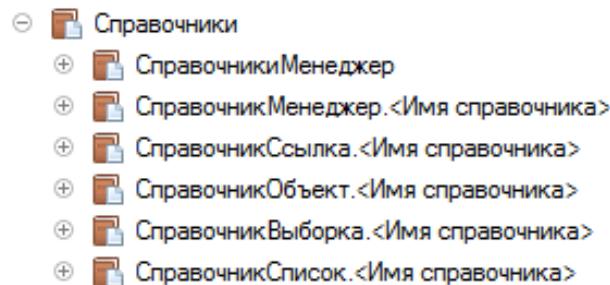


Рисунок 3.194. Типы для работы со справочниками

Это потому, что они описывают лишь общую «структуру» такого типа, правила, по которым будут формироваться типы для работы со справочниками. В каждой конфигурации, каждом прикладном решении они будут свои собственные, уникальные.

Пока вы не добавили в дерево объектов конфигурации ни одного справочника, будет существовать только тип *СправочникиМенеджер*. Больше никаких типов не будет.

Но как только вы добавите справочник Учителя, тут же появятся типы (рисунок 3.195):

- СправочникМенеджер.Учителя;
- СправочникСсылка.Учителя;
- СправочникОбъект.Учителя;
- СправочникВыборка.Учителя;
- СправочникСписок.Учителя.

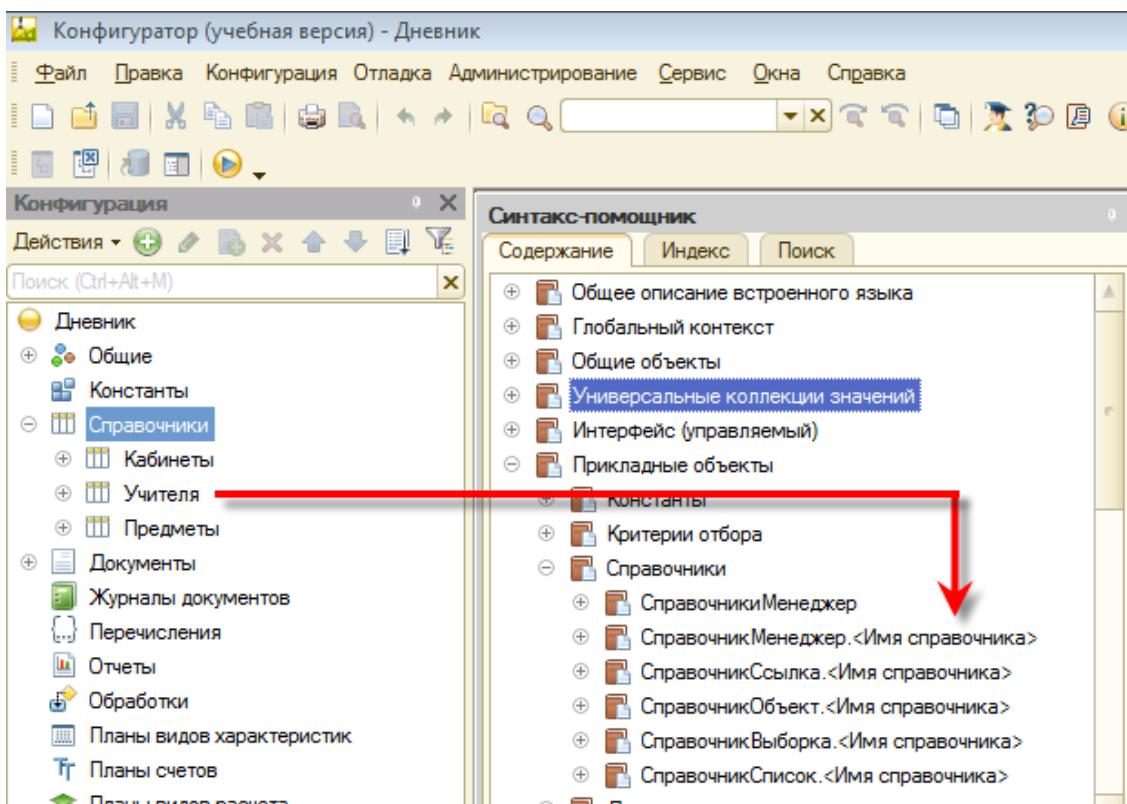


Рисунок 3.195. Типы для работы со справочником «Учителя»

Тут у вас наверняка возникнет два вопроса. Во-первых, почему для работы с одним справочником нужно целых пять разных типов? А во-вторых, почему для работы со справочником Учителя и со справочником Предметы нужны разные типы?

Пять разных типов нужны потому, что они предназначены для разных задач.

Тип *СправочникМенеджер.Учителя* нужен для того, чтобы создавать новые элементы, находить существующие элементы, выбирать все или некоторые элементы справочника.

Тип *СправочникСсылка.Учителя* нужен для того, чтобы отображать элементы справочника в интерфейсе: в списках, в полях ввода. Также он нужен для того, чтобы указывать на конкретный элемент справочника.

Тип *СправочникОбъект.Учителя* нужен для того, чтобы изменять элемент справочника.

Тип *СправочникВыборка.Учителя* нужен для того, чтобы последовательно, друг за другом, перебирать те элементы, которые были выбраны из базы данных.

А тип *СправочникСписок.Учителя* — это наследие прошлых версий платформы, которое сейчас практически нигде не используется.

Теперь о том, почему при добавлении в дерево объектов конфигурации нового справочника *Предметы* появляются новые типы. Проще всего это будет понять на примере типа *СправочникОбъект.<Имя справочника>*.

Если вы раскроете свойства этого типа, то увидите, что одно из свойств обозначается *<Имя реквизита>* (рисунок 3.196).

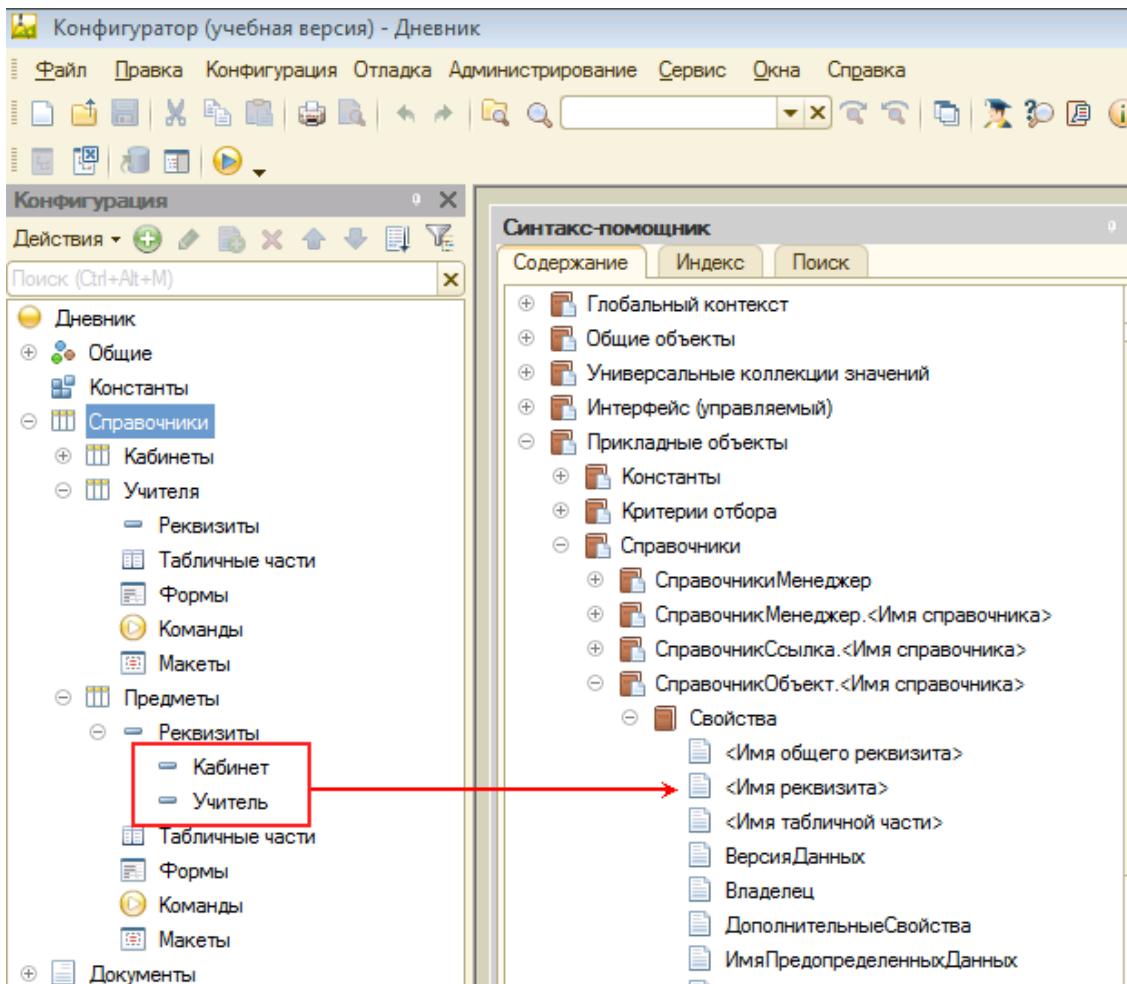


Рисунок 3.196. Свойства типа «СправочникОбъект»

Это значит, что у объекта встроенного языка, имеющего тип *СправочникОбъект.Предметы*, будет два свойства: *Кабинет* и *Учитель*. А у справочника *Учителя* реквизитов нет. Значит, у объекта типа *СправочникОбъект.Учителя* таких свойств не будет.

И это только часть отличий. А ещё есть табличные части. А ещё реквизиты с одними и теми же именами могут иметь разные свойства у разных объектов. Например, реквизит

Примечание у справочника Учителя может быть строкой неограниченной длины. Чтобы сохранить любую дополнительную информацию об учителе. А у справочника Предметы реквизит с таким же именем будет длиной всего 20 символов. Только для того, чтобы отметить, что этот предмет является не основным, а дополнительным.

Поэтому хорошо и правильно, что платформа создаёт для каждого объекта конфигурации свой собственный уникальный набор типов встроенного языка.

3.11.4 Объектные данные

Все данные, которые 1С:Предприятие сохраняет в базе данных, можно разделить на две большие группы: *объектные данные* и *необъектные данные*. Разницу между ними легко будет понять, если вспомнить, что в самом начале книги вы сравнивали объекты конфигурации с комнатами. А данные этих объектов конфигурации вы сравнивали с людьми, которые находятся в этих комнатах (рисунок 3.197).

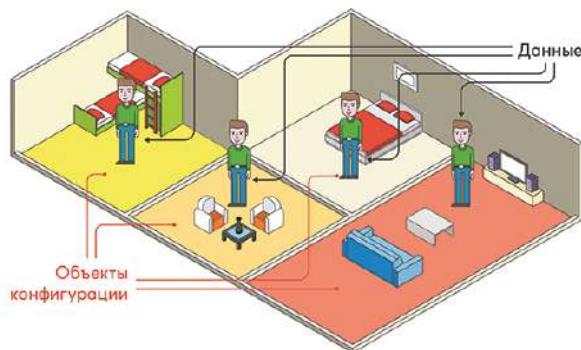


Рисунок 3.197. Объекты конфигурации — комнаты, люди — данные

Так вот. Бывают ситуации, когда для вас важно, какие именно это люди. Например, вы позвали гостей на день рождения. В этом случае для вас важно, чтобы к вам в гости пришёл именно Серёжа Соколов, который учится или работает вместе с вами. А не какой-то его тёзка или однофамилец. Который, может быть, и одет так же, и внешне на него похож.

В этом случае ваши гости являются объектными данными. Каждый из них уникален и важен сам по себе. Даже если он оделся в другую одежду и сильно вырос за то время, что вы его не видели. Всё равно он остался Серёжей Соколовым, а не стал кем-то другим (рисунок 3.198).



Рисунок 3.198. Гости — объектные данные

Другая ситуация. Вы заходите в маршрутку или автобус, и вам нужно понять, много там пассажиров или нет. Поедете вы на этом автобусе или подождёте следующего. В этом случае вам совершенно не важно, какие именно люди находятся в автобусе, как их зовут. Не важно, кто именно стоит в проходе с большими сумками, Сергей Николаевич или Марья Ивановна. Важно только то, что проход занят и дальше пройти нельзя.

В этом случае пассажиры автобуса являются необъектными данными. Их уникальность не имеет для вас никакого значения. Важно лишь, что они есть и что они занимают место в автобусе (рисунок 3.199).

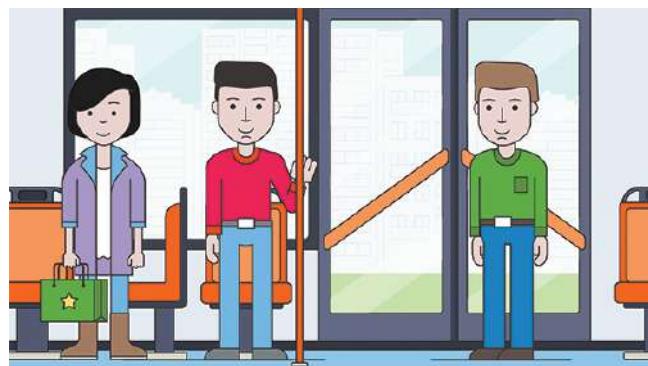


Рисунок 3.199. Пассажиры — необъектные данные

Какие данные в 1С:Предприятии объектные, а какие необъектные, понять очень просто. Это зависит от того объекта конфигурации, которому они принадлежат.

Если это справочник или документ, то в них всегда хранятся только объектные данные.

Если это регистр сведений или регистр накопления (с ними вы познакомитесь чуть позже), то в них бывают только необъектные данные.

Заметка

Кроме справочников и документов для описания объектных данных используются планы видов характеристик, планы счетов, планы видов расчёта, бизнес-процессы, задачи, планы обмена.

Для описания необъектных данных кроме перечисленных регистров используются регистры бухгалтерии, регистры расчёта, последовательности и перерасчёты регистров расчёта.

Итак, вернёмся к объектным данным. «Паспорт», «идентификатор», по которому можно отличить один объект данных от другого, — это его ссылка. Во встроенном языке для работы со ссылками используются типы *СправочникСсылка.<Имя справочника>*, *ДокументСсылка.<Имя документа>* и так далее. С этими типами вы уже знакомы.

Вспомните: когда вы добавляли в конфигурацию справочник *Предметы*, вы создавали у него реквизит *Учитель*. Он нужен для того, чтобы указать, какой именно учитель проводит занятия по этому предмету. И в качестве типа этого реквизита вы выбирали тип *СправочникСсылка.Учителя* (рисунок 3.200).

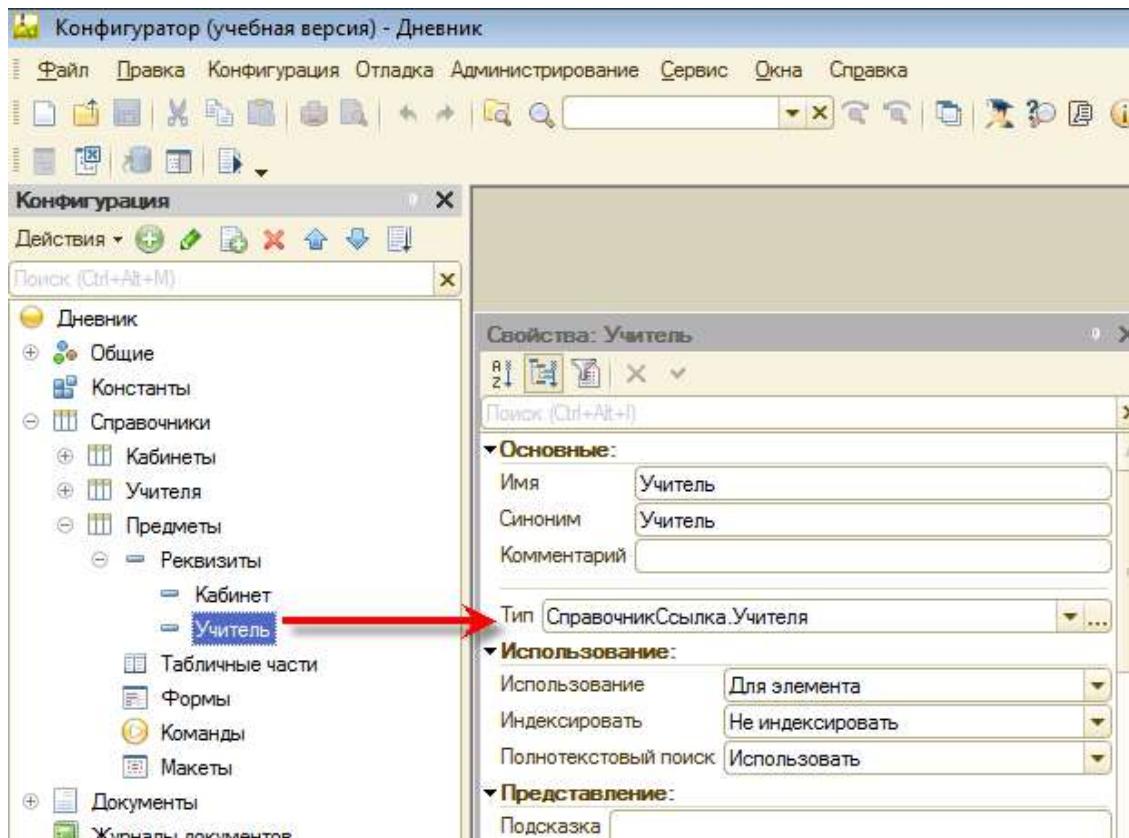


Рисунок 3.200. Тип реквизита «СправочникСсылка.Учителя»

Именно благодаря тому, что вы выбрали такой тип, вы имеете возможность выбирать в этом реквизите одного конкретного учителя из нескольких (рисунок 3.201).

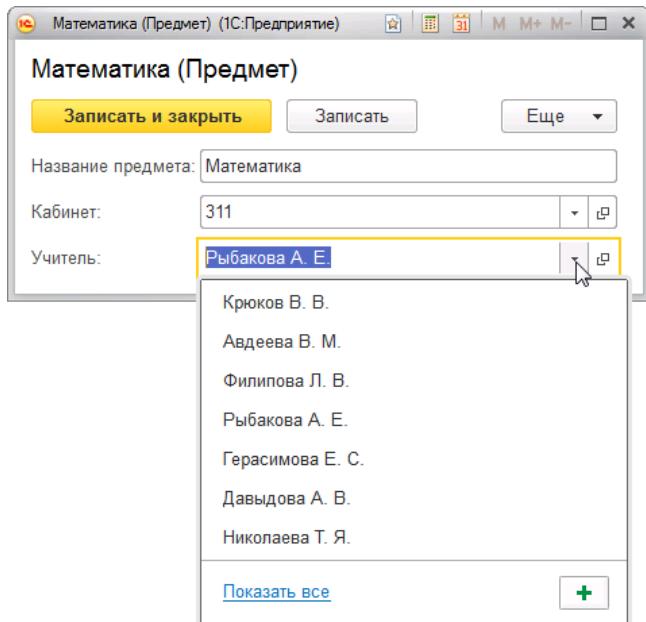


Рисунок 3.201. Выбор одного учителя из нескольких

Подробнее

Подробнее о работе с объектными данными вы можете прочитать в статье информационно-технологического сопровождения (ИТС) «[Особенности использования типов данных, предназначенных для манипулирования объектами базы данных](#)».

3.11.5 Как устроен документ

Информационная база

Если в вашей информационной базе нет данных (кабинетов, учителей, учебных дней), вы можете для выполнения дальнейших примеров взять демонстрационную базу «03 Визуальное Конструирование Итог.dt». Как её подключить, написано в разделе [А.1 «Как подключить демонстрационную базу»](#) на странице 543.

Теперь, когда вы знаете о базе данных, сервере и объектных данных, вы можете посмотреть, как до них «добраться» и как они выглядят.

Принципы работы со всеми объектными данными одинаковы. Вы будете знакомиться с ними на примере документа *УчебныйДень*. Он выбран ещё и потому, что прямо в следующей главе вы сделаете небольшую доработку в вашей конфигурации, которая значительно облегчит его использование.

Итак, чтобы познакомиться с документами, вам нужно попасть на сервер. До сих пор вы выполняли все задания в модуле управляемого приложения. Этот модуль всегда используется только на клиенте. Чтобы из него попасть на сервер, вы добавите в конфигурацию модуль, который исполняется только на сервере. А затем из модуля управляемого приложения вызовете процедуру, которую напишете в этом серверном модуле. На схеме это будет выглядеть следующим образом (рисунок 3.202).

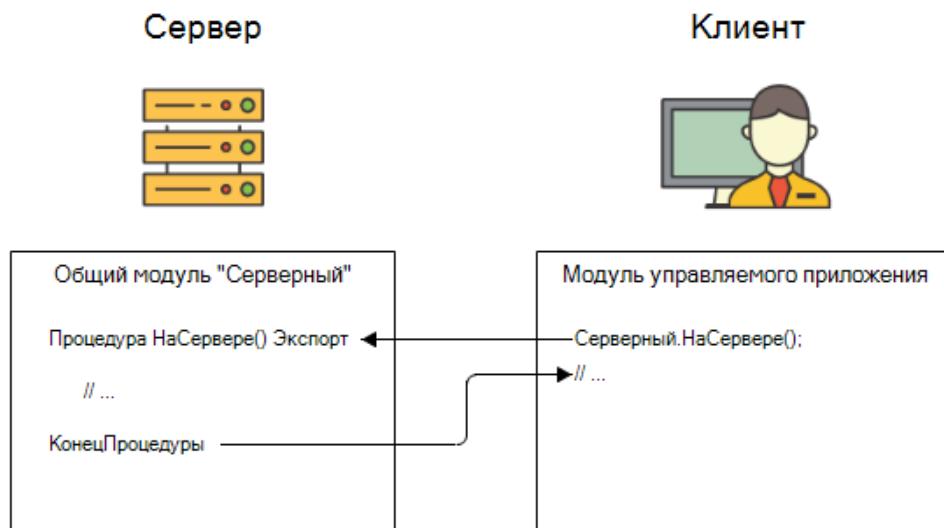


Рисунок 3.202. Передача исполнения с клиента на сервер

Таким модулем, который исполняется только на сервере, будет *общий модуль*. Общие модули не существуют в конфигурации заранее. Вы их создаёте сами. Столько, сколько вам нужно. Общие модули доступны из всех других модулей конфигурации. В общих модулях вы можете описывать собственные алгоритмы, которые вы используете в разных местах конфигурации.

Общему модулю можно указать, где он будет исполняться: на клиенте или на сервере. Поэтому одно назначение общих модулей заключается в том, что именно с их помощью вы можете перейти на сервер. Например, чтобы получить какие-то данные из базы данных. Этим вы сейчас и займётесь.

Общие модули располагаются в ветке конфигурации *Общие*. Добавьте в конфигурацию общий модуль и назовите его *Серверный* (рисунок 3.203).

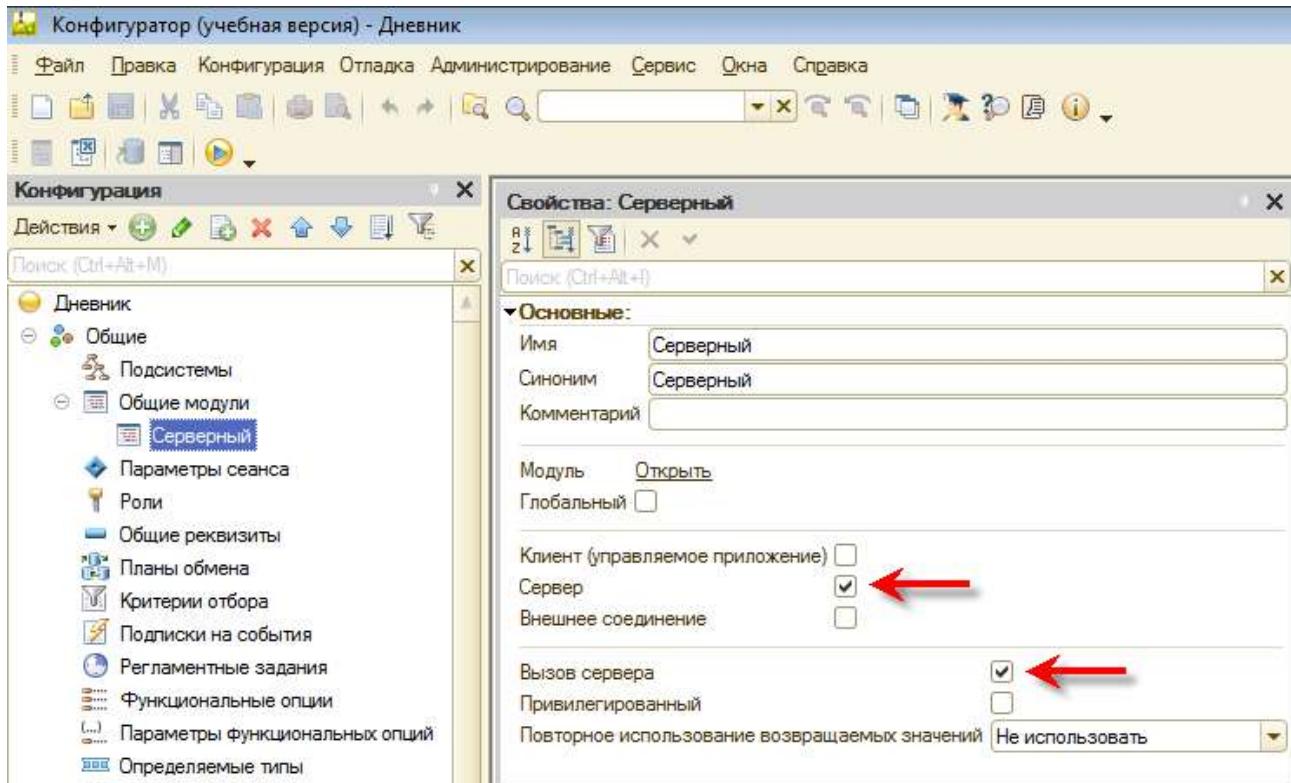


Рисунок 3.203. Общий модуль «Серверный»

После добавления у него стандартно уже будет установлен флажок *Сервер*. Это означает, что все процедуры и функции, написанные в этом модуле, будут исполняться на сервере. Это то, что вам нужно.

Но кроме этого вам ещё нужно попасть в этот модуль с клиента. А для этого вам понадобится установить флажок *Вызов сервера*. Именно он позволит вызывать процедуры и функции этого модуля с клиента. Установите этот флажок.

Общий модуль уже открыт у вас. Напишите в нём пустую процедуру с именем *НаСервере*. У этой процедуры не будет параметров. А в описании процедуры, после круглых скобок, добавьте через пробел слово *Экспорт* (листинг 3.68).

Листинг 3.68. Пустая процедура «НаСервере()»

Процедура НаСервере() Экспорт

КонецПроцедуры

Ключевое слово *Экспорт*, которое вы здесь использовали, позволяет расширить область видимости этой процедуры. Без этого слова процедура видна только в пределах этого модуля. А с этим словом процедура станет видна и из других модулей вашей конфигурации.

Обновите конфигурацию базы данных и установите точку останова на строке *КонецПроцедуры*. Пока вам этот модуль больше не понадобится.

Откройте модуль управляемого приложения. Если у вас там остался какой-то текст от предыдущих примеров, можете его удалить — кроме первой строки, естественно. Установка краткого заголовка приложения пускай остаётся.

После этой инструкции нужно вызвать процедуру из общего модуля. Делается это просто. Сначала вы пишете имя общего модуля, а затем, через точку, имя процедуры.

Контекстная подсказка вам поможет. В результате у вас получится такая инструкция (листинг 3.69).

Листинг 3.69. Вызов процедуры из общего модуля

```
Серверный.НаСервере();
```

Отлично. Теперь у вас всё готово для того, чтобы попасть на сервер. Запустите 1С: Предприятие в режиме отладки. Вы остановитесь в общем модуле на строке *КонецПроцедуры*.

Для знакомства с документами вы будете использовать окно вычисления выражений. Хоть сейчас у вас ничего не выделено в модуле, вы можете нажать **Shift+F9**, чтобы вычислить выражение.

Окно вычисления выражений позволяет вычислять не только то, что вы выделили в модуле, но и то, что вы написали в поле *Выражение*. Другими словами, в поле *Выражение* вы можете писать всё то же самое, что вы написали бы в модуле.

Напишите там *Документы* и нажмите **Enter** (рисунок 3.204).

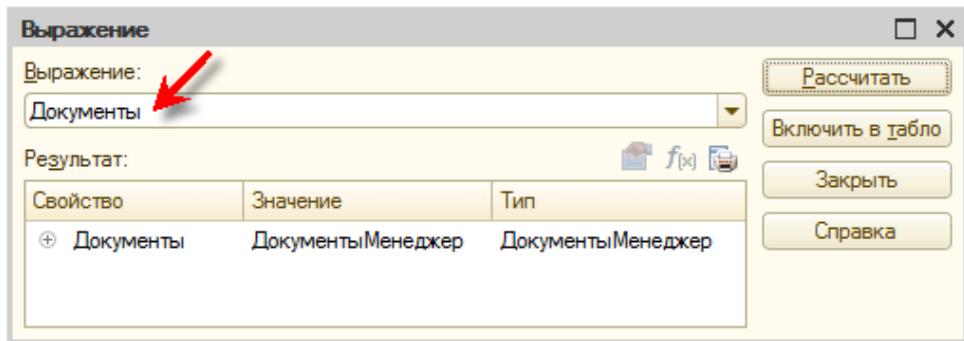


Рисунок 3.204. Свойство «Документы»

У вас получился объект типа *ДокументыМенеджер*.

Откуда взялось слово *Документы*, которое вы написали в поле *Выражение*? И почему нужно было написать именно его?

Тут надо вспомнить про контекст, который вы изучали, когда знакомились с процедурами и функциями. Тогда я рассказывал только о локальном контексте модуля, который состоит из переменных, процедур и функций, которые определены в этом модуле. Но кроме локального контекста во встроенном языке 1С:Предприятия существует ещё и *глобальный контекст*.

Глобальный контекст — это набор свойств, процедур и функций, которые доступны в любых модулях конфигурации. Например, раньше именно благодаря глобальному контексту вы могли использовать функции работы с датой, такие как *НачалоДня()*, *КонецДня()*, *Дата()* и другие.

Весь глобальный контекст описан в синтакс-помощнике в ветке, которая так и называется. В том числе у него есть свойства, и одно из свойств называется *Документы*. Именно его вы и набрали в поле *Выражение*. Сейчас перейти в синтакс-помощник вы не сможете, окно вычисления выражений вам не позволит. Но потом у вас будет возможность всё это посмотреть.

Итак, для всех разновидностей объектов конфигурации в глобальном контексте есть одноимённые свойства: *Справочники*, *Документы*, *РегистрыСведений*, *РегистрыНакопления* и так далее. Эти свойства возвращают объект, который сам по себе практически никогда не используется. Он содержит в себе коллекцию менеджеров, управляющих конкретными справочниками, документами и так далее.

Раскройте свойство *Документы*, и вы увидите, что в коллекции есть единственный элемент *ДокументМенеджер.УчебныйДень* (рисунок 3.205).

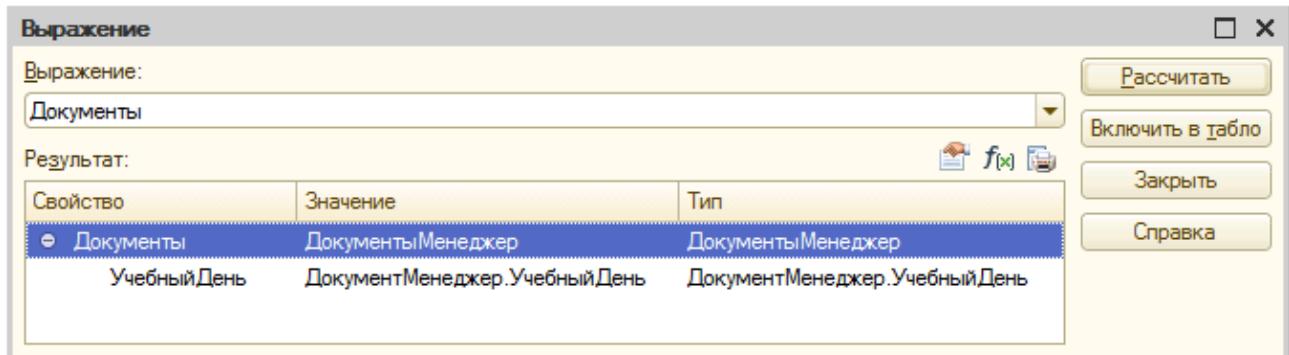


Рисунок 3.205. Менеджер документа «УчебныйДень»

Он один потому, что в вашей конфигурации только один документ — *УчебныйДень*.

Если вы хотите посмотреть, как аналогичная коллекция выглядит у справочников, наберите *Справочники* в поле *Выражение* (рисунок 3.206).

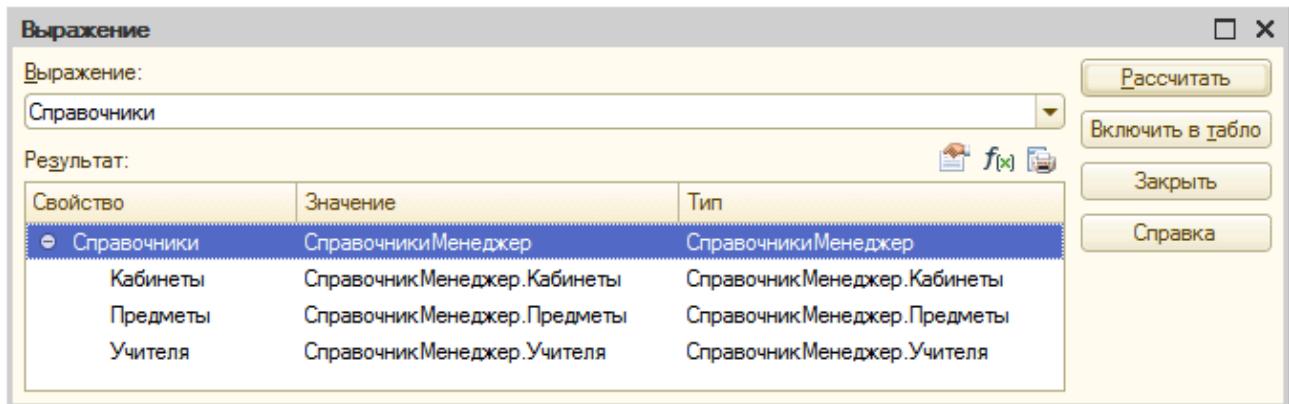


Рисунок 3.206. Менеджеры справочников

Менеджеров справочников будет три штуки, потому что в вашей конфигурации три справочника.

Теперь закройте окно вычисления выражений и перейдите в синтакс-помощник. Посмотрите, как всё то же самое выглядит там.

Откройте свойства глобального контекста и дважды щёлкните по свойству *Документы*. Вы увидите его описание (рисунок 3.207).

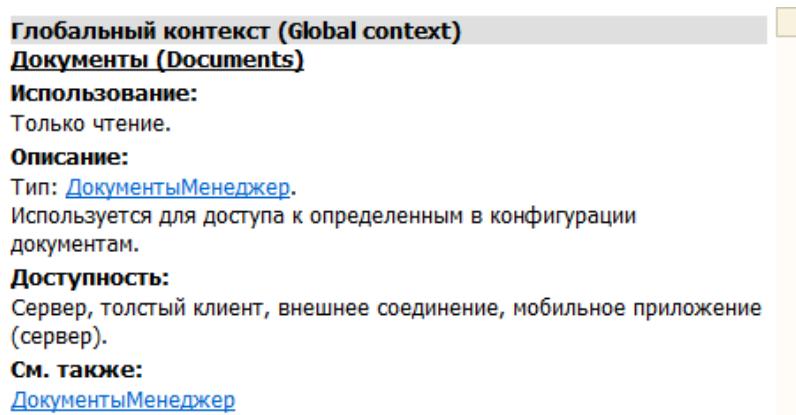


Рисунок 3.207. Описание свойства «Документы»

Это свойство вернёт вам объект *ДокументыМенеджер*. Чтобы узнать, что это за объект, щёлкните по гиперссылке на нём (рисунок 3.208).

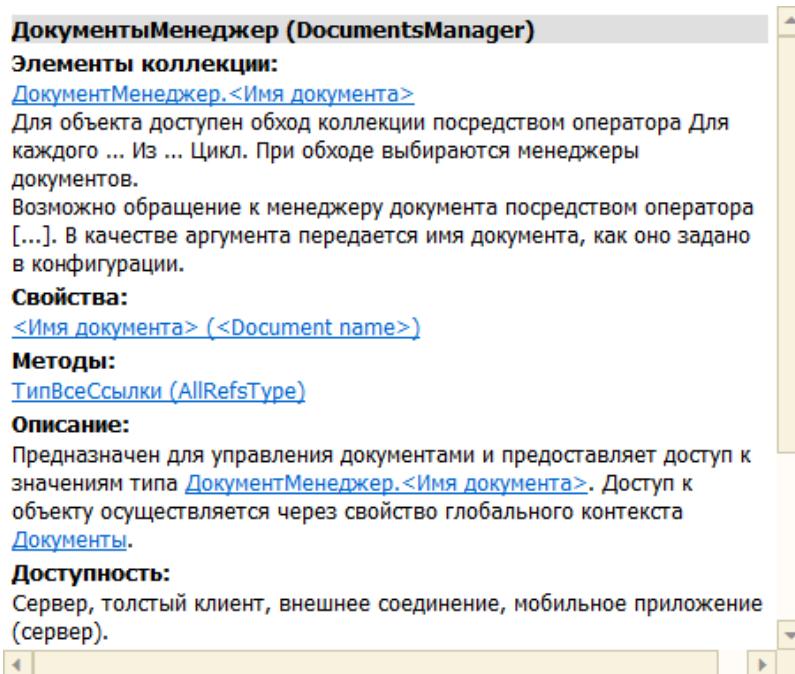


Рисунок 3.208. Описание типа «ДокументыМенеджер»

Тут вы увидите, что этот объект является коллекцией, которая состоит из менеджеров конкретных документов. Чтобы узнать, что за объекты содержатся в этой коллекции, щёлкните по гиперссылке на типе элемента коллекции (рисунок 3.209).

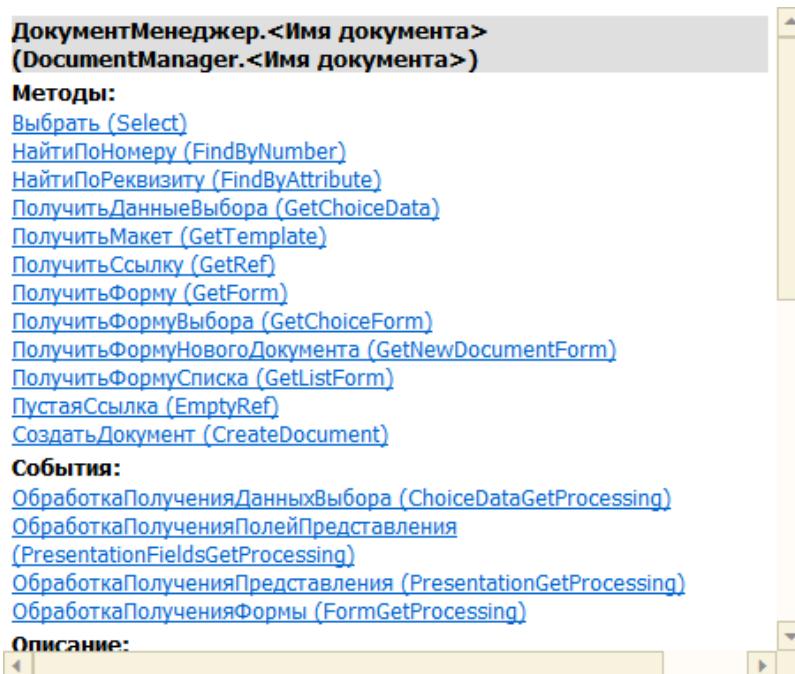


Рисунок 3.209. Описание типа «ДокументМенеджер.<Имя документа>»

Кстати, если вы вернётесь назад, то увидите, что у менеджера всех документов в качестве свойства можно указать имя нужного вам объекта конфигурации *Документ*. Таким образом, чтобы получить менеджер конкретного документа, вам не нужно перебирать всю коллекцию. Можно написать просто *Документы.УчебныйДень*.

Этот принцип — через свойство получить доступ к менеджеру всех объектов, а затем по имени получить менеджер нужного вам объекта конфигурации — одинаков и для объектных, и для необъектных данных. Различия начнутся дальше.

Поэтому сейчас самостоятельно потренируйтесь ещё раз. Посмотрите эту цепочку с помощью окна вычисления выражения. Для справочников, для документов. А затем тоже самое посмотрите с помощью синтаксис-помощника. Вы должны понимать, как то, что вы видите в окне вычисления выражений, связано с тем, что вы видите в синтаксис-помощнике. Это важно.

Чтобы вам было проще, я покажу несколько картинок. Они помогут вам в самостоятельных упражнениях.

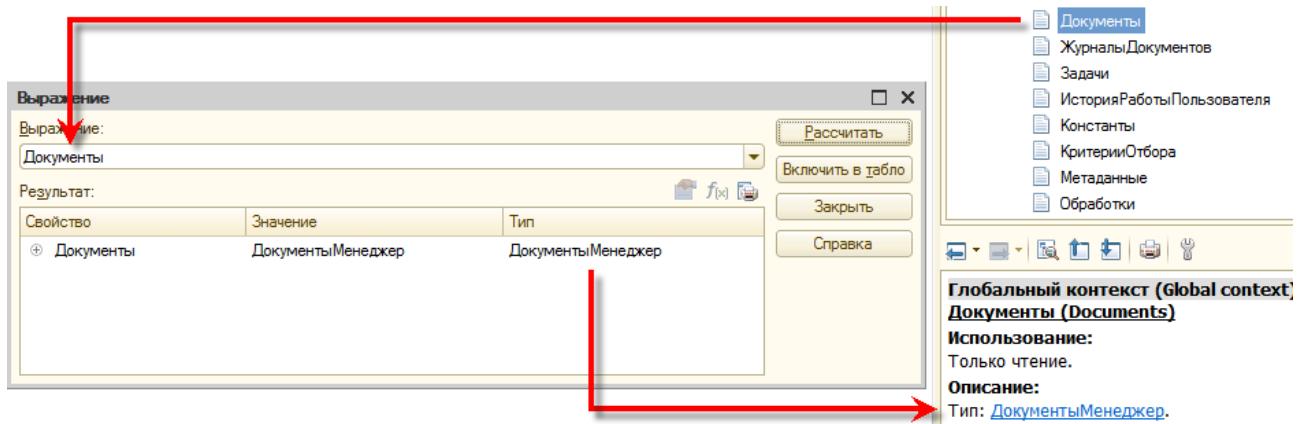


Рисунок 3.210. С помощью свойства получаете менеджер всех объектов

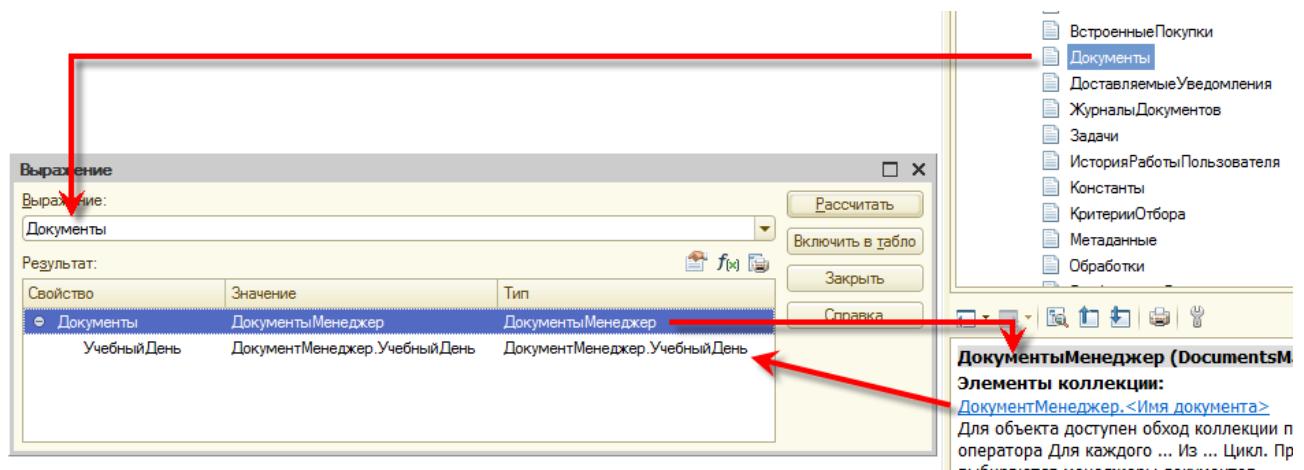


Рисунок 3.211. Он является коллекцией менеджеров конкретных объектов конфигурации

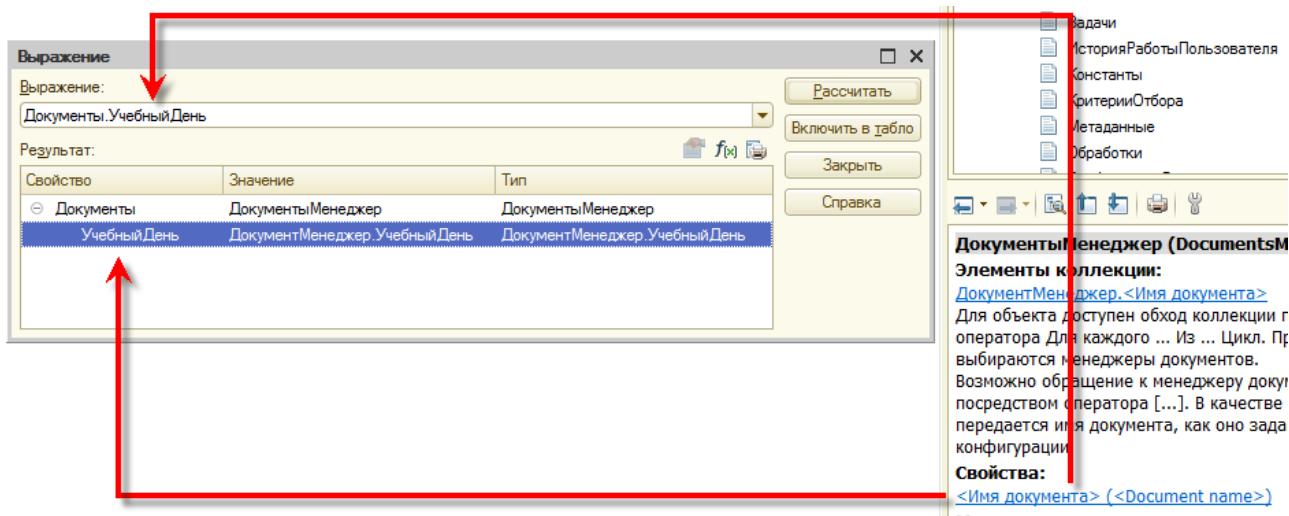


Рисунок 3.212. К каждому менеджеру можно получить доступ по его имени в конфигураторе

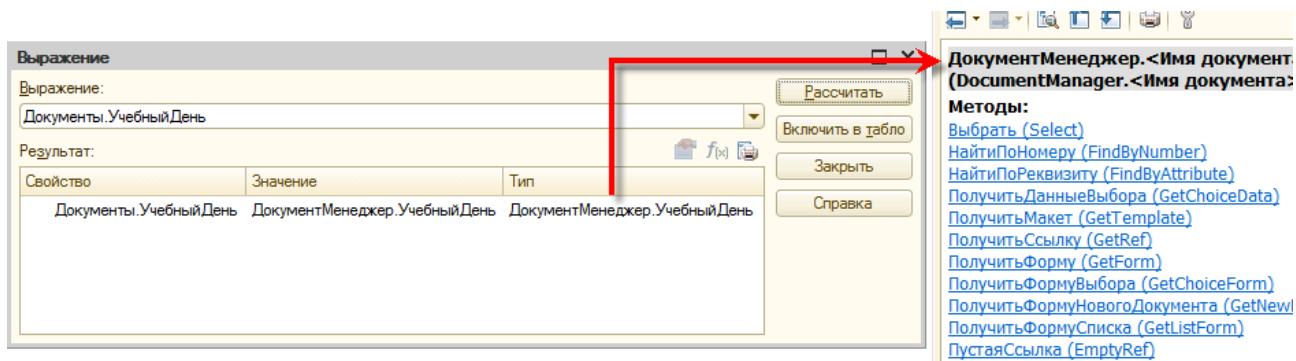


Рисунок 3.213. Результатом выполнения «Документы.УчебныйДень» является менеджер этого документа

К рисунку 3.213 можно сделать небольшое пояснение. Менеджер документа не имеет свойств, только методы и события. Поэтому, к сожалению, в окне вычисления выражения вы ничего не увидите. Нужно пользоваться синтаксис-помощником.

Чтобы знакомиться с документами дальше, придётся написать небольшую программу. Завершите отладку и откройте общий модуль *Серверный*.

Сейчас вы выберете несколько документов. Для этого вам понадобится метод *Выбрать()*, который хорошо видно на рисунке 3.213. И вы наверняка уже знаете, как должна выглядеть инструкция, которая его использует. Потому что начало её уже написано у вас в поле *Выражение* на рисунке 3.213.

Посмотрите в синтаксис-помощнике, что делает метод *Выбрать()* объекта *ДокументМенеджер.<Имя документа>*. Ему можно указать несколько параметров, но все они необязательные. Поэтому вы их сейчас использовать не будете. А возвращает он выборку документов. Поэтому вы можете написать так, как показано в листинге 3.70.

Листинг 3.70. Выборка документов

```
Выборка = Документы.УчебныйДень.Выбрать();
```

Объект *ДокументВыборка.<Имя документа>*, который вы таким образом получите, имеет метод *Следующий()*. Посмотрите его описание в синтаксис-помощнике. Этот метод позволяет обходить всю полученную выборку, и выглядит это так (листинг 3.71). Сделайте этот пример в своей конфигурации.

Листинг 3.71. Обход выборки документов

```
Выборка = Документы.УчебныйДень.Выбрать();
Пока Выборка.Следующий() Цикл
КонецЦикла;
```

После того как вы получили выборку методом *Выбрать()*, она не спозиционирована ни на каком своём элементе. Чтобы перейти к первому элементу этой выборки или к следующему элементу, нужно выполнить метод *Следующий()*. Если первый или следующий элемент получен, метод вернёт значение *Истина*. Если элементов больше нет, метод вернёт значение *Ложь*.

Именно поэтому для обхода выборки используется цикл *Пока Цикл*, на каждой итерации которого выполняется метод *Следующий()*. Как только выборка закончится и метод вернёт *Ложь*, выполнение цикла тоже прекратится.

Установите точку останова на строке *Пока Выборка.Следующий() Цикл* и запустите 1С:Предприятие в режиме отладки. Посмотрите, как это работает.

До того, как первый раз выполнится метод *Следующий()*, в объекте выборки нет никаких данных (рисунок 3.214).

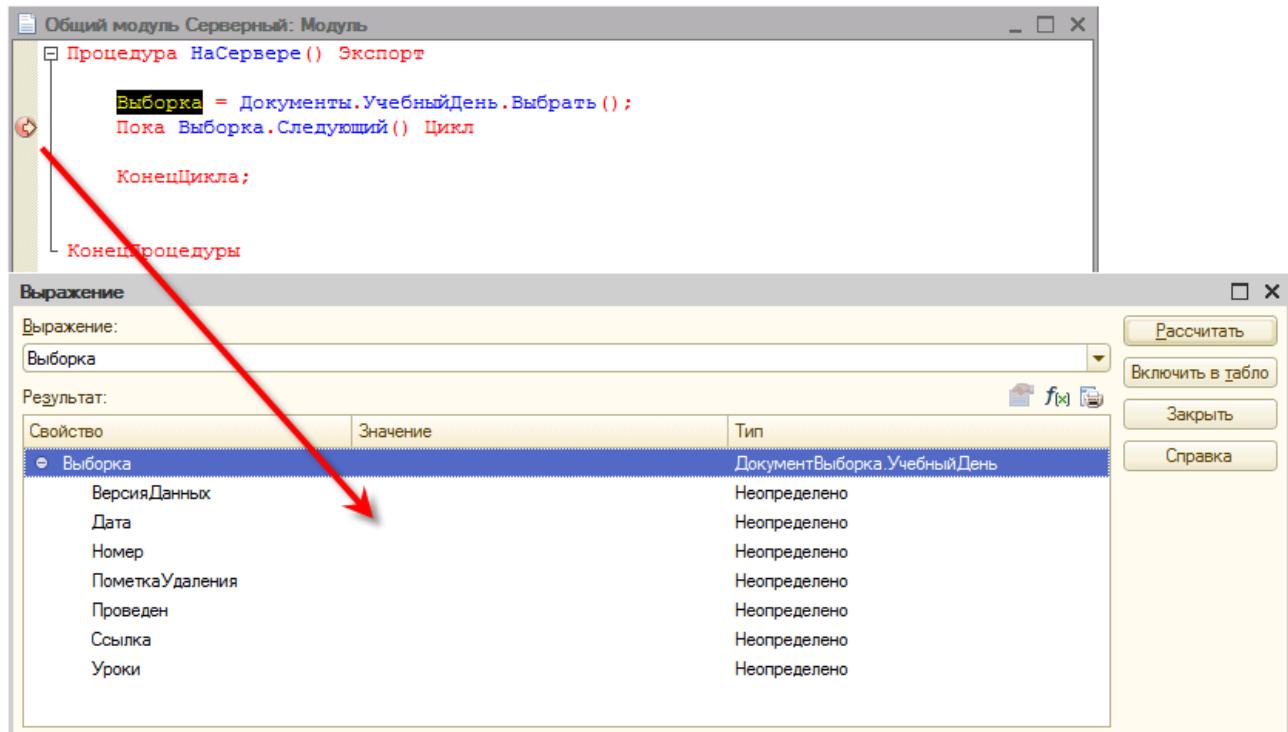


Рисунок 3.214. В объекте выборки нет данных

Закройте окно вычисления выражений, сделайте один шаг и снова посмотрите переменную *Выборка*. Вы увидите, что в ней появятся данные одного из документов (рисунок 3.215).

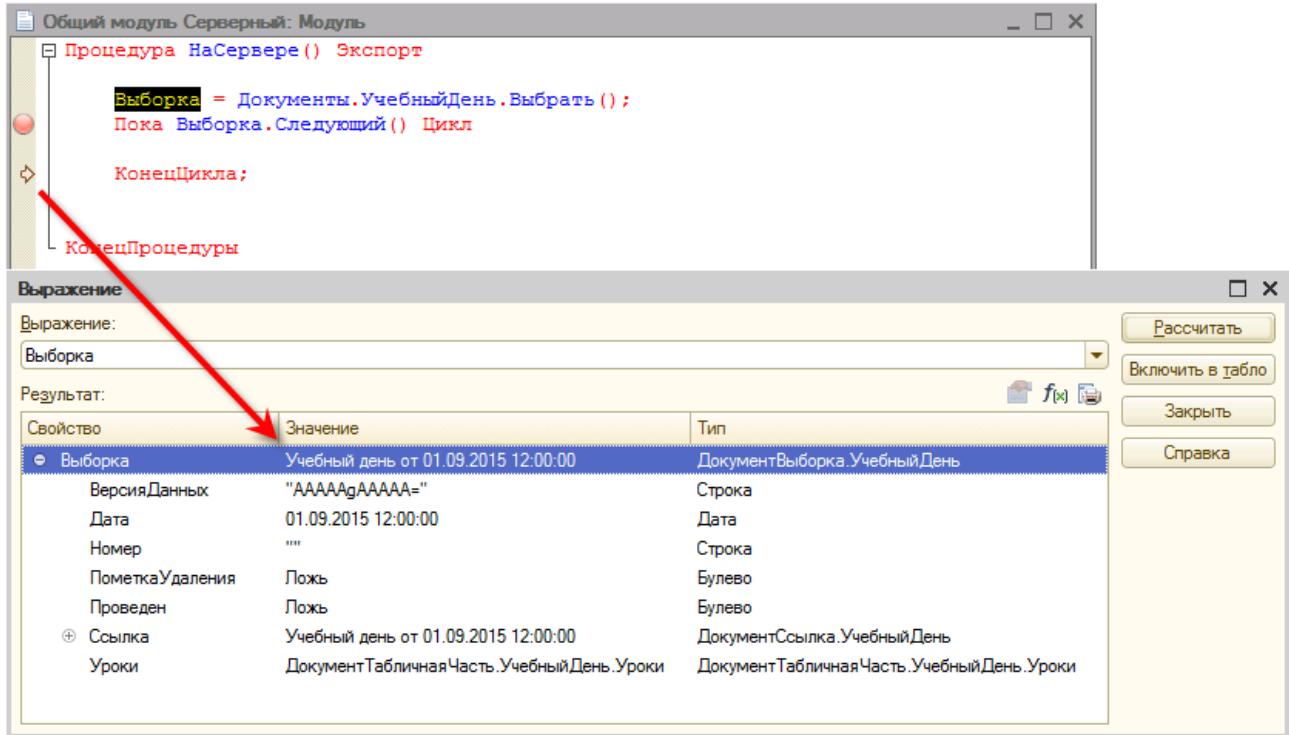


Рисунок 3.215. В объекте выборки данные документа

При выполнении метода *Выбрать()* вы не указывали порядок, в котором нужно выбирать документы. Поэтому неизвестно, в каком порядке они будут появляться, но вам это и не нужно сейчас.

Выборка содержит значения всех свойств документа, а также все его табличные части. К каждой табличной части вы можете обратиться по её имени.

Объект табличной части содержит коллекцию строк. Каждая строка содержит значения реквизитов, которые есть у табличной части (рисунок 3.216).

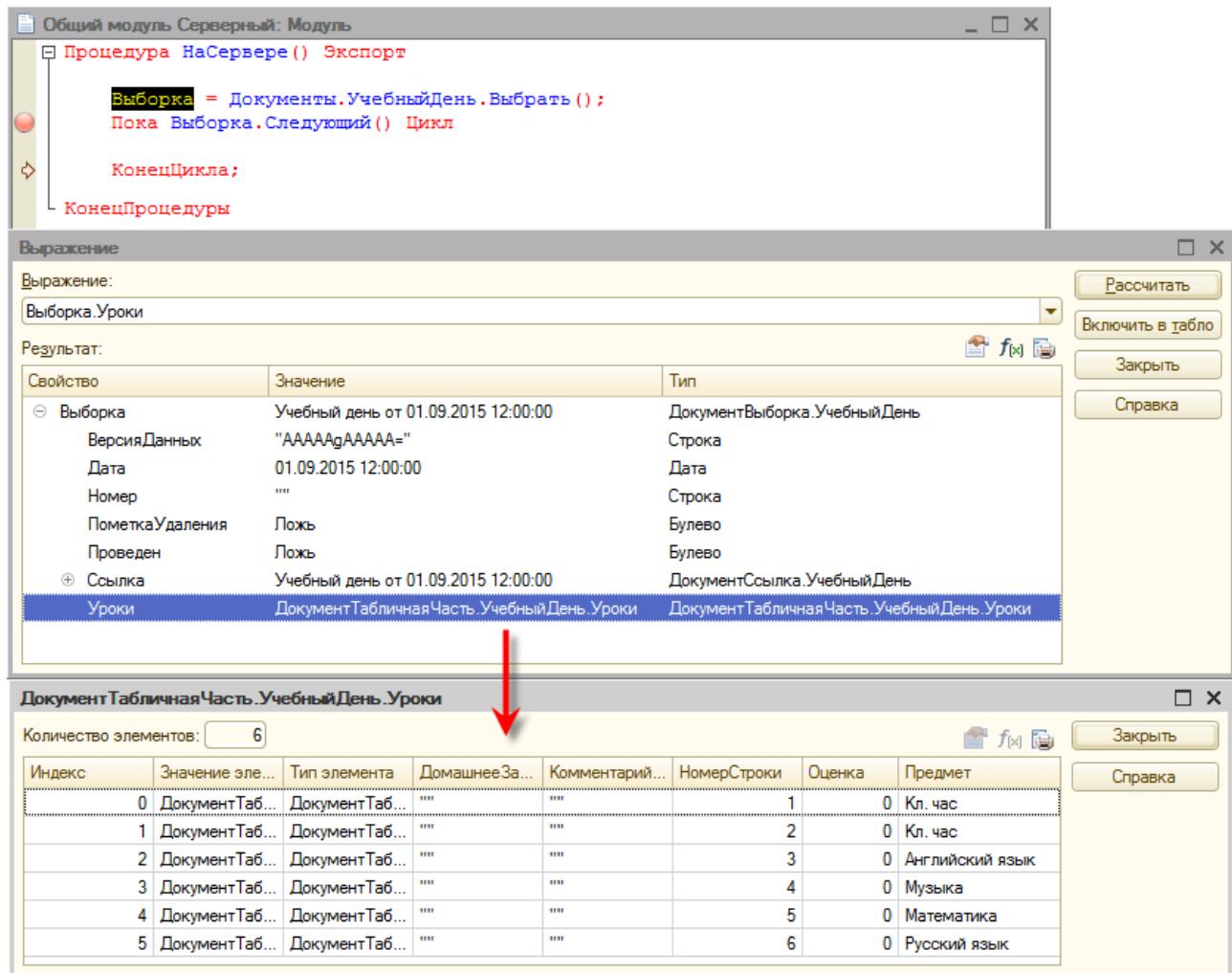


Рисунок 3.216. Табличная часть документа

Табличная часть — это индексированная коллекция. Поэтому к каждой строке табличной части вы можете обращаться по её индексу. Это вы можете увидеть на рисунке 3.217 в поле Выражение. Табличную часть вы получаете по имени (Уроки), а дальше, по индексу, получаете первую её строку.

Выражение

Выражение: Выборка Уроки[0]

Результат:

Свойство	Значение	Тип
Выборка Уроки[0]	ДокументТаб...	ДокументТабличнаяЧастьСтрока.УчебныйДень.Уроки
ДомашнееЗадание	""	Строка
КомментарийУчителя	""	Строка
НомерСтроя	1	Число
Оценка	0	Число
Предмет	Кл. час	СправочникСсылка.Предметы

Рисунок 3.217. Стока табличной части

Вот таким нехитрым способом, «путешествуя» только с помощью отладчика, вы можете самостоятельно узнать очень многое об устройстве объектов встроенного языка. А

после этого недостающую информацию об их методах вы можете получить из синтакс-помощника.

Для этого достаточно на закладке *Индекс* начать вводить имя того типа, который вы увидели в отладчике. И синтакс-помощник сразу же подберёт вам типы, которые начинаются с той строки, которую вы ищете (рисунок 3.218).

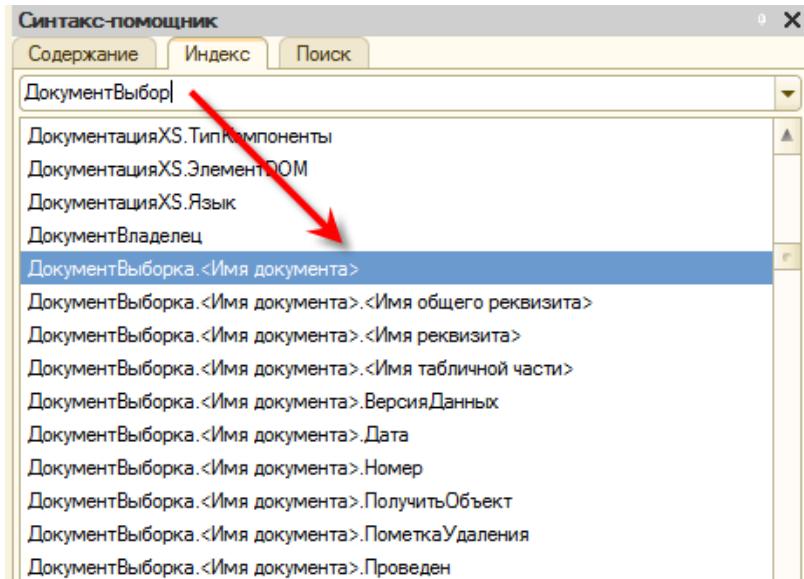


Рисунок 3.218. Поиск в синтакс-помощнике по имени типа

К сожалению, этот способ не поможет вам найти типы, описывающие табличную часть. Потому что они могут начинаться с разных слов: *ДокументТабличнаяЧасть...*, *СправочникТабличнаяЧасть...* и так далее.

Объекты встроенного языка, описывающие табличную часть и её строки, одинаковы для всех объектов конфигурации, у которых могут быть табличные части. Поэтому в синтакс-помощнике они описаны в ветке *Универсальные объекты* у прикладных объектов (рисунок 3.219).

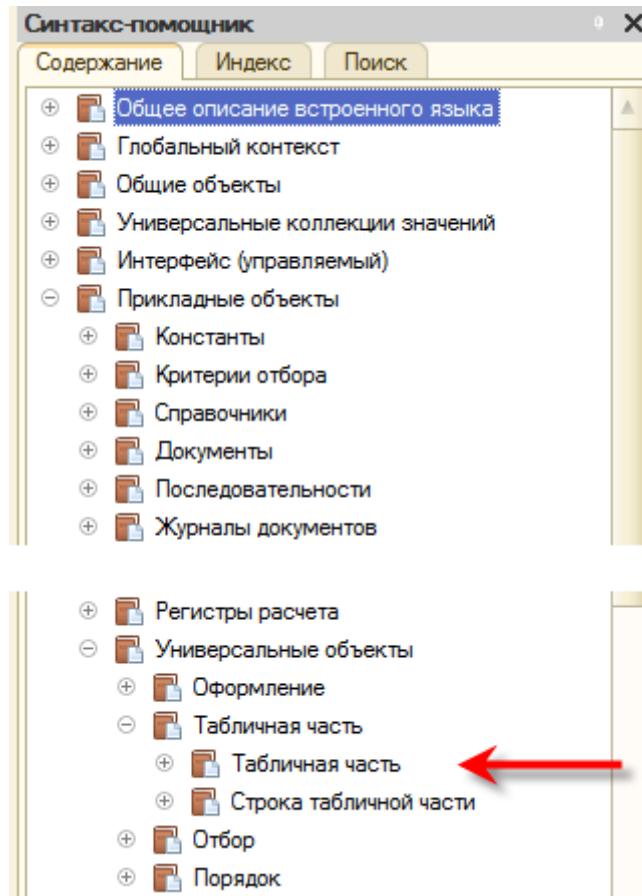


Рисунок 3.219. Типы табличной части в синтакс-помощнике

Может быть, на слух вам пока сложно воспринимать всё это многообразие объектов. Ничего страшного.

Сейчас вы сделаете небольшую доработку в своей конфигурации. Это поможет вам лучше понять работу с прикладными объектами. А после этого вы сможете выполнить несколько самостоятельных упражнений, чтобы закрепить свои знания.

3.11.6 Номер документа УчебныйДень

В самом начале, когда вы создавали объекты конфигурации, вы удалили номер у документа УчебныйДень. Главная причина этого заключалась совсем не в том, что номер не нужен.

На самом деле он нужен, и сейчас вы будете возвращать его обратно. Просто в тот момент вы не обладали достаточными знаниями для того, чтобы сделать этот номер правильным. А раз так, то я попросил вас убрать его совсем, чтобы он вас не смущал.

Вспомните, почему нужен номер документа. Номер нужен потому, что это единственный прикладной признак, который позволяет отличить один документ от другого. Кроме того, платформа умеет автоматически контролировать уникальность номеров документов. Чтобы в базе случайно не оказалось двух документов с одинаковыми номерами.

Именно этой возможности вам сейчас очень не хватает. По смыслу нашей задачи в каждый учебный день у вас должен существовать единственный документ УчебныйДень. Сейчас вы отличаете один документ от другого по его дате. Но ничто не мешает вам сейчас создать несколько документов с одинаковой датой.

Если это случится, то логика работы вашего электронного дневника будет нарушена.

Восстановить её будет сложно, потому что непонятно, какой из документов, созданных в этот день, правильный, а какой нет.

Чтобы исключить эту ситуацию, вы сейчас сделаете так, чтобы номер документа формировался автоматически и однозначно соответствовал бы той дате, которая есть у документа.

Номер документа будет у вас строкой. Эта строка будет содержать год, месяц и день от той даты, которая есть у документа. То есть если документ имеет дату 1 сентября 2015 года, то его номер будет 20150901. Для месяца и дня вы будете использовать лидирующие нули, чтобы длина номера у всех документов была одинаковая.

Сначала нужно вернуть номер документа. Откройте окно редактирования документа *УчебныйДень*, перейдите на закладку *Нумерация*. Здесь:

- установите длину номера — 8;
- включите флагок *Контроль уникальности*.

На всякий случай проверьте, что тип номера *Строка* и что другие свойства установлены у вас так же, как на рисунке 3.220.

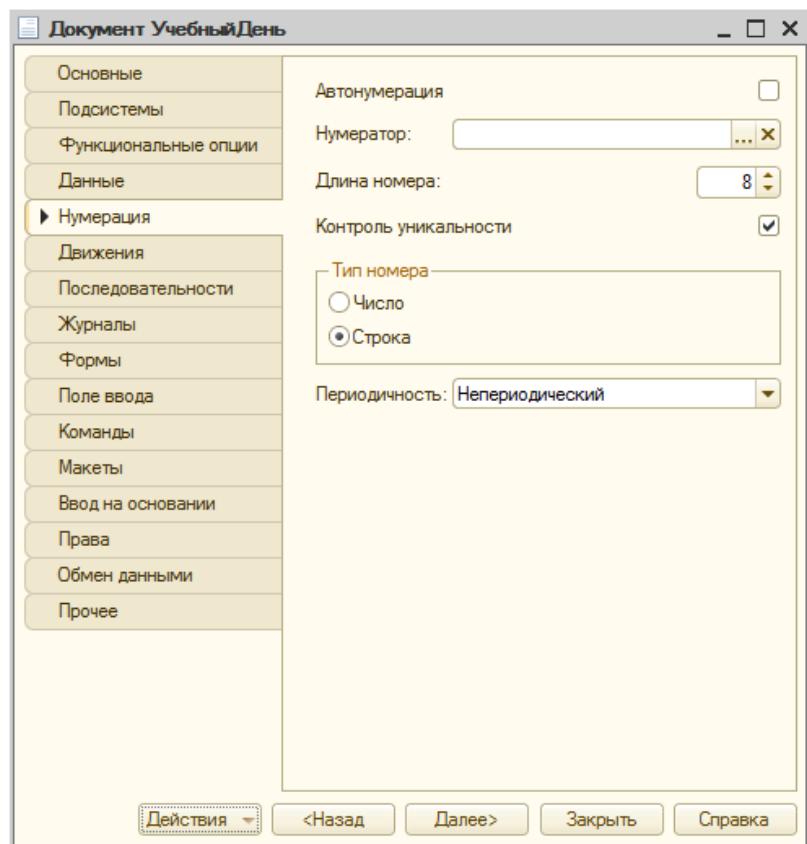


Рисунок 3.220. Свойства документа «УчебныйДень»

Нажмите *Закрыть* и обновите конфигурацию базы данных.

Платформа выдаст вам предупреждение о том, что номер документа стал неуникальным. Всё правильно, нажмите *Принять*.

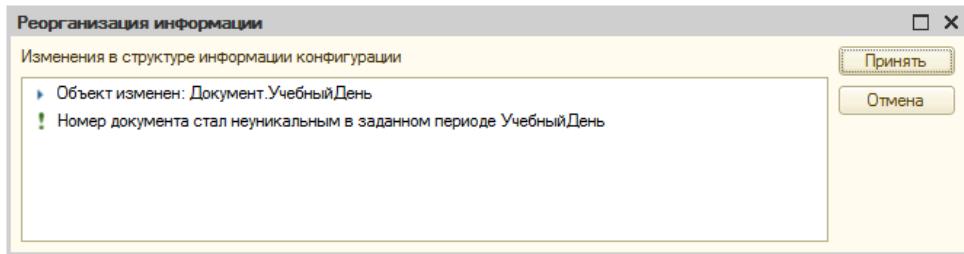


Рисунок 3.221. Номер документа стал неуникальным

Платформа ещё раз на всякий случай спросит вас, и вы снова ответьте ей *Да* (рисунок 3.222).

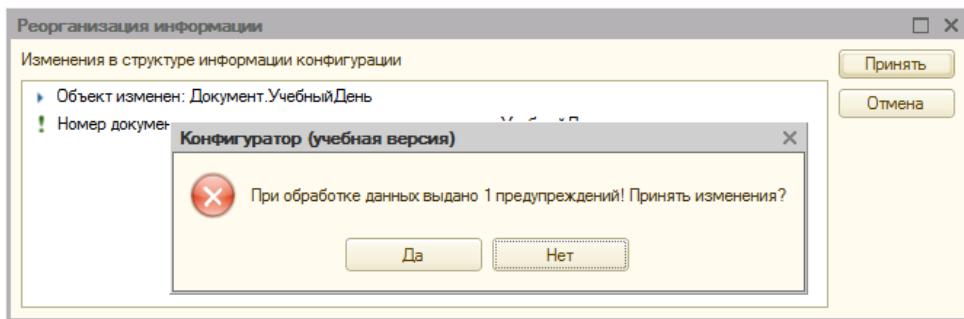


Рисунок 3.222. Вопрос о принятии изменений

Всё, теперь у документа *УчебныйДень* есть номер. Что нужно сделать дальше?

А дальше нужно сделать две вещи. Сначала установить правильный номер тем документам, которые уже есть в вашей информационной базе. А затем написать небольшую программу, которая автоматически будет устанавливать правильный код новым документам.

Чтобы установить номер существующим документам, вы будете использовать тот пример с выборкой документов, который у вас уже есть. Вы его немного доработаете.

Сначала откройте общий модуль. Тут нужно будет создать функцию, которая сформирует вам правильный номер документа. А передавать ей вы будете дату. Эта функция понадобится вам не только в этом модуле, поэтому не забудьте написать слово *Экспорт* (листинг 3.72). Тело функции пока не пишите, сначала прочитайте следующий абзац.

Листинг 3.72. Функция «ПолучитьНомер()»

Функция ПолучитьНомер(ДатаДокумента) Экспорт

```
НомерДокумента = Формат(ДатаДокумента, "ДФ=ууууММdd");
```

```
Возврат НомерДокумента;
```

КонецФункции

Функция *Формат()*, использованная в листинге, позволяет получить из даты строку нужного вам вида. Первым параметром ей передаётся значение, а вторым — форматная строка. Для создания форматной строки используйте конструктор, который вам уже знаком по первой части книги.

Чтобы вызвать его в модуле, откройте контекстное меню в том месте, где должна быть форматная строка, и выберите *Конструктор форматной строки* (рисунок 3.223).

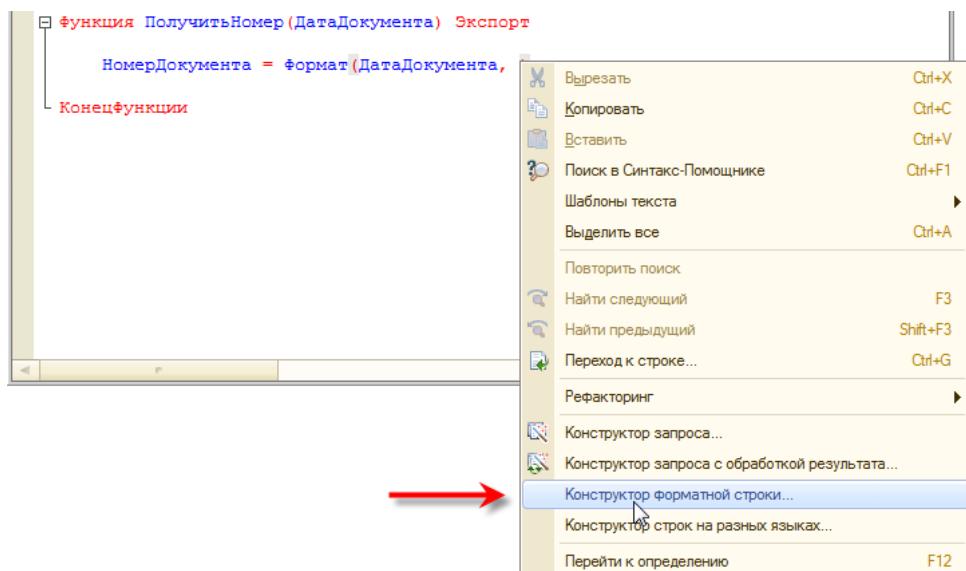


Рисунок 3.223. Вызов конструктора форматной строки

На закладке *Дата* выберите наиболее подходящий формат, а окончательно отредактировать его можно прямо в том поле, в котором вы его выбирали (рисунок 3.224).

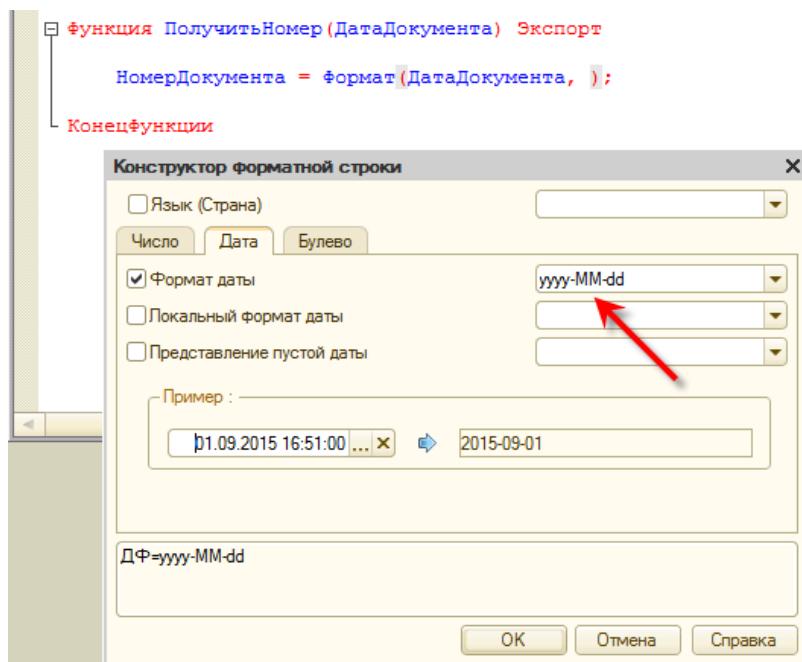


Рисунок 3.224. Выбор и редактирование формата даты

В результате ваша функция будет выглядеть, как в листинге 3.72.

Теперь перейдите к процедуре *НаСервере()*, которую вы делали в предыдущем примере. Там у вас уже есть готовый цикл перебора всех документов. Теперь внутрь этого цикла вам нужно дописать присвоение правильного номера и сохранение документа.

Для того чтобы изменять данные документа, записанного в базу данных, нужно обязательно использовать объект *ДокументОбъект.<Имя документа>*. Только он позволяет изменять данные. Все остальные объекты для работы с документами позволяют только читать данные. Это справедливо и для всех остальных объектных данных.

Получить этот объект можно прямо из выборки. У неё есть метод, который так и называется — *ПолучитьОбъект()* (листинг 3.73).

Листинг 3.73. Получить объект из выборки

```
ДокументОбъект = Выборка.ПолучитьОбъект();
```

После этого можно установить тот номер документа, который нужен (листинг 3.74).

Листинг 3.74. Установить номер документа

```
ДокументОбъект = Выборка.ПолучитьОбъект();
```

```
ДокументОбъект.Номер = ПолучитьНомер(ДокументОбъект.Дата);
```

И теперь, чтобы ваши изменения не пропали, нужно сохранить их в базе данных. Для этого у прикладных объектов есть метод *Записать()* (листинг 3.75).

Листинг 3.75. Записать объект документа

```
ДокументОбъект = Выборка.ПолучитьОбъект();
```

```
ДокументОбъект.Номер = ПолучитьНомер(ДокументОбъект.Дата);
```

```
ДокументОбъект.Записать();
```

В результате общий модуль будет выглядеть у вас так, как показано в листинге 3.76. Проверьте.

Листинг 3.76. Общий модуль «Серверный»

Процедура НаСервере() Экспорт

```
Выборка = Документы.УчебныйДень.Выбрать();
```

```
Пока Выборка.Следующий() Цикл
```

```
ДокументОбъект = Выборка.ПолучитьОбъект();
```

```
ДокументОбъект.Номер = ПолучитьНомер(ДокументОбъект.Дата);
```

```
ДокументОбъект.Записать();
```

КонецЦикла;

КонецПроцедуры

Функция ПолучитьНомер(ДатаДокумента) Экспорт

```
НомерДокумента = Формат(ДатаДокумента, "ДФ=yyyyMMdd");
```

```
Возврат НомерДокумента;
```

КонецФункции

Вся эта программа должна будет отработать при запуске вашего электронного дневника. Чтобы вы сразу могли увидеть изменения, вам нужно перенести в форму списка документов поле *Номер*.

Для этого откройте форму списка у документа *УчебныйДень*, раскройте реквизит *Объект* и перетащите мышью поле *Номер* в список, к полю *Дата* (рисунок 3.225).

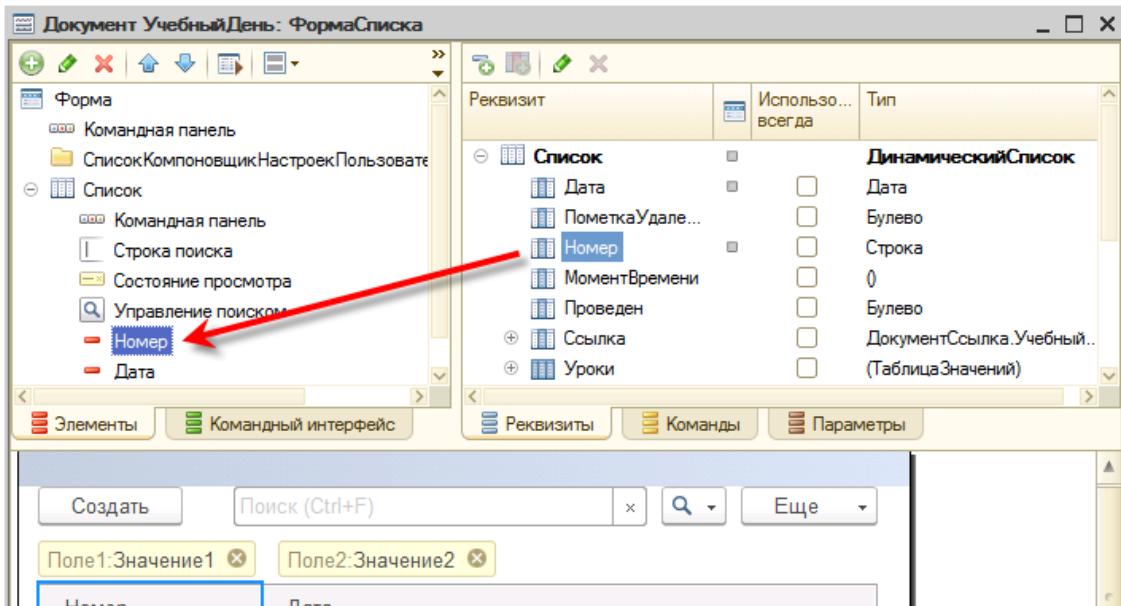


Рисунок 3.225. Перетащите поле «Номер» в форму

После этого отключите все точки останова, если они у вас есть, и запустите 1С:Предприятие.

Сначала на начальной странице колонка *Номер* будет пустая. Просто потому, что формы начальной страницы создаются раньше, чем выполняется обработчик *ПриНачалоРаботыСистемы()* (рисунок 3.226).

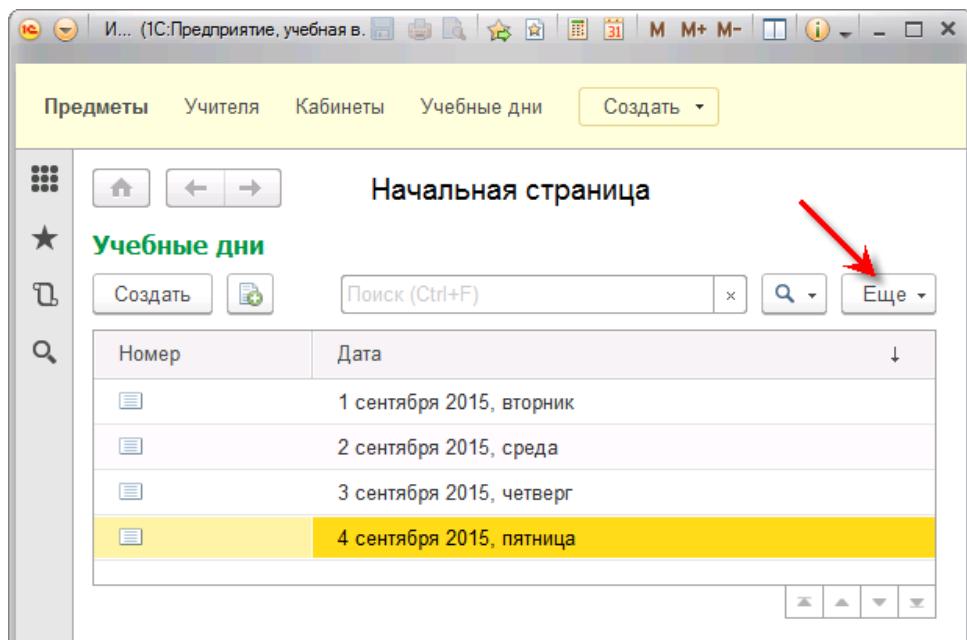


Рисунок 3.226. Начальная страница после запуска

Чтобы обновить данные в списке, нажмите F5 или выполните команду *Обновить* из меню *Ещё* списка. В колонке появятся номера документов (рисунок 3.227).

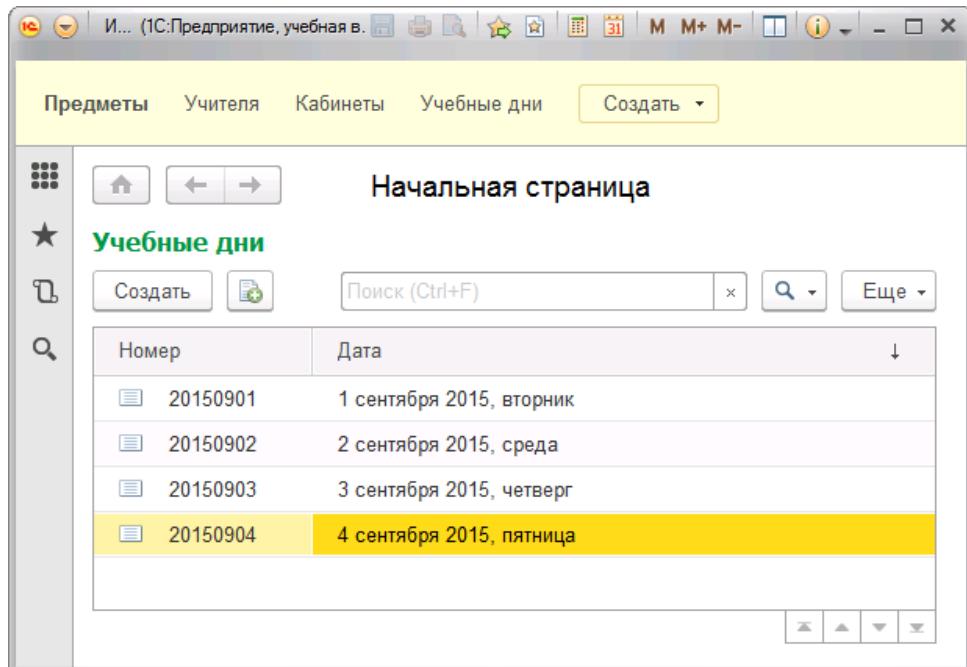


Рисунок 3.227. Номера документов в списке

Как видите, номера документов именно такие, как вы и задумывали.

Это действие, простановка кодов существующим документам, понадобилось вам для того, чтобы «восстановить порядок» в тех данных, которые уже есть в вашей базе. Это действие нужно сделать один раз, и оно не должно выполняться каждый раз при старте программы.

Вы вызвали его из модуля управляемого приложения только для того, чтобы не усложнять пример. На самом деле для выполнения таких «административных» действий делают обработки, которые запускают только тогда, когда это нужно.

Поэтому сейчас вам нужно будет удалить ту часть программы, которая устанавливает документам их номера. Это процедура *НаСервере()* в общем модуле и её вызов в модуле управляемого приложения. Процедуру *ПолучитьНомер()* в общем модуле не удаляйте, потому что она понадобится вам в дальнейшем (листинги 3.77 и 3.78).

Листинг 3.77. Текст, который нужно удалить из модуля управляемого приложения

Процедура ПриНачалеРаботыСистемы()

```
УстановитьКраткийЗаголовокПриложения("Иванов Петя");
```

```
Серверный.НаСервере();
```

```
КонецПроцедуры
```

Листинг 3.78. Текст, который нужно удалить из общего модуля «Серверный»

```
Процедура НаСервере() Экспорт
```

```
Выборка = Документы.УчебныйДень.Выбрать();
Пока Выборка.Следующий() Цикл
    ДокументОбъект = Выборка.ПолучитьОбъект();
    ДокументОбъект.Номер = ПолучитьНомер(ДокументОбъект.Дата);
    ДокументОбъект.Записать();

КонецЦикла;
```

КонецПроцедуры

```
Функция ПолучитьНомер(ДатаДокумента) Экспорт
```

```
НомерДокумента = Формат(ДатаДокумента, "ДФ=yyyyMMdd");
```

```
Возврат НомерДокумента;
```

КонецФункции

Также откройте форму списка документа и удалите из неё поле *Номер*, которое вы добавляли. Вы будете использовать его только программно, а для отображения в форме прекрасно подходит поле *Дата*. Чтобы удалить поле, выделите его и нажмите кнопку *Удалить текущий* (рисунок 3.228).

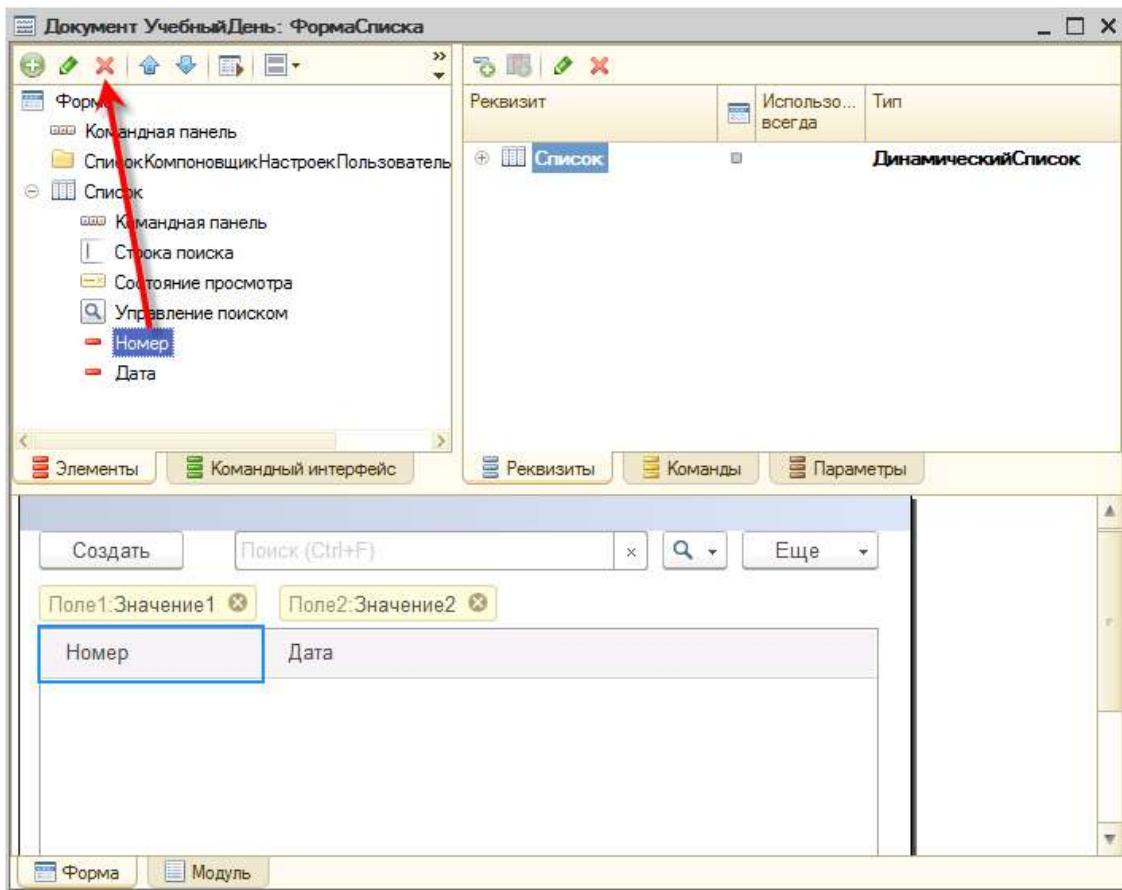


Рисунок 3.228. Удалите поле «Номер»

Обновите конфигурацию базы данных.

3.11.7 События объектов

Номера старых документов вы обновили. Теперь нужно сделать что-то, чтобы у новых документов, которые будут появляться в вашем электронном дневнике, автоматически создавались правильные номера.

Для этого нужно сделать небольшое отступление о том, как встроенный язык связан с тем, что делает платформа 1С:Предприятия.

Если вы думаете, что встроенный язык — это «для вас», а платформа использует какие-то свои особенные способы работы, вы ошибаетесь. Когда в режиме 1С:Предприятие вы открываете документ *Учебный день*, когда нажимаете в нём кнопку *Записать и закрыть*, платформа использует те же самые объекты встроенного языка, с которыми вы только что познакомились: *ДокументСсылка.УчебныйДень* и *ДокументОбъект.УчебныйДень*. Посмотрите на рисунок 3.229.

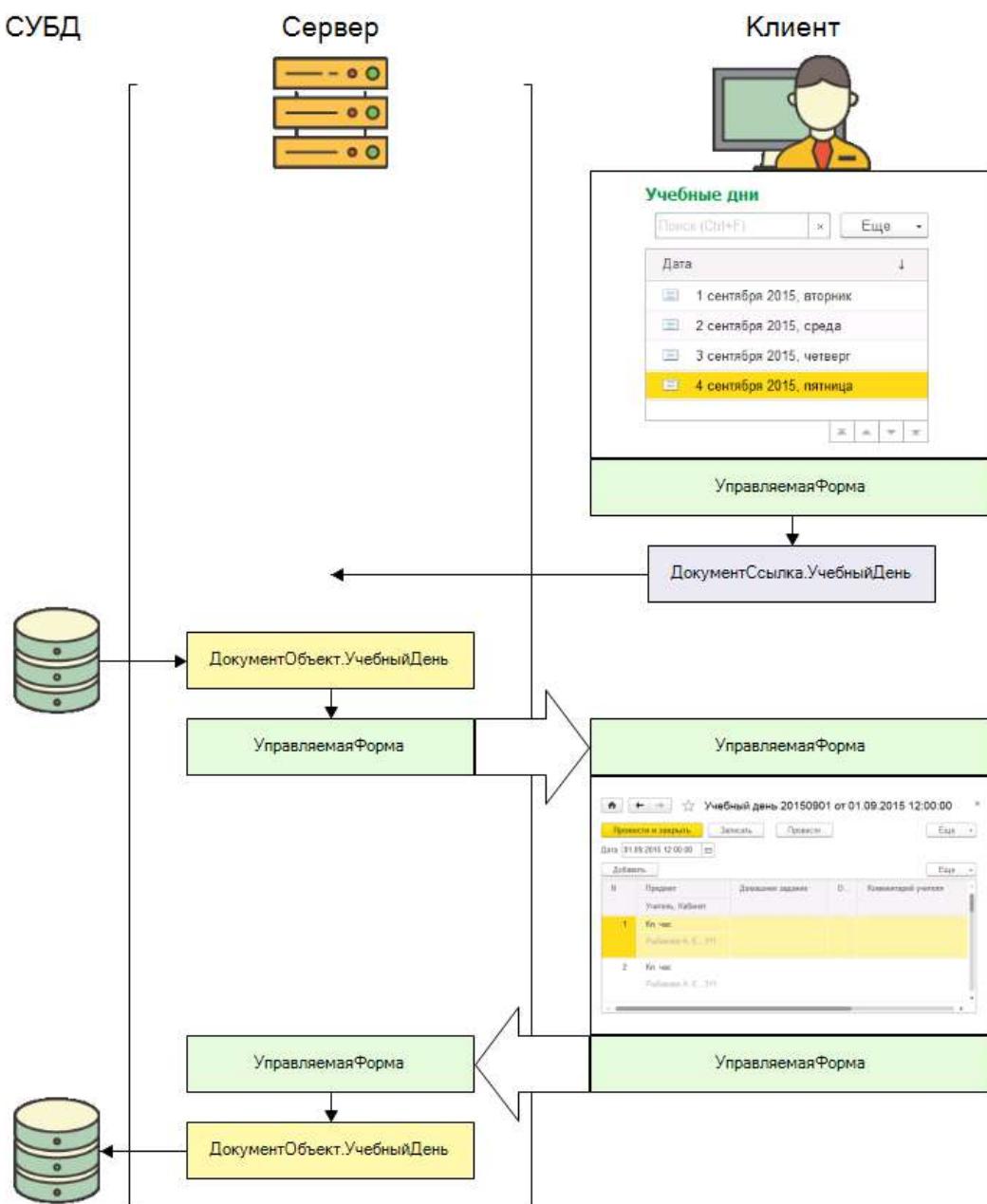


Рисунок 3.229. Открытие и запись документа

Когда вы видите на экране список документов *Учебный день*, на самом деле вы видите

объект встроенного языка *УправляемаяФорма*. Когда вы дважды щёлкаете мышью по строке этого списка, платформа отправляет на сервер команду о том, что нужно открыть форму учебного дня. Какой именно день нужно открыть, платформа определяет по ссылке на документ (*ДокументСсылка.УчебныйДень*). Эта ссылка есть в каждой строке списка документов.

Сервер, имея ссылку, читает из базы данных всю информацию, которая относится к этой ссылке. Помещает её в объект (*ДокументОбъект.УчебныйДень*). Затем из данных этого объекта платформа создаёт управляемую форму, которая отправляется на клиент.

Вы, например, что-то изменяете в этом документе. Например, указываете свои оценки. И нажимаете *Записать и закрыть*.

Управляемая форма отправляется на сервер. Там из её данных платформа создаёт объект (*ДокументОбъект.УчебныйДень*) и записывает его в базу данных.

Теперь вспомните: в самом начале занятий встроенным языком я говорил о том, что у платформы есть события. Это такие моменты, когда вы можете вмешаться в работу прикладного решения. В процессе действий, которые я только что описал, происходит большое количество событий. И все эти события связаны не с конфигурацией вообще, а с конкретными объектами встроенного языка. С теми объектами, которые участвуют в этих действиях.

Такой подход очень удобен, потому что исчезает разница между тем, что делает пользователь, и тем, что делаете вы, когда пишете программу. Не важно, в результате чего началась запись документа. Может быть, это пользователь нажал кнопку *Записать*, может быть, это вы в своей программе написали *ДокументОбъект.Записать()*. В любом случае у объекта встроенного языка *ДокументОбъект.УчебныйДень* возникнут события, связанные с записью документа. И если вы в каком-то из этих событий напишете свою программу, она обязательно выполнится.

Вы хотите, чтобы номер документа формировался и записывался автоматически. Исходя из того, какая дата у документа. Значит, вам нужно вмешаться в процесс записи в том месте, которое показано на рисунке 3.230. Прямо перед тем, как данные будут записаны в базу данных.

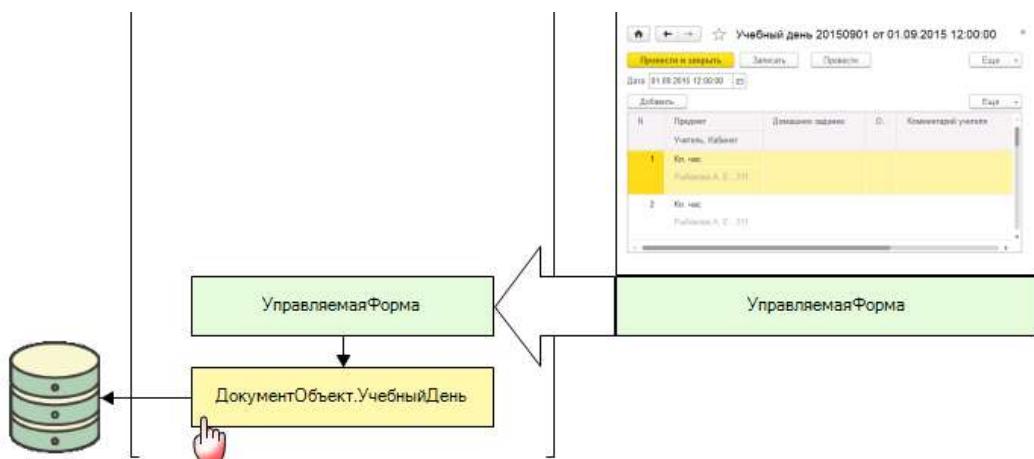


Рисунок 3.230. Момент перед записью документа

Откройте синтакс-помощник и посмотрите, какие события есть у типа *ДокументОбъект.<Имя документа>*. Из того, что может относиться к «записи», вы увидите только два события: *ПередЗаписью* и *ПриЗаписи*. Если вы прочтёте их описание, вам встретится незнакомое слово «транзакция». Не обращайте пока на него внимания. Главное для вас сейчас то, что событие *ПередЗаписью* возникает до того, как данные записываются в базу

данных. А событие *ПриЗаписи* возникает уже после того, как данные записаны (рисунок 3.231).

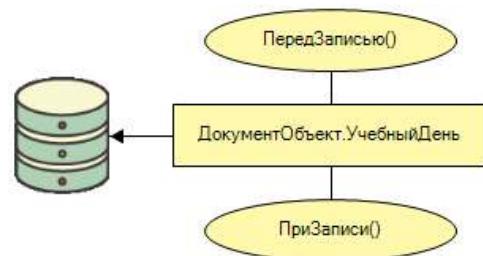


Рисунок 3.231. События «ПередЗаписью» и «ПриЗаписи»

Значит, вам, чтобы сформировать правильный номер, надо успеть до того, как всё начнёт записываться, то есть нужно обработать событие *ПередЗаписью*.

Итак, что нужно делать, вы уже поняли. Осталось понять, где это делать. В каком месте конфигурации написать свои инструкции.

3.11.8 Установка номера для новых документов

Для того чтобы вы могли обработать события, связанные с объектом *ДокументОбъект.<Имя документа>*, у каждого объекта конфигурации *Документ* существует свой собственный модуль, который называется *модуль объекта*. Открыть этот модуль вы можете из контекстного меню нужного вам объекта конфигурации. Команда называется *Открыть модуль объекта* (рисунок 3.232).

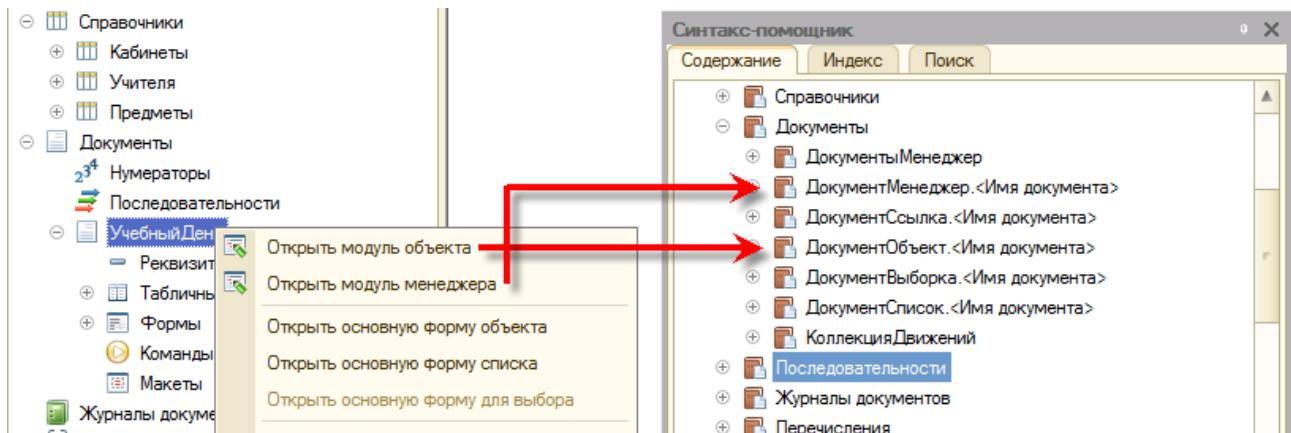


Рисунок 3.232. Команды «Открыть модуль объекта» и «Открыть модуль менеджера»

Кстати, тут же находится команда *Открыть модуль менеджера*. В модуле менеджера вы можете обработать события объекта *ДокументМенеджер.<Имя документа>*.

Другие объекты конфигурации, которые используются для описания объектных данных, устроены так же. Например, у справочников есть аналогичные команды и аналогичные модули (рисунок 3.233).

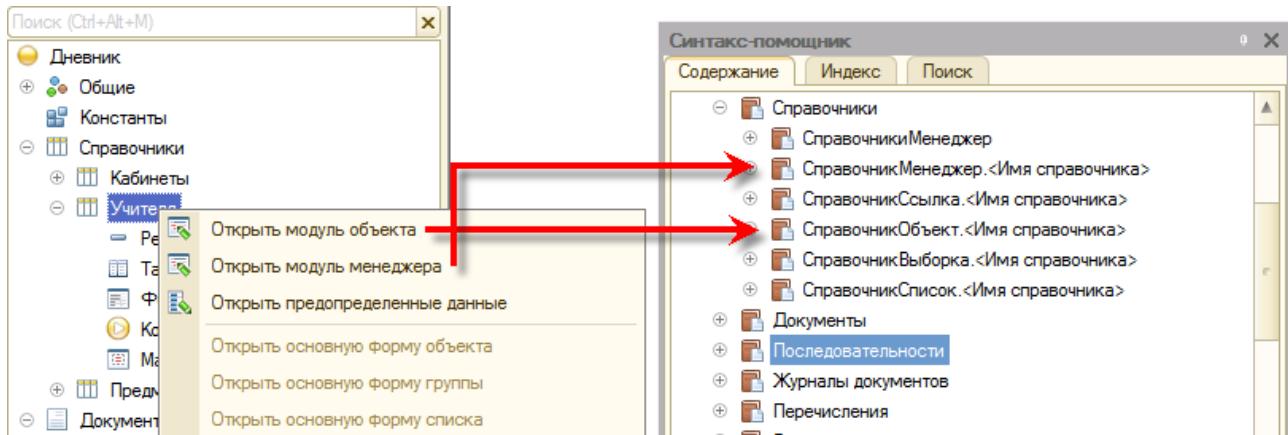


Рисунок 3.233. Команды справочника

Итак, вы хотите обработать событие объекта документа *ПередЗаписью*, поэтому откройте модуль объекта вашего документа *УчебныйДень*.

Чтобы создать обработчик события, раскройте выпадающий список в поле *Процедуры и функции* и выберите интересующее вас событие — *ПередЗаписью* (рисунок 3.234).

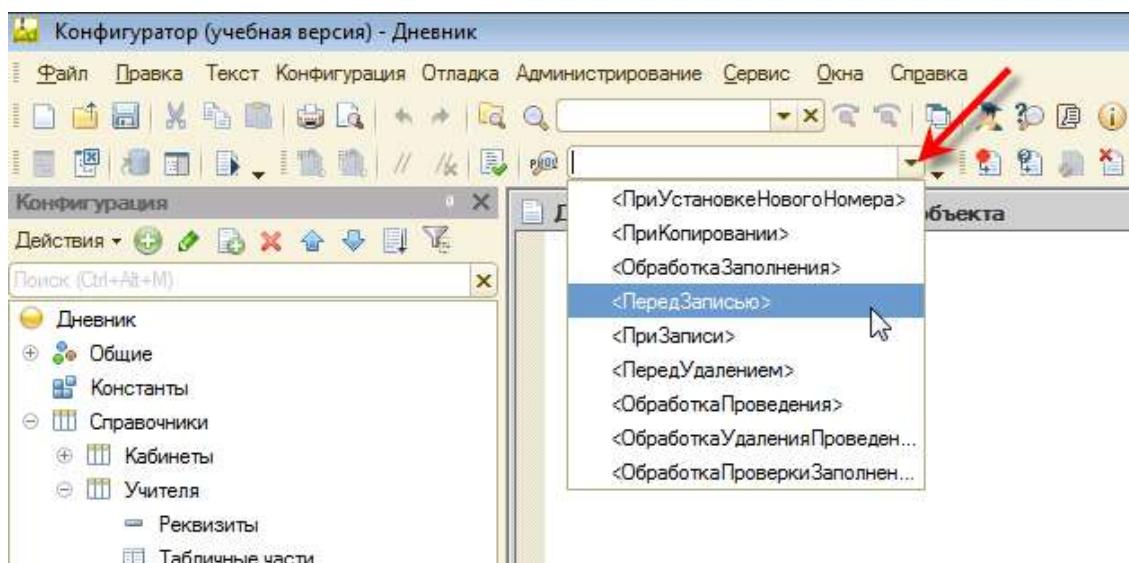


Рисунок 3.234. Создать обработчик события «ПередЗаписью»

Платформа добавит в модуль заготовку процедуры, обрабатывающей это событие (листинг 3.79).

Листинг 3.79. Заготовка обработчика события

```
Процедура ПередЗаписью(Отказ, РежимЗаписи, РежимПроведения)
    // Вставить содержимое обработчика.
КонецПроцедуры
```

У неё уже будут существовать все параметры, которые необходимы для обработки этого события (рисунок 3.235).

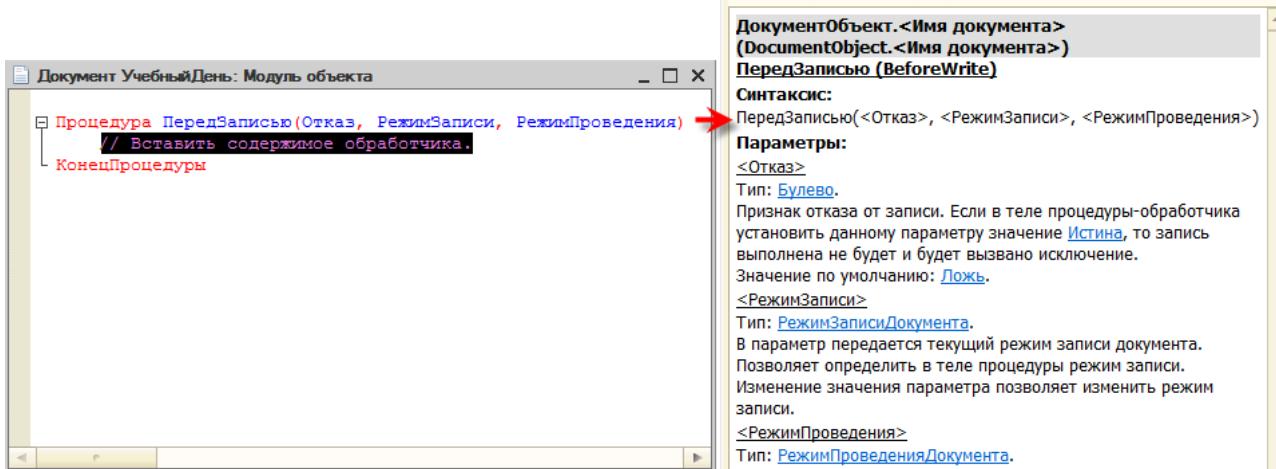


Рисунок 3.235. Заготовка процедуры и описание события в синтакс-помощнике

Вам останется только добавить свои инструкции в эту процедуру. Всё, что вам нужно сделать, это присвоить правильный номер документу. Делается это так (листинг 3.80).

Листинг 3.80. Присвоить номер документу

```
Номер = Серверный.ПолучитьНомер(Дата);
```

Поскольку вы находитесь в модуле объекта, все свойства документа доступны вам «напрямую», то есть просто по их имени. Поэтому слева от знака равенства вы пишете *Номер*. Справа вы обращаетесь к функции, которую вы написали раньше, в общем модуле *Серверный*. Она формирует вам правильный номер документа.

Записывать здесь ничего не нужно, потому что платформа сама выполнит запись объекта, после того как выполнится эта процедура.

Запустите 1С:Предприятие и проверьте, как работает эта инструкция. Добавьте новый документ, ничего не заполняйте в нём, а только нажмите кнопку *Записать*. Вы сразу увидите, что заголовок документа изменится и в нём появится номер документа (рисунок 3.236).

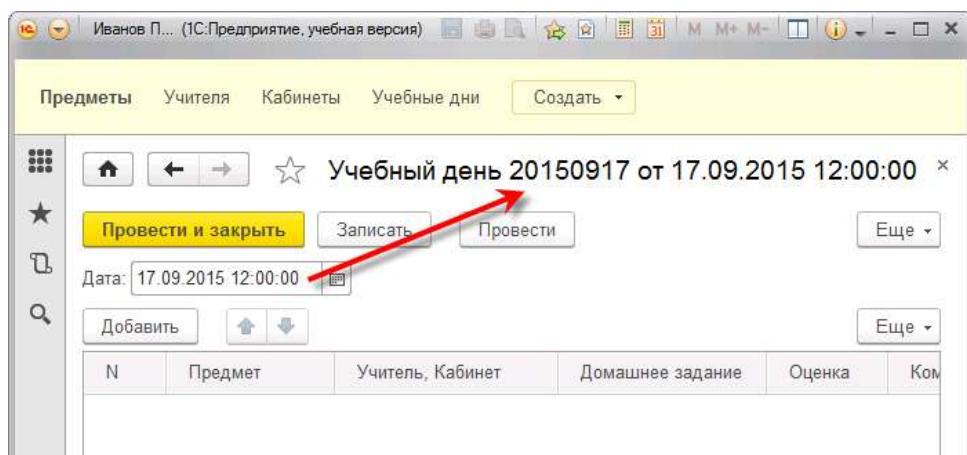


Рисунок 3.236. Номер нового документа соответствует его дате

Если вы всё сделали правильно, то номер нового документа будет соответствовать его дате.

Чтобы этот тестовый проверочный документ не мешал вам, вы можете сразу его удалить. Для этого можете выполнить команду *Удалить* из меню *Ещё* (рисунок 3.237).

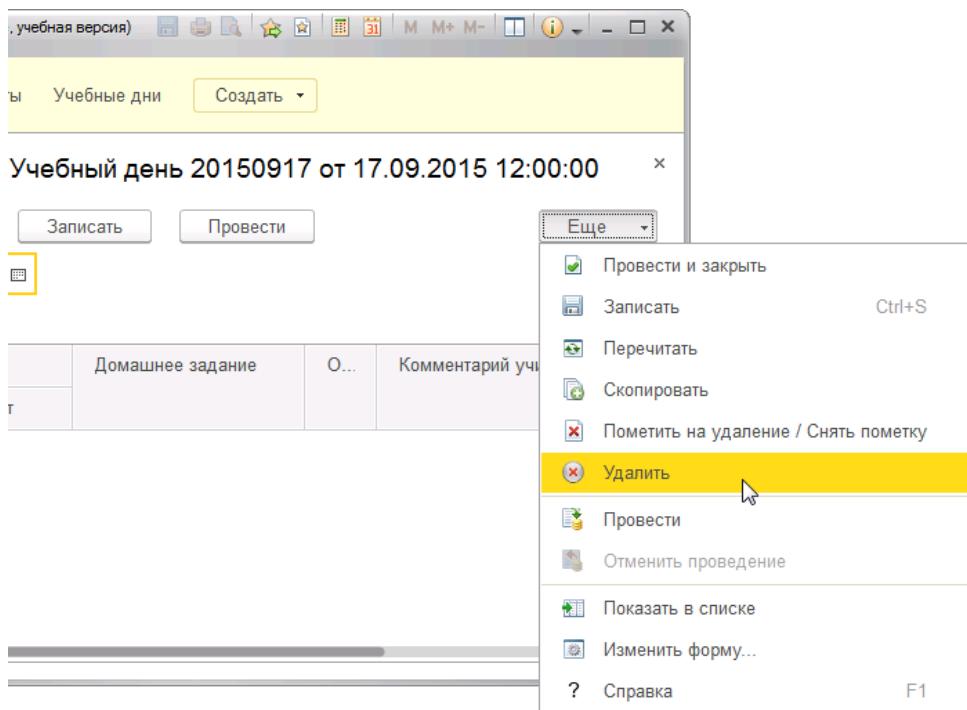


Рисунок 3.237. Удалить документ

Совет

Не забывайте, что модули всех объектов существуют только на сервере. Поэтому в них можно использовать только те процедуры, функции и объекты встроенного языка, которые доступны в контексте сервера.

Теперь вы можете выполнить несколько самостоятельных примеров, чтобы закрепить свои знания. Для выполнения заданий создайте новую информационную базу и загрузите в неё демонстрационную базу «04 НомераДокументов.dt». Как это сделать, написано в разделе А.1 «Как подключить демонстрационную базу» на странице 543.

Информационная база

Прикладное решение, которое должно получиться у вас к этому моменту, содержится в демонстрационной базе «04 НомераДокументов.dt». Как её подключить, написано в разделе А.1 «Как подключить демонстрационную базу» на странице 543.

Задание 3.49

В каждом документе УчебныйДень удалите последний урок. В каждом учебном дне должно остаться по 5 уроков.

Для выполнения этого задания создайте новую процедуру в общем модуле Серверный. Перед запуском и проверкой своей программы запишите где-нибудь, какой урок должен оказаться последним в каждый из дней. Чтобы у вас была возможность проверить правильность работы.

Задание 3.50

Получите документ за 2 сентября 2015 г. и удалите в нём первый урок. Для этого самостоятельно найдите в синтакс-помощнике метод, который позволяет найти документ по номеру. Обработайте ситуацию, что документа с таким номером может и не быть в базе.

Перед выполнением запомните, какой урок был первым, чтобы потом проверить. После проверки в тексте программы измените номер документа на несуществующий. Проверьте, как работает ваша программа в случае, когда искомый документ отсутствует.

Подсказка 1. Чтобы определить, является ли ссылка на документ пустой, у типа *ДокументСсылка.<Имя документа>* существует метод *Пустая()*.

Подсказка 2. Чтобы, находясь на сервере, сообщить что-нибудь клиенту, используйте объект *СообщениеПользователю*. Самостоятельно найдите его описание в синтакс-помощнике. Вам понадобится создать этот объект, заполнить свойство *Текст* и выполнить метод *Сообщить()*.

Задание 3.51

Создайте документ 8 сентября 2015 г. и запишите в него два урока информатики. Самостоятельно найдите метод, который позволяет создать документ.

Подсказка 1. У документа нужно обязательно заполнить дату.

Подсказка 2. Способы работы со справочниками такие же, как и с документами. В справочнике *Предметы* название предмета хранится в поле *Код*.

Подсказка 3. Если вы всё сделали правильно, после запуска приложения вы не увидите свой новый документ в списке. Это связано с тем, что вы делаете учебный пример в обработчике *ПриНачалеРаботыСистемы()*. Но ваш новый документ есть в списке. Чтобы его увидеть, выполните команду *Ещё — Обновить*.

Глава 4

Автоматическое заполнение расписания

Не читайте эту главу!

Если с помощью встроенного языка вы можете:

- перебрать все дни недели;
- найти в базе нужные документы;
- скопировать их данные в новые документы;
- сохранить новые документы в базе;
- обновить список документов;

смело переходите к главе 5 «Регистры и отчёты» на странице 373.

Информационная база

Прикладное решение, которое должно получиться у вас к этому моменту, содержится в демонстрационной базе «04 НомераДокументов.dt». Как её подключить, написано в разделе A.1 «Как подключить демонстрационную базу» на странице 543.

Теперь у вас есть достаточное количество знаний для того, чтобы сделать первую большую доработку в вашем электронном дневнике. Сейчас работать с ним не очень удобно, потому что каждую неделю приходится вручную создавать документы УчебныйДень для следующей недели.

Обычно из недели в неделю расписание постоянное, не меняется. Поэтому было бы гораздо удобнее, чтобы программа сама создавала документы УчебныйДень на следующую неделю. С такими же предметами, которые были на этой неделе. А если появятся какие-то изменения, то вы сделаете их вручную.

Этот алгоритм вы оформите в виде команды документа. И будете вызывать его нажатием на кнопку в списке документов.

Команды в дереве объектов конфигурации расположены в разных местах. Если команда выполняет общие действия, не связанные с конкретными объектами конфигурации, то такие команды вы можете добавлять в ветке *Общие — Общие команды*. А если команда связана с определённым объектом конфигурации, как в вашем случае, то такие команды добавляются в ветку *Команды*, которая есть у каждого объекта конфигурации (рисунок 4.1).

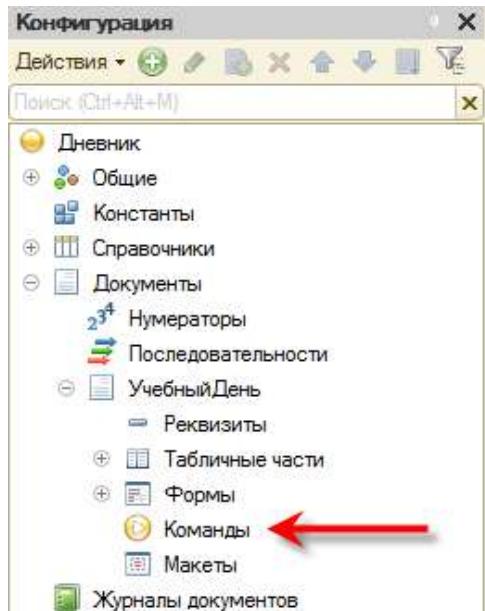


Рисунок 4.1. Команды документа «УчебныйДень»

Добавьте новую команду. Платформа сразу же откроет вам модуль этой команды и палитру свойств. В модуле будет заготовка команды, она вам пока не нужна. Займитесь свойствами команды.

Задайте имя, например *ЗаполнитьРасписаниеНаСледующуюНеделю*. Теперь вам нужно заполнить два важных свойства команды: *Группа* и *Тип параметра команды* (рисунок 4.2).

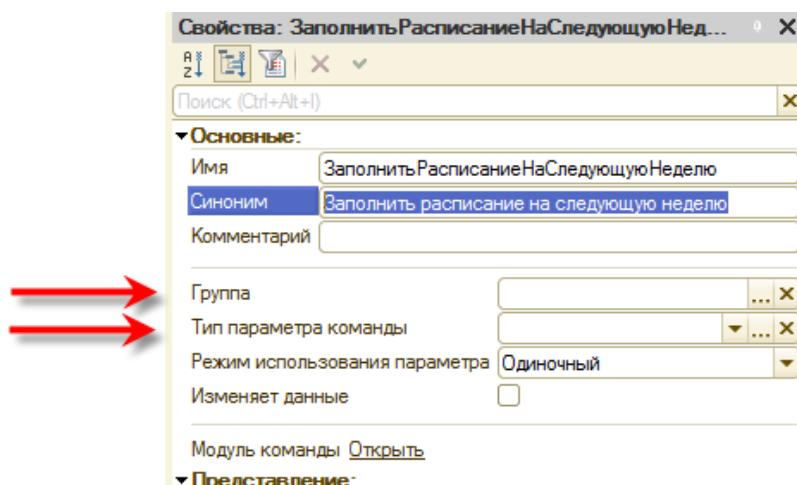


Рисунок 4.2. Свойства команды

Дело в том, что платформа самостоятельно размещает команды в нужных местах интерфейса вашей программы. Чтобы понять, в каком месте нужно отобразить вашу команду, она анализирует именно эти два свойства.

Свойство *Тип параметра команды* определяет, какие данные нужны вашей команде. Платформа поместит вашу команду в те формы, где есть данные этого типа.

Например, если команда создаёт копию учебного дня, то ей нужно знать, какой именно учебный день следует скопировать. То есть ей нужно знать ссылку на конкретный учебный день. Значит, в этом свойстве вы укажете тип *ДокументСсылка.УчебныйДень*.

Тогда платформа поместит эту команду в те формы, где есть такие данные: в форму списка документов и в форму документа. Если вы нажмёте её в форме документа, она

скопирует вам этот документ. А если вы нажмёте её в форме списка, то она скопирует тот документ, который в данный момент выделен в списке. Вот так это работает.

В вашем случае команде не требуется никаких дополнительных данных. Она сама вычислит, какая неделя сейчас, какие документы есть на этой неделе, и скопирует их. Но вы зададите тип параметра команды, для того чтобы платформа знала, в какие формы поместить эту команду. Раз команда копирует учебные дни, будет удобно, чтобы она находилась в списке учебных дней. Значит, задайте тип параметра команды как *ДокументСсылка.УчебныйДень*.

Второе важное свойство, *Группа*, влияет на то, в каком месте формы появится команда. Если вы нажмёте кнопку выбора в этом свойстве, то платформа предложит вам несколько вариантов размещения вашей команды (рисунок 4.3).

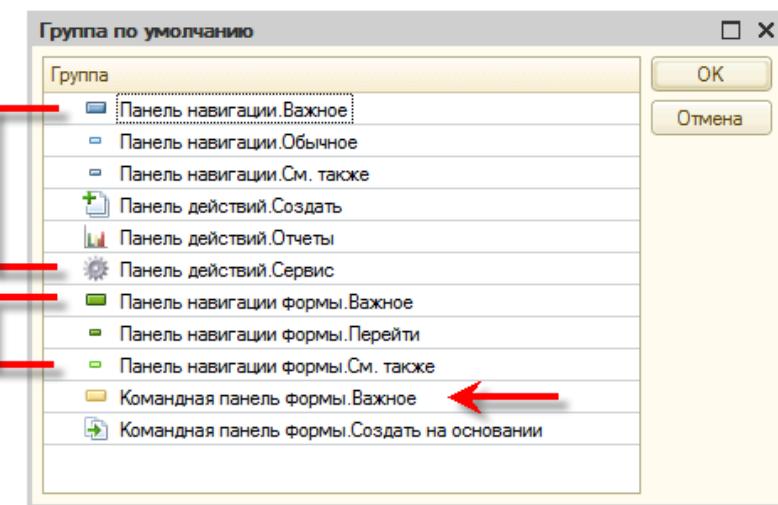


Рисунок 4.3. Варианты размещения команды

Варианты под цифрой 1 используются для общих команд. То есть для команд, которые «не привязаны» к конкретным формам. Остальные варианты описывают два места: панель навигации (цифра 2) и командную панель формы.

В панели навигации обычно располагают команды, которые «переносят» пользователя в другую точку программы: открывают ему список других объектов, показывают список вспомогательных данных и так далее. В командной панели обычно размещают команды, которые «не меняют положение пользователя» в программе, но выполняют какие-то действия с данными.

Для вашего случая как раз подходит командная панель. Потому выберите *Командная панель формы.Важное* (рисунок 4.4).

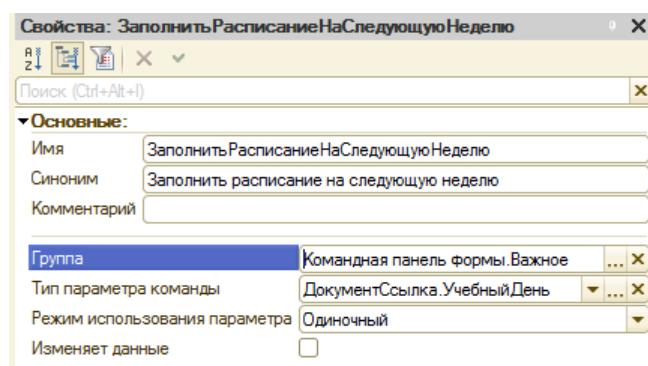


Рисунок 4.4. Свойства команды

Теперь вы можете посмотреть, как это выглядит в программе. Обновите конфигурацию базы данных и запустите 1С:Предприятие. Вы сразу увидите, что ваша команда появилась в списке документов, который находится на начальной странице (рисунок 4.5).

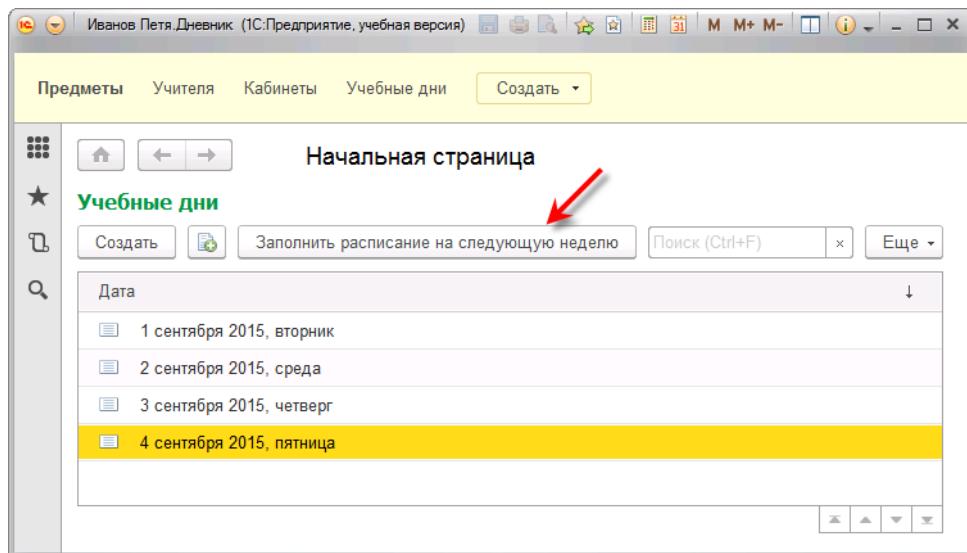


Рисунок 4.5. Команда в списке документов

Совет

Если вы не видите команду в списке документов, возможно, она просто не поместилась по ширине. Растворите форму вправо или влево, и команда появится. Когда для команды не хватает места в форме, она отображается в общем списке команд формы, который вы можете открыть с помощью кнопки *Ещё*.

Таким образом, ваша команда оказалась там, где вы и хотели. В списке документов. Теперь откройте любой из документов. В нём вы тоже увидите свою команду (рисунок 4.6).

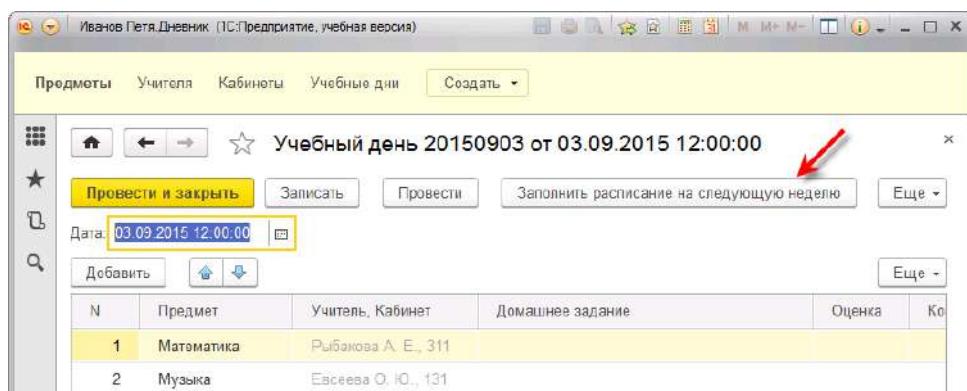


Рисунок 4.6. Команда в форме документа

Это не совсем то, что вы хотели. В документе эта команда не нужна. В этом месте ей пользоваться совсем неудобно. Поэтому из документа её нужно убрать. Делается это очень просто.

Закройте 1С:Предприятие, вернитесь в конфигуратор. Откройте форму документа Учебный День. Перейдите на закладку *Командный интерфейс* (рисунок 4.7).

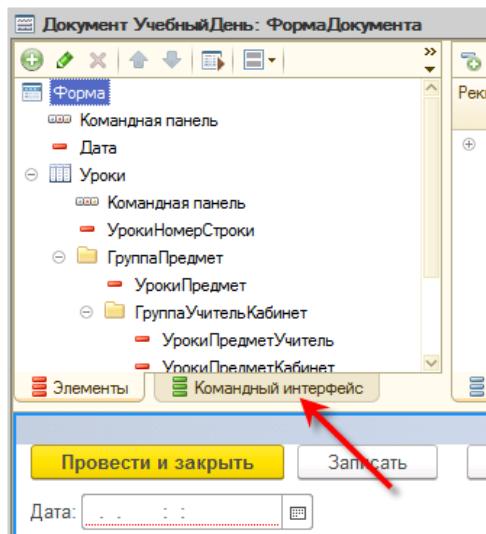


Рисунок 4.7. Закладка «Командный интерфейс»

Чтобы вам лучше было видно, растяните окно команд в ширину. Вот ваша команда (рисунок 4.8).

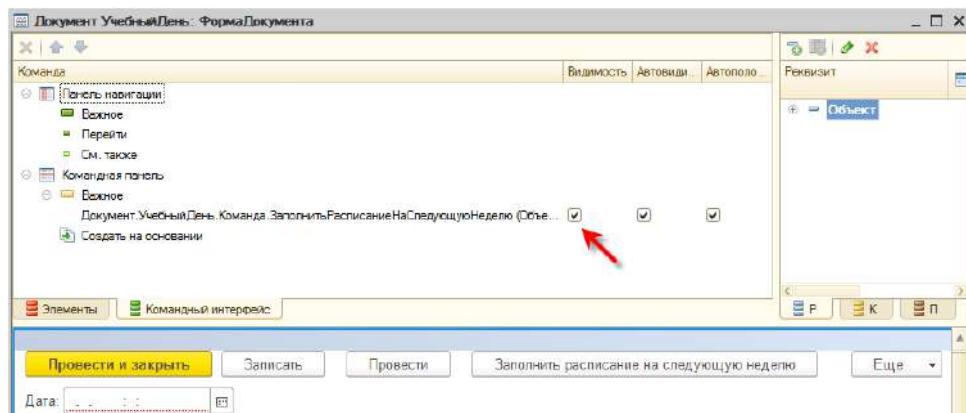


Рисунок 4.8. Команда в интерфейсе формы

Убрать эту команду очень просто. Нужно снять флаг *Видимость*, и она исчезнет из формы.

Обновите конфигурацию базы данных, запустите 1С:Предприятие и убедитесь, что команда исчезла (рисунок 4.9).

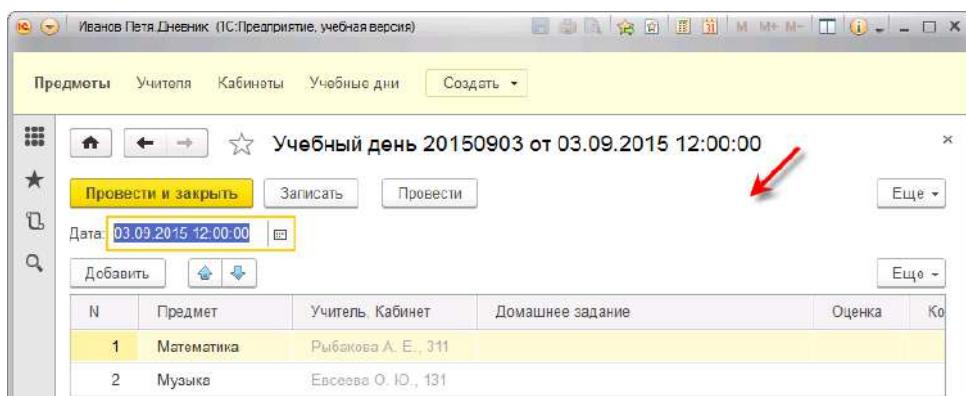


Рисунок 4.9. Команда исчезла из формы документа

Теперь вы можете спокойно заняться модулем команды. Закройте 1С:Предприятие и вернитесь в конфигуратор.

Модуль команды, в отличие от других модулей, с которыми вы уже работали, существует и на клиенте, и на сервере. Поэтому в нём для каждой процедуры в отдельности нужно указывать, в каком контексте она будет исполняться.

Как это делается, вы уже видите. В той «заготовке», которую создала платформа, есть директива компиляции `&НаКлиенте` (листинг 4.1).

Листинг 4.1. Директива компиляции «&НаКлиенте»

```
&НаКлиенте
Процедура ОбработкаКоманды(ПараметрКоманды, ПараметрыВыполненияКоманды)
    //Вставить содержимое обработчика.
    //ПараметрыФормы = Новый Структура("", );
    //ОткрытьФорму("Документ.УчебныйДень.ФормаСписка", ПараметрыФормы, ПараметрыВыполнени...
КонецПроцедуры
```

Директивы компиляции — это специальные ключевые слова. Они указывают платформе, в каком контексте должна исполняться функция или процедура. В модуле команды могут использоваться три такие директивы:

- `&НаКлиенте` — процедура может исполняться только на клиенте;
- `&НаСервере` — процедура может исполняться только на сервере;
- `&НаКлиентеНаСервере` — процедура может исполняться и на клиенте, и на сервере.

Подробнее

На самом деле директив компиляции больше. Обо всех них вы можете прочитать в синтакс-помощнике в разделе *Общее описание встроенного языка — Директивы компиляции*.

Любая команда — это интерактивное действие, которое всегда инициируется пользователем. Это значит, что «начинается» команда на клиенте. Именно поэтому платформа и указала, что обработчик команды должен исполняться на клиенте.

Но ваша команда должна будет прочитать данные из базы данных и записать в неё новые документы. А с базой данных можно работать только на сервере.

Поэтому первое, что вам нужно сделать, это передать исполнение программного кода на сервер. Делается это просто (листинг 4.2).

Листинг 4.2. Процедура «ЗаполнитьРасписаниеНаСервере»

```
&НаКлиенте
Процедура ОбработкаКоманды(ПараметрКоманды, ПараметрыВыполненияКоманды)
    ЗаполнитьРасписаниеНаСервере();
КонецПроцедуры

&НаСервере
Процедура ЗаполнитьРасписаниеНаСервере()

КонецПроцедуры
```

Ниже, в этом же модуле, создайте процедуру и предварите её директивой `&НаСервере`. А в клиентской процедуре `ОбработкаКоманды` вместо закомментированного кода вставьте вызов этой процедуры.

Теперь в серверной процедуре вам нужно написать программу, которая будет копировать учебные дни. Все программисты по-разному подходят к написанию программ. Я покажу вам один из возможных способов.

Сначала, чтобы не забыть и не запутаться, вы с помощью комментариев описываете всё, что нужно сделать. А потом постепенно дополняете комментарии нужными фрагментами кода.

На «языке комментариев» ваш алгоритм может выглядеть следующим образом (листинг 4.3).

Листинг 4.3. Алгоритм «на языке комментариев»

```
// Получить начало этой недели.  
  
// Перебрать все дни этой недели.  
  
// Проверить, что в базе есть старый документ-оригинал.  
  
// Вычислить дату и номер для нового документа.  
  
// Проверить, что в базе еще нет нового документа на эту дату.  
  
// Создать и записать новый документ.
```

Что вам предстоит сделать в соответствии с таким алгоритмом? Прежде всего нужно узнать, когда начинается текущая неделя. После этого нужно перебрать все дни текущей недели и для каждого из них сделать следующее:

- Убедиться в том, что в базе данных есть документ, который вы будете копировать. Потому что, если документа нет, что вы будете копировать?
- Вычислить дату и номер для нового документа. Дата нового документа должна быть на неделю больше.
- Убедиться в том, что в будущей неделе ещё нет документа для этой даты. Откуда он может там появиться? Например, вы скопировали документы один раз. Потом забыли, что вы это делали, и нажали на кнопку ещё раз. Конечно, платформа не даст вам создать ещё один документ с такой же датой и, значит, с таким же номером. Но это будет сопровождаться сообщениями об ошибке, появлением диалогов, в которых пользователю нужно будет нажимать кнопки... Будет гораздо лучше, если такую ситуацию вы обработаете самостоятельно и создадите только те документы, которых ещё нет.
- И, наконец, создать новый документ, заполнить его табличную часть теми предметами, которые есть в документе-оригинале, и записать его в базу данных.

А теперь, когда вы всё записали и уже ничего не забудете, можно последовательно, одну за другой, выполнять записанные задачи.

Первая задача — узнать начало текущей недели. Это просто, вы наверняка можете сделать это самостоятельно (листинг 4.4).

Листинг 4.4. Узнать начало текущей недели

```
// Получить начало этой недели.  
НачалоЭтойНедели = НачалоНедели(ТекущаяДатаСеанса());
```

Теперь вам нужно перебрать все дни текущей недели. Для этого используйте цикл, который каждый раз к началу недели будет прибавлять нужное количество дней. Первый раз — ничего, второй раз — один день (получится вторник), третий раз — два дня и так

далее, включая пятницу. Будем считать, что в субботу и воскресенье занятий нет, поэтому даже не рассматривайте эти дни.

На встроенным языке это будет выглядеть следующим образом (листинг 4.5).

Листинг 4.5. Перебор дней недели

```
// Получить начало этой недели.  
НачалоЭтойНедели = НачалоНедели(ТекущаяДатаСеанса());  
  
// Перебрать все дни этой недели.  
Для КоличествоДобавляемыхДней = 0 По 4 Цикл  
    ДатаСтарого = НачалоЭтойНедели + 60 * 60 * 24 * КоличествоДобавляемыхДней;  
  
    // Проверить, что в базе есть старый документ-оригинал.  
  
    // Вычислить дату и номер для нового документа.  
  
    // Проверить, что в базе еще нет нового документа на эту дату.  
  
    // Создать и записать новый документ.  
  
КонецЦикла;
```

Комментарий. Переменная цикла изменяется от нуля до четырёх. Потому что к понедельнику ничего добавлять не надо, а чтобы получить пятницу, нужно добавить четыре дня.

В теле цикла вы сразу же вычисляете дату старого документа-оригинала. Того самого, который вы будете копировать. $60 * 60 * 24$ — это количество секунд, содержащихся в одних сутках.

Что нужно делать дальше? Дальше вам нужно будет найти документ, который есть в этот день. Как вы помните, уникальным идентификатором, по которому можно найти документ, является его номер. А кроме этого в общем модуле у вас есть функция, которая формирует правильный номер документа из его даты.

Поэтому дальше вы вычисляете номер документа, который собираетесь найти, и находите этот документ (листинг 4.6).

Листинг 4.6. Найти документ-оригинал

```
// Получить начало этой недели.  
НачалоЭтойНедели = НачалоНедели(ТекущаяДатаСеанса());  
  
// Перебрать все дни этой недели.  
Для КоличествоДобавляемыхДней = 0 По 4 Цикл  
    ДатаСтарого = НачалоЭтойНедели + 60 * 60 * 24 * КоличествоДобавляемыхДней;  
    НомерСтарого = Серверный.ПолучитьНомер(ДатаСтарого);  
  
    // Проверить, что в базе есть старый документ-оригинал.  
    СтарыйДокумент = Документы.УчебныйДень.НайтиПоНомеру(НомерСтарого);  
  
    // Вычислить дату и номер для нового документа.  
  
    // Проверить, что в базе еще нет нового документа на эту дату.  
  
    // Создать и записать новый документ.
```

КонецЦикла;

Тут нужно вспомнить о том, что документ может быть в базе, а может и не быть. Почему его может не быть? Потому, что вы его не создавали, забыли. Или потому, что были каникулы и занятий не было. Эту ситуацию нужно обработать. Если документа-оригинала нет, то и нет смысла пытаться его копировать. Можно сразу же переходить к следующему дню. Значит, напишите инструкцию *Если* и обработайте эту ситуацию (листинг 4.7).

Листинг 4.7. Обработка ошибочной ситуации

```
// Получить начало этой недели.  
НачалоЭтойНедели = НачалоНедели(ТекущаяДатаСеанса());  
  
// Перебрать все дни этой недели.  
Для КоличествоДобавляемыхДней = 0 По 4 Цикл  
    ДатаСтарого = НачалоЭтойНедели + 60 * 60 * 24 * КоличествоДобавляемыхДней;  
    НомерСтарого = Серверный.ПолучитьНомер(ДатаСтарого);  
  
    // Проверить, что в базе есть старый документ-оригинал.  
    СтарыйДокумент = Документы.УчебныйДень.НайтиПоНомеру(НомерСтарого);  
  
    // В базе нет документа-оригинала - перейти к следующему дню недели.  
    Если СтарыйДокумент.Пустая() Тогда  
        Продолжить;  
  
    КонецЕсли;  
  
    // Вычислить дату и номер для нового документа.  
  
    // Проверить, что в базе еще нет нового документа на эту дату.  
  
    // Создать и записать новый документ.  
КонецЦикла;
```

Но просто так, молча, пропускать этот день нехорошо. Может быть, вы действительно ошиблись и не создали документ для этого дня. Значит, нужно как-то сообщить пользователю о том, что вы не нашли документ-оригинал.

В тот момент, когда вы находитесь на сервере, нет никакой возможности что-то сообщить пользователю. Но вы можете сформировать нужное вам сообщение. И когда исполнение встроенного языка вернётся на клиент, пользователь увидит это сообщение.

Для этого используется объект встроенного языка, который называется *Сообщение-Пользователю*. Вы легко можете найти его в синтакс-помощнике (с помощью закладки *Индекс*).

Этот объект умеет больше, чем просто показать сообщение пользователю. Но вы сейчас будете использовать его в самом простом варианте. Сначала, с помощью конструктора, создадите объект. Потом в свойство *Текст* установите то сообщение, которое вы хотите показать. И в заключение выполните его метод *Сообщить()* (листинг 4.8).

Листинг 4.8. Сообщение пользователю

```
// Получить начало этой недели.  
НачалоНедели = НачалоНедели(ТекущаяДатаСеанса());  
  
// Перебрать все дни этой недели.  
Для КоличествоДобавляемыхДней = 0 По 4 Цикл  
    ДатаСтарого = НачалоНедели + 60 * 60 * 24 * КоличествоДобавляемыхДней;  
    НомерСтарого = Серверный.ПолучитьНомер(ДатаСтарого);  
  
    // Проверить, что в базе есть старый документ-оригинал.  
    СтарыйДокумент = Документы.УчебныйДень.НайтиПоНомеру(НомерСтарого);  
  
    // В базе нет документа-оригинала - перейти к следующему дню недели.  
    Если СтарыйДокумент.Пустая() Тогда  
        Сообщение = Новый СообщениеПользователю;  
        Сообщение.Текст = "На " + Формат(ДатаСтарого, "ДФ='д ММММ гггг, дддд'") +  
            " нет учебного дня.";  
        Сообщение.Сообщить();  
  
        Продолжить;  
  
    КонецЕсли;  
  
    // Вычислить дату и номер для нового документа.  
  
    // Проверить, что в базе еще нет нового документа на эту дату.  
  
    // Создать и записать новый документ.  
  
КонецЦикла;
```

Комментарий. Текст, который увидит пользователь, будет выглядеть примерно так: «На такое-то число нет учебного дня». Функция *Формат()* используется для того, чтобы дата была написана в том же самом виде, как она указана в списке документов. Вспомните: когда вы «облагораживали» форму списка документа УчебныйДень, вы для даты использовали точно такую же форматную строку (рисунок 4.10).

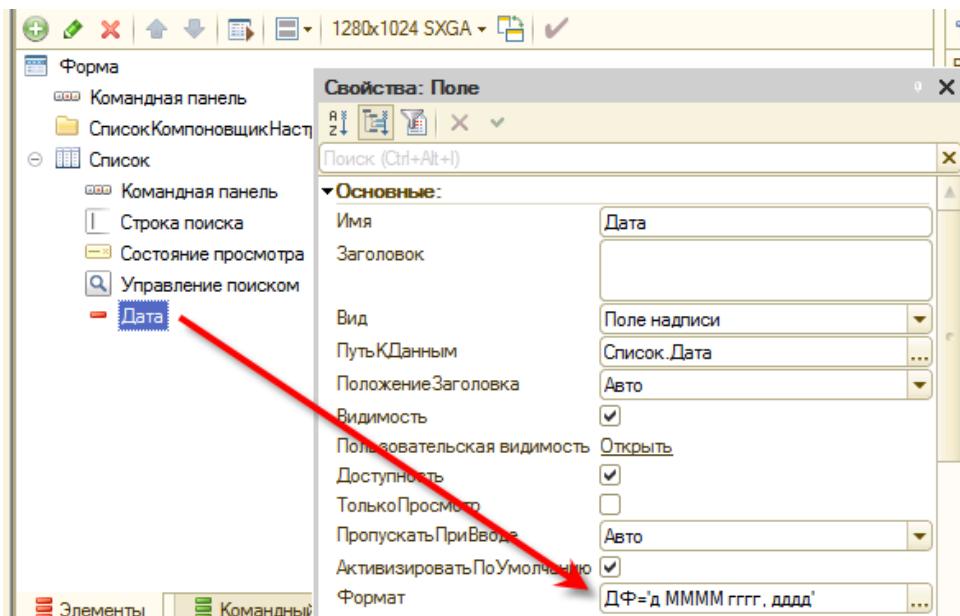


Рисунок 4.10. Формат даты в списке документов

Итак, в «плохом» случае, если документа-оригинала нет, вы сообщили об этом пользователю и перешли к следующему дню. Теперь нужно написать, что делать в «хорошем» случае.

Судя по комментариям, нужно вычислить дату и номер для нового документа, который вы будете создавать. Сделать это просто (листинг 4.9).

Листинг 4.9. Вычислить дату и номер для нового документа

```
// Получить начало этой недели.  
НачалоНедели = НачалоНедели(ТекущаяДатаСеанса());  
  
// Перебрать все дни этой недели.  
Для КоличествоДобавляемыхДней = 0 По 4 Цикл  
    ДатаСтарого = НачалоНедели + 60 * 60 * 24 * КоличествоДобавляемыхДней;  
    НомерСтарого = Серверный.ПолучитьНомер(ДатаСтарого);  
  
    // Проверить, что в базе есть старый документ-оригинал.  
    СтарыйДокумент = Документы.УчебныйДень.НайтиПоНомеру(НомерСтарого);  
  
    // В базе нет документа-оригинала - перейти к следующему дню недели.  
    Если СтарыйДокумент.Пустая() Тогда  
        Сообщение = Новый СообщениеПользователю;  
        Сообщение.Текст = "На " + Формат(ДатаСтарого, "ДФ='д ММММ гггг, дддд'") +  
            " нет учебного дня.";  
        Сообщение.Сообщить();  
  
        Продолжить;  
  
КонецЕсли;  
  
// Вычислить дату и номер для нового документа.  
ДатаНового = ДатаСтарого + 60 * 60 * 24 * 7; // к дате старого прибавить неделю  
НомерНового = Серверный.ПолучитьНомер(ДатаНового);  
  
// Проверить, что в базе еще нет нового документа на эту дату.  
  
// Создать и записать новый документ.  
  
КонецЦикла;
```

Комментарий. Чтобы узнать дату нового документа, нужно к дате документа-оригинала прибавить одну неделю. То есть семь раз по одним суткам. А номер для нового документа вы снова получите с помощью процедуры, которая есть у вас в общем модуле.

Что нужно сделать дальше? Чтобы проверить, что такого документа ещё нет, нужно его попробовать найти. Если это закончится неудачей и вы его не найдёте, значит, всё хорошо. Можно создавать новый документ. А если поиск закончится удачей, значит, новый документ создавать не надо. Так и напишите (листинг 4.10).

Листинг 4.10. Проверить, что в базе ещё нет нового документа

```

// Получить начало этой недели.
НачалоНедели = НачалоНедели(ТекущаяДатаСеанса()) ;

// Перебрать все дни этой недели.
Для КоличествоДобавляемыхДней = 0 По 4 Цикл
    ДатаСтарого = НачалоНедели + 60 * 60 * 24 * КоличествоДобавляемыхДней;
    НомерСтарого = Серверный.ПолучитьНомер(ДатаСтарого);

    // Проверить, что в базе есть старый документ-оригинал.
    СтарыйДокумент = Документы.УчебныйДень.НайтиПоНомеру(НомерСтарого);

    // В базе нет документа-оригинала - перейти к следующему дню недели.
    Если СтарыйДокумент.Пустая() Тогда
        Сообщение = Новый СообщениеПользователю;
        Сообщение.Текст = "На " + Формат(ДатаСтарого, "ДФ='д ММММ гггг, дддд'"') +
            " нет учебного дня." ;
        Сообщение.Сообщить();

        Продолжить;

    КонецЕсли;

    // Вычислить дату и номер для нового документа.
    ДатаНового = ДатаСтарого + 60 * 60 * 24 * 7; // к дате старого прибавить неделю
    НомерНового = Серверный.ПолучитьНомер(ДатаНового);

    // Проверить, что в базе еще нет нового документа на эту дату.
    Если Документы.УчебныйДень.НайтиПоНомеру(НомерНового).Пустая() Тогда

        // Создать и записать новый документ.

    Иначе

        // Уже есть новый документ на эту дату - перейти к следующему дню недели.

    КонецЕсли;

КонецЦикла;

```

Комментарий. Первый раз, когда вы искали старый документ, вы сохраняли его в отдельной переменной. И потом анализировали эту переменную. Здесь вы сразу анализируете результат поиска, не сохраняя его ни в какие переменные. Почему так?

Потому, что старый документ вам сейчас пригодится. Из него нужно будет скопировать табличную часть. Поэтому вы и сохранили его в переменной. А новый документ, даже если вы его найдёте, ни для чего не нужен. Поэтому вы и не сохраняете его в переменной.

Что делать дальше? Теперь вы можете сначала обработать ошибочную ситуацию, когда новый документ уже есть. Тут нужно вывести пользователю сообщение, похожее на то, которое вы уже писали. Поэтому можете просто скопировать те три строки и изменить текст сообщения (листинг 4.11).

Листинг 4.11. Сообщение о том, что новый документ уже существует

```

// Получить начало этой недели.
НачалоНедели = НачалоНедели(ТекущаяДатаСеанса()) ;

// Перебрать все дни этой недели.
Для КоличествоДобавляемыхДней = 0 По 4 Цикл
    ДатаСтарого = НачалоНедели + 60 * 60 * 24 * КоличествоДобавляемыхДней;
    НомерСтарого = Серверный.ПолучитьНомер(ДатаСтарого);

    // Проверить, что в базе есть старый документ-оригинал.
    СтарыйДокумент = Документы.УчебныйДень.НайтиПоНомеру(НомерСтарого);

    // В базе нет документа-оригинала - перейти к следующему дню недели.
    Если СтарыйДокумент.Пустая() Тогда
        Сообщение = Новый СообщениеПользователю;
        Сообщение.Текст = "На " + Формат(ДатаСтарого, "ДФ='д ММММ гггг, дддд'") +
            " нет учебного дня." ;
        Сообщение.Сообщить();

        Продолжить;

    КонецЕсли;

    // Вычислить дату и номер для нового документа.
    ДатаНового = ДатаСтарого + 60 * 60 * 24 * 7; // к дате старого прибавить неделю
    НомерНового = Серверный.ПолучитьНомер(ДатаНового);

    // Проверить, что в базе еще нет нового документа на эту дату.
    Если Документы.УчебныйДень.НайтиПоНомеру(НомерНового).Пустая() Тогда

        // Создать и записать новый документ.

    Иначе

        // Уже есть новый документ на эту дату - перейти к следующему дню недели.
        Сообщение = Новый СообщениеПользователю;
        Сообщение.Текст = "Уже есть учебный день " +
            Формат(ДатаНового, "ДФ='д ММММ гггг, дддд'") ;
        Сообщение.Сообщить();

    КонецЕсли;

КонецЦикла;

```

Комментарий. Здесь инструкция *Продолжить* не нужна, так как после этих строк нет исполняемых инструкций и в любом случае будет выполнен переход к следующей итерации цикла.

Теперь вы подошли к самому главному. Нужно создать новый документ и скопировать в него предметы, которые есть в табличной части документа-оригинала.

Создайте новый документ и обязательно установите ему дату. Номер можно не устанавливать, потому что он вычислится автоматически при записи документа (листинг 4.12).

Листинг 4.12. Создать новый документ

```

// Получить начало этой недели.
НачалоНедели = НачалоНедели(ТекущаяДатаСеанса()) ;

// Перебрать все дни этой недели.
Для КоличествоДобавляемыхДней = 0 По 4 Цикл
    ДатаСтарого = НачалоНедели + 60 * 60 * 24 * КоличествоДобавляемыхДней;
    НомерСтарого = Серверный.ПолучитьНомер(ДатаСтарого);

    // Проверить, что в базе есть старый документ-оригинал.
    СтарыйДокумент = Документы.УчебныйДень.НайтиПоНомеру(НомерСтарого);

    // В базе нет документа-оригинала - перейти к следующему дню недели.
    Если СтарыйДокумент.Пустая() Тогда
        Сообщение = Новый СообщениеПользователю;
        Сообщение.Текст = "На " + Формат(ДатаСтарого, "ДФ='д ММММ гггг, дддд')");
        Сообщение.Текст += " нет учебного дня." ;
        Сообщение.Сообщить();

        Продолжить;

    КонецЕсли;

    // Вычислить дату и номер для нового документа.
    ДатаНового = ДатаСтарого + 60 * 60 * 24 * 7; // к дате старого прибавить неделю
    НомерНового = Серверный.ПолучитьНомер(ДатаНового);

    // Проверить, что в базе еще нет нового документа на эту дату.
    Если Документы.УчебныйДень.НайтиПоНомеру(НомерНового).Пустая() Тогда

        // Создать и записать новый документ.
        НовыйДокумент = Документы.УчебныйДень.СоздатьДокумент();
        НовыйДокумент.Дата = ДатаНового;

    Иначе

        // Уже есть новый документ на эту дату - перейти к следующему дню недели.
        Сообщение = Новый СообщениеПользователю;
        Сообщение.Текст = "Уже есть учебный день " +
            Формат(ДатаНового, "ДФ='д ММММ гггг, дддд')");
        Сообщение.Сообщить();

    КонецЕсли;

КонецЦикла;

```

Теперь вам нужно перебрать всю табличную часть старого документа. На каждой итерации цикла создайте в новом документе строку табличной части и запишите в неё предмет из документа-оригинала (листинг 4.13).

Листинг 4.13. Обход табличной части и копирование предметов в новый документ

```

// Получить начало этой недели.
НачалоНедели = НачалоНедели(ТекущаяДатаСеанса()) ;

// Перебрать все дни этой недели.
Для КоличествоДобавляемыхДней = 0 По 4 Цикл
    ДатаСтарого = НачалоНедели + 60 * 60 * 24 * КоличествоДобавляемыхДней;
    НомерСтарого = Серверный.ПолучитьНомер(ДатаСтарого);

    // Проверить, что в базе есть старый документ-оригинал.
    СтарыйДокумент = Документы.УчебныйДень.НайтиПоНомеру(НомерСтарого);

    // В базе нет документа-оригинала - перейти к следующему дню недели.
    Если СтарыйДокумент.Пустая() Тогда
        Сообщение = Новый СообщениеПользователю;
        Сообщение.Текст = "На " + Формат(ДатаСтарого, "ДФ='д ММММ гггг, дддд'"') +
            " нет учебного дня." ;
        Сообщение.Сообщить();

        Продолжить;

    КонецЕсли;

    // Вычислить дату и номер для нового документа.
    ДатаНового = ДатаСтарого + 60 * 60 * 24 * 7; // к дате старого прибавить неделю
    НомерНового = Серверный.ПолучитьНомер(ДатаНового);

    // Проверить, что в базе еще нет нового документа на эту дату.
    Если Документы.УчебныйДень.НайтиПоНомеру(НомерНового).Пустая() Тогда

        // Создать и записать новый документ.
        НовыйДокумент = Документы.УчебныйДень.СоздатьДокумент();
        НовыйДокумент.Дата = ДатаНового;
        Для Каждого СтрокаСтарого Из СтарыйДокумент.Уроки Цикл
            НоваяСтрока = НовыйДокумент.Уроки.Добавить();
            НоваяСтрока.Предмет = СтрокаСтарого.Предмет;

        КонецЦикла;

    Иначе

        // Уже есть новый документ на эту дату - перейти к следующему дню недели.
        Сообщение = Новый СообщениеПользователю;
        Сообщение.Текст = "Уже есть учебный день " +
            Формат(ДатаНового, "ДФ='д ММММ гггг, дддд'"');
        Сообщение.Сообщить();

    КонецЕсли;

КонецЦикла;

```

И, наконец, самое важно и главное, что осталось сделать, — это записать новый документ (листинг 4.14).

Листинг 4.14. Записать документ

```

// Получить начало этой недели.
НачалоНедели = НачалоНедели(ТекущаяДатаСеанса()) ;

// Перебрать все дни этой недели.
Для КоличествоДобавляемыхДней = 0 По 4 Цикл
    ДатаСтарого = НачалоНедели + 60 * 60 * 24 * КоличествоДобавляемыхДней;
    НомерСтарого = Серверный.ПолучитьНомер(ДатаСтарого);

    // Проверить, что в базе есть старый документ-оригинал.
    СтарыйДокумент = Документы.УчебныйДень.НайтиПоНомеру(НомерСтарого);

    // В базе нет документа-оригинала - перейти к следующему дню недели.
    Если СтарыйДокумент.Пустая() Тогда
        Сообщение = Новый СообщениеПользователю;
        Сообщение.Текст = "На " + Формат(ДатаСтарого, "ДФ='д ММММ гггг, дддд')");
        Сообщение.Текст += " нет учебного дня." ;
        Сообщение.Сообщить();

        Продолжить;

    КонецЕсли;

    // Вычислить дату и номер для нового документа.
    ДатаНового = ДатаСтарого + 60 * 60 * 24 * 7; // к дате старого прибавить неделю
    НомерНового = Серверный.ПолучитьНомер(ДатаНового);

    // Проверить, что в базе еще нет нового документа на эту дату.
    Если Документы.УчебныйДень.НайтиПоНомеру(НомерНового).Пустая() Тогда

        // Создать и записать новый документ.
        НовыйДокумент = Документы.УчебныйДень.СоздатьДокумент();
        НовыйДокумент.Дата = ДатаНового;
        Для Каждого СтрокаСтарого Из СтарыйДокумент.Уроки Цикл
            НоваяСтрока = НовыйДокумент.Уроки.Добавить();
            НоваяСтрока.Предмет = СтрокаСтарого.Предмет;

        КонецЦикла;

        НовыйДокумент.Записать();

    Иначе

        // Уже есть новый документ на эту дату - перейти к следующему дню недели.
        Сообщение = Новый СообщениеПользователю;
        Сообщение.Текст = "Уже есть учебный день " +
            Формат(ДатаНового, "ДФ='д ММММ гггг, дддд')");
        Сообщение.Сообщить();

    КонецЕсли;

КонецЦикла;

```

Вроде бы пример готов. Но есть ещё один последний штрих, который нужно сделать. Команда, которую вы сейчас написали, будет вызываться из списка учебных дней. Она создаст ещё пять новых учебных дней. И вы, наверное, ожидаете сразу же увидеть их в списке. Но этого не произойдёт.

Дело в том, что сервер не может «по своему желанию» ничего сообщить клиенту. В частности, он не имеет возможности как-то известить клиента о том, что в базе данных появились новые учебные дни и их надо показать в списке.

Но это можете сделать вы. Когда исполнение встроенного языка вернётся с сервера на клиента, вы можете попросить платформу, чтобы она обновила список учебных дней. Проще всего это сделать следующим образом (листинг 4.15).

Листинг 4.15. Оповестить списки об изменении

```
&НаКлиенте
Процедура ОбработкаКоманды(ПараметрКоманды, ПараметрыВыполненияКоманды)
```

```
ЗаполнитьРасписаниеНаСервере();
ОповеститьОбИзменении(ПараметрКоманды);
```

```
КонецПроцедуры
```

```
&НаСервере
Процедура ЗаполнитьРасписаниеНаСервере()
```

```
// Получить начало этой недели.
НачалоЭтойНедели = НачалоНедели(ТекущаяДатаСеанса());
```

В клиентскую процедуру *ОбработкаКоманды()* после вызова серверной процедуры добавить вызов функции *ОповеститьОбИзменении()*. Описание этой функции вы можете прочитать в синтаксис-помощнике. В двух словах, она делает следующее. Вы передаёте ей ссылку. Она оповещает все списки, которые открыты на клиенте и которые могут содержать эту ссылку, о том, что им нужно обновить свои данные. А какую ссылку вы ей передаёте? Ту, что находится в параметре команды. А там находится, если вы помните, та строка, которая выделена в списке учебных дней. И хотя именно эта строка в вашем примере никак не меняется, главное, что платформа обновит список учебных дней и вы увидите в нём новые учебные дни.

Всё. Теперь пример готов. Обновите конфигурацию базы данных.

На всякий случай, перед тем как запустить и проверить этот пример, сделайте копию вашей базы данных. Она пригодится вам в том случае, если вы допустили в примере какую-нибудь ошибку. Например, по ошибке вместо семи новых документов у вас создаются семьдесят документов. Тогда вы просто скопируете обратно базу данных, и у вас снова будут те данные, которые есть сейчас. Как найти файл базы данных, можете посмотреть в разделе 3.11.1 «База данных» на страницах 310–311.

Итак, запустите 1С:Предприятие в режиме отладки и проверьте, как работает ваша команда.

Если у вас нет учебных дней для текущей недели, это хорошо. Тогда можно проверить, как работает проверка их отсутствия. Если у вас есть все документы для текущей недели, задайте одному из них другую дату и сохраните. Чтобы проверить, обнаружит программа его отсутствие или нет.

В моей базе данных нет учебных дней для текущей недели, поэтому команда сообщит следующее (рисунок 4.11).

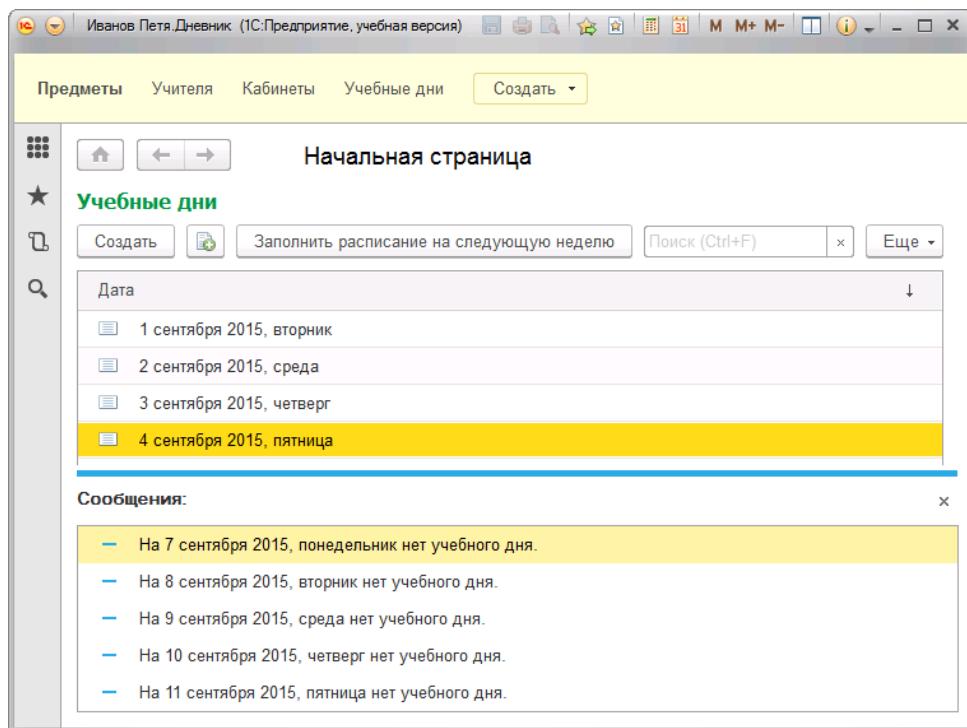


Рисунок 4.11. Нет учебных дней для текущей недели

Закройте окно сообщений и создайте четыре учебных дня копированием. Один день всё-таки пропустите (рисунок 4.12).

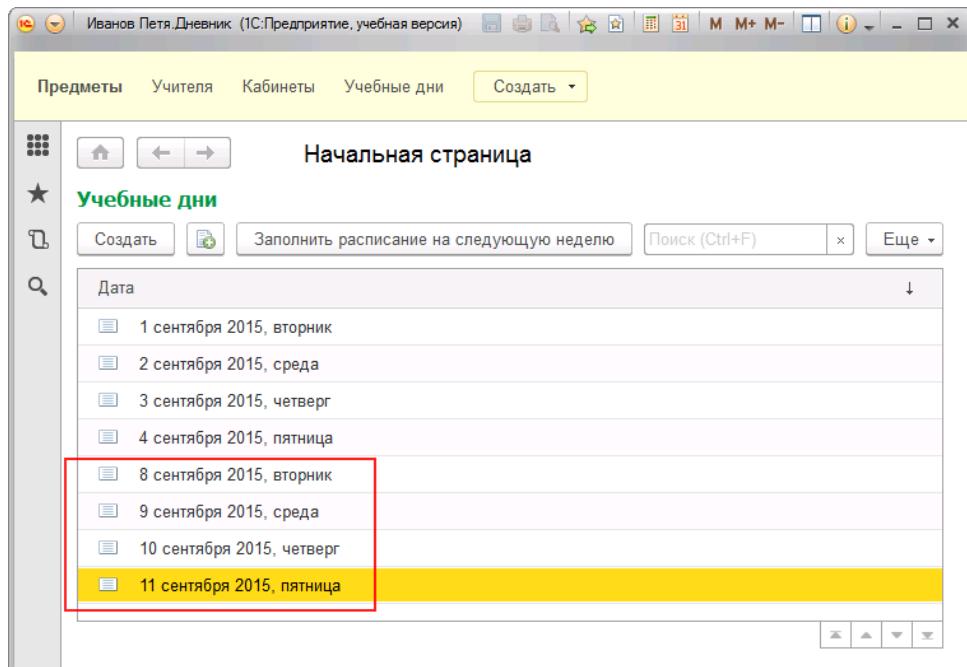


Рисунок 4.12. Создадим четыре учебных дня

Снова выполните команду. Появятся четыре новых документа и сообщение (рисунок 4.13).

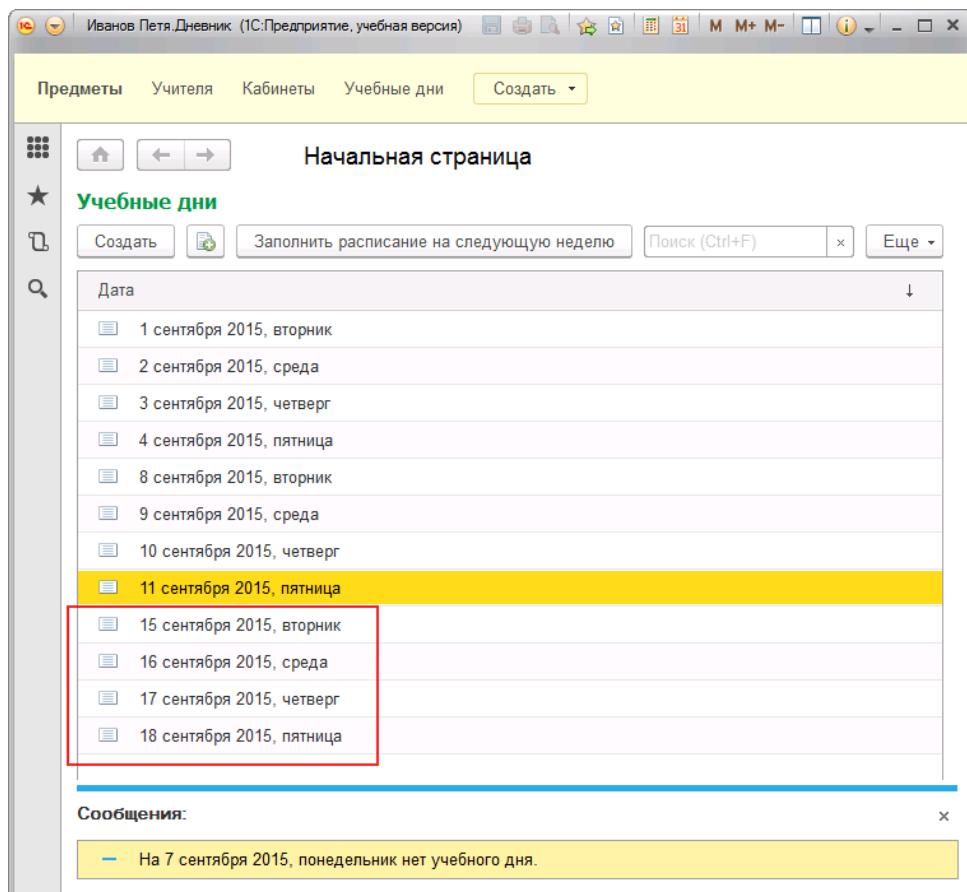


Рисунок 4.13. Четыре новых документа

Очень хорошо. Значит, новые документы создаются как надо. Осталось проверить, что команда правильно поведёт себя в том случае, когда часть новых документов уже есть.

Создайте недостающий учебный день 7 сентября и ещё раз выполните команду (рисунок 4.14).

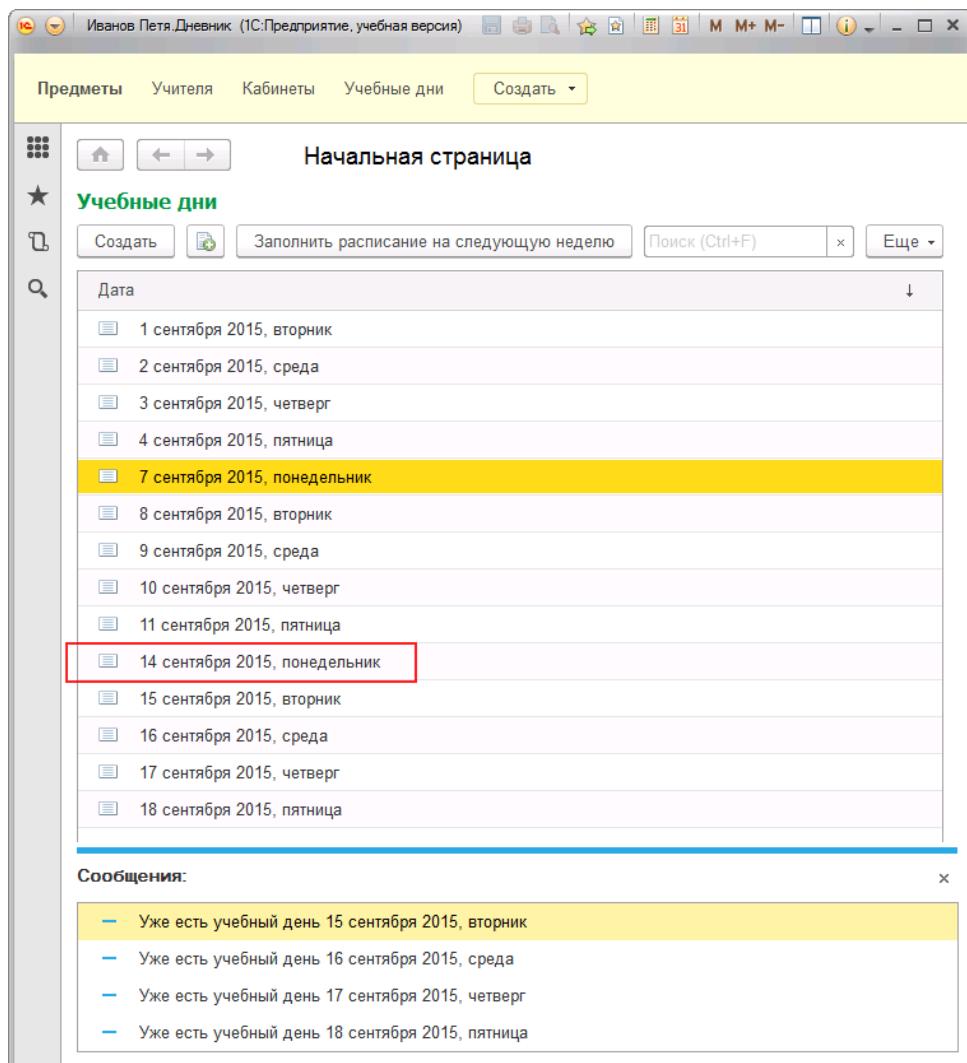


Рисунок 4.14. Правильное поведение, когда новые учебные дни уже есть

В списке появится недостающий понедельник будущей недели, а про остальные дни программа скажет, что они уже есть, и записывать их не станет.

Отлично! Ваша команда работает правильно!

Информационная база

Прикладное решение, которое должно получиться у вас к этому моменту, содержится в демонстрационной базе «05 Копирование Учебных Дней.dt». Как её подключить, написано в разделе [А.1 «Как подключить демонстрационную базу»](#) на странице [543](#).

Задание 4.1

Измените алгоритм создания новых учебных дней так, чтобы они копировались не с текущей недели, а с той недели, день которой выделен в списке. Для выполнения задания создайте новую информационную базу и загрузите в неё демонстрационную базу «05 КопированиеУчебныхДней.dt». Как её подключить, написано в разделе [A.1 «Как подключить демонстрационную базу»](#) на странице [543](#).

Подсказка 1. День, выделенный в списке, содержится в параметре *ПараметрКоманды*.

Подсказка 2. На клиенте свойства объекта *ДокументСсылка.<Имя документа>* недоступны. Их можно прочитать только на сервере.

Подсказка 3. Раньше, чтобы создать новый документ, вы к дате старого документа прибавляли неделю. Теперь этот интервал будет другим, и его нужно вычислить в начале алгоритма.

Глава 5

Регистры и отчёты

Не читайте эту главу!

Если вы знаете:

- можно ли обойтись без регистров;
- зачем проводить документы;
- для чего нужны ресурсы, а для чего — измерения;
- почему система компоновки данных похожа на хорошую хозяйку;
- как создать структуру отчёта;
- чем группировка отличается от детальных записей;
- как нарисовать диаграмму;
- когда нужен регистр сведений, а когда — накопления;
- как самому записать движения в регистр;

смело переходите к главе 6 «Язык запросов» на странице 449.

Информационная база

Прикладное решение, которое должно получиться у вас к этому моменту, содержится в демонстрационной базе «05 КопированиеУчебныхДней.dt». Как её подключить, написано в разделе A.1 «Как подключить демонстрационную базу» на странице 543.

На протяжении последних двух глав вы очень близко и подробно занимались самыми мелкими деталями 1С:Предприятия, то есть встроенным языком. Теперь настало время выбраться из «глубин» системы и снова посмотреть на всё дерево прикладного решения в целом (рисунок 5.1).

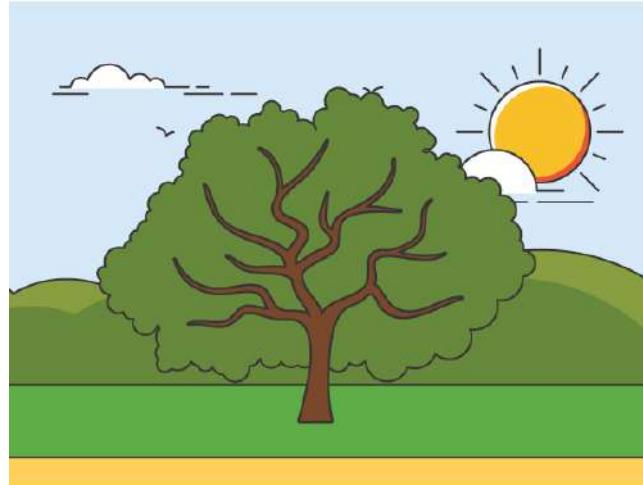


Рисунок 5.1. Сейчас вы тут

В этой главе вам предстоит познакомиться с двумя новыми большими ветками, которые крепятся к его стволу, — это регистры и отчёты. Эти ветки очень важны для любого прикладного решения. И сейчас как раз настал подходящий момент, чтобы познакомиться с ними.

5.1 Зачем нужны регистры

Первые объекты конфигурации, с которыми вы будете знакомиться в этой главе, называются *регистры*. В конфигурации могут быть регистры нескольких видов (рисунок 5.2).

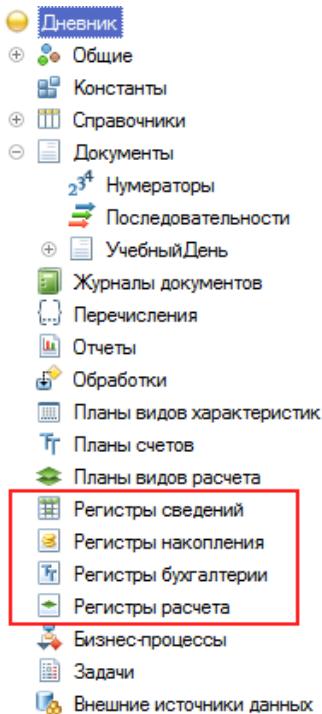


Рисунок 5.2. Регистры

Каждый из них имеет свои особенности и своё назначение. Однако по большому счёту смысл их существования в конфигурации одинаков. Поэтому сейчас, не вдаваясь в подробности, я расскажу, зачем вообще понадобились в конфигурации какие-то регистры.

Смотрите: сейчас, казалось бы, в вашей конфигурации есть всё, что нужно. Вы можете создать учителей, кабинеты, предметы. Можете записать, какие занятия проходят у вас в каждый день недели. Можете записать полученные оценки и домашние задания.

Но через некоторое время вам обязательно захочется узнать, какой у вас, например, средний балл по математике. Или сколько занятий по литературе было с начала учебного года. Или какие задания задали по информатике на следующую неделю.

Чтобы ответить на эти вопросы, вам понадобятся *отчёты*. Это пока незнакомый для вас объект конфигурации. Но ничего страшного. Даже из названия, «отчёты», вы понимаете: это что-то такое, что выберет из базы данных нужную вам информацию и покажет её в удобном виде.

Теоретически, вы можете создать один или несколько таких отчётов уже сейчас. Ведь в документах УчебныйДень у вас есть вся необходимая информация. Какие занятия прошли. Какие оценки вы получили. Что вам задали.

Таким образом, вы можете (теоретически) с помощью отчёта проанализировать всю эту информацию и получить необходимые итоговые данные (рисунок 5.3).

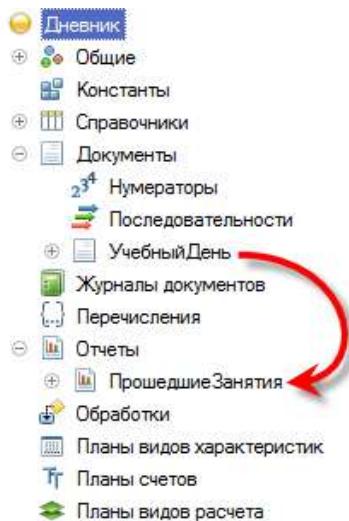


Рисунок 5.3. Неправильный вариант использования отчётов

Но это будет работать до тех пор, пока вы не решите усовершенствовать вашу конфигурацию. А это обязательно произойдёт. Например, вы захотите вести учёт более широко. И учитывать не только те занятия, которые проходят «по программе», но и внеклассные занятия: кружки, игры, секции, олимпиады и так далее.

Для этого вам придётся добавить в конфигурацию ещё один документ. Пусть он будет называться *ВнеклассноеЗанятие*. А это значит, что придётся переделывать и ваш отчёт. Ведь он учитывает только документ УчебныйДень, который существовал ранее (рисунок 5.4).

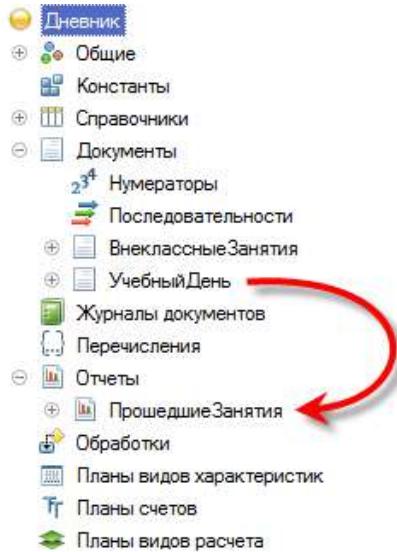


Рисунок 5.4. При добавлении новых видов занятий придётся переделывать отчёт

Чтобы избежать такой ситуации и чтобы не собирать сложным образом итоговые данные по многим документам, в 1С:Предприятии существуют регистры.

Регистры специальным образом хранят учётные данные, поставляемые, например, документами. А документы имеют возможность *проведения*. При проведении документ по некоторому алгоритму записывает свои данные в регистр. А отчёты оперируют уже не данными документов, а теми данными, которые хранятся в регистрах (рисунок 5.5).

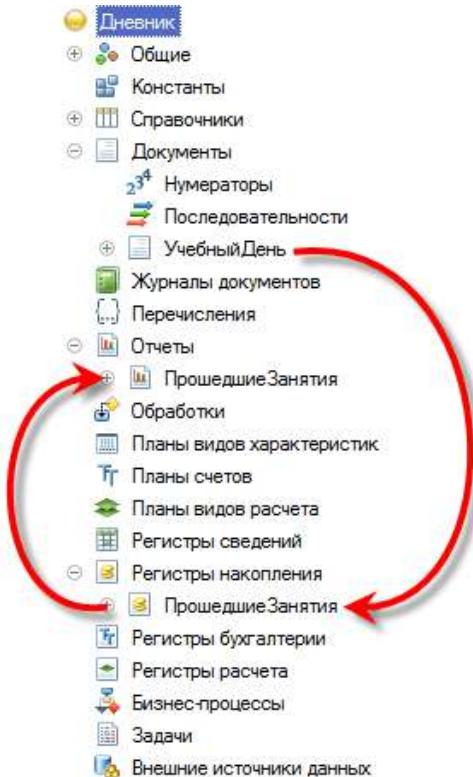


Рисунок 5.5. Отчёты оперируют данными, которые хранятся в регистрах

Таким образом, дело отчёта — выбрать из регистра данные в нужном вам разрезе. А дело документа — поместить свои данные в регистр в соответствии с прикладной логикой.

Регистры специальным образом обрабатывают данные, которые в них помещают документы. Так, чтобы отчёты могли быстро получить нужные им итоговые значения. А

добавление в конфигурацию новых документов сводится к тому, чтобы правильно поместить данные документа в нужные регистры. При этом имеющиеся отчёты остаются без изменений (рисунок 5.6).

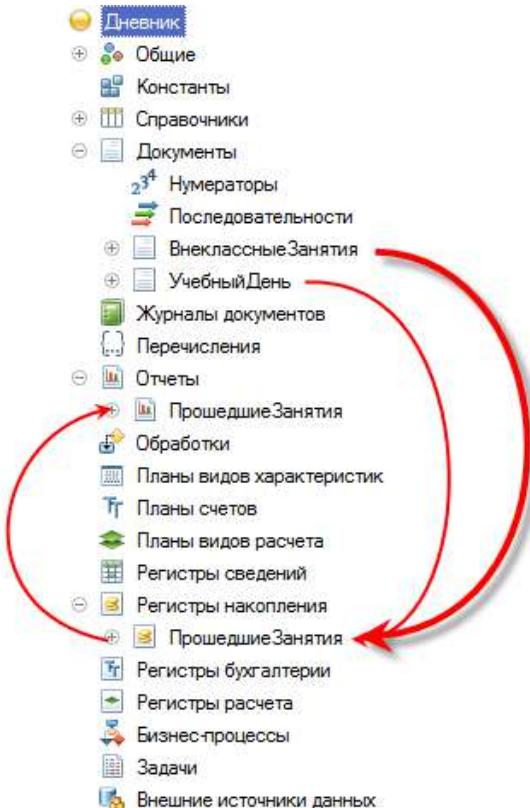


Рисунок 5.6. Новый документ должен правильно поместить свои данные в регистр

5.2 Что будет в этой главе

Дальше в этой главе вы создадите два отчёта. Для каждого из этих отчётов вы сделаете свою «систему хранения данных», свой регистр.

Первый отчёт, *Успеваемость*, будет показывать ваши оценки по разным предметам и считать средний балл. Для этого отчёта вы сделаете регистр *Оценки*.

Второй отчёт, *Прошедшие Занятия*, будет подсчитывать количество занятий по разным предметам. Для этого отчёта вы создадите регистр с таким же именем — *Прошедшие Занятия*.

В процессе создания этих отчётов вам понадобится написать несколько строк на встроенным языке, чтобы заполнить регистры данными. Но я не буду подробно останавливаться на этих приёмах, потому что вашей основной целью будет знакомство с отчётами.

А после этого вы создадите ещё один регистр — *Домашние Задания*. И уже на его примере вы познакомитесь с тем, как работать с регистрами во встроенном языке. Потому что эти знания понадобятся вам в следующей главе.

5.3 Регистр сведений

Для создания первого отчёта, *Успеваемость*, вам понадобится *регистр сведений*. С регистром сведений проще всего знакомиться тогда, когда он есть у вас перед глазами,

вместе с данными, и вы можете его «потрогать». Поэтому сейчас вы поступите не совсем обычным образом.

Сначала, с самыми минимальными пояснениями, вы создадите регистр и заполните его данными. А уже после этого вы будете его изучать, знакомиться с его устройством и с его назначением. Поэтому сейчас не старайтесь понять незнакомые термины, которые вам встретятся. Все эти термины я объясню после того, как вы заполните регистр данными.

5.3.1 Регистр сведений Оценки

Перейдите в конфигуратор и в ветку *Регистры сведений* добавьте новый регистр. Назовите его *Оценки* (рисунок 5.7).

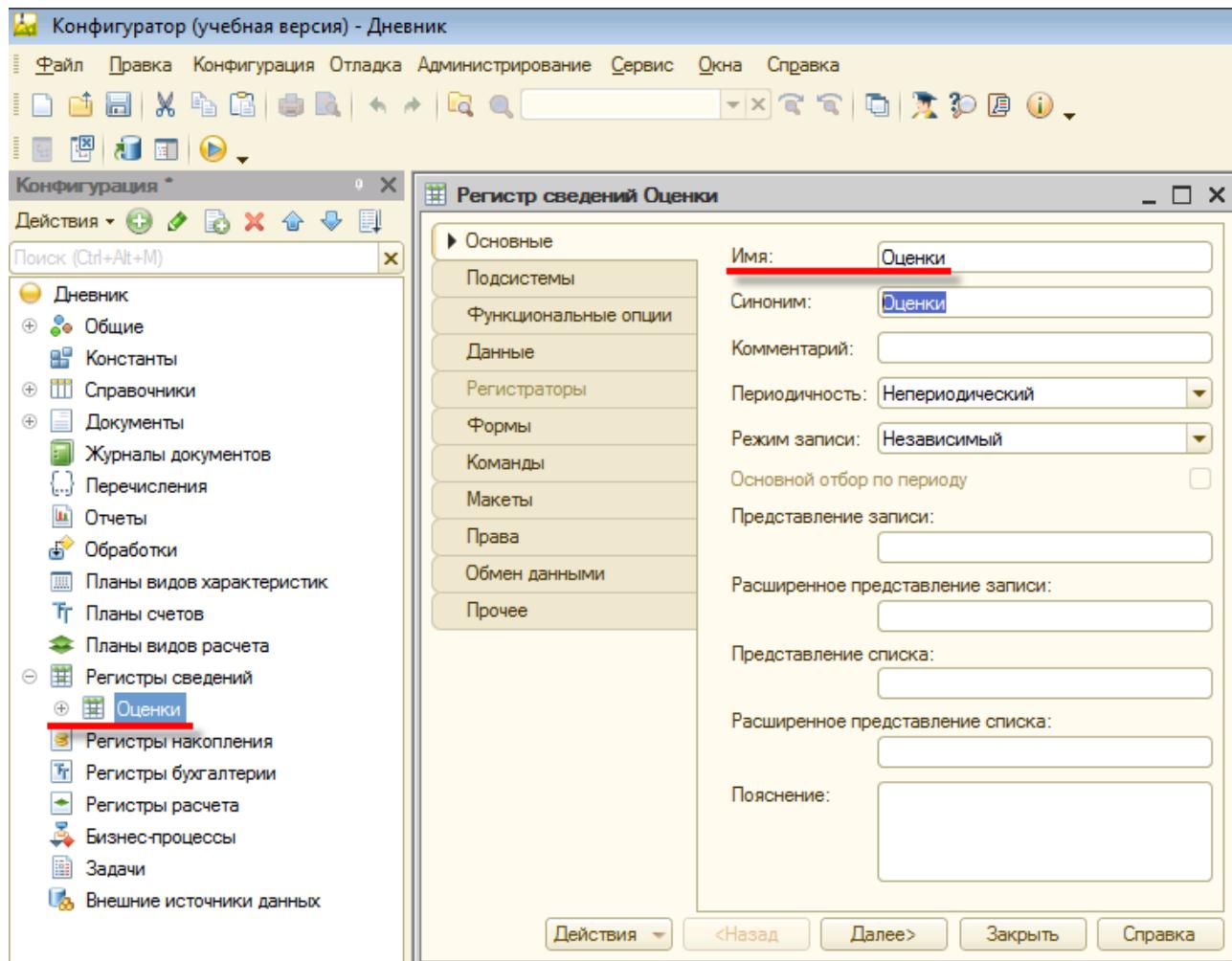


Рисунок 5.7. Регистр сведений «Оценки»

Этот регистр будет существовать не сам по себе, а будет подчиняться регистратору. Поэтому свойство *Режим записи* установите в значение *Подчинение регистратору*. А свойство *Периодичность* установите в значение *В пределах дня* (рисунок 5.8).

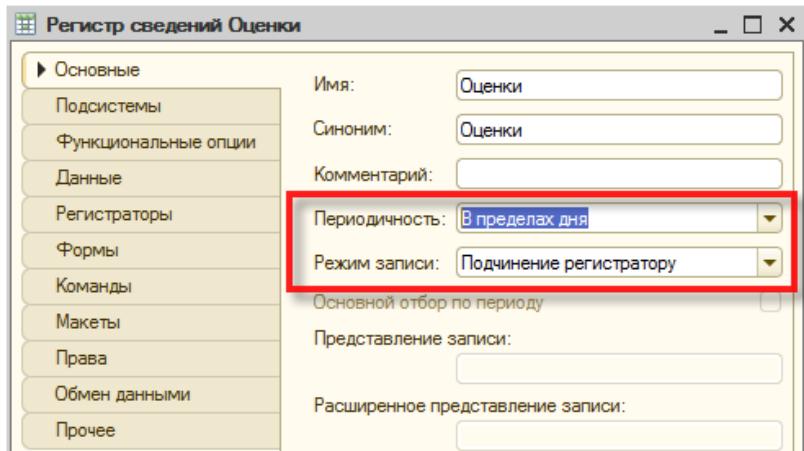


Рисунок 5.8. Режим записи и периодичность

В пределах дня нужно выбрать потому, что ваши оценки не могут изменяться чаще, чем раз в день.

Перейдите на закладку *Данные*. Здесь вы создадите собственно саму структуру регистра. Всё то же самое вы могли бы сделать и в дереве конфигурации, раскрыв ветку этого регистра. Но для разнообразия сделайте это в окне редактирования регистра.

Добавьте *измерение* регистра и назовите его *Предмет*. Тип у этого измерения, как вы, наверное, догадываетесь, нужно установить в значение *СправочникСсылка.Предметы* (рисунок 5.9).

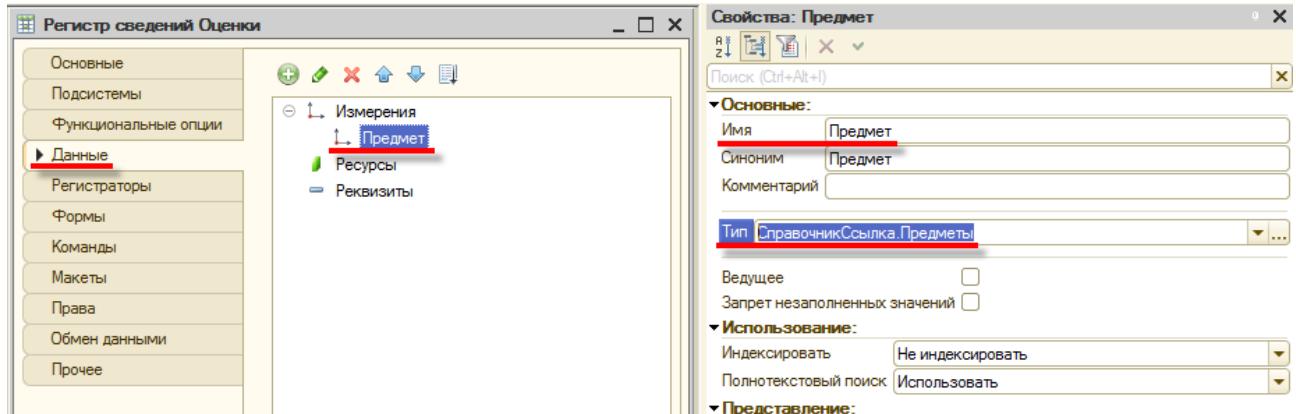


Рисунок 5.9. Измерение «Предмет»

В это поле вы будете записывать предмет, по которому получена оценка.

Добавьте ещё одно измерение и назовите его *НомерУрока*. Для этого измерения выберите:

- тип *Число*;
- *Длина 2*;
- *Неотрицательное* (рисунок 5.10).

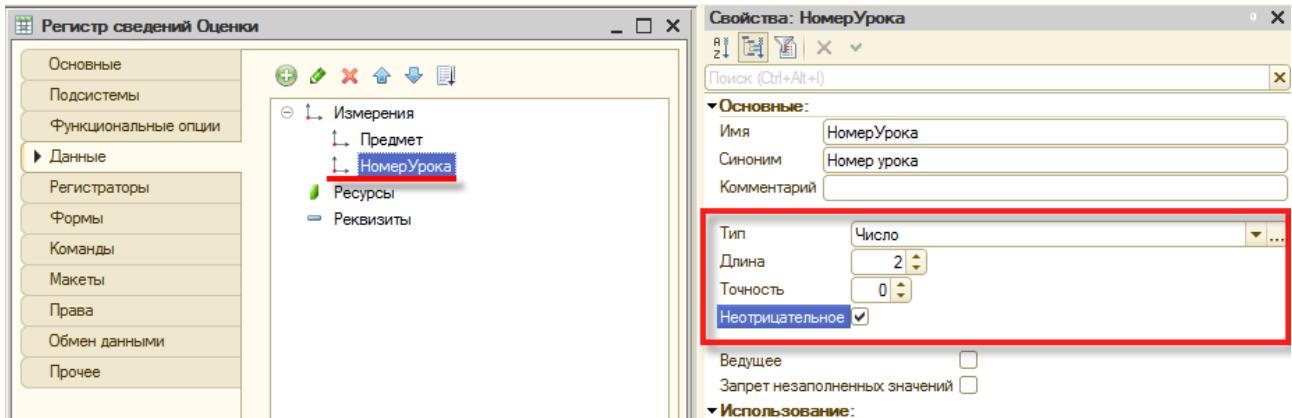


Рисунок 5.10. Измерение «НомерУрока»

В это поле вы будете записывать порядковый номер урока в течение дня.

Теперь добавьте *ресурс*. Назовите его *Оценка*. Для ресурса установите:

- тип Число;
- Длина 1;
- Неотрицательное (рисунок 5.11).

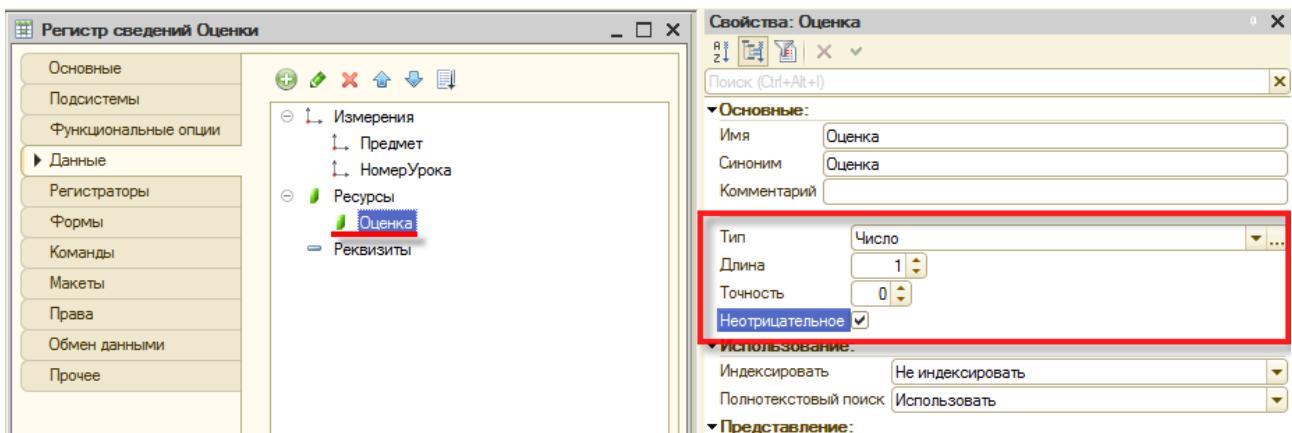


Рисунок 5.11. Ресурс «Оценка»

В это поле вы будете записывать оценку, которая вами получена.

Всё, структура регистра готова. Теперь нужно создать процедуру, которая будет помещать в этот регистр нужные вам данные.

5.3.2 Процедура проведения документов

Перейдите на закладку *Регистраторы*.

Как я говорил в начале, этот регистр будет существовать не сам по себе, а подчиняться регистратору. Регистратор — это документ. Платформа предлагает вам выбрать один из имеющихся документов.

Документ у вас единственный, именно он вам и нужен. Поэтому просто отметьте его (рисунок 5.12).

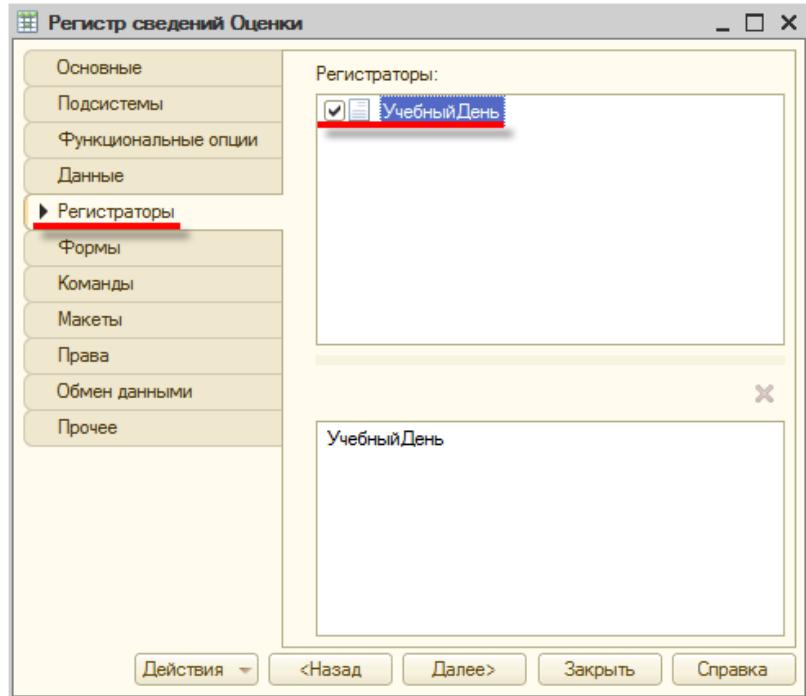


Рисунок 5.12. Регистратор

Всё. На этом все действия с регистром сведений закончены. Закройте его окно.

Данные в регистр сведений будет записывать его регистратор, документ УчебныйДень. Эти данные называются *движения*. Откройте окно редактирования документа УчебныйДень и перейдите на закладку *Движения* (рисунок 5.13).

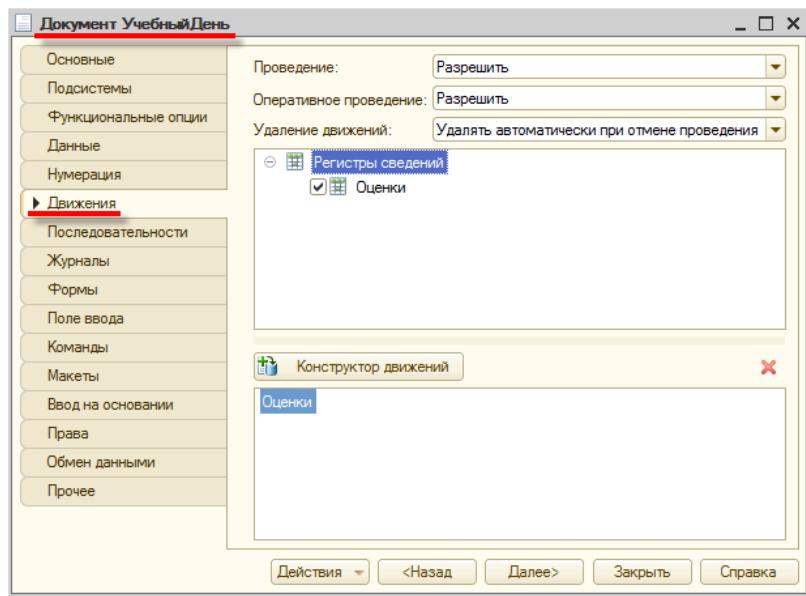


Рисунок 5.13. Закладка «Движения»

В верхней части окна находятся три свойства: *Проведение*, *Оперативное проведение* и *Удаление движений*. Их менять не нужно, вас полностью устраивают их стандартные значения.

Короткое пояснение просто для информации. В верхнем окне, если вы развернёте ветку, показаны все регистры, которые есть в вашей конфигурации. Флажком отмечены те регистры, для которых этот документ является регистратором. В нижнем окне показаны только те регистры, для которых этот документ является регистратором. Таким образом,

если вы захотите, чтобы этот документ стал регистратором для каких-то других регистров, это можно сделать прямо в этом окне. Не нужно для этого «ходить» в каждый регистр и указывать это там.

Чтобы создать процедуру, заполняющую регистр данными, нажмите кнопку Конструктор движений. Откроется конструктор (рисунок 5.14).

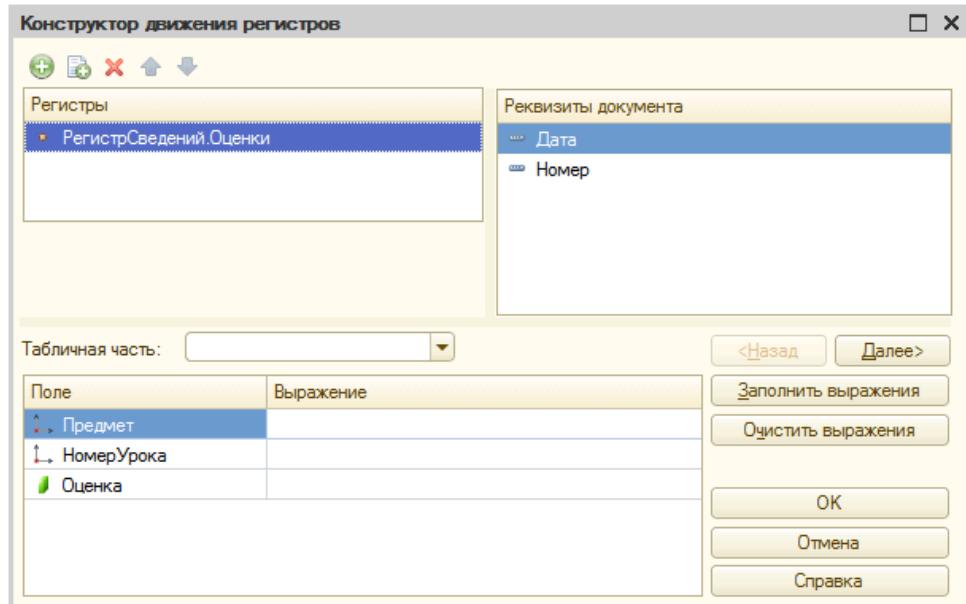


Рисунок 5.14. Конструктор движений

Этот конструктор сам напишет за вас процедуру на встроенным языке, которая будет добавлять данные в регистр. Нужно только правильно указать, что и куда вы хотите добавить.

Слева вверху указано, что сейчас вы будете создавать движения для регистра сведений Оценки. Документ, вообще говоря, может записывать свои данные в несколько разных регистров. Тогда все они будут перечислены в этом окне, и для каждого из них вы сможете указать собственный «алгоритм».

Справа вверху перечислены реквизиты, которые есть у вашего документа. Но только у самого документа. А ведь главная информация документа находится в его табличной части. Чтобы увидеть и реквизиты табличной части тоже, выберите её в поле *Табличная часть* (рисунок 5.15).

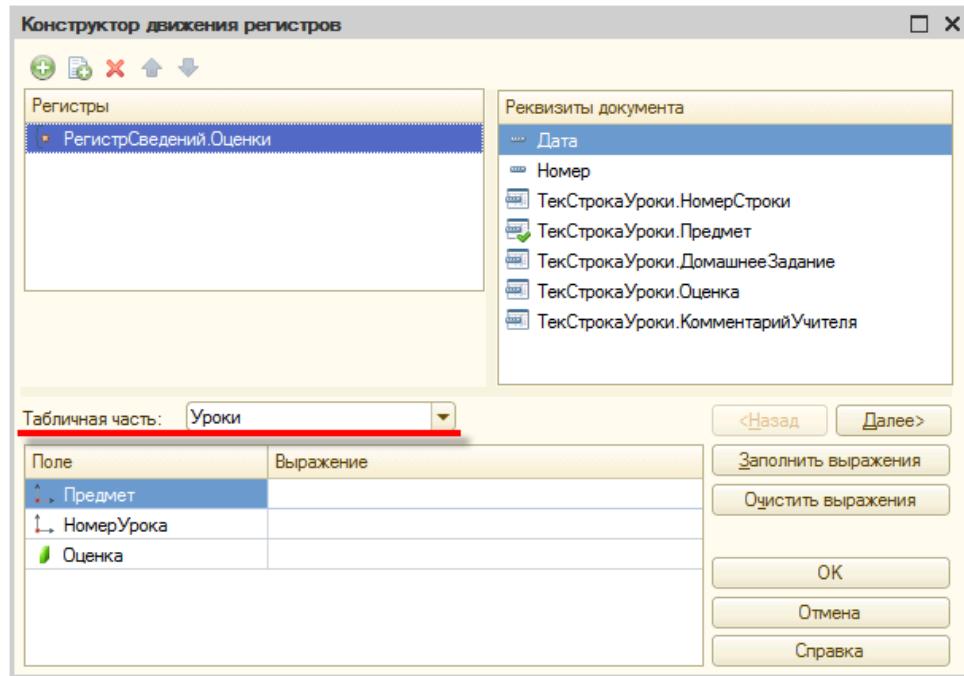


Рисунок 5.15. Выбор табличной части

Вы сразу увидите, что количество доступных вам реквизитов документа значительно увеличилось.

Слева внизу перечислены поля регистра, в которые вы будете что-то записывать. Как видите, тут есть оба ваших измерения и ресурс.

Этот конструктор достаточно умный, и он может сам сообразить, что в какое поле записывать. Попробуйте. Нажмите кнопку *Заполнить выражения* (рисунок 5.16).

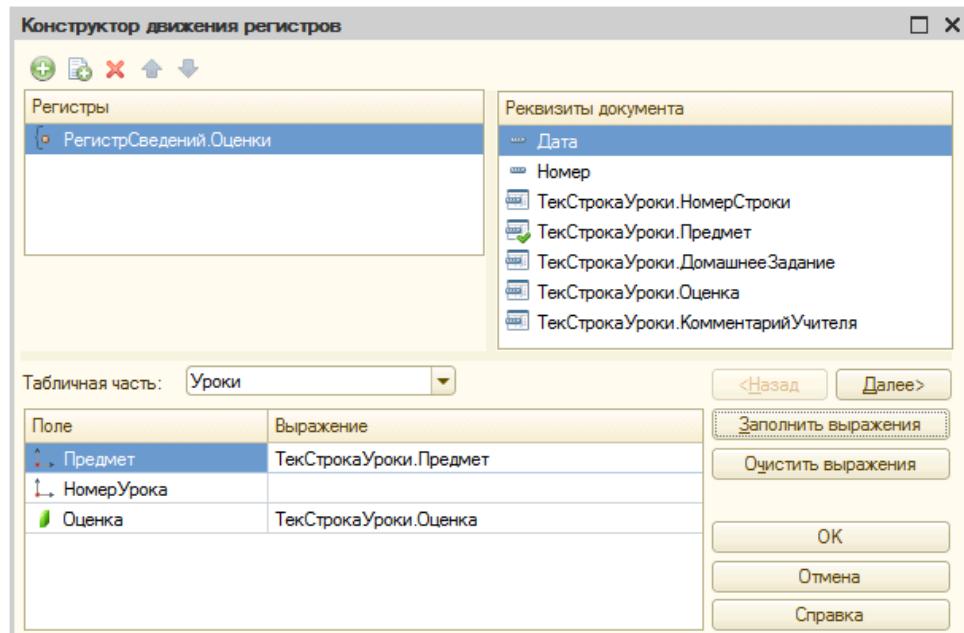


Рисунок 5.16. Конструктор заполнил выражения

Конструктор предложит вам в поле *Предмет* записывать предмет из табличной части документа, а в поле *Оценка* — оценку из табличной части документа. Это правильно, именно так вам и нужно.

Единственное, чего не смог сообразить конструктор, — что записывать в поле *НомерУрока*. Но это знаете вы. Туда нужно записать номер строки табличной части. Потому что он как раз и совпадает с порядковым номером урока в течение дня.

Установите курсор в пустую ячейку *Выражение* рядом с полем *НомерУрока*, а потом дважды щёлкните мышью на реквизите документа *ТекСтрокаУроки.НомерСтроки*. Это выражение появится в ячейке (рисунок 5.17).

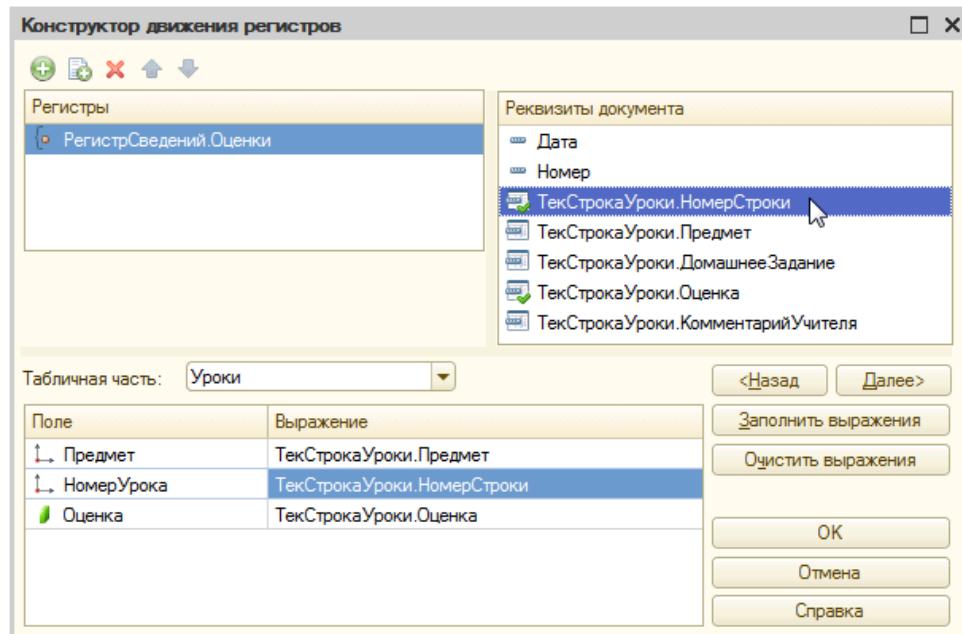


Рисунок 5.17. Выражение для номера урока

В конструкторе больше ничего делать не надо, нажмите *OK*. Платформа закроет конструктор, и вы увидите процедуру, которая записывает данные в регистр (рисунок 5.18).

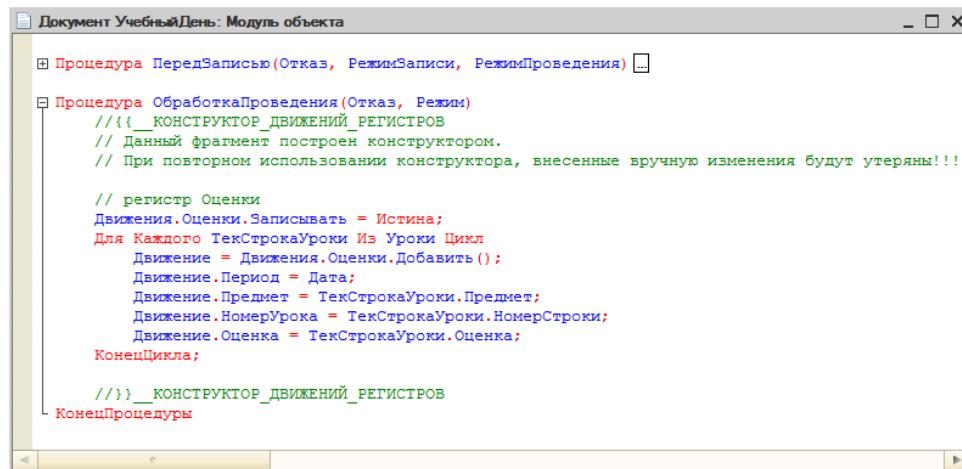


Рисунок 5.18. Процедура проведения документа

Сейчас не нужно вникать в её текст. Позже вы отдельно займётесь тем, как работать с регистром из встроенного языка. Но пока вы видите, что ничего сложного тут нет.

У объекта *ДокументОбъект.<Имя документа>* существует событие *ОбработкаПроведения*. В этом событии выполняются все действия, которые должны происходить при проведении документа. В вашем случае в цикле обходится табличная часть документа, *Уроки*. Для каждой её строки создаётся одна запись в регистре. Указывается обязательный реквизит записи *Период* — это дата вашего документа. А кроме этого записываются значения предмета, номера урока и оценка. Все они берутся из строки табличной части.

Теперь всё готово для того, чтобы заполнить регистр данными. Запустите 1С:Предприятие в режиме отладки.

5.3.3 Заполнение регистра данными

В вашей базе уже есть некоторое количество документов *Учебный День*. Когда вы их добавляли, вы нажимали кнопку *Записать*, а потом закрывали документ (рисунок 5.19).

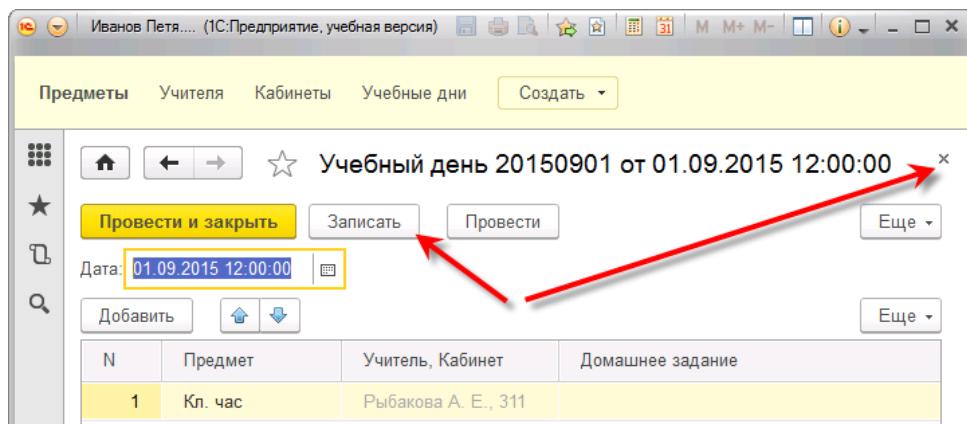


Рисунок 5.19. Записать и закрыть документ

Таким образом вы «подготавливали» данные документа. То есть они сохранялись в базе данных, но при этом не оказывали никакого влияния на состояние учёта. Теперь же вы в нескольких документах проставите оценки и *проводёте* эти документы. При этом выполнится обработка проведения, и данные документа попадут в регистр. То есть изменят состояние учёта.

Поскольку данные вам нужны только для примера, возьмите два дня: первое и второе сентября. Пусть первого сентября по английскому языку и математике будут пятёрки, а по музыке и русскому языку четвёрки (рисунок 5.20).

Учебный день 20150901 от 01.09.2015 12:00:00 *					
<input type="button" value="Провести и закрыть"/> <input type="button" value="Записать"/> <input type="button" value="Провести"/>					
<input type="text" value="01.09.2015 12:00:00"/> <input type="button" value="..."/>					
<input type="button" value="Добавить"/> <input type="button" value="↑"/> <input type="button" value="↓"/>					
N	Предмет	Учитель, Кабинет	Домашнее задание	Оценка	Комментарий учителя
1	Кл. час	Рыбакова А. Е., 311			
2	Кл. час	Рыбакова А. Е., 311			
3	Английский язык	Николаева Т. Я., 401		5	
4	Музыка	Евсеева О. Ю., 131		4	
5	Математика	Рыбакова А. Е., 311		5	
6	Русский язык	Авдеева В. М., 409		4	

Рисунок 5.20. Оценки 1 сентября

Теперь нажмите кнопку *Провести и закрыть*. Документ будет проведён, и его данные изменят состояние учёта в вашей системе. О том, что документ проведён, будет говорить иконка с зелёным флагом рядом с документом (рисунок 5.21).

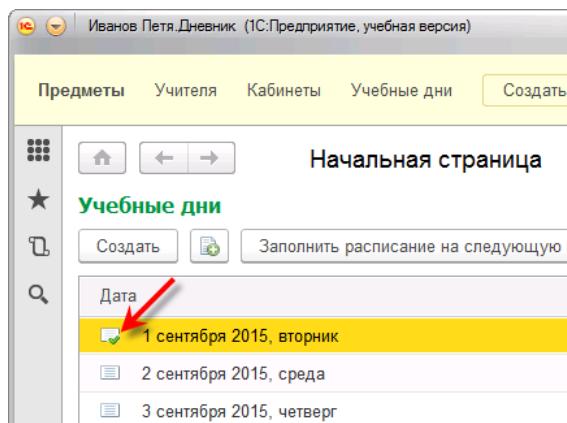


Рисунок 5.21. Документ проведен

Теперь заполните оценки за второе сентября. Пусть этот день будет не таким удачным, как первый. По некоторым предметам будут тройки (рисунок 5.22).

N	Предмет	Учитель, Кабинет	Домашнее задание	Оценка	Комментарий учителя
1	Математика	Рыбакова А. Е., 311		4	
2	Английский язык	Николаева Т. Я., 401		3	
3	Природоведение	Филиппова Л. В., 220		5	
4	Природоведение	Филиппова Л. В., 220		4	
5	Русский язык	Авдеева В. М., 409		3	
6	Литература	Авдеева В. М., 409		5	

Рисунок 5.22. Оценки 2 сентября

Тоже проведите и закройте этот документ.

Теперь вы будете смотреть на то, что записано в регистр, и знакомиться с его устройством.

5.3.4 Хранение данных в таблицах

Поскольку регистр, с точки зрения пользователя, содержит «вспомогательные» данные, платформа не включает его автоматически в интерфейс прикладного решения. Но вы не просто пользователь, но ещё и разработчик прикладного решения. Поэтому сейчас я расскажу вам о способе, которым вы сможете «добраться» до любых данных вашего приложения. Даже если команды перехода к ним нет в прикладном интерфейсе.

Для этого нужно открыть главное меню и выполнить команду *Все функции...* (рисунок 5.23).

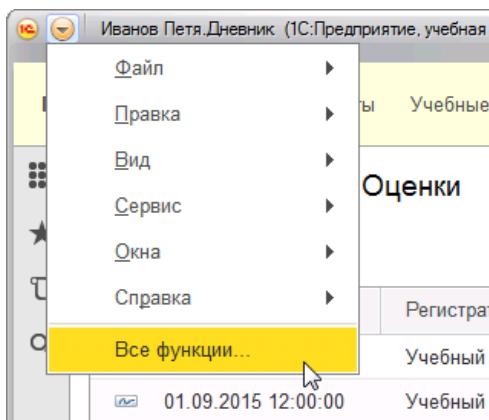


Рисунок 5.23. Команда «Все функции...»

Совет

Может так случиться, что этой команды нет в главном меню. Значит, она отключена. Чтобы включить её, через это же меню откройте параметры конфигуратора — *Сервис — Параметры* и включите флажок *Отображать команду «Все функции»*.

В окне *Все функции* дважды щёлкните мышью на регистре сведений *Оценки* (рисунок 5.24).

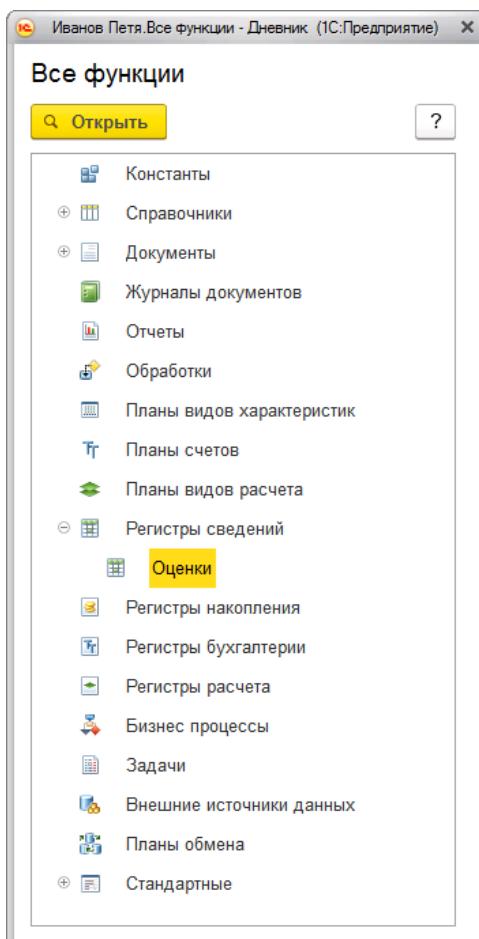


Рисунок 5.24. Регистр сведений «Оценки»

На экране вы увидите все данные, которые хранятся в этом регистре (рисунок 5.25).

Период ↓	Регистратор	Номер строки	Предмет	Номер урока	Оценка
01.09.2015	Учебный день 20150901 от 01.09.2015 12:00:00	1	Кл. час	1	
01.09.2015	Учебный день 20150901 от 01.09.2015 12:00:00	2	Кл. час	2	
01.09.2015	Учебный день 20150901 от 01.09.2015 12:00:00	3	Английский язык	3	5
01.09.2015	Учебный день 20150901 от 01.09.2015 12:00:00	4	Музыка	4	4
01.09.2015	Учебный день 20150901 от 01.09.2015 12:00:00	5	Математика	5	5
01.09.2015	Учебный день 20150901 от 01.09.2015 12:00:00	6	Русский язык	6	4
02.09.2015	Учебный день 20150902 от 02.09.2015 12:00:00	1	Математика	1	4
02.09.2015	Учебный день 20150902 от 02.09.2015 12:00:00	2	Английский язык	2	3
02.09.2015	Учебный день 20150902 от 02.09.2015 12:00:00	3	Природоведение	3	5
02.09.2015	Учебный день 20150902 от 02.09.2015 12:00:00	4	Природоведение	4	4
02.09.2015	Учебный день 20150902 от 02.09.2015 12:00:00	5	Русский язык	5	3
02.09.2015	Учебный день 20150902 от 02.09.2015 12:00:00	6	Литература	6	5

Рисунок 5.25. Данные регистра «Оценки»

То, что вы видите сейчас на экране, очень похоже на то, как хранятся данные в базе данных. Правда, в базе данных колонки имеют другие имена, есть некоторые служебные колонки, которых здесь не видно. Но в целом именно так данные и хранятся. Любые данные 1С:Предприятия.

Такой способ хранения данных называется *табличным*. Вообще данные вашего прикладного решения хранятся в нескольких разных *таблицах*. А то, что вы видите сейчас на экране, это одна таблица.

У таблицы есть колонки и есть строки. Колонки таблицы называются *поля*. Строки таблицы называются *записи*. То есть структура таблицы определяется её полями, а содержимое таблицы — это её записи (рисунок 5.26).

Период ↓	Регистратор
Запись →	01.09.2015 Учебный день 20150901 от 01.09.2015 12:00:00
Запись →	01.09.2015 Учебный день 20150901 от 01.09.2015 12:00:00
Запись →	01.09.2015 Учебный день 20150901 от 01.09.2015 12:00:00
Запись →	01.09.2015 Учебный день 20150901 от 01.09.2015 12:00:00

Рисунок 5.26. Поля и записи

Для каждого объекта конфигурации в базе данных обычно создаётся одна таблица. Если объект конфигурации имеет сложную структуру, то может быть и не одна таблица. Но это сейчас не очень важно. А важно то, что в одной таблице находятся данные одного объекта конфигурации.

Структура этой таблицы, её поля, во-первых, определяются тем, что это за объект конфигурации. А во-вторых, тем, какие реквизиты вы добавляли этому объекту в дереве конфигурации. Например, в той таблице, которая у вас на экране, поля *Период*, *Регистратор* и *НомерСтроки* появились потому, что это регистр сведений, который подчинён регистратору. А поля *Предмет*, *НомерУрока* и *Оценка* — это измерения и ресурсы, которые этому регистру добавили вы (рисунок 5.27).

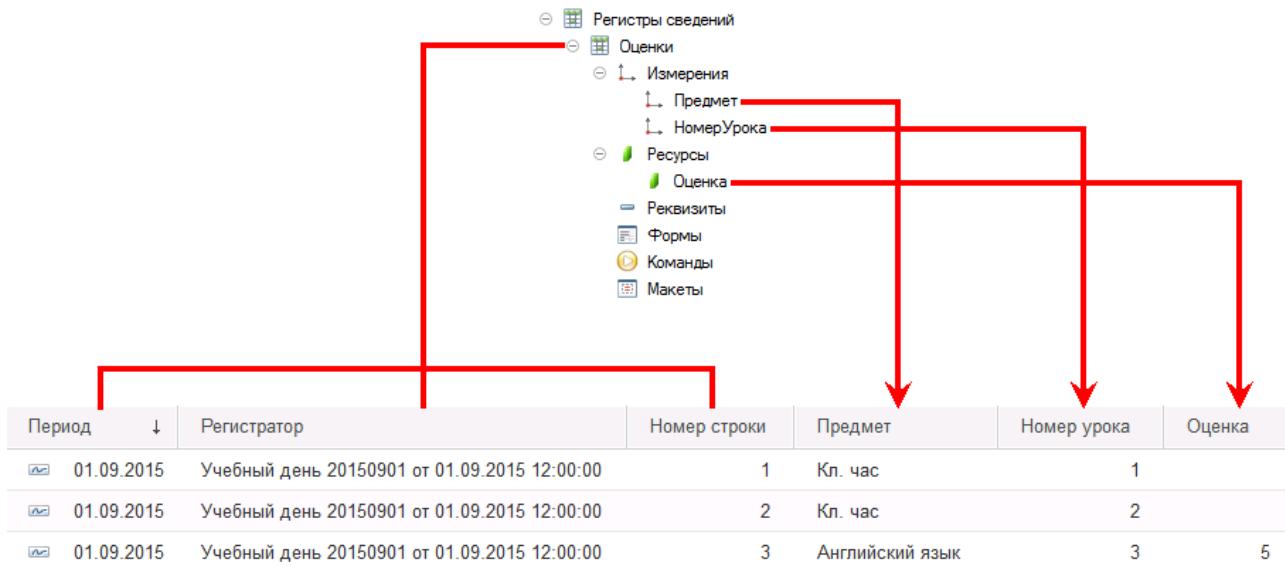


Рисунок 5.27. Состав полей определяется объектом конфигурации

Теперь можно разобраться с тем, почему у регистра именно такой набор полей и зачем они нужны.

5.3.5 Устройство регистра сведений

Регистр сведений может быть *независимым*, а может быть *подчинён регистратору*. Разница между такими регистрами только в полях *Регистратор* и *НомерСтроки*. Сейчас ваш регистр подчинён регистратору. Но если из него убрать эти поля, то он станет независимым (рисунок 5.28).

Период	Регистратор	Номер строки	Предмет	Номер урока	Оценка
01.09.2015	Учебный день 20150901 от 01.09.2015 12:00:00	1	Кл. час	1	
01.09.2015	Учебный день 20150901 от 01.09.2015 12:00:00	2	Кл. час	2	
01.09.2015	Учебный день 20150901 от 01.09.2015 12:00:00	3	Английский язык	3	5
01.09.2015	Учебный день 20150901 от 01.09.2015 12:00:00	4	Музыка	4	4
01.09.2015	Учебный день 20150901 от 01.09.2015 12:00:00	5	Математика	5	5
01.09.2015	Учебный день 20150901 от 01.09.2015 12:00:00	6	Русский язык	6	4
02.09.2015	Учебный день 20150902 от 02.09.2015 12:00:00	1	Математика	1	4
02.09.2015	Учебный день 20150902 от 02.09.2015 12:00:00	2	Английский язык	2	3
02.09.2015	Учебный день 20150902 от 02.09.2015 12:00:00	3	Природоведение	3	5
02.09.2015	Учебный день 20150902 от 02.09.2015 12:00:00	4	Природоведение	4	4
02.09.2015	Учебный день 20150902 от 02.09.2015 12:00:00	5	Русский язык	5	3
02.09.2015	Учебный день 20150902 от 02.09.2015 12:00:00	6	Литература	6	5

Рисунок 5.28. В независимом регистре нет поля «Регистратор»

Если регистр сведений независимый, то записи в него вы добавляете (или удаляете) по своему желанию. Когда хотите и как хотите. С помощью встроенного языка или «руками» (интерактивно) в режиме 1С:Предприятие.

Если регистр сведений подчинён регистратору, всё то же самое. Но в дополнение к этому ещё и платформа берёт на себя некоторые сервисные функции, чтобы данные, находящиеся в регистре, соответствовали тому, что содержится в регистраторе.

Регистратор — это документ. Документ, некоторые данные которого вы хотите поместить в регистр. Какие сервисные функции в этом случае предоставляет вам платформа?

Во-первых, это механизм *проведения* документа. Пока документ просто записан, его данных нет в регистре. Как только вы провели документ, его данные появились в регистре. Если вы отмените проведение документа, его данные снова исчезнут из регистра, но останутся в базе данных. Если вы совсем удалите проведённый документ, то его данные исчезнут и из регистра, и из базы данных. Всё это платформа делает автоматически.

Стандартно, когда вы добавляете в конфигурацию новый документ, он использует механизм проведения. Это указано в свойстве *Проведение*. Но, вообще говоря, вы можете и отказаться от использования этого механизма, если так нужно для вашей задачи (рисунок 5.29).

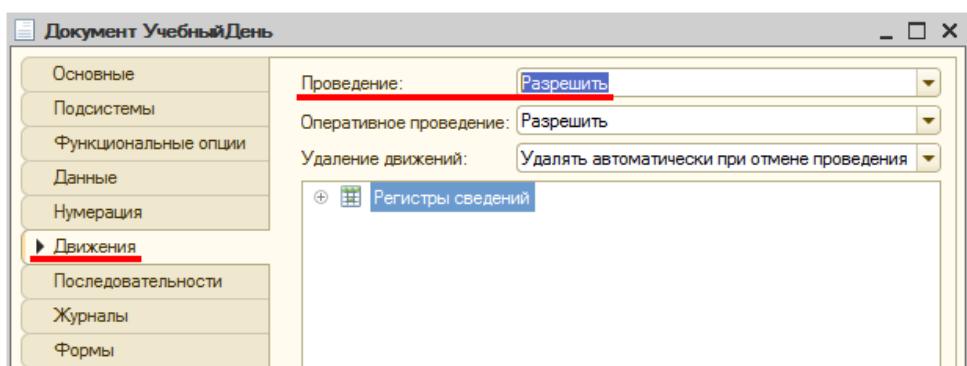


Рисунок 5.29. Проведение документа разрешено

Во-вторых, платформа предоставляет вам события *ОбработкаПроведения* и *ОбработкаУдаленияПроведения*, которые есть у типа *ДокументОбъект.<Имя документа>*. Обрабатывая эти события, вы можете написать собственный алгоритм, который будет добавлять данные этого документа в регистр или удалять их. Вы уже воспользовались событием *ОбработкаПроведения*.

Чтобы платформа могла предоставлять вам такой «сервис», в таблице регистра и существует поле *Регистратор*. Именно по этому полю платформа может найти записи, относящиеся к документу, который вы изменяете или удаляете. Записи регистра, принадлежащие одному документу, называются его *движениями* (рисунок 5.30).

Регистратор	Номер строки	Предмет	Номер урока	Оценка
Учебный день 20150901 от 01.09.2015 12:00:00	1	Кл. час	1	
Учебный день 20150901 от 01.09.2015 12:00:00	2	Кл. час	2	
Учебный день 20150901 от 01.09.2015 12:00:00	3	Английский язык	3	5
Учебный день 20150901 от 01.09.2015 12:00:00	4	Музыка	4	4
Учебный день 20150901 от 01.09.2015 12:00:00	5	Математика	5	5
Учебный день 20150901 от 01.09.2015 12:00:00	6	Русский язык	6	4
Учебный день 20150902 от 02.09.2015 12:00:00	1	Математика	1	4
Учебный день 20150902 от 02.09.2015 12:00:00	2	Английский язык	2	3
Учебный день 20150902 от 02.09.2015 12:00:00	3	Природоведение	3	5
Учебный день 20150902 от 02.09.2015 12:00:00	4	Природоведение	4	4
Учебный день 20150902 от 02.09.2015 12:00:00	5	Русский язык	5	3
Учебный день 20150902 от 02.09.2015 12:00:00	6	Литература	6	5

Рисунок 5.30. Движения документа

Когда вы проводите документ, он добавляет свои движения в регистр. Когда вы отменяете проведение документа или вообще удаляете его, платформа удаляет движения этого документа из регистра.

Стандартно для новых документов указывается, что движения должны удаляться в тот момент, когда вы отменяете проведение документа. Это указывается в свойстве *Удаление Движений*. Но вы можете выбрать и другой алгоритм, если так нужно для вашей задачи (рисунок 5.31).

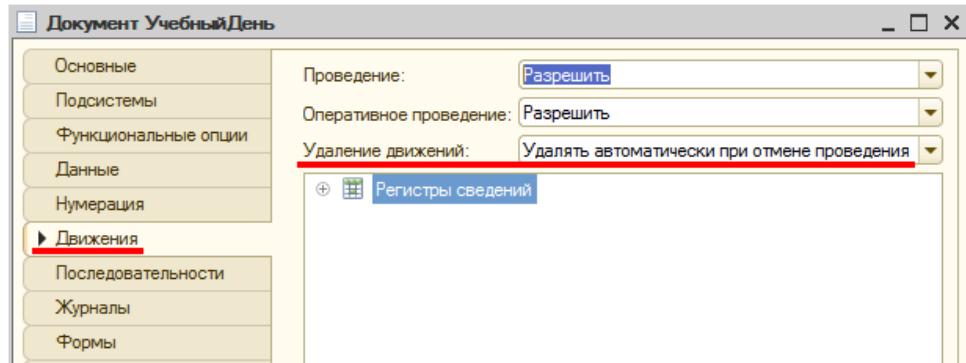


Рисунок 5.31. Движения удаляются при отмене проведения

Вот так устроена в платформе связка документа и регистра.

Теперь нужно разобраться, почему вы выбрали именно регистр сведений, а не какой-то другой. И почему одни реквизиты у него называются измерениями, а другие ресурсами. Хотя и то и другое в таблице — это просто колонки, которые по внешним признакам ничем не отличаются друг от друга (рисунок 5.32).

Предмет	Номер урока	Оценка
Кл. час	1	
Кл. час	2	
Английский язык	3	5
Музыка	4	4
Математика	5	5
Русский язык	6	4
Математика	1	4
Английский язык	2	3
Природоведение	3	5
Природоведение	4	4
Русский язык	5	3
Литература	6	5

Рисунок 5.32. Измерения и ресурс

Попробуйте сформулировать: зачем вы создавали регистр *Оценки*? Чтобы, например, узнать оценки по математике. Теперь слово «оценки» замените на «ресурс», а слово «математика» замените на «измерение». И вам всё сразу станет понятно (рисунок 5.33).

**Рисунок 5.33.** Ресурсы и измерения

Измерения — это уточняющая информация, которую вы передадите регистру, когда будете его о чём-то спрашивать. А *ресурсы* — это то, что регистр вам ответит. То, что вы хотите узнать в результате своего вопроса.

Вы хотите знать оценки, поэтому создали ресурс *Оценка*. Если бы кроме этого вы хотели знать продолжительность ответов на вопрос учителя, вы бы создали ещё один реквизит — *ПродолжительностьОтвета*. И тогда ваш вопрос звучал бы так: «Какие оценки и сколько времени я потратил на ответы по математике?».

Вы хотите знать оценки по каждому предмету. Поэтому вы создали измерение *Предмет*. Если бы вы хотели кроме этого знать, за устный ответ получена оценка или за письменную работу, вы бы создали одно измерение — *ВидОтвета*. И тогда ваш вопрос звучал бы так: «Какие оценки у меня за устные ответы по математике?».

Примечание

Состав измерений и ресурсов регистра определяется только тем, какую информацию вы хотите получать из регистра и в каких «разрезах». Разрезы — это и есть измерения.

«Всё это хорошо, — скажете вы, — но откуда взялось измерение *НомерУрока*? Ведь мне совсем неинтересно знать, первым уроком была математика или третьим. Зачем хранить эти данные?»

А тут дело в другом. Это измерение вы добавили не потому, что оно нужно вам для вашей прикладной задачи, а потому, что оно нужно платформе.

Регистр сведений не может хранить в себе «одинаковые» записи. Все записи с его точки зрения должны быть «разными». Как регистр определяет, какие записи одинаковые, а какие разные?

Регистр сведений, подчинённый регистратору, использует для этого поле *Регистратор* и все измерения. То есть в вашем регистре не может быть двух записей, у которых совпадают все три поля: *Регистратор*, *Предмет* и *НомерУрока* (рисунок 5.34).

Период ↓	Регистратор	Номер строки	Предмет	Номер урока	Оценка
01.09.2015	Учебный день 20150901 от 01.09.2015 12:00:00	1	Кл. час	1	
01.09.2015	Учебный день 20150901 от 01.09.2015 12:00:00	2	Кл. час	2	
01.09.2015	Учебный день 20150901 от 01.09.2015 12:00:00	3	Английский язык	3	5
01.09.2015	Учебный день 20150901 от 01.09.2015 12:00:00	4	Музыка	4	4
01.09.2015	Учебный день 20150901 от 01.09.2015 12:00:00	5	Математика	5	5
01.09.2015	Учебный день 20150901 от 01.09.2015 12:00:00	6	Русский язык	6	4
02.09.2015	Учебный день 20150902 от 02.09.2015 12:00:00	1	Математика	1	4
02.09.2015	Учебный день 20150902 от 02.09.2015 12:00:00	2	Английский язык	2	3
02.09.2015	Учебный день 20150902 от 02.09.2015 12:00:00	3	Природоведение	3	5
02.09.2015	Учебный день 20150902 от 02.09.2015 12:00:00	4	Природоведение	4	4
02.09.2015	Учебный день 20150902 от 02.09.2015 12:00:00	5	Русский язык	5	3
02.09.2015	Учебный день 20150902 от 02.09.2015 12:00:00	6	Литература	6	5

Рисунок 5.34. Записи регистра уникальны по регистратору и измерениям

Если бы у вас не было измерения *НомерУрока*, то 2 сентября у вас возникла бы проблема. В этот день было два урока природоведения. И на обоих уроках вы получили оценку. Но записать её вы бы не смогли, так как значения полей *Регистратор* и *Предмет* у таких записей совпадают (рисунок 5.35).

Период ↓	Регистратор	Номер строки	Предмет	Номер урока	Оценка
01.09.2015	Учебный день 20150901 от 01.09.2015 12:00:00	1	Кл. час	1	
01.09.2015	Учебный день 20150901 от 01.09.2015 12:00:00	2	Кл. час	2	
01.09.2015	Учебный день 20150901 от 01.09.2015 12:00:00	3	Английский язык	3	5
01.09.2015	Учебный день 20150901 от 01.09.2015 12:00:00	4	Музыка	4	4
01.09.2015	Учебный день 20150901 от 01.09.2015 12:00:00	5	Математика	5	5
01.09.2015	Учебный день 20150901 от 01.09.2015 12:00:00	6	Русский язык	6	4
02.09.2015	Учебный день 20150902 от 02.09.2015 12:00:00	1	Математика	1	4
02.09.2015	Учебный день 20150902 от 02.09.2015 12:00:00	2	Английский язык	2	3
02.09.2015	Учебный день 20150902 от 02.09.2015 12:00:00	3	Природоведение	3	5
02.09.2015	Учебный день 20150902 от 02.09.2015 12:00:00	4	Природоведение	4	4
02.09.2015	Учебный день 20150902 от 02.09.2015 12:00:00	5	Русский язык	5	3
02.09.2015	Учебный день 20150902 от 02.09.2015 12:00:00	6	Литература	6	5

Рисунок 5.35. Без измерения «НомерУрока» нельзя записать две оценки по одному предмету в один день

Платформа сказала бы вам, что запись с такими значениями полей в регистре уже есть, и отказалась бы её добавлять.

Поэтому вы создали измерение *НомерУрока*, благодаря которому такие записи для регистра становятся «разными» и он замечательно их записывает (рисунок 5.36).

Период ↓	Регистратор	Номер строки	Предмет	Номер урока	Оценка
01.09.2015	Учебный день 20150901 от 01.09.2015 12:00:00	1	Кл. час	1	
01.09.2015	Учебный день 20150901 от 01.09.2015 12:00:00	2	Кл. час	2	
01.09.2015	Учебный день 20150901 от 01.09.2015 12:00:00	3	Английский язык	3	5
01.09.2015	Учебный день 20150901 от 01.09.2015 12:00:00	4	Музыка	4	4
01.09.2015	Учебный день 20150901 от 01.09.2015 12:00:00	5	Математика	5	5
01.09.2015	Учебный день 20150901 от 01.09.2015 12:00:00	6	Русский язык	6	4
02.09.2015	Учебный день 20150902 от 02.09.2015 12:00:00	1	Математика	1	4
02.09.2015	Учебный день 20150902 от 02.09.2015 12:00:00	2	Английский язык	2	3
02.09.2015	Учебный день 20150902 от 02.09.2015 12:00:00	3	Природоведение	3	5
02.09.2015	Учебный день 20150902 от 02.09.2015 12:00:00	4	Природоведение	4	4
02.09.2015	Учебный день 20150902 от 02.09.2015 12:00:00	5	Русский язык	5	3
02.09.2015	Учебный день 20150902 от 02.09.2015 12:00:00	6	Литература	6	5

Рисунок 5.36. Измерение «НомерУрока» позволяет записывать несколько оценок по предмету в день

Наконец вы подошли к самому интересному вопросу. Почему вы создали регистр сведений, а не какой-то другой регистр? Ведь есть регистры накопления, регистры бухгалтерии и регистры расчёта?

Регистры бухгалтерии и регистры расчёта — это сложные объекты, которые предназначены для решения совершенно определённых прикладных задач. Регистр бухгалтерии используется тогда, когда в программе нужно вести бухгалтерский учёт. Регистр расчёта используется во всех задачах, связанных с начислением заработной платы. Эти регистры я не буду рассматривать в этой книге.

Остаётся регистр накопления. Вы рассмотрите его совсем скоро. В чём же отличие регистра сведений от регистра накопления?

Регистр сведений просто хранит данные. Когда вы спрашиваете его: «Какие у меня есть оценки по математике?» — он просто перечисляет вам эти оценки.

Регистр накопления не просто хранит данные, а накапливает значения ресурсов. Он их суммирует. Для того примера, который вы делаете сейчас, нет никакого смысла знать сумму всех оценок по математике. Но вот для следующей задачи, когда вы захотите знать количество занятий по разным предметам, вам понадобится регистр накопления. Потому что там действительно нужно будет по каждому предмету прибавлять и прибавлять новые занятия каждый день.

Подробнее

Подробнее вы можете прочитать про регистр сведений в документации «[Руководство разработчика 8.3. Раздел 5.14.2. Регистры сведений](#)».

5.3.6 Оперативное проведение

Небольшое отступление нужно сделать по поводу проведения документов. Может быть, вы уже с этим столкнулись, а может быть, столкнётесь в будущем. Но при проведении документов УчебныйДень вы можете получить такое сообщение от платформы (рисунок 5.37).

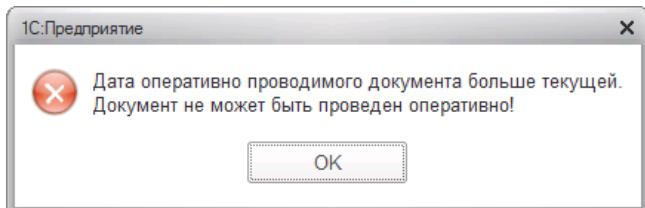


Рисунок 5.37. Оперативное проведение будущей датой невозможно

Что это значит и что с этим делать?

Стандартно при добавлении нового документа его свойство *Оперативное Проведение* устанавливается в значение *Разрешить* (рисунок 5.38).

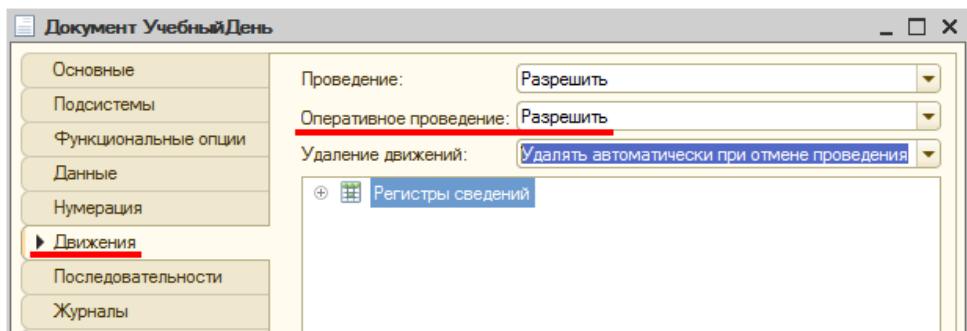


Рисунок 5.38. Оперативное проведение разрешено

Это значит, что механизм проведения документов настроен на работу в режиме «реального времени». То есть вы можете проводить документы, дата которых совпадает с текущей датой. Или меньше её.

Провести документ, дата которого больше чем текущая дата, платформа вам не даст. И вы получите то сообщение, которое показано на рисунке 5.37.

Такой подход полностью соответствует вашим задачам.

Сначала вы создаёте документы УчебныйДень. Как правило, вы создаёте их будущими датами, на следующую неделю. Это предварительные данные, которые вы просто сохраняете в базе данных. И они не влияют и не могут повлиять на состояние учёта. Ведь это занятия, которые ещё не произошли, а только планируются. Поэтому и проводить эти документы не нужно.

Но когда вы заносите в существующий документ полученные оценки, вы делаете это в тот же день или после того, как событие произошло. И проводите этот документ, то есть изменяете состояние учёта.

Поэтому нет ничего страшного в том, что вы вдруг получили то сообщение, которое показано на рисунке 5.37. Это платформа страхует вас от того, чтобы вы случайно не зарегистрировали в учётной системе то событие, которое ещё не произошло в действительности. В этом случае нужно просто сохранить и закрыть документ. Не проводя его. А провести его вы сможете тогда, когда наступит дата, указанная в документе.

Подробнее

Подробнее вы можете прочитать про проведение документов в документации «Руководство разработчика 8.3. Раздел 5.9.3. Механизм проведения документов».

5.3.7 Отчёт Успеваемость

Наконец вы подошли к тому, ради чего создавался регистр *Оценки*. Сейчас на основе данных этого регистра вы сделаете отчёт, который будет показывать вашу успеваемость.

Процесс создания и выполнения отчёта в 1С:Предприятии очень похож на процесс приготовления еды, который вам прекрасно знаком. Чтобы приготовить вкусное блюдо, нужно знать рецепт. В рецепте написано, какие понадобятся продукты и как их приготовить (рисунок 5.39).



Рисунок 5.39. Рецепт приготовления блюда

Хозяйка берёт рецепт и идёт с ним в магазин. Чтобы купить те продукты, которые потребуются (рисунок 5.40).



Рисунок 5.40. Хозяйка покупает продукты по рецепту

После этого она, по этому же рецепту, готовит их (рисунок 5.41).



Рисунок 5.41. Хозяйка готовит по рецепту

В результате получается готовое блюдо, например пицца (рисунок 5.42).



Рисунок 5.42. Готовое блюдо

В 1С:Предприятии в качестве хозяйки выступает специальный механизм, который называется *система компоновки данных*. Этот механизм умеет выбирать из базы данных информацию, нужным образом отбирать, группировать, сортировать её и представлять в виде таблиц или диаграмм.

Чтобы система компоновки данных знала, какие данные взять и как их «приготовить», у неё существует свой «рецепт», который называется *схема компоновки данных*.

Тут не запутайтесь, потому что на первый взгляд названия похожие. Итак, система — это «хозяйка», а схема — это «рецепт».

Чтобы пользователь мог запустить и увидеть какой-нибудь отчёт, в дереве объектов конфигурации существует ветка, которая так и называется — *Отчёты*. Вместе с каждым отчётом хранится и его «рецепт», то есть схема компоновки данных. И отчёт, и его схему компоновки данных создаёте вы в процессе разработки прикладного решения.

Дальше процесс выполнения отчёта выглядит следующим образом (рисунок 5.43).

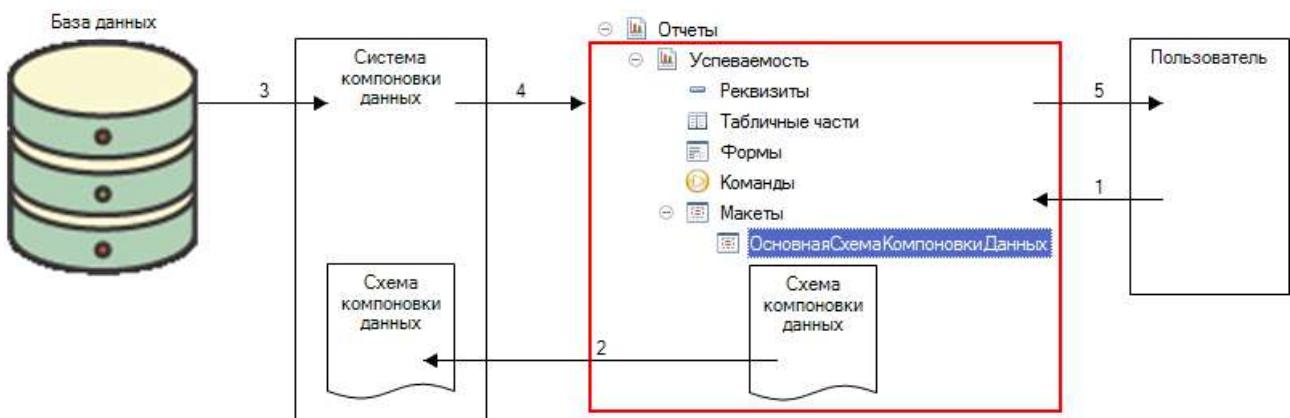


Рисунок 5.43. Процесс выполнения отчёта

1. Пользователь вызывает ваш отчёт и нажимает в нём кнопку *Сформировать*.
2. Система компоновки данных берёт схему компоновки данных, которая содержится в отчёте.
3. В соответствии с тем, что написано в этой схеме, система компоновки данных выбирает данные из базы данных, а затем «готовит» их.
4. После того как все данные «приготовлены», она передаёт их отчёту.
5. Отчёт показывает готовые данные пользователю.

Таким образом, отчёт здесь выступает в роли своеобразного «официанта», который сам ничего не готовит, но умеет красиво преподнести готовое блюдо посетителю (рисунок 5.44).



Рисунок 5.44. Отчёт — «официант»

Как видите, ничего сложного тут нет, поэтому можно сразу приступить.

Добавьте новый отчёт и назовите его *Успеваемость*. Больше ничего задавать не нужно, поэтому сразу приступайте к созданию «рецепта», то есть схемы компоновки данных. Для этого нажмите кнопку *открытия* у поля *Основная схема компоновки данных* (рисунок 5.45).

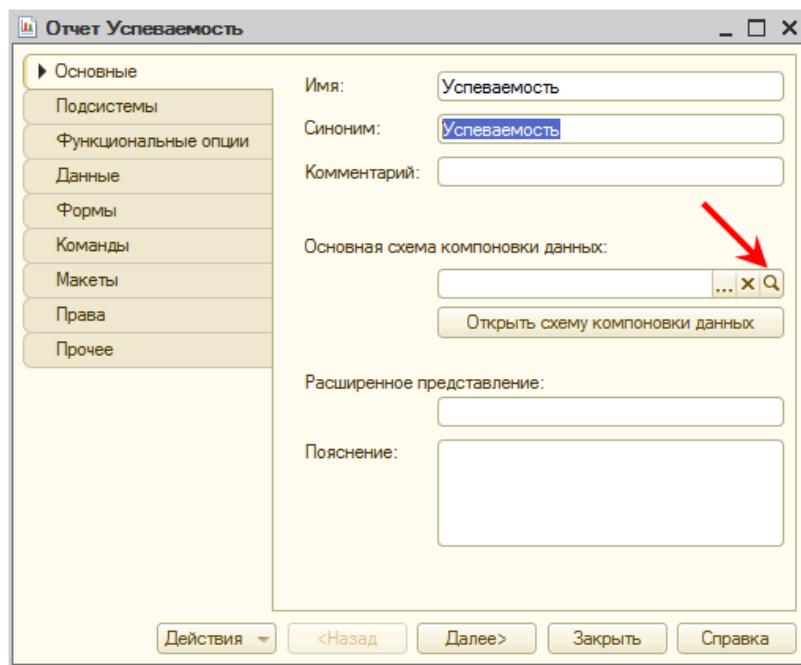


Рисунок 5.45. Создание основной схемы компоновки данных

Схема компоновки данных хранится у отчёта в макете. Поэтому платформа предложит вам создать макет единственного типа — типа *Схема компоновки данных*. Тут выбирать не из чего, поэтому просто согласитесь и нажмите кнопку *Готово*.

Платформа откроет вам *конструктор схемы компоновки данных*. Это специальный интерактивный инструмент, который позволяет создать схему компоновки только с помощью мыши (рисунок 5.46).

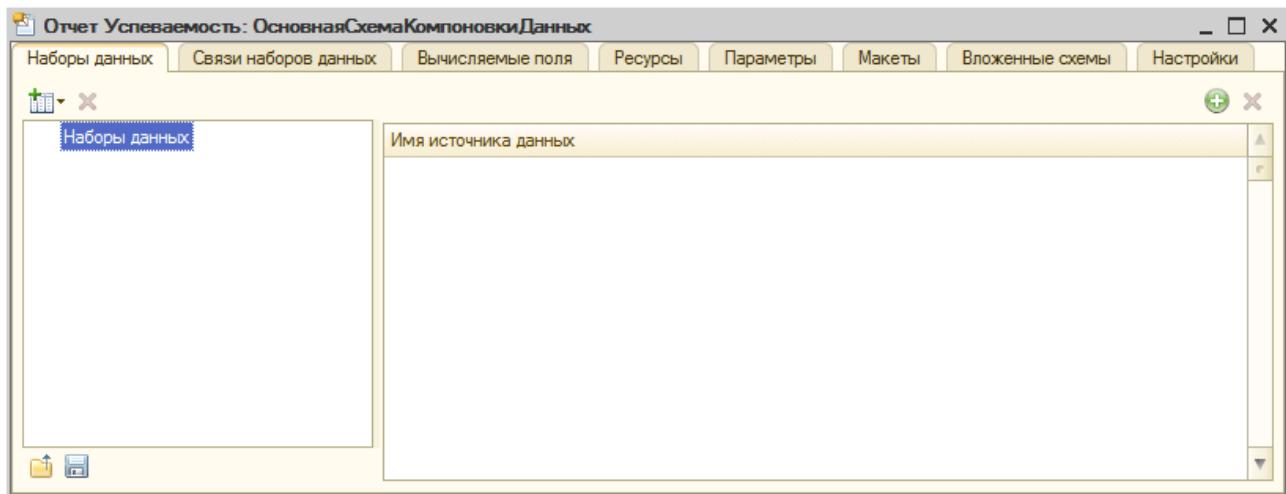


Рисунок 5.46. Пустая схема компоновки данных

Сейчас схема компоновки данных, то есть ваш «рецепт» приготовления отчёта, пустая. Начните её заполнять.

Закладка *Наборы данных* описывает то, что нужно получить из базы данных. То есть «в какой магазин пойти, какие продукты купить».

«Список магазинов» задаётся в окне *Наборы данных*. Нажмите кнопку *Добавить набор данных* и выберите пункт *Добавить набор данных — запрос* (рисунок 5.47).

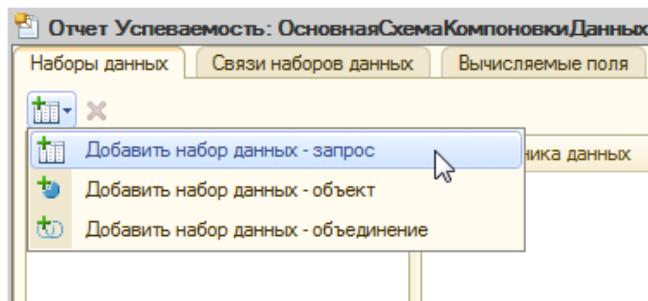


Рисунок 5.47. Добавление набора данных — запроса

Заметка

У схемы компоновки данных могут быть три разных вида источников данных. В подавляющем большинстве случаев используется «набор данных — запрос». Именно его вы и будете использовать во всех отчётах. Остальные виды источников используются гораздо реже и в этой книге не рассматриваются.

Платформа добавит пустой набор данных. Его стандартное имя, *НаборДанных1*, вас вполне устраивает. Менять его не нужно (рисунок 5.48).

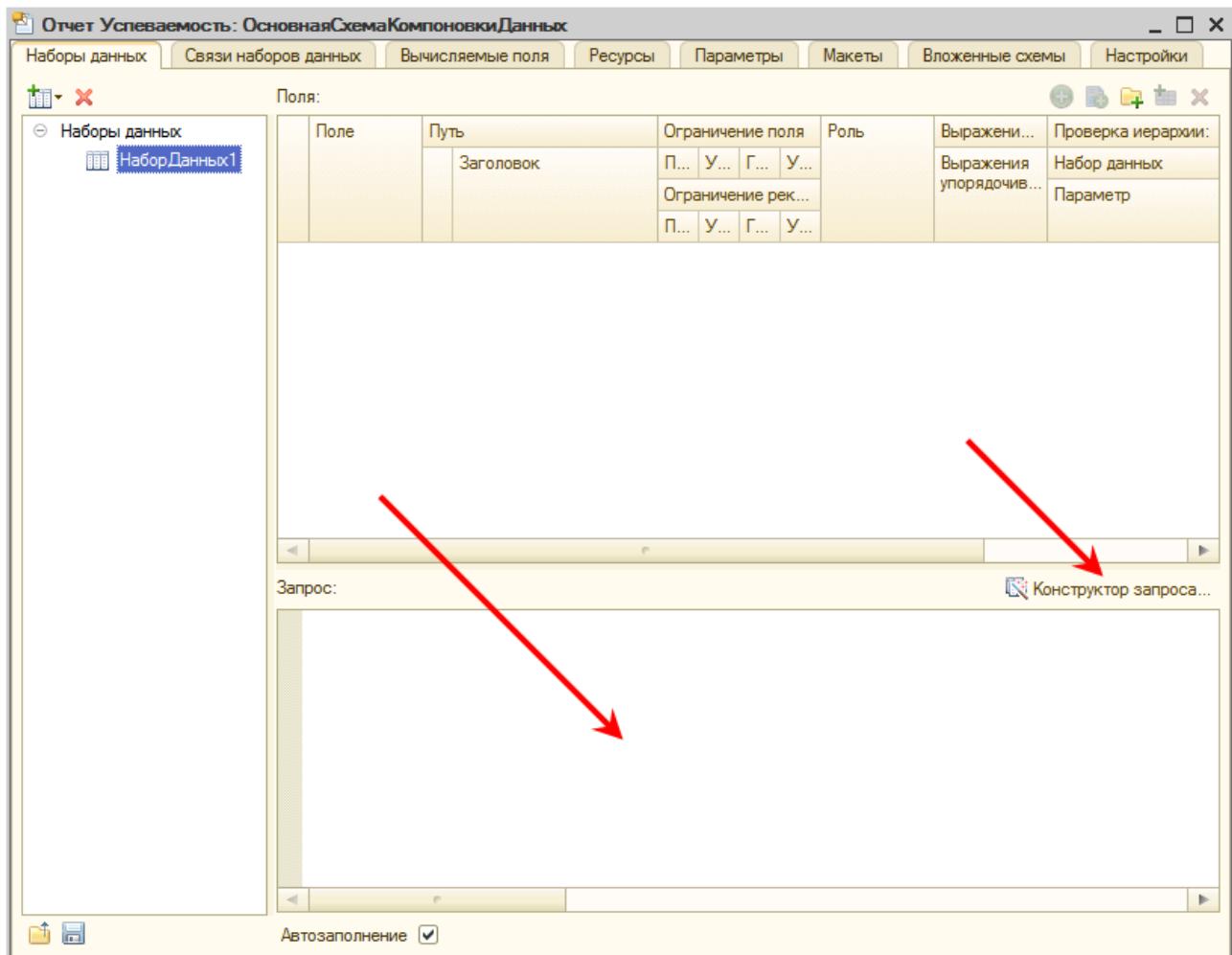


Рисунок 5.48. Пустой набор данных — запрос

Теперь вам нужно описать, откуда брать данные. Для этого используется специальный язык, с которым вы пока не знакомы, — *язык запросов*. Очень скоро, уже в следующей главе, вы познакомитесь с ним. А пока воспользуйтесь *конструктором запроса*. Он позволит вам составить запрос только с помощью мыши (рисунок 5.49).

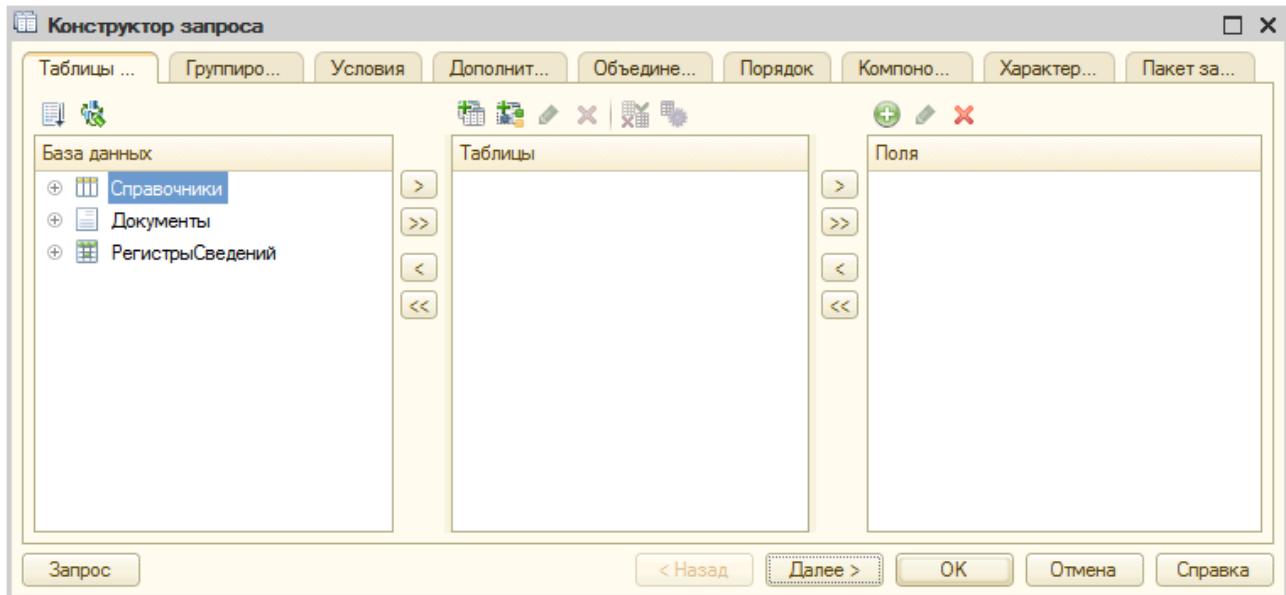


Рисунок 5.49. Конструктор запроса

Вы хотите анализировать свои оценки. Поэтому раскройте ветку *РегистрыСведений* и дважды щёлкните по регистру *Оценки*, чтобы выбрать его таблицу (рисунок 5.50).

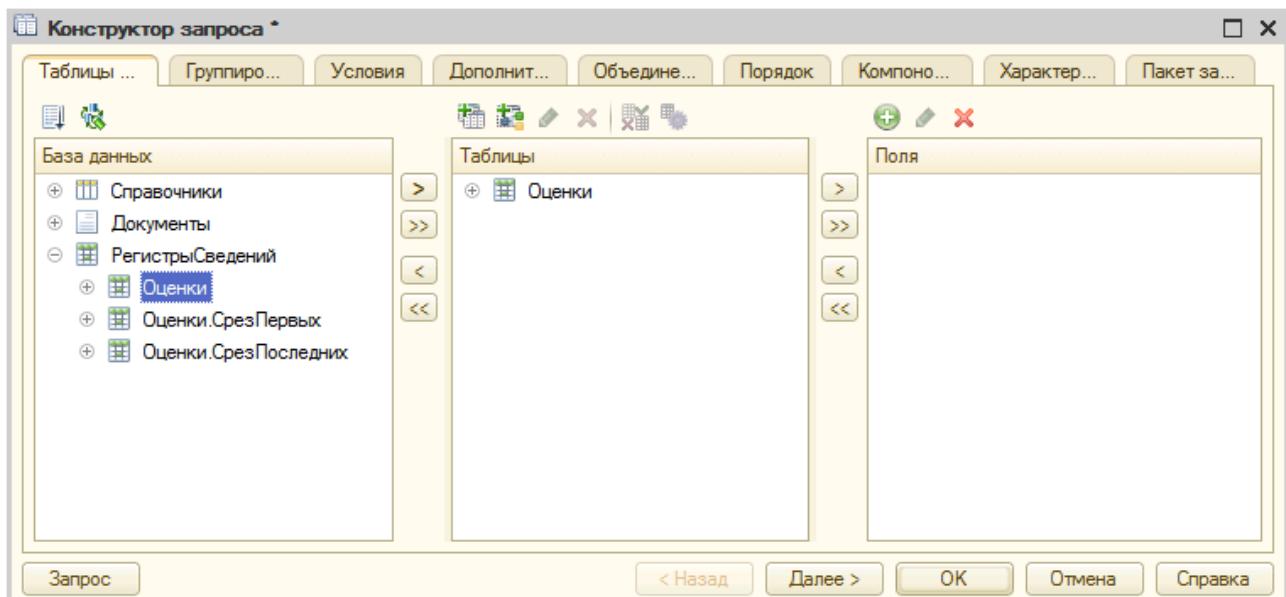


Рисунок 5.50. Выбрать таблицу «Оценки»

А теперь уже в поле *Таблицы* раскройте таблицу *Оценки* и выберите из неё поля *Предмет*, *Оценка* и *Регистратор* (рисунок 5.51).

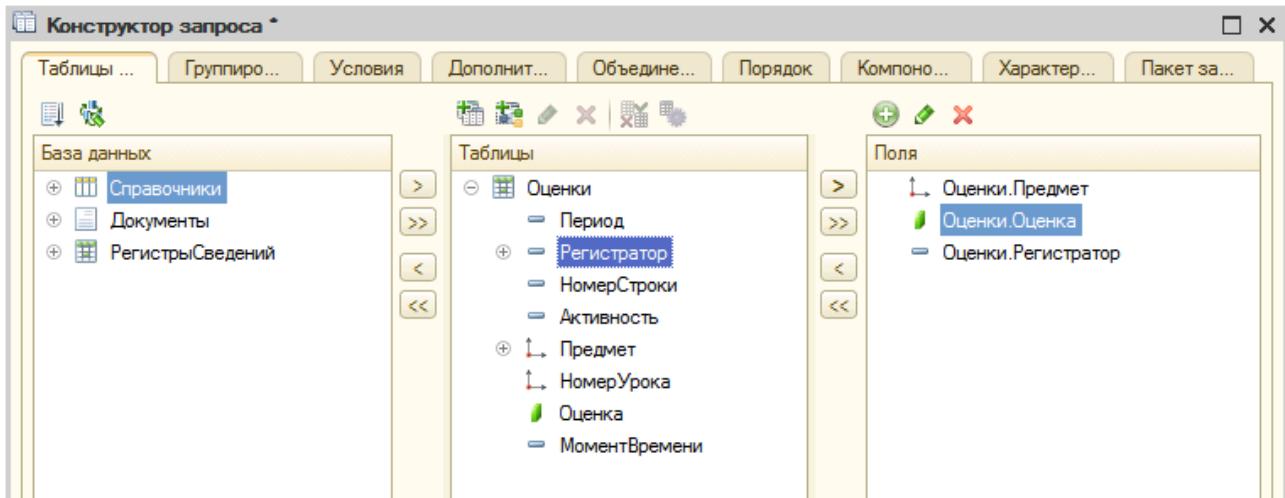


Рисунок 5.51. Выбрать все поля из таблицы «Оценки»

На этом ваш запрос почти готов. Осталось добавить последний штрих.

1С:Предприятие может использовать разные базы данных. Не только файловую базу данных, с которой сейчас работаете вы. Порядок, в котором одинаковые данные хранятся в разных СУБД, может различаться.

Поэтому практически всегда, когда вы создаёте какой-нибудь запрос к базе данных, обязательным правилом хорошего тона является указание порядка, в котором будут расположены данные, выбранные из базы.

Перейдите на закладку *Порядок* и выберите двойным щелчком поле *Регистратор.Дата* (*Все поля — Оценки — Регистратор — Дата*) (рисунок 5.52).

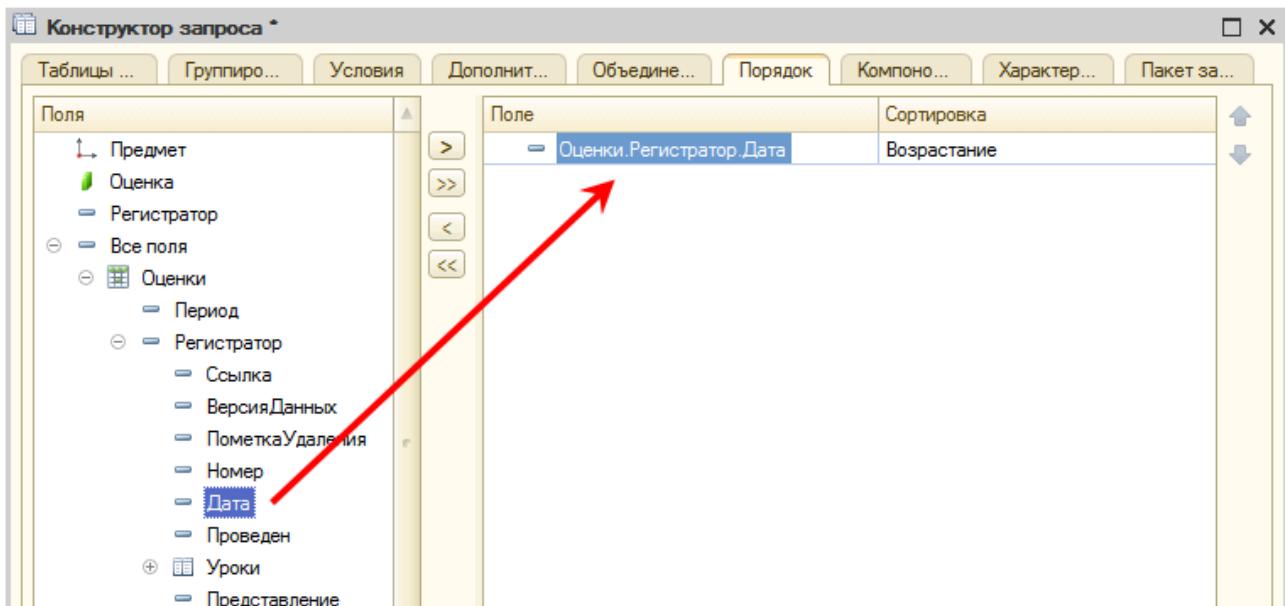


Рисунок 5.52. Порядок

Это значит, что оценки будут расположены в порядке возрастания даты, в которую они получены.

Запрос готов. Нажмите *OK* (рисунок 5.53).

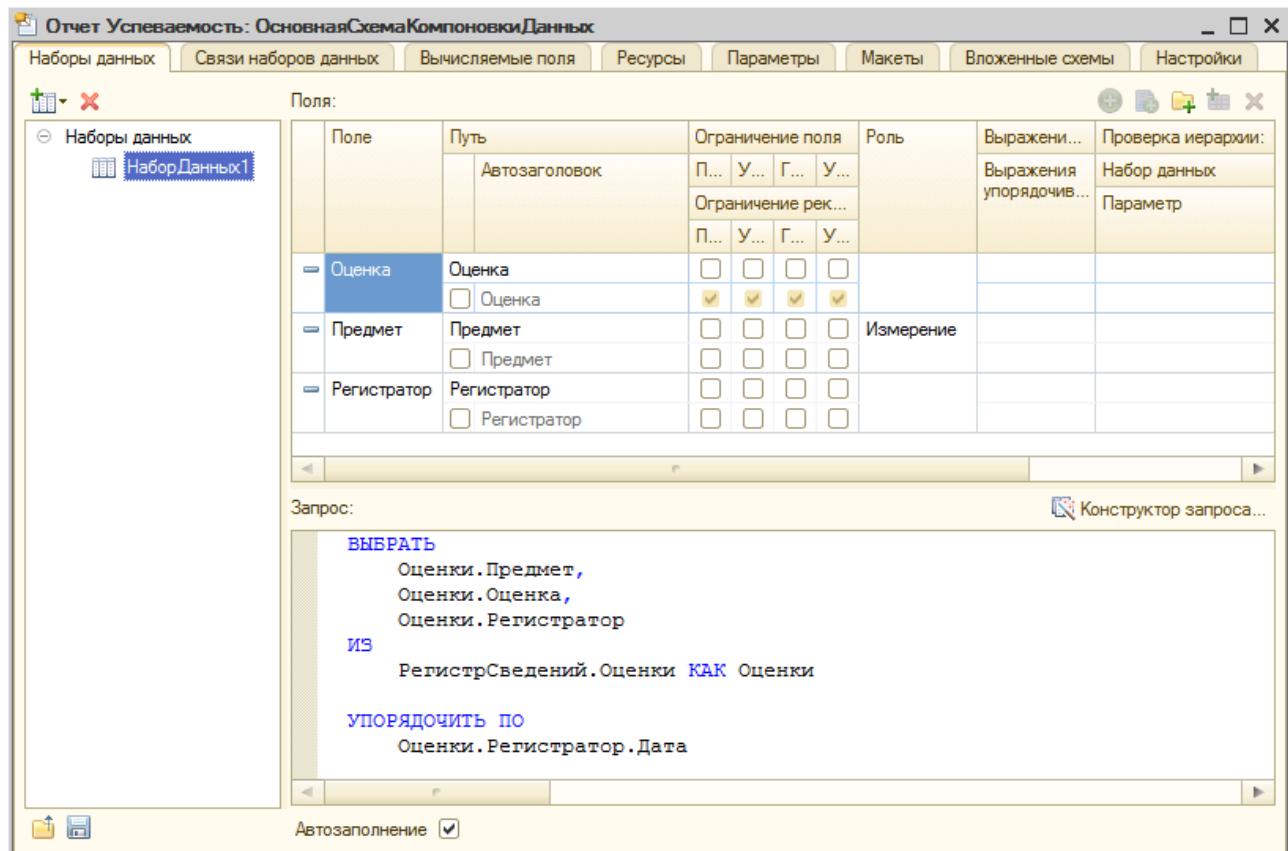


Рисунок 5.53. Текст запроса

В нижнем окне платформа автоматически сформирует текст запроса. А в верхнем окне она заполнит поля системы компоновки. То есть те поля, с которыми вы будете работать в этом отчёте.

Итак, этим запросом вы сказали, «какие продукты нужно купить». Теперь надо сказать, как их «приготовить».

Готовить вы будете очень просто. Перейдите на закладку *Ресурсы*. Здесь дважды щёлкните по полю *Оценка*. Затем, дважды щёлкнув в ячейке *Выражение*, измените её на *Среднее(Оценка)* (рисунок 5.54).

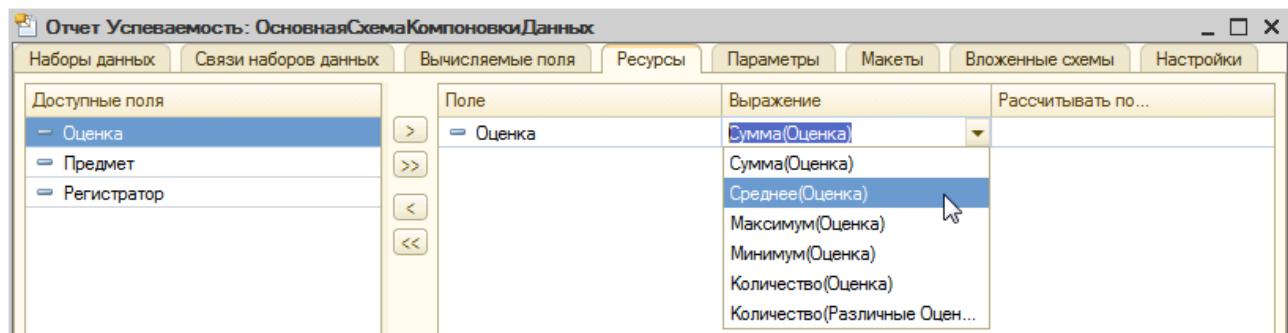


Рисунок 5.54. Ресурсы схемы компоновки

Тем самым вы указали, что система компоновки данных должна будет посчитать вам среднее значение оценки, средний балл.

Теперь, после того как вы сказали, как «приготовить продукты», нужно сделать последнее действие. Нужно указать «хозяйке», как эти приготовленные продукты «выложить на тарелку». Это делается на закладке *Настройки* (рисунок 5.55).

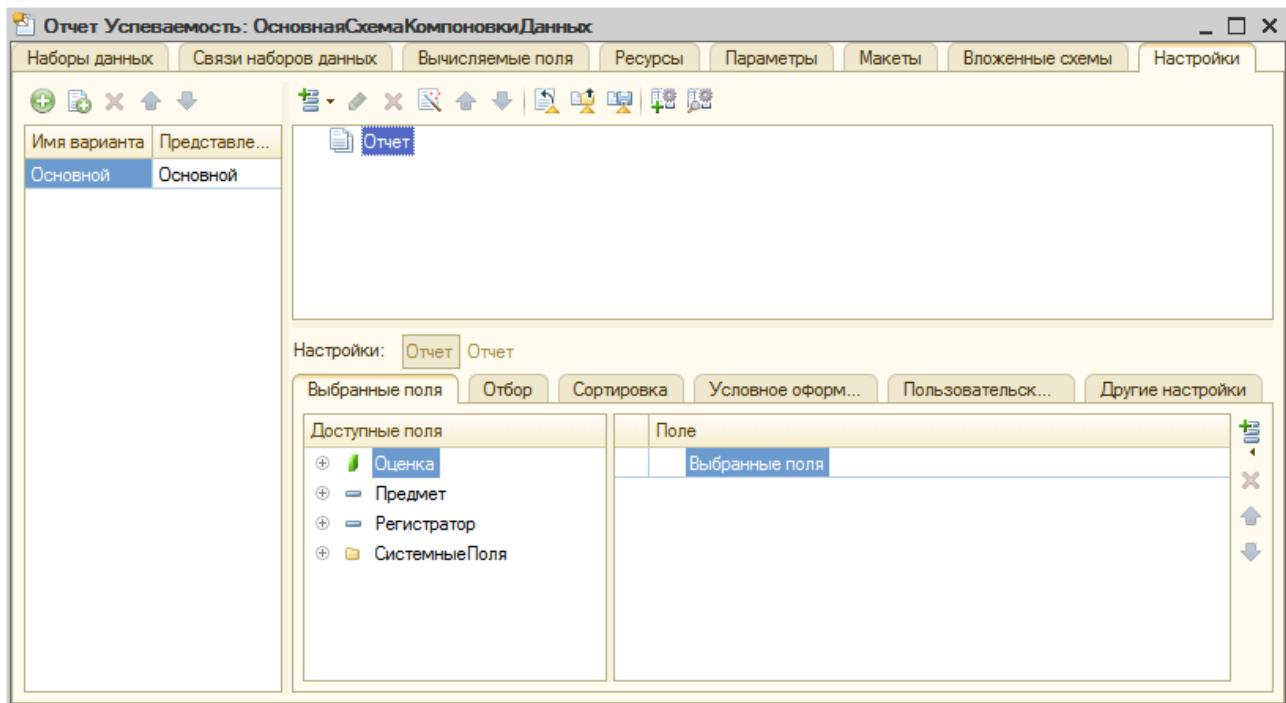


Рисунок 5.55. Настройки компоновки данных

Настройки компоновки данных полностью определяют то, как пользователь увидит данные вашего отчёта. В виде таблицы или в виде диаграммы, какие колонки будут у этой таблицы, и так далее.

Сначала нужно указать выбранные поля. Это те поля, которые вы хотите увидеть в отчёте. Если представить весь отчёт в виде таблицы, то эти поля будут её колонками.

Выберите поля *Предмет*, *Регистратор* и *Оценка* (рисунок 5.56).

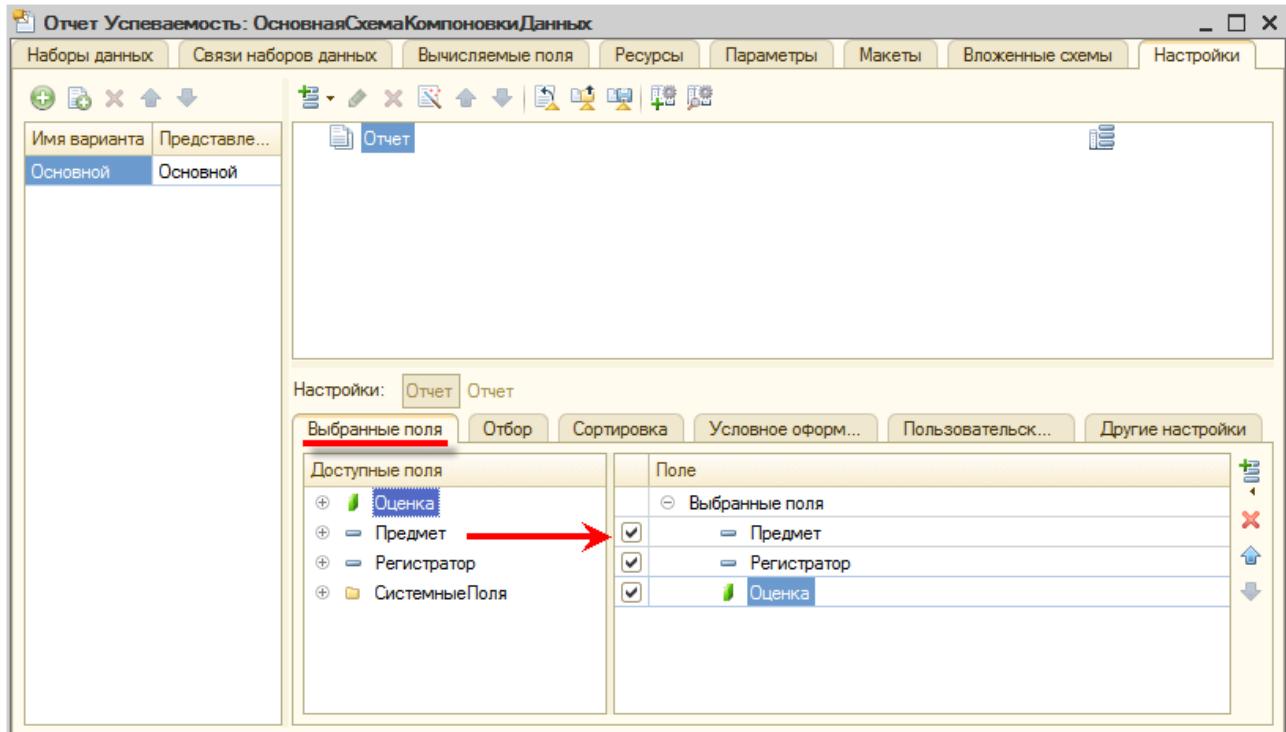


Рисунок 5.56. Выбранные поля отчёта

Теперь можно создать саму структуру отчёта. Пусть отчёт будет содержать у вас два

элемента. Сначала таблицу с оценками по предметам, а затем диаграмму с оценками по дням.

В верхнем поле вызовите контекстное меню на элементе *Отчёт* и выберите команду *Новая группировка...* (рисунок 5.57).

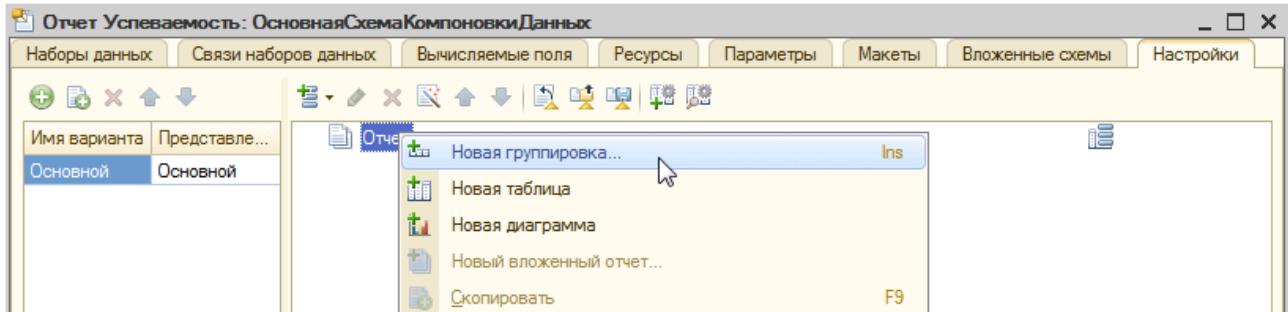


Рисунок 5.57. Добавление группировки

Появится небольшое окно для описания группировки. Вы хотите, чтобы средний балл был посчитан по каждому предмету, поэтому в поле *Поле* выберите *Предмет* (рисунок 5.58).

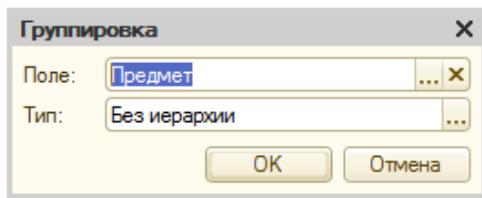


Рисунок 5.58. Группировка «Предмет»

Нажмите *OK*. Теперь уже на элементе *Предмет* вызовите контекстное меню и опять добавьте новую группировку. Но на этот раз не указывайте поле, а просто нажмите *OK*. Платформа добавит элемент, который обозначит как <Детальные записи> (рисунок 5.59).

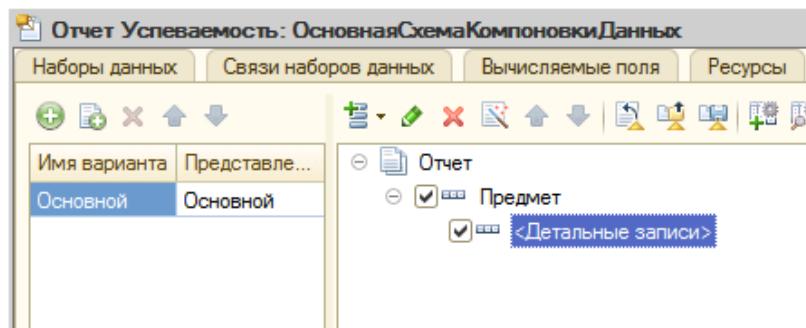


Рисунок 5.59. Группировка «Детальные записи»

Первая часть вашего отчёта уже готова, и вы можете посмотреть, как она выглядит. Запустите 1С:Предприятие в режиме отладки (рисунок 5.60).

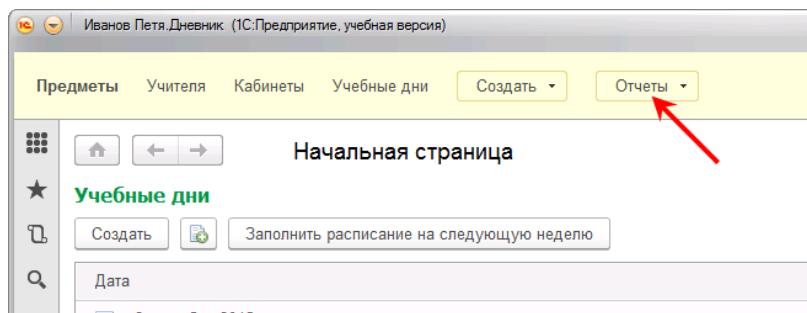


Рисунок 5.60. Панель разделов

В панели разделов появилась кнопка *Отчёты*. Если вы нажмёте на неё, вы увидите команду для запуска своего отчёта *Успеваемость*. Запустите его и нажмите кнопку *Сформировать* (рисунок 5.61).

Предмет	Оценка
Регистратор	
Английский язык	4
Учебный день 20150901 от 01.09.2015 12:00:00	5
Учебный день 20150902 от 02.09.2015 12:00:00	3
Кл. час	
Учебный день 20150901 от 01.09.2015 12:00:00	
Учебный день 20150901 от 01.09.2015 12:00:00	
Литература	5
Учебный день 20150902 от 02.09.2015 12:00:00	5
Математика	4,5
Учебный день 20150901 от 01.09.2015 12:00:00	5
Учебный день 20150902 от 02.09.2015 12:00:00	4
Музыка	4
Учебный день 20150901 от 01.09.2015 12:00:00	4
Природоведение	4,5
Учебный день 20150902 от 02.09.2015 12:00:00	5
Учебный день 20150902 от 02.09.2015 12:00:00	4
Русский язык	3,5
Учебный день 20150901 от 01.09.2015 12:00:00	4
Учебный день 20150902 от 02.09.2015 12:00:00	3
Итого	3,5

Рисунок 5.61. Отчёт «Успеваемость»

Стандартно отчёт устроен таким образом, что после выполнения он разворачивает все свои группировки. Можно сворачивать их, нажимая мышью на значки группировок, а можно свернуть сразу все. Для этого вызовите контекстное меню в левой верхней ячейке таблицы и выполните команду *Уровень группировок — Уровень 1* (рисунок 5.62).

Предмет	Оценка
Вырезать Ctrl+X	4
Копировать Ctrl+C	5
Вставить Ctrl+V	3
Специальная вставка...	
Объединить Ctrl+M	5
Раздвинуть	4,5
Разбить ячейку	5
Удалить	4
Очистить	4,5
Вставить примечание	5
Уровни группировок	4
Свойства Alt+Enter	3,5
Уровень 1	
Уровень 2	3,5

Рисунок 5.62. Команда отображения группировок первого уровня

Все группировки свернутся, и вы увидите только средние оценки по каждому предмету, а внизу таблицы, — общий средний балл по всем предметам (рисунок 5.63).

Предмет	Оценка
Регистратор	
Английский язык	4
Кл. час	5
Литература	4,5
Математика	4
Музыка	4,5
Природоведение	3,5
Русский язык	3,5
Итого	3,5

Рисунок 5.63. Средний балл по предметам

Если вы хотите узнать, из каких оценок получился средний балл 4 по английскому языку, можете развернуть эту группировку (рисунок 5.64).

Предмет	Оценка
Регистратор	
Английский язык	4
Учебный день 20150901 от 01.09.2015 12:00:00	5
Учебный день 20150902 от 02.09.2015 12:00:00	3
Кл. час	
Литература	5
Математика	4,5
Музыка	4
Природоведение	4,5
Русский язык	3,5
Итого	3,5

Рисунок 5.64. Оценки по английскому языку

На этой картинке как раз хорошо видно структуру отчёта, которую вы создавали в конфигураторе. Если расположить её слева от отчёта, получится такая картинка (рисунок 5.65).

Предмет	Оценка
Регистратор	
Английский язык	4
Учебный день 20150901 от 01.09.2015 12:00:00	5
Учебный день 20150902 от 02.09.2015 12:00:00	3
Кл. час	
Литература	5
Математика	4,5
Музыка	4
Природоведение	4,5
Русский язык	3,5
Итого	3,5

Рисунок 5.65. Оценки по английскому языку

Строка Английский язык — это и есть группировка Предмет, то есть по предмету. И в этой группировке посчитано среднее значение ресурса, то есть оценки.

А ниже неё находятся две *детальные записи*. За один и за другой учебный день. В них просто показано значение ресурса, то есть оценка, которая имеется в этих записях.

Если вы раскроете Классный час, то увидите, что по этому предмету оценок не было, но записи в регистре есть (рисунок 5.66).

Предмет	Оценка
Регистратор	
Английский язык	4
+ Кл. час	
Учебный день 20150901 от 01.09.2015 12:00:00	
Учебный день 20150901 от 01.09.2015 12:00:00	
+ Литература	5
+ Математика	4,5
+ Музыка	4
+ Природоведение	4,5
+ Русский язык	3,5
Итого	3,5

Рисунок 5.66. Оценки за классный час

Так произошло потому, что обработку проведения вы создавали автоматически, с помощью конструктора. И эту особенность конструктор, естественно, учсть не мог. Значит, нужно будет это поправить самостоятельно.

Аналогичным образом вы можете посмотреть, из чего складываются средние оценки и по другим предметам (рисунок 5.67).

Предмет	Оценка
Регистратор	
Английский язык	4
+ Кл. час	
Литература	5
Учебный день 20150902 от 02.09.2015 12:00:00	5
Математика	4,5
Музыка	4
Природоведение	4,5
Учебный день 20150902 от 02.09.2015 12:00:00	5
Учебный день 20150902 от 02.09.2015 12:00:00	4
Русский язык	3,5
Итого	3,5

Рисунок 5.67. Оценки по другим предметам

А если вы дважды щёлкнете на любом из регистраторов, перечисленных в таблице, то платформа откроет этот документ (рисунок 5.68).

Предмет	Оценка
Регистратор	4
Английский язык	5
Кл. час	5
Литература	4,5
Математика	4
Музыка	4,5
Природоведение	5
Русский язык	3,5
Итого	3,5

Рисунок 5.68. Платформа откроет документ

Закройте 1С:Предприятие и вернитесь в конфигуратор.

Сначала вы немного измените обработку проведения документов (чтобы не было записей без оценок). А затем завершите создание отчёта, добавив в него диаграмму оценок по дням.

Откройте модуль документа *УчебныйДень* и раскройте его процедуру проведения. В цикл добавьте проверку того, что значение *Оценка* заполнено (листинг 5.1).

Листинг 5.1. Проверка того, что оценка заполнена

```
// регистр Оценки
Движения.Оценки.Записывать = Истина;
Для Каждого ТекСтрокаУроки Из Уроки Цикл
    Если ЗначениеЗаполнено(ТекСтрокаУроки.Оценка) Тогда
        Движение = Движения.Оценки.Добавить();
        Движение.Период = Дата;
        Движение.Предмет = ТекСтрокаУроки.Предмет;
        Движение.НомерУрока = ТекСтрокаУроки.НомерСтрочки;
        Движение.Оценка = ТекСтрокаУроки.Оценка;

    КонецЕсли;
КонецЦикла;
```

То есть записываться будут только такие движения, в которых есть какая-нибудь оценка.

Теперь, чтобы не забыть, запустите 1С:Предприятие в режиме отладки и *перепроведите* оба документа. Для этого откройте каждый документ и нажмите *Провести* и закрыть.

Старые движения, которые есть в регистре, платформа заменит новыми, в соответствии с изменившейся программой.

Сразу и проверьте. Сформируйте отчёт *Успеваемость*. Записей про классный час в нём уже не будет (рисунок 5.69).

Предмет	Оценка
Регистратор	
Английский язык	4
Учебный день 20150901 от 01.09.2015 12:00:00	5
Учебный день 20150902 от 02.09.2015 12:00:00	3
Литература	5
Учебный день 20150902 от 02.09.2015 12:00:00	5
Математика	4,5
Учебный день 20150901 от 01.09.2015 12:00:00	5
Учебный день 20150902 от 02.09.2015 12:00:00	4
Музыка	4
Учебный день 20150901 от 01.09.2015 12:00:00	4
Природоведение	4,5
Учебный день 20150902 от 02.09.2015 12:00:00	4
Учебный день 20150902 от 02.09.2015 12:00:00	5
Русский язык	3,5
Учебный день 20150901 от 01.09.2015 12:00:00	4
Учебный день 20150902 от 02.09.2015 12:00:00	3
Итого	4,2

Рисунок 5.69. Отчёт, построенный по новым движениям документа

Теперь вернитесь в конфигуратор и закончите отчёт.

Как вы помните, вы хотели видеть в отчёте таблицу, а под ней — диаграмму. На элементе *Отчёт* вызовите контекстное меню и выполните команду *Новая диаграмма* (рисунок 5.70).

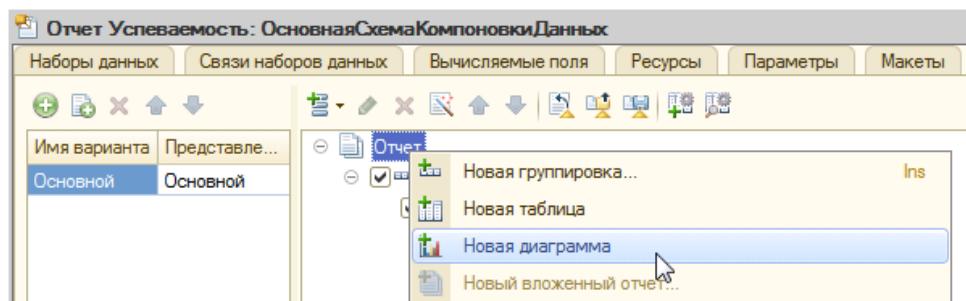


Рисунок 5.70. Добавление диаграммы

У диаграммы есть точки и серии. Что это такое, вы поймёте потом, когда увидите, как выглядит диаграмма. А пока в точки диаграммы с помощью контекстного меню добавьте группировку *Регистратор* (рисунок 5.71).

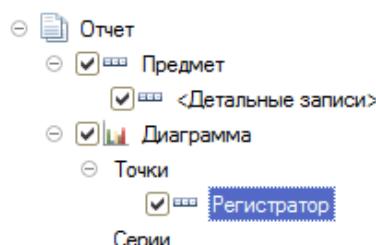


Рисунок 5.71. Точки диаграммы

А в серии диаграммы добавьте группировку *Предмет* (рисунок 5.72).

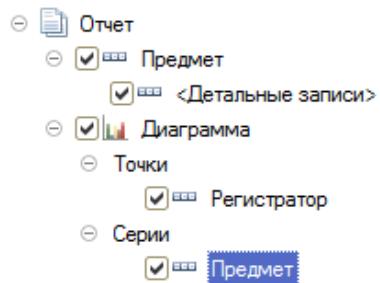


Рисунок 5.72. Серии диаграммы

Всё, диаграмма готова. Теперь запустите 1С:Предприятие в режиме отладки, сформируйте отчёт и прокрутите его вниз. Вы увидите диаграмму (рисунок 5.73).

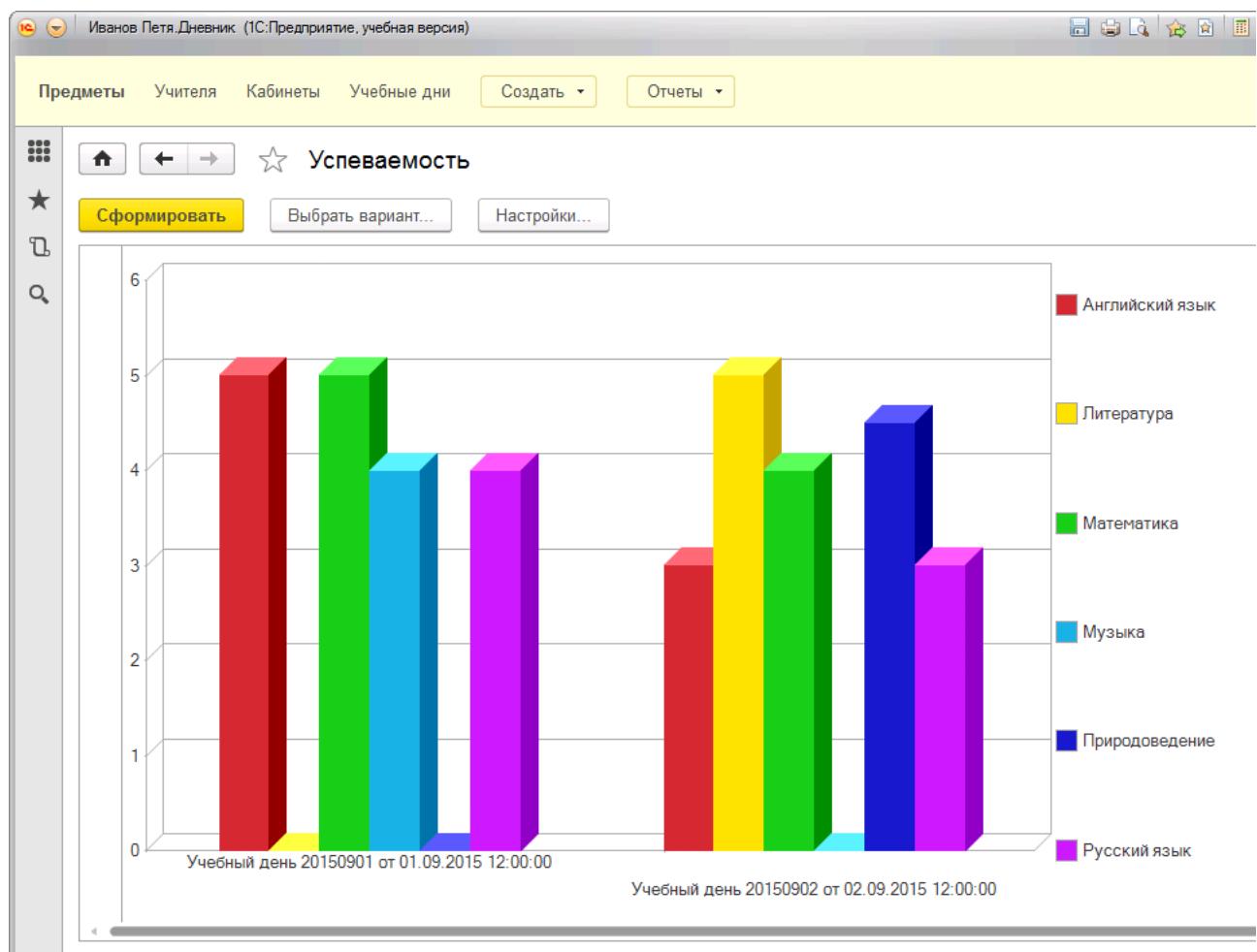


Рисунок 5.73. Диаграмма

По горизонтали в ней показаны учебные дни. Оценки по каждому предмету показаны своим цветом. Английский язык — красным, литература — жёлтым и так далее. В каждый день показаны все полученные оценки.

Если к этой диаграмме добавить её структуру из конфигуратора, то получится следующая картинка (рисунок 5.74).

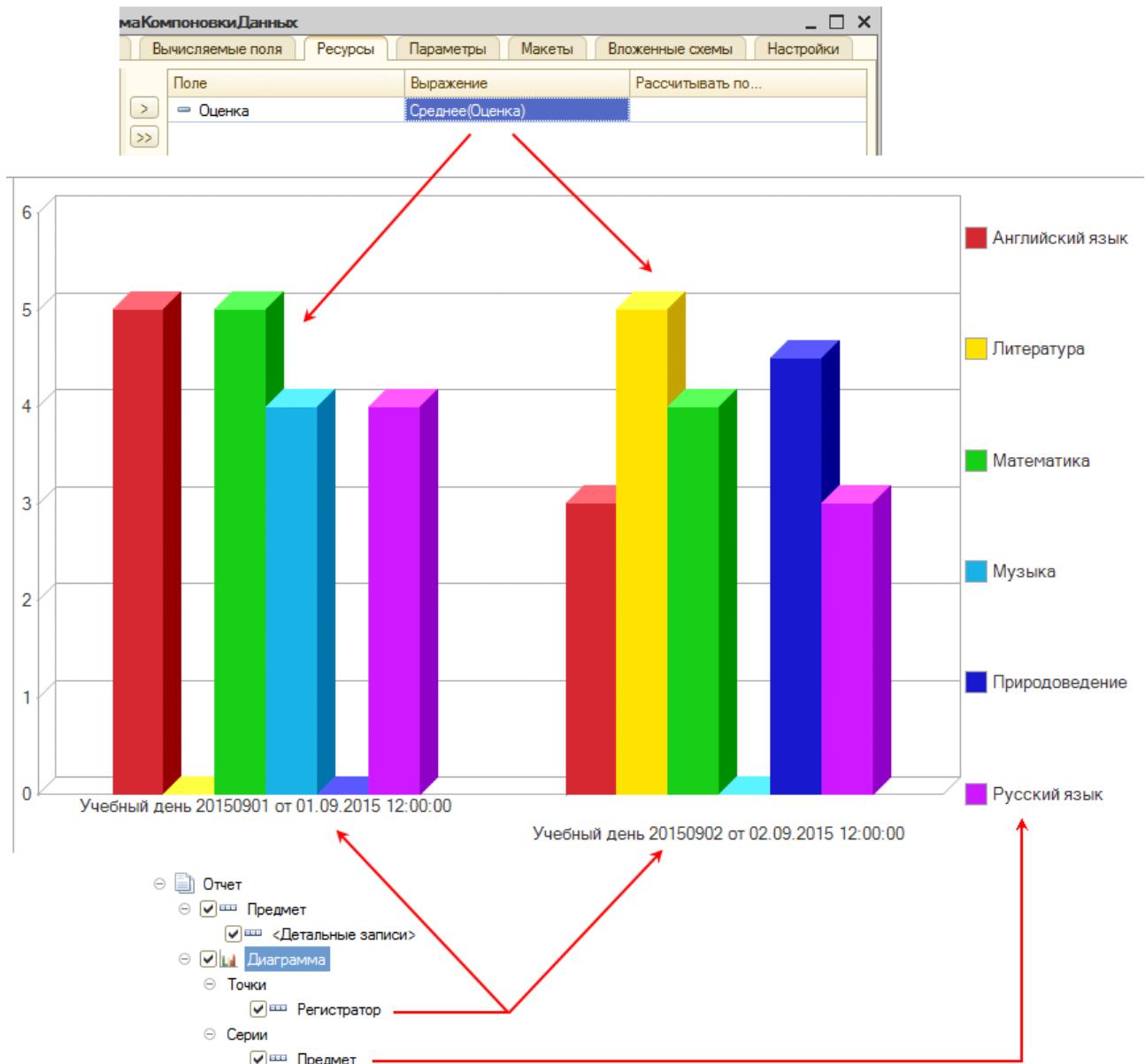


Рисунок 5.74. Структура диаграммы

Точки диаграммы — это отметки, указываемые на горизонтальной оси. Серии диаграммы — это наборы значений, которые вы хотите показать в диаграмме. Набор оценок по английскому языку, набор оценок по литературе и так далее. А «внутри» диаграммы, в её теле, отображается ресурс, который рассчитывается в вашем отчёте, тот, который вы указали на закладке Ресурсы. То есть средний балл.

Информационная база

Прикладное решение, которое должно получиться у вас к этому моменту, содержится в демонстрационной базе «06 ОтчетУспеваемость.dt». Как её подключить, написано в разделе A.1 «Как подключить демонстрационную базу» на странице 543.

Задание 5.1

Для выполнения заданий создайте новую информационную базу и загрузите в неё демонстрационную базу «06 ОтчетУспеваемость.dt».

В таблице отчёта Успеваемость отключите детальные записи, а в диаграмме отключите точки.

Запустите отчёт. Объясните, какие данные теперь вы видите в таблице и диаграмме.

Задание 5.2

Используйте отчёт, который получился у вас в задании 5.1. В таблице вместо группировки по предмету создайте группировку по учебному дню, включите детальные записи.

Объясните, какие данные теперь вы видите в таблице.

Задание 5.3

Используйте отчёт, который получился у вас в задании 5.1: в таблице используется группировка Предмет и детальные записи отключены.

Вместо объёмной гистограммы представьте данные в измерительной диаграмме. Подсказки:

1. В структуре отчёта выделите корень, Отчёт, и откройте закладку *Другие настройки*.
2. Выберите тип диаграммы *Измерительная*, задайте *Базовое значение 2* и *Минимальное значение тоже 2*.
3. Создайте *Полосы измерительной диаграммы*: 2–3 красная, 3–4 жёлтая, 4–5 зелёная.

Подробнее

Подробнее вы можете прочитать в документации:

- про систему компоновки данных «Руководство разработчика 8.3. Глава 10. Система компоновки данных»;
- про схему компоновки данных «Руководство разработчика 8.3. Раздел 10.3. Схема компоновки данных»;
- про конструктор схемы компоновки данных «Руководство разработчика 8.3. Раздел 10.3.5. Конструктор схемы компоновки данных».

5.4 Регистр накопления

Теперь познакомьтесь с ещё одним видом регистра и сделайте отчёт, который будет использовать его данные.

Отчёт будет называться *Прошедшие Занятия*. Он будет показывать, сколько учебных занятий по каждому из предметов прошло с того времени, как вы начали их записывать в свою базу. Для того чтобы получить такие данные, нужно каждый раз, как вы проводите документ *Учебный День*, прибавлять сколько-то занятий к тому, что уже имеется.

Для таких целей хорошо подходит регистр накопления. Как я уже говорил раньше, он не просто хранит те данные, которые вы в него записываете, а суммирует значения своих ресурсов.

Регистры накопления бывают двух видов: *регистр остатков* и *регистр оборотов*.

Регистр остатков нужен тогда, когда вы хотите знать, «сколько осталось». Например, у вас есть кошельёк с деньгами. Их количество может увеличиваться, может уменьшаться. Вас прежде всего интересует, сколько денег осталось в кошельке. Кроме этого вам, может быть, интересно знать, сколько вы потратили. Для такой задачи хорошо подходит регистр остатков. Он подсчитывает и остатки («сколько было», «сколько стало»), и обороты («сколько потратили»).

Регистр оборотов нужен тогда, когда вы хотите знать только обороты. Когда понятие остатка («сколько было», «сколько стало») не имеет смысла. Когда учитываемое значение изменяется только в одну сторону (только увеличивается или только уменьшается).

Например, вы занимаетесь физкультурой и выполняете упражнения. Каждый день вам нужно выполнять какое-то количество упражнений. Тут вас интересует только то, сколько упражнений вы выполнили сегодня или вчера. Количество выполненных упражнений не может уменьшаться, оно только увеличивается. Для вас нет никакого смысла в том, чтобы знать остатки упражнений, например «сколько упражнений я выполнил к началу этого дня». Для таких задач хорошо подходит регистр оборотов. Он подсчитывает только изменения ресурсов. Что вчера ресурс изменился на 10 (вы выполнили десять упражнений), а сегодня он изменился на 15 (вы выполнили 15 упражнений).

Примечание

Если вам нужно знать, «сколько осталось», используйте регистр остатков.

Если вам нужно знать только то, «на сколько изменилось» за какой-то период, или изменение происходит только в одну сторону (только увеличивается), то тогда используйте регистр оборотов.

Ваша задача — подсчитывать количество учебных занятий, которые прошли в тот или иной день. Поэтому для такой задачи вы будете использовать регистр оборотов.

5.4.1 Регистр накопления ПрошедшиеЗанятия

Перейдите в конфигуратор. Добавьте регистр накопления и назовите его *Прошедшие Занятия* (рисунок 5.75).

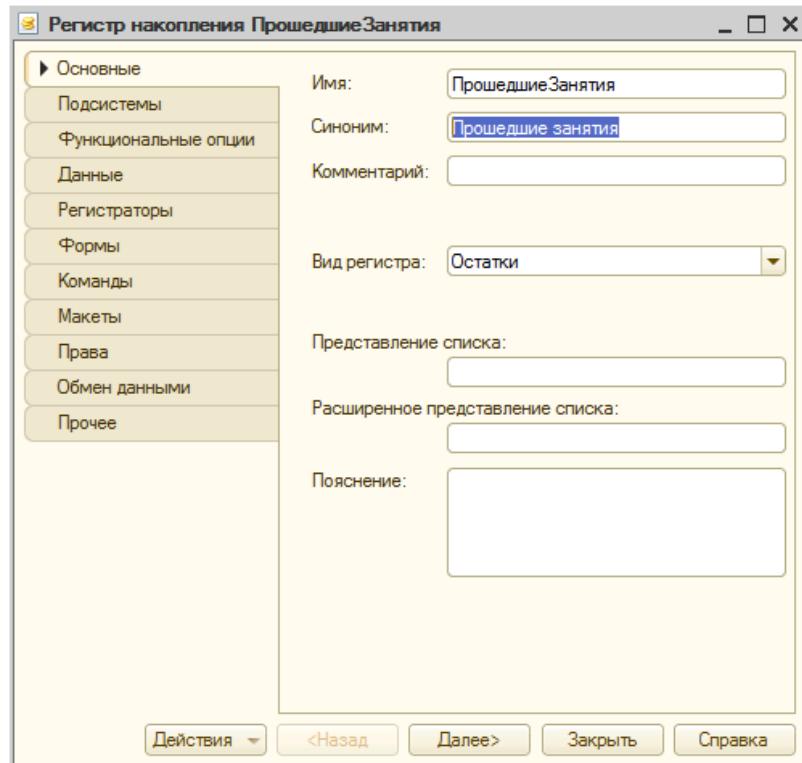


Рисунок 5.75. Регистр накопления «ПрошедшиеЗанятия»

Это будет регистр оборотов. Поэтому в свойстве *Вид регистра* выберите *Обороты* (рисунок 5.76).

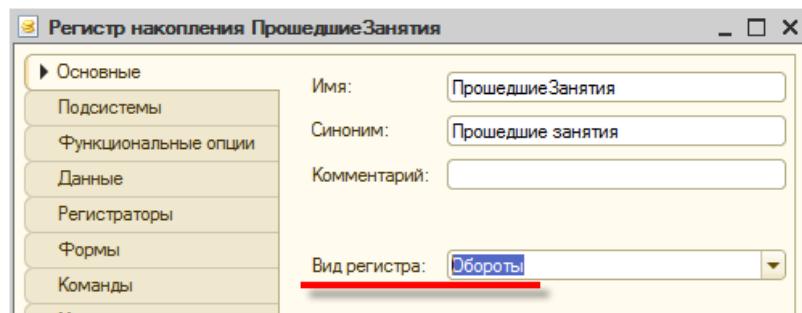


Рисунок 5.76. Регистр оборотов

Перейдите на закладку *Данные* и создайте структуру регистра. Добавьте:

- измерение *Предмет*, тип *СправочникСсылка.Предметы*;
- ресурс *КоличествоУроков*, тип *Число* (рисунок 5.77).

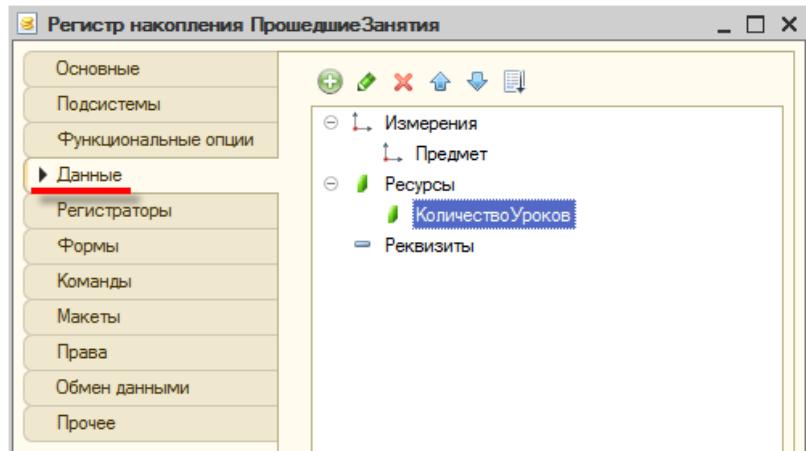


Рисунок 5.77. Измерение и ресурс

Регистр готов.

Теперь нужно изменить процедуру проведения документа УчебныйДень, чтобы он добавлял свои движения и в этот регистр.

Откройте модуль документа и раскройте процедуру ОбработкаПроведения. Раньше вы создавали движения с помощью конструктора. И конструктор вставил в модуль свои комментарии. Чтобы при повторном его использовании он мог заменить старый текст новым.

Но вы уже изменили текст, созданный конструктором. Вы добавили проверку того, что поле Оценка заполнено. Поэтому не имеет смысла снова пользоваться конструктором, а проще дописать алгоритм движений для нового регистра вручную. Тем более что он будет практически таким же.

Сначала удалите комментарии, чтобы процедура приобрела такой вид (листинг 5.2).

Листинг 5.2. Процедура без комментариев

Процедура ОбработкаПроведения(Отказ, Режим)

```
// регистр Оценки
Движения.Оценки.Записывать = Истина;
Для Каждого ТекСтрокаУроки Из Уроки Цикл
    Если ЗначениеЗаполнено(ТекСтрокаУроки.Оценка) Тогда
        Движение = Движения.Оценки.Добавить();
        Движение.Период = Дата;
        Движение.Предмет = ТекСтрокаУроки.Предмет;
        Движение.НомерУрока = ТекСтрокаУроки.НомерСтроки;
        Движение.Оценка = ТекСтрокаУроки.Оценка;

    КонецЕсли;
КонецЦикла;
```

КонецПроцедуры

Теперь немного измените оставшийся текст. Цикл обхода табличной части у вас будет один, но в этом цикле вы будете записывать движения как в один, так и в другой регистр. Поэтому перенесите комментарий про регистр Оценки внутрь цикла (листинг 5.3).

Листинг 5.3. Комментарий внутри цикла

```
Движения.Оценки.Записывать = Истина;
Для Каждого ТекСтрокаУроки Из Уроки Цикл

    // Регистр Оценки
    Если ЗначениеЗаполнено(ТекСтрокаУроки.Оценка) Тогда
        Движение = Движения.Оценки.Добавить();
        Движение.Период = Дата;
        Движение.Предмет = ТекСтрокаУроки.Предмет;
        Движение.НомерУрока = ТекСтрокаУроки.НомерСтрочки;
        Движение.Оценка = ТекСтрокаУроки.Оценка;

    КонецЕсли;

КонецЦикла;
```

Теперь скопируйте первую строку процедуры и замените в ней *Оценки* на *ПрошедшиеЖанятия* (листинг 5.4).

Листинг 5.4. Записывать движения регистра «ПрошедшиеЖанятия»

```
Движения.Оценки.Записывать = Истина;
Движения.ПрошедшиеЖанятия.Записывать = Истина;
Для Каждого ТекСтрокаУроки Из Уроки Цикл

    // Регистр Оценки
    Если ЗначениеЗаполнено(ТекСтрокаУроки.Оценка) Тогда
        Движение = Движения.Оценки.Добавить();
        Движение.Период = Дата;
        Движение.Предмет = ТекСтрокаУроки.Предмет;
        Движение.НомерУрока = ТекСтрокаУроки.НомерСтрочки;
        Движение.Оценка = ТекСтрокаУроки.Оценка;

    КонецЕсли;

КонецЦикла;
```

Буквально в следующем разделе вы познакомитесь с тем, как работать с регистрами из встроенного языка. Поэтому сейчас просто выполните это без объяснений.

Теперь после инструкции *Если* напишите комментарий *Регистр ПрошедшиеЖанятия* и добавьте следующие строки (листинг 5.5).

Листинг 5.5. Движения в регистре «ПрошедшиеЗанятия»

```
Движения.Оценки.Записывать = Истина;
Движения.ПрошедшиеЗанятия.Записывать = Истина;
Для Каждого ТекСтрокаУроки Из Уроки Цикл
```

```
// Регистр Оценки
Если ЗначениеЗаполнено(ТекСтрокаУроки.Оценка) Тогда
    Движение = Движения.Оценки.Добавить();
    Движение.Период = Дата;
    Движение.Предмет = ТекСтрокаУроки.Предмет;
    Движение.НомерУрока = ТекСтрокаУроки.НомерСтроки;
    Движение.Оценка = ТекСтрокаУроки.Оценка;
```

КонецЕсли;

```
// Регистр ПрошедшиеЗанятия
Движение = Движения.ПрошедшиеЗанятия.Добавить();
Движение.Период = Дата;
Движение.Предмет = ТекСтрокаУроки.Предмет;
Движение.КоличествоУроков = 1;
```

КонецЦикла;

Процедура проведения готова.

Последнее, что осталось сделать, — указать, что этот документ будет создавать движения и по регистру *ПрошедшиеЗанятия*. Откройте окно редактирования документа, перейдите на закладку *Движения* и отметьте регистр накопления *ПрошедшиеЗанятия* (рисунок 5.78).

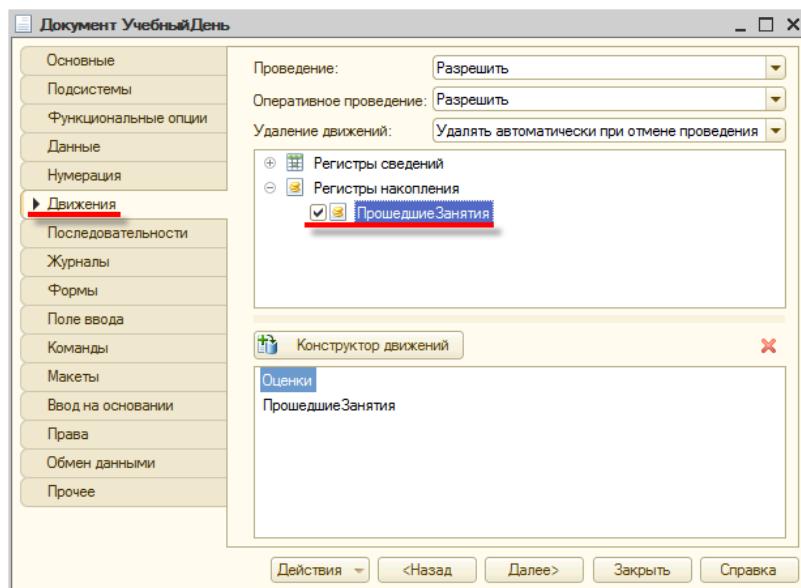


Рисунок 5.78. Документ делает движения по регистру накопления

Теперь у вас готово всё. Запустите 1С:Предприятие в режиме отладки и перепроверните документы *УчебныйДень*. Для того чтобы отработал новый алгоритм проведения.

Ради интереса можете посмотреть, какие записи появились в регистре накопления. Открыть его можно так же, как и регистр сведений, через *Главное меню – Все функции...* (рисунок 5.79).

Период ↓	Регистратор	Номер строки	Предмет	Количество уроков
• 01.09.2015 12:00:00	Учебный день 20150901 от 01.09.2015 12:00:00	1	Кл. час	1
• 01.09.2015 12:00:00	Учебный день 20150901 от 01.09.2015 12:00:00	2	Кл. час	1
• 01.09.2015 12:00:00	Учебный день 20150901 от 01.09.2015 12:00:00	3	Английский язык	1
• 01.09.2015 12:00:00	Учебный день 20150901 от 01.09.2015 12:00:00	4	Музыка	1
• 01.09.2015 12:00:00	Учебный день 20150901 от 01.09.2015 12:00:00	5	Математика	1
• 01.09.2015 12:00:00	Учебный день 20150901 от 01.09.2015 12:00:00	6	Русский язык	1
• 02.09.2015 12:00:00	Учебный день 20150902 от 02.09.2015 12:00:00	1	Математика	1
• 02.09.2015 12:00:00	Учебный день 20150902 от 02.09.2015 12:00:00	2	Английский язык	1
• 02.09.2015 12:00:00	Учебный день 20150902 от 02.09.2015 12:00:00	3	Природоведение	1
• 02.09.2015 12:00:00	Учебный день 20150902 от 02.09.2015 12:00:00	4	Природоведение	1
• 02.09.2015 12:00:00	Учебный день 20150902 от 02.09.2015 12:00:00	5	Русский язык	1
• 02.09.2015 12:00:00	Учебный день 20150902 от 02.09.2015 12:00:00	6	Литература	1

◀ ▲ ▾ ▶

Рисунок 5.79. Данные регистра «ПрошедшиеЗанятия»

Как видите, всё очень похоже на регистр сведений. Те же «служебные» поля *Период*, *Регистратор* и *Номер строки*. Поле измерения *Предмет* и поле ресурса *Количество уроков*. Только использует он эти поля немного иначе, и вы это сейчас увидите.

Подробнее

Подробнее вы можете прочитать про регистр накопления в документации «Руководство разработчика 8.3. Раздел 5.14.3. Регистры накопления».

5.4.2 Отчёт ПрошедшиеЗанятия

Создайте ещё один отчёт. Он будет использовать данные регистра накопления. Поэтому так и назовите его — *ПрошедшиеЗанятия*.

Откройте его схему компоновки данных, добавьте набор данных — запрос и вызовите конструктор запроса.

Раскройте ветки регистров сведений и регистров накопления (рисунок 5.80).

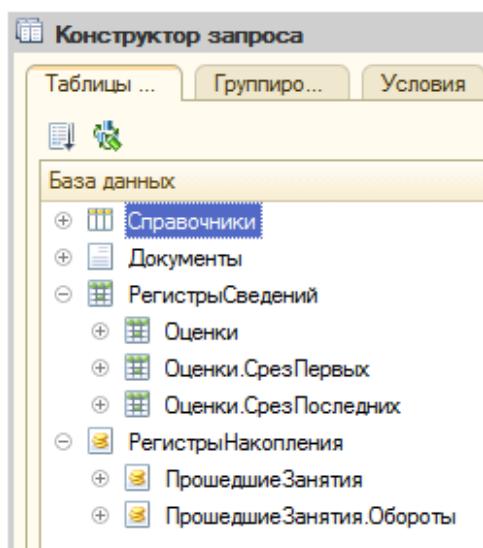


Рисунок 5.80. Таблицы регистров в конструкторе запроса

Здесь хорошо видны различия между этими регистрами. Регистр сведений *Оценки* предоставляет три таблицы. Первой из них, *Оценки*, вы уже пользовались. В ней находятся движения в том виде, в котором вы их записывали в регистр. Две другие таблицы позволяют выбрать самые последние или, наоборот, самые первые оценки на какую-нибудь дату. Для ваших задач такая возможность не нужна, поэтому вы не рассматривали эти таблицы. Грубо говоря, можно считать, что регистр сведений предоставляет вам только таблицу, в которой содержатся движения.

А у обратного регистра накопления две таблицы. В первой, *ПрошедшиеЗанятия*, тоже находятся движения в том виде, как вы их записывали. А вот во второй таблице как раз находятся обороты, то есть те самые «изменения» учитываемых ресурсов, ради которых вы всё это затевали.

Выберите двойным щелчком таблицу *ПрошедшиеЗанятияОбороты* и раскройте её в поле *Таблицы* (рисунок 5.81).

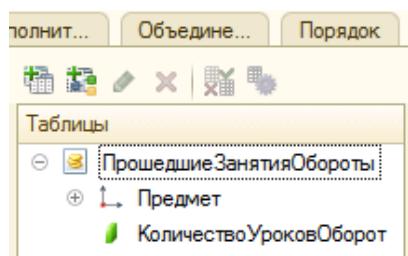


Рисунок 5.81. Доступные поля таблицы оборотов

Сейчас в этой таблице вам доступны поле измерения *Предмет* и обороты, посчитанные по полю *КоличествоУроков*. Но если взять их в таком виде, то вы узнаете только общее количество прошедших занятий за всё время. А хотелось бы видеть его в разбивке по дням.

Регистр накопления предоставляет такую возможность, но её нужно включить. Выделите таблицу *ПрошедшиеЗанятияОбороты* в поле *Таблицы* и нажмите кнопку *Параметры виртуальной таблицы* (рисунок 5.82).

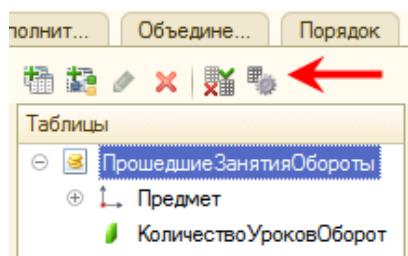


Рисунок 5.82. Параметры виртуальной таблицы

Здесь для поля *Периодичность* выберите значение *Регистратор* (рисунок 5.83).

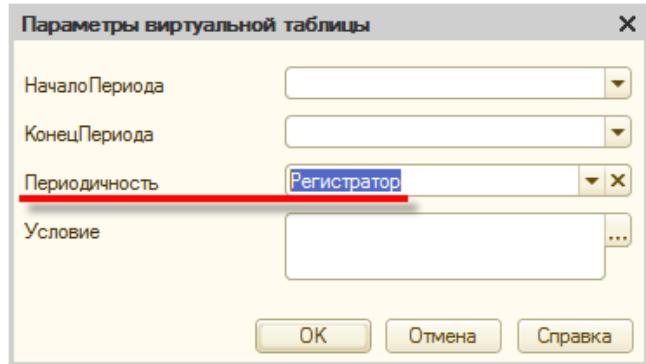


Рисунок 5.83. Периодичность по регистратору

Это значит, что обороты в отчёте вы сможете посмотреть с точностью до регистратора.

После того как вы сделаете такую настройку и нажмёте *OK*, в виртуальной таблице вам станут доступны ещё два поля: *Период* и *Регистратор* (рисунок 5.84).

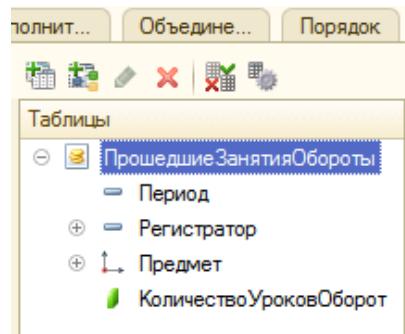


Рисунок 5.84. Поля «Период» и «Регистратор»

Теперь выберите поля *Предмет*, *КоличествоУроковОборот* и *Регистратор* (рисунок 5.85).

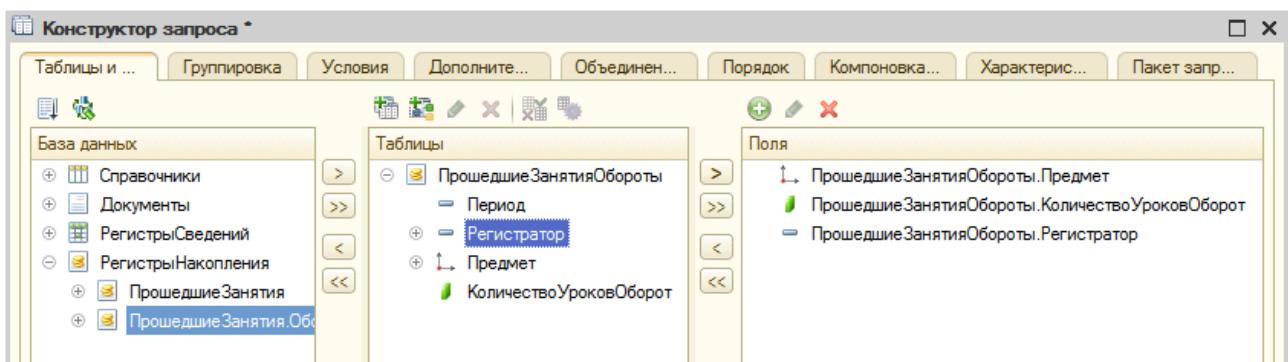


Рисунок 5.85. Выбранные поля

На этом создание запроса закончено, нажмите *OK*. В конструкторе схемы компоновки данных вы увидите запрос, который получился, и поля системы компоновки, которые заполнились автоматически (рисунок 5.86).

Запрос:

```

ВЫБРАТЬ
    ПрошедшиеЗанятияОбороты.Предмет,
    ПрошедшиеЗанятияОбороты.КоличествоУроковОборот,
    ПрошедшиеЗанятияОбороты.Регистратор
ИЗ
    РегистрНакопления.ПрошедшиеЗанятия.Обороты(, , Регистратор, ) КАК ПрошедшиеЗанятияОбороты

```

Автозаполнение

Рисунок 5.86. Текст запроса и поля компоновки данных

Чтобы вам было проще конструировать отчёт и чтобы он выглядел понятно для пользователя, измените имена некоторых полей. Поле *Регистратор* назовите *УчебныйДень*, а поле *КоличествоУроковОборот* назовите просто *КоличествоУроков* (рисунок 5.87).

Рисунок 5.87. Новые имена полей

Теперь нужно создать ресурсы. Перейдите на закладку *Ресурсы* и выберите поле *КоличествоУроков*. Платформа автоматически использует функцию суммирования, что вам и нужно (рисунок 5.88).

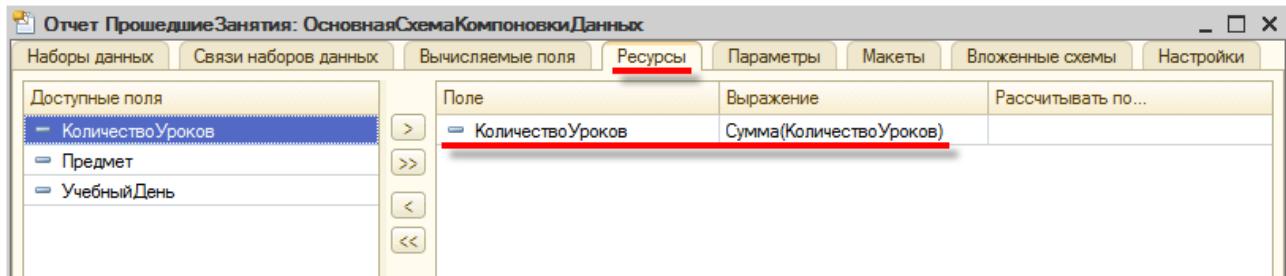


Рисунок 5.88. Ресурс «КоличествоУроков»

После этого переместитесь на закладку *Настройки*, чтобы создать структуру отчёта.

На закладке *Выбранные поля* выберите УчебныйДень, Предмет и КоличествоУроков (рисунок 5.89).

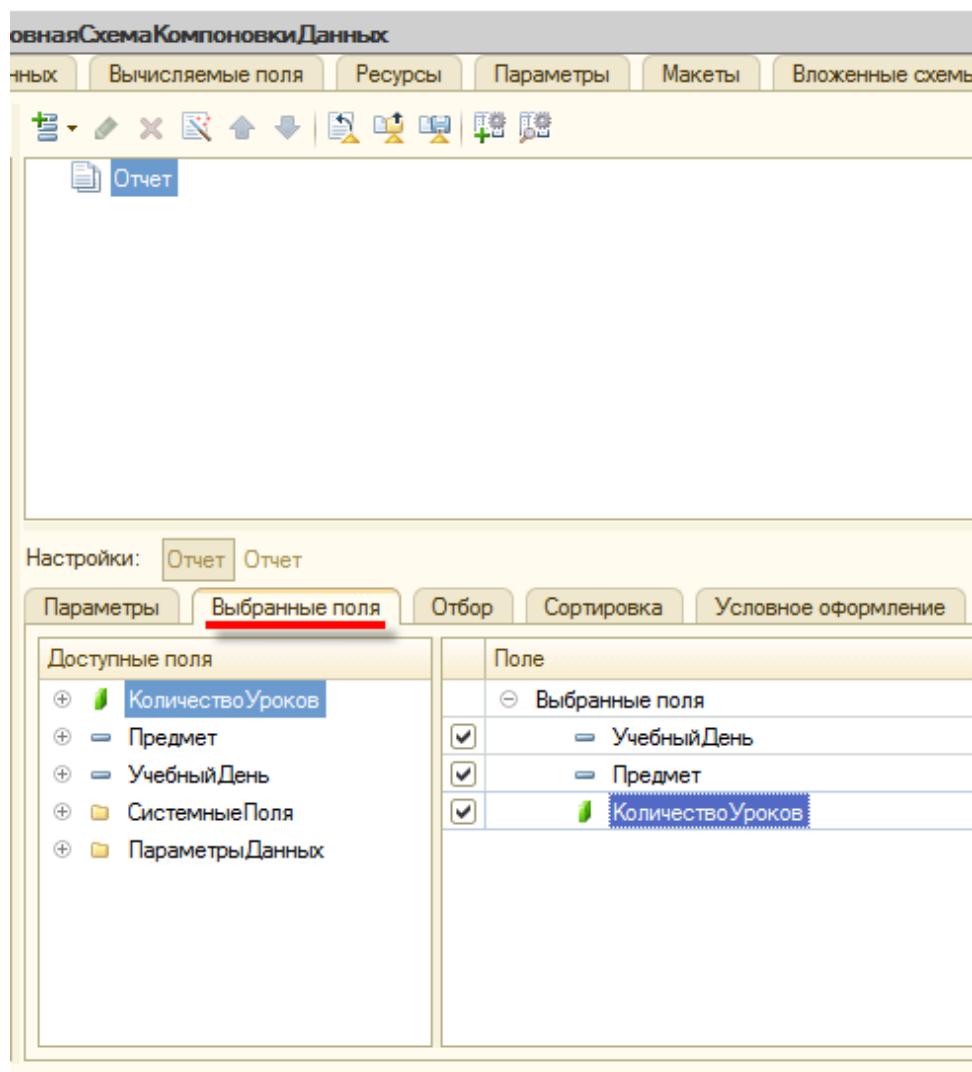


Рисунок 5.89. Выбранные поля

В этом отчёте у вас тоже будут диаграмма и таблица. Сначала добавьте диаграмму. В точках у неё будет группировка Предмет. Серии не задавайте, их не будет (рисунок 5.90).

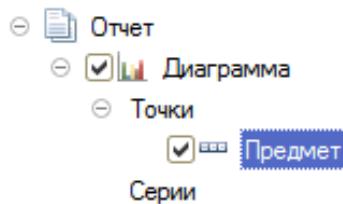


Рисунок 5.90. Диаграмма

Под диаграммой добавьте группировку *Предмет*, а в неё — группировку *УчебныйДень*. Детальные записи вы здесь использовать не будете (рисунок 5.91).

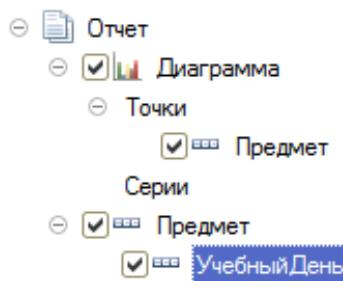


Рисунок 5.91. Группировки «Предмет» и «УчебныйДень»

Хочется, чтобы предметы в нижней таблице располагались не в алфавитном порядке, а так, чтобы сначала шли предметы, по которым проведено наибольшее количество занятий. Для этого нужно установить сортировку для группировки *Предмет*.

Выделите эту группировку в структуре, внизу отметьте, что вас интересуют настройки только для этой группировки, и перейдите на закладку *Сортировка* (рисунок 5.92).

Рисунок 5.92. Сортировка для группировки «Предмет»

Удалите поле *<Авто>* и добавьте вместо него поле *КоличествоУроков*. А направление сортировки укажите *По убыванию* (рисунок 5.93).

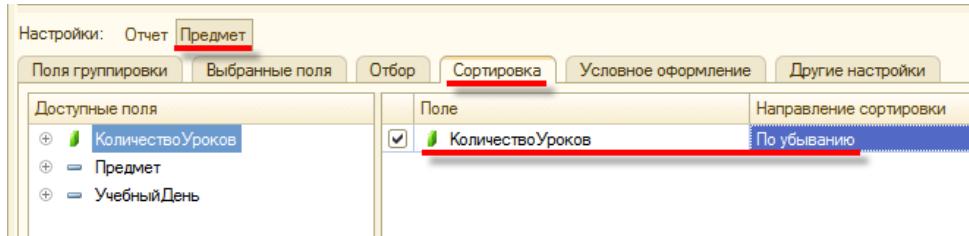


Рисунок 5.93. Сортировка по полю «КоличествоУроков»

Теперь последний штрих. Чтобы таблица не была чёрно-белой, а выглядела красиво, выберите макет её оформления. Для этого, оставаясь в настройках группировки *Предмет*, перейдите на закладку *Другие настройки*. На ней выберите макет оформления — *Море* (рисунок 5.94).

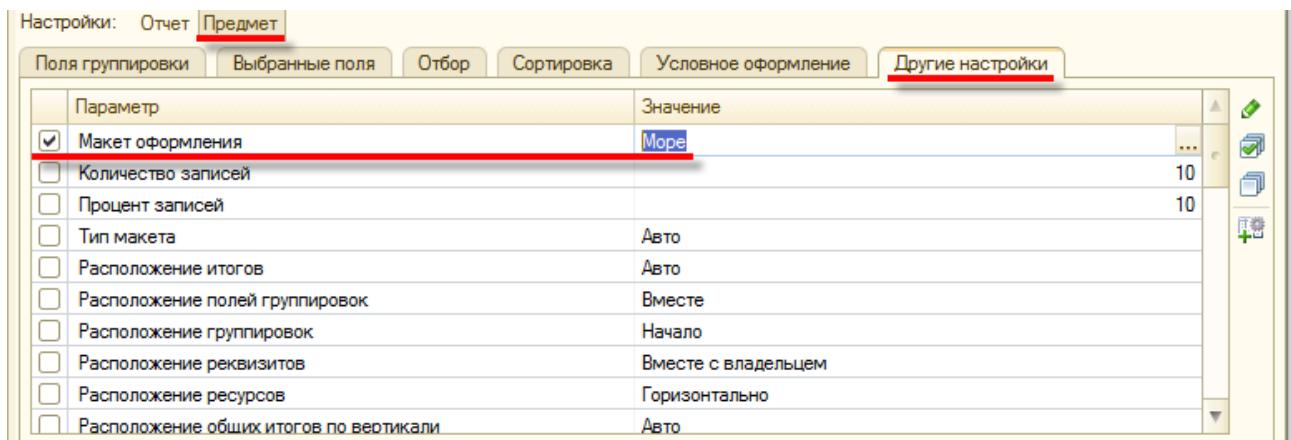


Рисунок 5.94. Макет оформления «Море»

Всё, отчёт готов.

Запустите 1С:Предприятие в режиме отладки, откройте отчёт *Прошедшие занятия* и выполните его. С помощью контекстного меню отобразите только первый уровень группировок (рисунок 5.95).

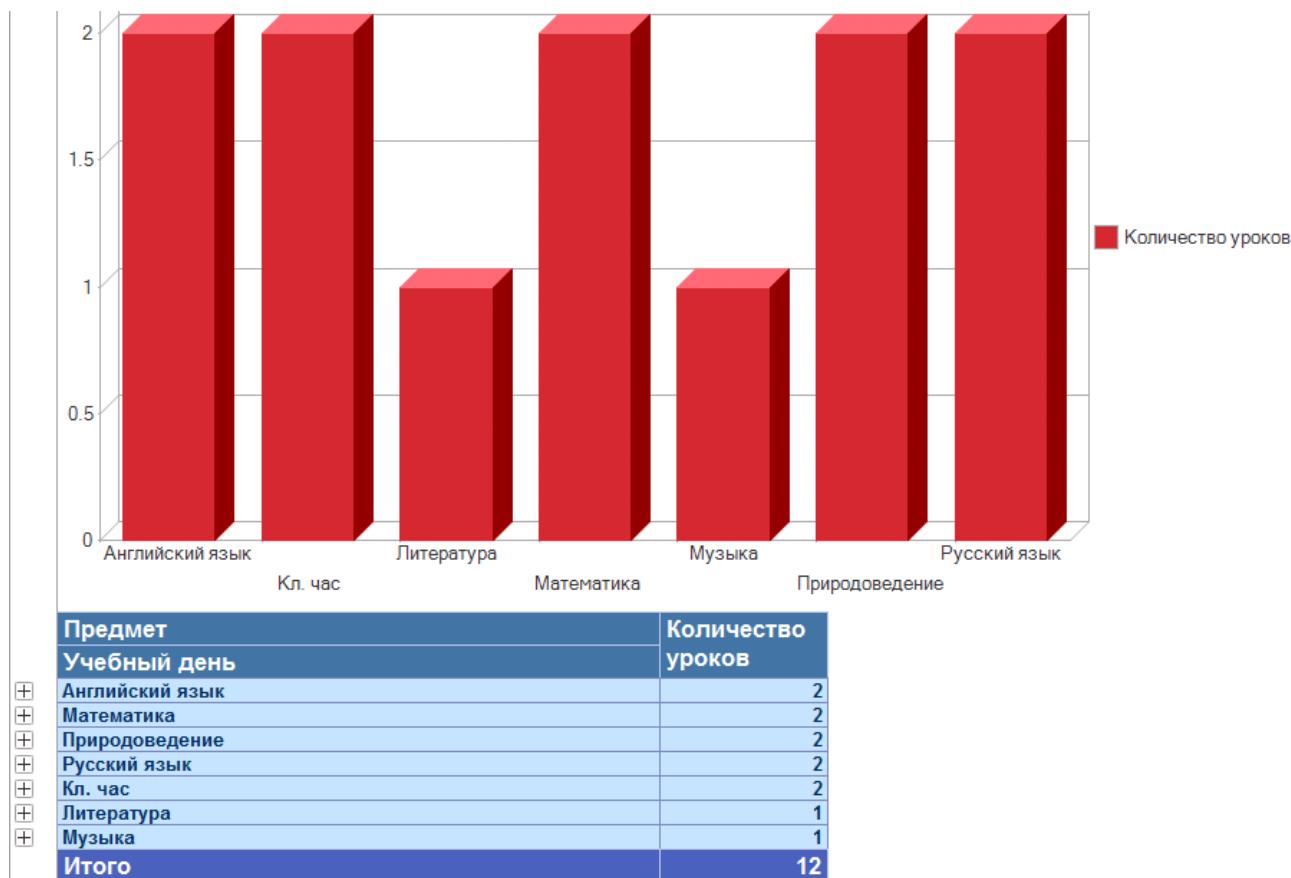


Рисунок 5.95. Отчёт «Прошедшие занятия»

Что вы видите? Наверху, в диаграмме, в алфавитном порядке расположены предметы. И для каждого предмета показано количество занятий за всё время.

В нижней таблице находятся предметы в порядке убывания количества занятий. Если вы раскроете любой из предметов, то увидите учебные дни, в которые проходило занятие по этому предмету.

Это хорошо, но хотелось бы иметь возможность анализировать количество занятий не за всё время, а за какой-то выбранный промежуток. Например, за неделю или за месяц. Это можно легко сделать.

Вернитесь в конфигуратор. В схеме компоновки данных выделите корень структуры, Отчёт, и внизу откройте закладку Параметры (рисунок 5.96).

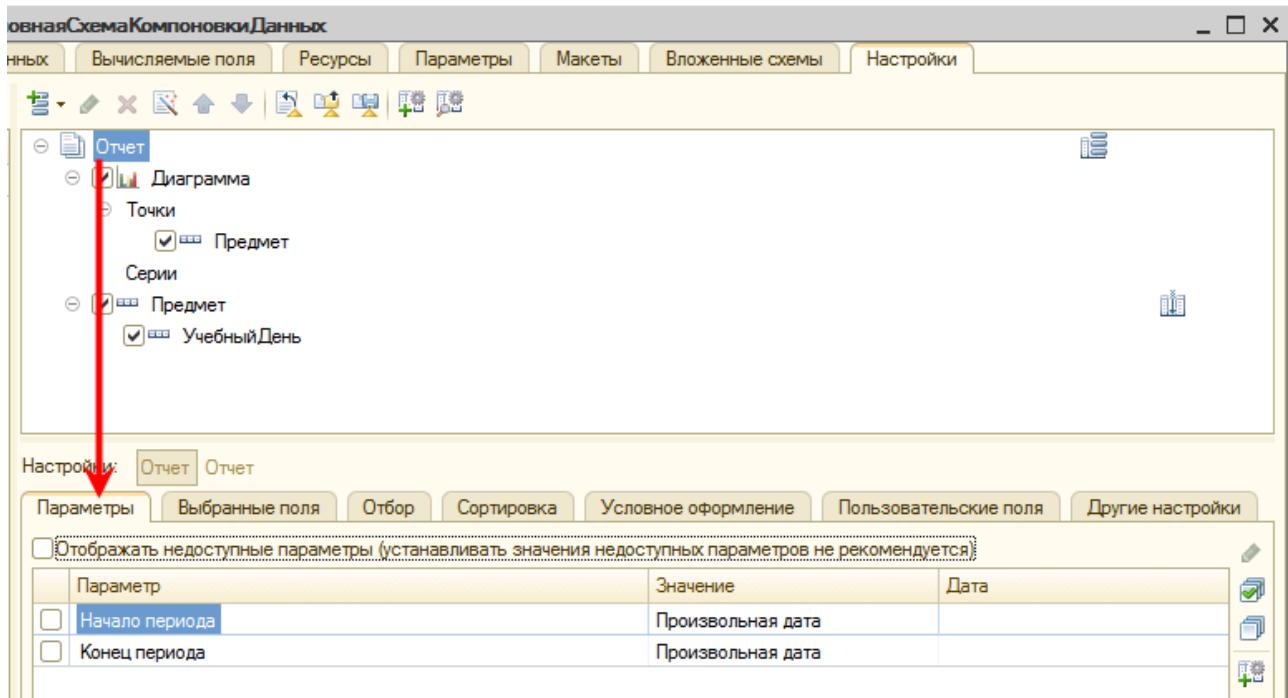


Рисунок 5.96. Параметры

Когда вы делали отчёт на основе регистра сведений, такой закладки не было. Но сейчас вы используете регистр накопления. Он не просто хранит данные, но и некоторым образом обрабатывает их. Именно поэтому вы можете легко выбрать периодичность, с которой вы хотите видеть данные. Или, как на этой закладке, установить границы периода, в рамках которого нужно анализировать данные.

Параметры начала и конца периода предоставляются самим регистром, и стандартно они отключены, то есть не используются в отчёте.

Вам нужно просто их включить, установив флажки перед их названиями.

А после этого, чтобы можно было быстро и удобно их изменять, добавить их в *пользовательские настройки* (рисунок 5.97).

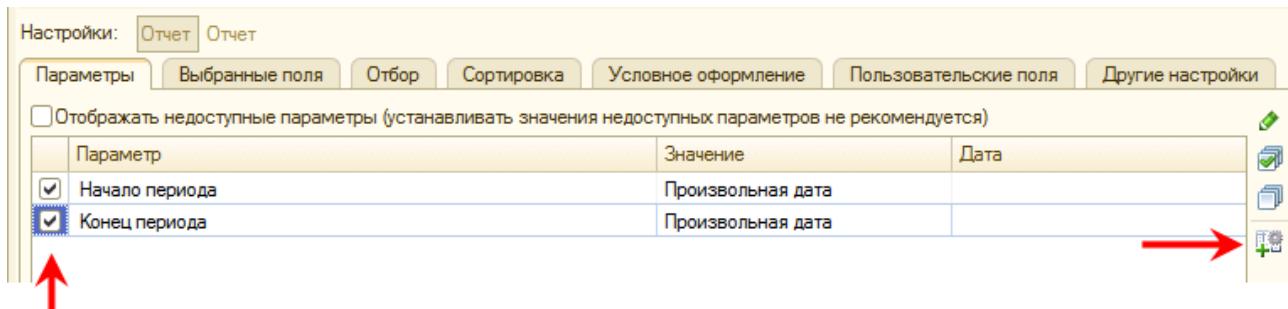


Рисунок 5.97. Свойства элемента пользовательских настроек

Добавление в пользовательские настройки нужно выполнять для каждого параметра отдельно. В диалоге включите флажок *Включать в пользовательские настройки*. А тот режим редактирования, который установлен стандартно, *Быстрый доступ*, как раз означает, что эти параметры будут показаны прямо в форме отчёта (рисунок 5.98).

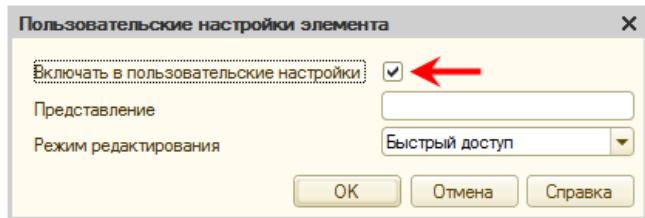


Рисунок 5.98. Пользовательские настройки

Теперь снова запустите 1С:Предприятие в режиме отладки и вызовите отчёт (рисунок 5.99).

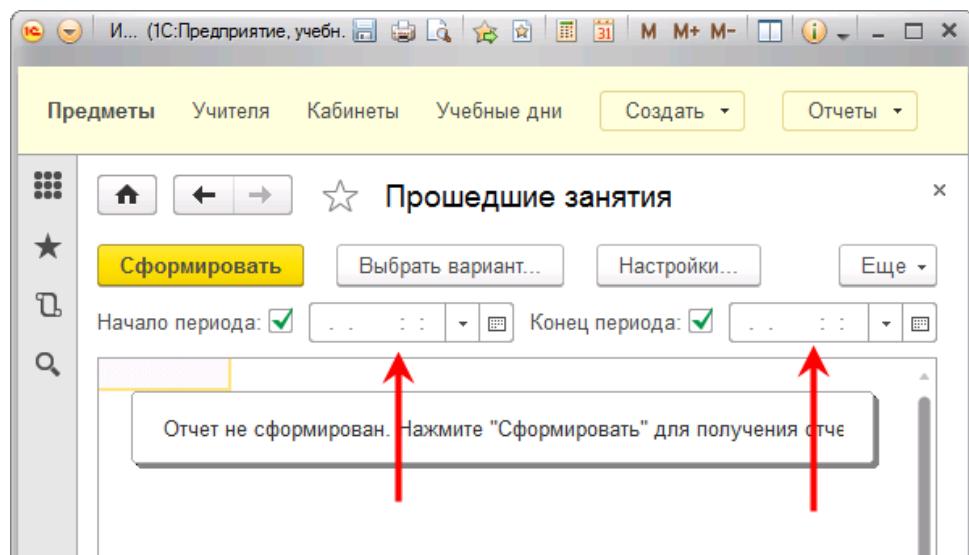


Рисунок 5.99. Параметры в форме отчёта

В его верхней части вы увидите оба параметра: *Начало периода* и *Конец периода*.

С помощью кнопки выбора из списка вы можете выбрать одно из стандартных значений, которые предоставляет вам платформа. Это удобно, потому что не нужно вводить числа (рисунок 5.100).

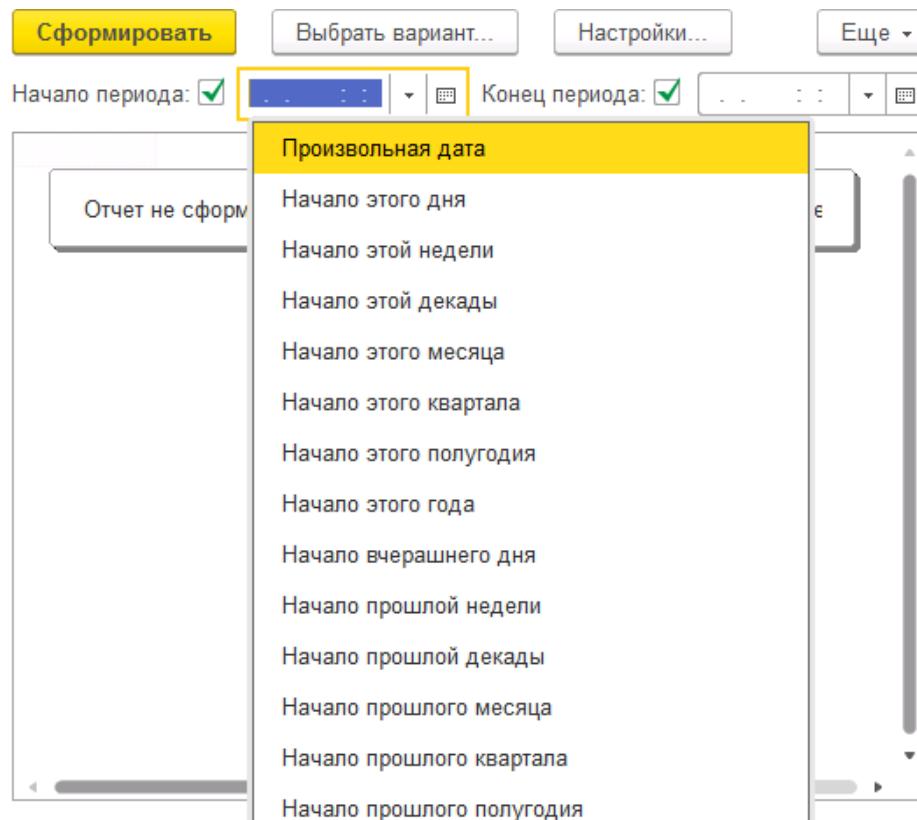


Рисунок 5.100. Стандартные даты, предоставляемые платформой

Но сейчас в вашей базе данные всего за два дня. Поэтому воспользуйтесь соседней кнопкой выбора и в качестве начала периода выберите 1 сентября, а в качестве конца периода — 2 сентября. Вы увидите данные только за первое сентября (рисунок 5.101).

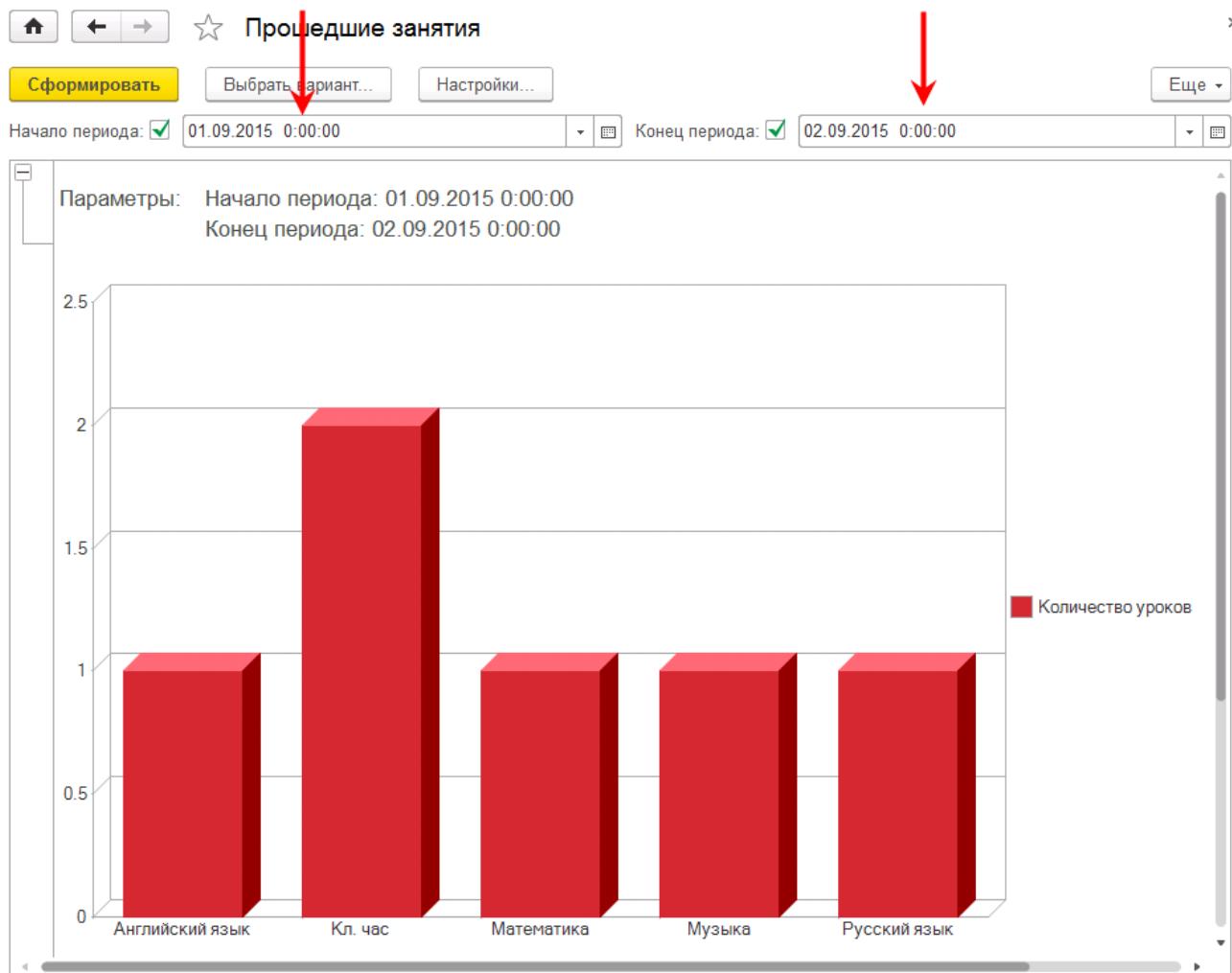


Рисунок 5.101. Данные за первое сентября

Обратите внимание, что, когда вы выбираете дату из календаря, платформа автоматически добавляет к ней время — 0:00:00. Поэтому, чтобы увидеть данные за день, во втором поле вы выбрали следующую дату.

Но если во втором поле вам обязательно хочется указать тоже первое сентября, тогда, после выбора даты, вручную нужно установить время 23:59:59 (рисунок 5.102).

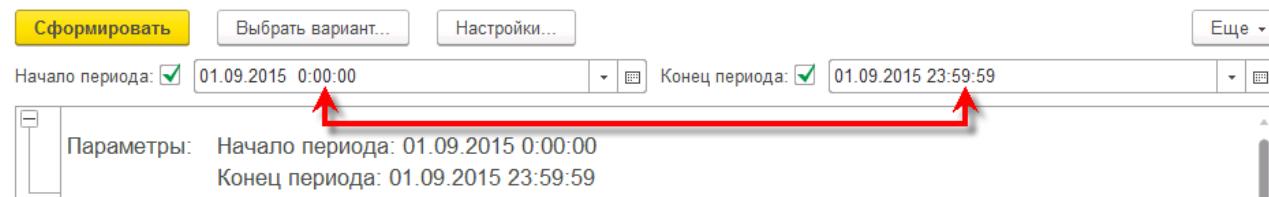


Рисунок 5.102. Установка даты и времени

Совет

Если вы хотите изучить систему компоновки данных подробнее, научиться создавать сложные отчёты, вам поможет книга Е. Ю. Хрусталевой «Разработка сложных отчётов в 1С:Предприятии 8.2. Система компоновки данных. Издание 2» (информация о книге).



Рисунок 5.103. Разработка сложных отчётов в 1С:Предприятии 8.2

Информационная база

Прикладное решение, которое должно получиться у вас к этому моменту, содержится в демонстрационной базе «07 ОтчетПрошедшиеЗанятия.dt». Как её подключить, написано в разделе [A.1 «Как подключить демонстрационную базу»](#) на странице [543](#).

5.5 Работа с регистрами из встроенного языка

Теперь, когда вы создали два отчёта, вы уже достаточно хорошо ориентируетесь в том, какие данные содержатся в регистрах и как они используются.

Настало время познакомиться с регистрами совсем близко, научиться записывать в них данные и читать их самостоятельно, без помощи платформы.

5.5.1 Необъектные данные

Сначала вспомните то занятие, когда вы только начинали учиться работать с документами из встроенного языка. Тогда я говорил о том, что есть *объектные данные* и *необъектные данные*.

Напомню. Объектные данные вы сравнивали с гостями, которых вы пригласили на свой день рождения (рисунок 5.104).



Рисунок 5.104. Гости — объектные данные

В этом случае для вас важно, чтобы к вам в гости пришёл именно Серёжа Соколов, который учится и работает вместе с вами. А не какой-то его тёзка и однофамилец. Который, может быть, и одет так же, и внешне на него похож.

Каждый из таких гостей уникален и важен сам по себе. Даже если он оделся в другую одежду и сильно вырос за то время, что вы его не видели. Всё равно он остался Серёжей Соколовым, а не стал кем-то другим.

Необъектные данные вы сравнивали с людьми в автобусе. Вам совершенно не важно, какие именно люди находятся в автобусе, как их зовут. Не важно, кто именно стоит в проходе с большими сумками, Сергей Николаевич или Марья Ивановна. Важно только то, что проход занят и дальше пройти нельзя (рисунок 5.105).

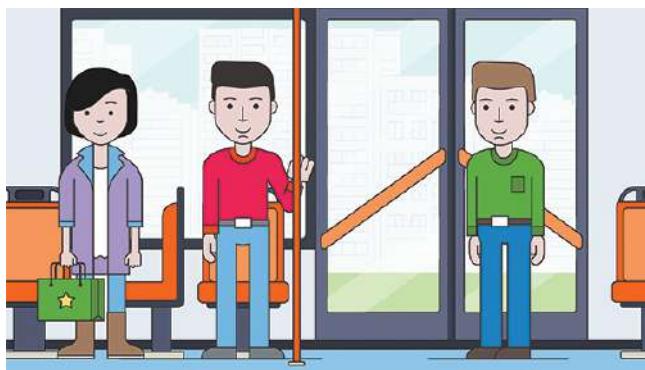


Рисунок 5.105. Пассажиры — необъектные данные

Как вы работаете с объектными данными? У каждого из них есть ссылка. По этой ссылке вы получаете объект. Изменяете его реквизиты. Записываете его в базу данных.

То есть с каждым объектом вы «возитесь» индивидуально. Каждому гостю, который пришёл к вам, вы уделяете внимание, называете его по имени, говорите: «Здравствуй, Серёжа Соколов, дорогой! Проходи, садись!»

А в автобусе вы никого не называете по имени. Если проход занят и вам не хватает места, вы обращаетесь сразу ко всем, кто стоит в проходе: «Граждане, стоящие в проходе, пройдите дальше в салон».

То же самое с необъектными данными. Когда вам нужно с ними что-то сделать, вы не работаете с каждой записью отдельно, а берёте сразу несколько записей и изменяете их. Вот эти несколько записей называются *набор записей*. А чтобы платформа знала, какие именно записи нужно взять, у каждого набора записей есть *отбор*.

Отбор — это условие, по которому нужно отобрать записи. Например, «граждане, стоящие в проходе», — это отбор. С таким отбором получится один набор записей. Он будет состоять из всех людей, которые стоят в проходе.

А «женщины, сидящие на первом сиденье», — это тоже отбор. С таким отбором получится уже другой набор записей. Он будет состоять из двух женщин.

Во встроенном языке, когда вы хотите прочитать необъектные данные, нужно создать набор записей, установить ему какой-то отбор и прочитать этот набор записей. Платформа заполнит набор записей данными из базы данных.

Когда вы хотите записать необъектные данные, тоже нужно создать набор записей и установить ему отбор. После этого заполнить набор теми записями, которые вам нужны, и записать этот набор записей. Платформа поместит ваши данные в базу данных.

Прямо сейчас вы всё это и попробуете. Но для того, чтобы было на чём учиться, вы создадите ещё один регистр сведений, в котором будут храниться домашние задания.

Подробнее

Подробнее о работе с необъектными данными вы можете прочитать в статье информационно-технологического сопровождения (ИТС) «[Особенности использования типов, предназначенных для манипулирования необъектными данными](#)».

5.5.2 Регистр сведений ДомашниеЗадания

Перейдите в конфигуратор, добавьте новый регистр сведений и назовите его *ДомашниеЗадания*. Регистр будет подчиняться регистратору и иметь периодичность в пределах дня (рисунок 5.106).

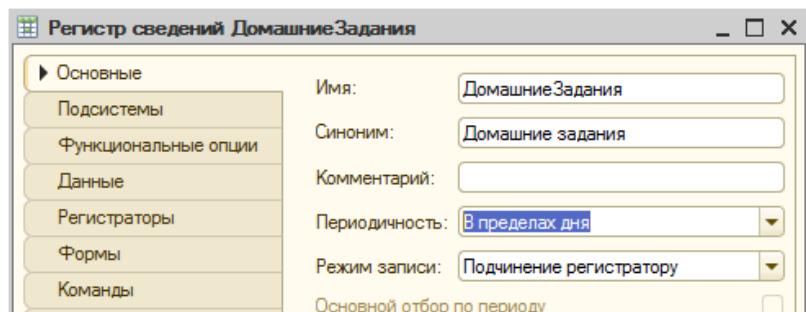


Рисунок 5.106. Регистр «ДомашниеЗадания»

У этого регистра будет одно измерение — *Предмет*. Тип — *СправочникСсылка.Предметы*. Чтобы знать, по какому предмету это домашнее задание.

Кроме этого у регистра будет два ресурса:

- *ДомашнееЗадание*, тип — *Строка, Неограниченная длина*;
- *Выполнено*, тип *Булево*.

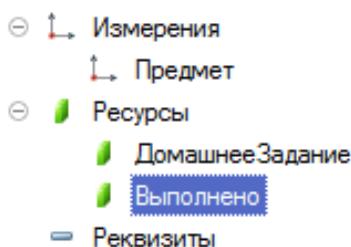


Рисунок 5.107. Измерение и ресурсы регистра

В одном ресурсе вы будете хранить текст домашнего задания. А в другом ресурсе — признак того, выполнено оно или нет. Чтобы в будущем была возможность автоматически составлять список домашних заданий, которые нужно выполнить.

В заключение на закладке *Регистраторы* отметьте, что регистратором будет являться документ *УчебныйДень*.

Вся информация, которая нам понадобится для записи в регистр, находится в документе *УчебныйДень*. Единственное, чего в нём нет, — отметки о том, выполнено домашнее задание или нет.

Поэтому добавьте реквизит *Выполнено* в табличную часть документа и задайте ему тип *Булево* (рисунок 5.108).

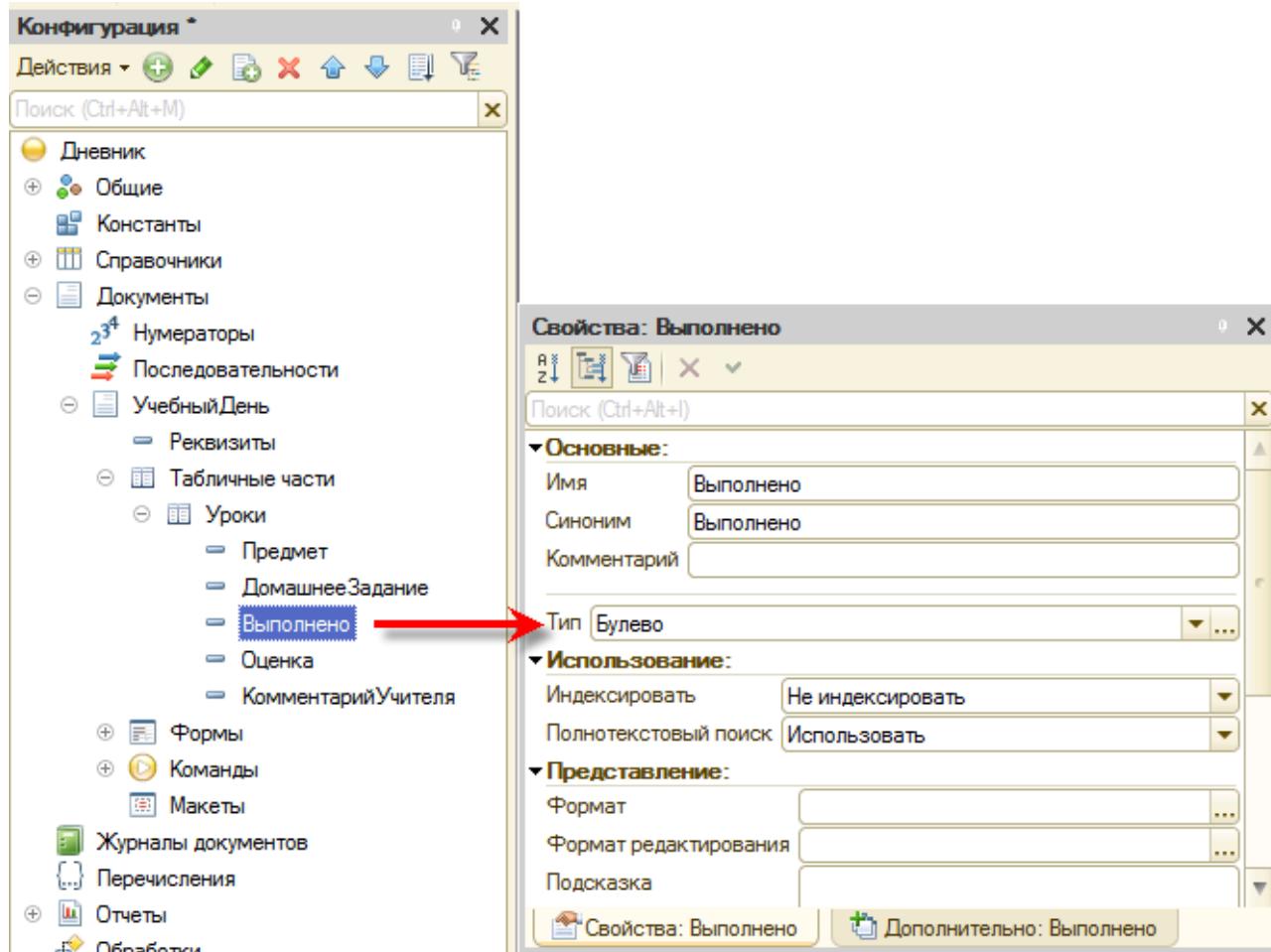


Рисунок 5.108. Реквизит «Выполнено» табличной части документа

И чтобы у вас была возможность проставлять эти отметки, добавьте этот реквизит в форму документа УчебныйДень. Потому что форму документа вы создавали раньше, и естественно, что в ней нет этого реквизита (рисунок 5.109).

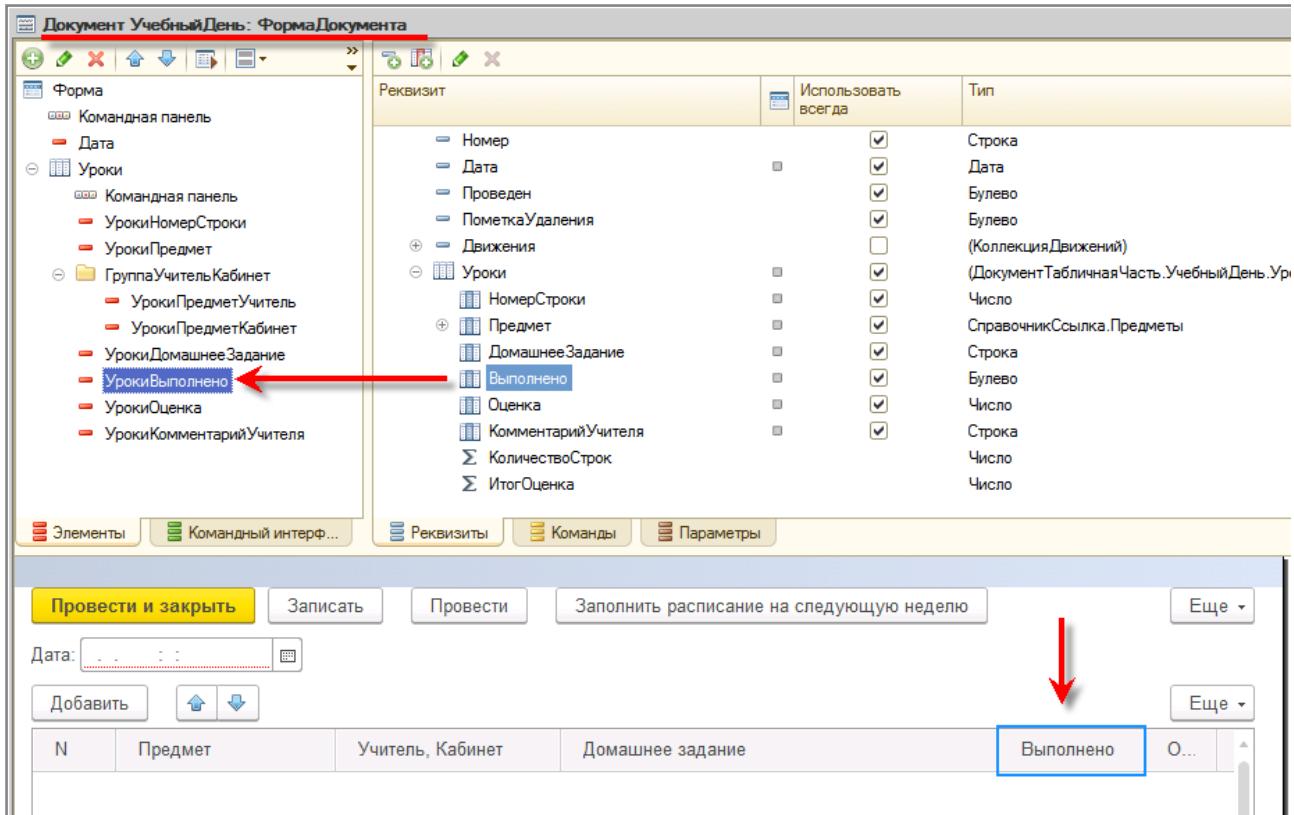


Рисунок 5.109. Добавление реквизита в форму документа

5.5.3 Запись в регистр ДомашниеЗадания

Теперь интересный вопрос. В какой момент информация должна записываться в этот регистр?

До сих пор вы записывали данные в регистр в момент проведения документа. То есть тогда, когда учебный день фактически завершён. Все занятия прошли, все полученные оценки вы занесли в документ.

Но домашние задания нужно записывать гораздо раньше, ещё до того, как начался соответствующий им учебный день. Единственный подходящий момент, который есть для этого, — это запись документа. То есть вы узнали, что вам задали, занесли эту информацию в нужный учебный день и нажали кнопку **Записать**. Вот в этот момент и нужно записать домашние задания в регистр.

Наверное, вы помните, что вы уже интересовались событиями документа. Когда искали место для того, чтобы присвоить новому документу правильный номер. Тогда я говорил о том, что у объекта документа есть два подходящих события: *ПередЗаписью* и *ПриЗаписи* (рисунок 5.110).

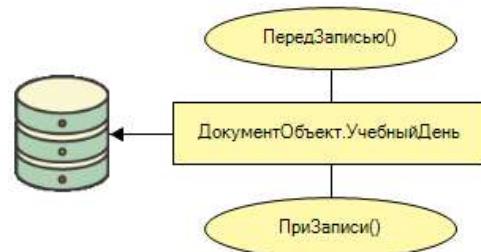


Рисунок 5.110. События «ПередЗаписью» и «ПриЗаписи»

Событие *ПередЗаписью* возникает до того, как данные записываются в базу данных. А событие *ПриЗаписи* возникает уже после того, как данные записаны.

В событии *ПередЗаписью* вы присваиваете документу правильный номер. Это хорошо и правильно, потому что делать это нужно действительно до того, как документ начнёт записываться в базу данных.

А вот записывать домашние задания нужно уже после того, как записан сам документ. Потому что это дополнительная информация, связанная с этим документом. И она не имеет смысла без самого документа.

Это значит, что если вдруг по каким-то причинам не удалось записать документ в базу данных, то и движения в регистрах, связанные с этим документом, тоже не должны быть записаны. Одно без другого не имеет смысла.

Поэтому в событиях, которые выполняются до записи документа в базу, нужно подготовить все данные самого документа, например его номер. А в событиях, которые выполняются после записи документа, нужно подготовить те данные, которые связаны с этим документом и зависят от него, например движения в регистрах (рисунок 5.111).

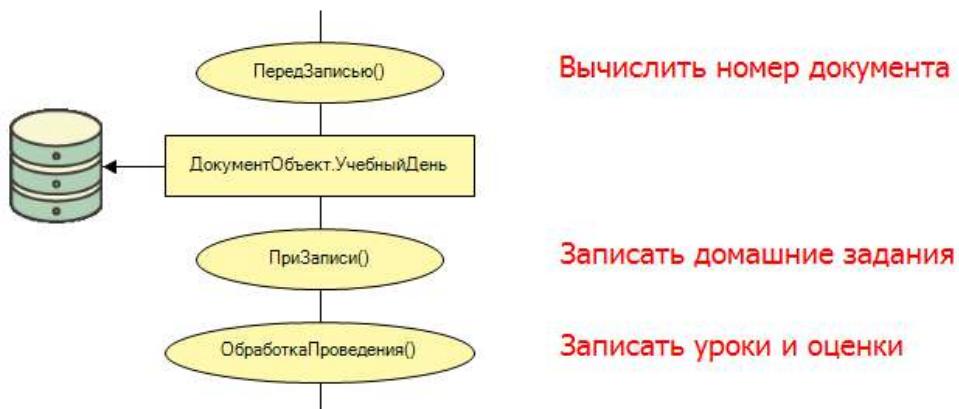


Рисунок 5.111. События, возникающие в транзакции записи документа

На рисунке 5.111 показаны события объекта *ДокументОбъект.<Имя документа>*. Если вы откроете описание любого из этих событий в синтакс-помощнике, то увидите: везде говорится о том, что эти события выполняются в *транзакции* записи. *Транзакция* — это способ, которым изменяются данные в базе данных.

Этот способ прекрасно известен вам из обычной жизни. Просто вы не знаете, что называется он именно так. Представьте, что база данных — это автобус. А семья — это данные. Как поступает семья, если ей нужно куда-то доехать на автобусе (рисунок 5.112)?



Рисунок 5.112. Семья садится в автобус

Родители обязательно проконтролируют, чтобы дети сели в автобус вместе с ними. Не может такого быть, что родители зашли, а дети замешкались и остались на остановке. Наоборот, также не может быть, что дети сели в автобус, а родителям места не хватило и они остались.

Если едут, то все вместе. Если у кого-то не получается, то все не едут и ждут следующего автобуса.

Транзакция работает точно так же. То, что все события на рисунке 5.111 выполняются в одной транзакции, в практическом смысле означает следующее.

Во-первых, не может случиться так, что документ не записался в базу данных, а движения — записались. Если в процессе записи документа возникла какая-то проблема, то и все последующие события выполнены не будут.

Во-вторых, не может быть так, что документ записался в базу данных, а движения — нет. Если проблема возникла в процессе записи движений (уже после того, как документ записан), то все изменения, которые были сделаны в базе данных до этого в этой же транзакции (запись документа), будут отменены.

Другими словами, будет записано либо всё, либо ничего.

Выполнение транзакций обеспечивает база данных. Вам как разработчикам не нужно предпринимать никаких специальных действий для того, чтобы при записи данных началась транзакция. Платформа делает это автоматически. Единственное, что вам нужно, это правильно выбрать событие, в котором записывать данные в регистр.

Поскольку при проведении документа уже поздно записывать домашние задания, вы будете записывать их при записи документа. Откройте модуль документа УчебныйДень и добавьте в него обработчик события ПриЗаписи. Если не помните, как это делать, посмотрите в разделе 3.11.8 «Установка номера для новых документов» на страницах 347–348.

В этом обработчике вам нужно будет сначала создать набор записей, потом установить ему отбор, затем заполнить его нужными записями и в конце записать в базу данных.

Создать набор записей можно с помощью менеджера регистра сведений. А добраться до менеджера регистра сведений можно так же, как и до менеджера документа. Через свойство глобального контекста и указание нужного регистра (листинг 5.6).

Листинг 5.6. Создать набор записей

Процедура ПриЗаписи(Отказ)

```
// Регистр ДомашниеЗадания.  
НаборЗаписей = РегистрыСведений.ДомашниеЗадания.СоздатьНаборЗаписей();
```

КонецПроцедуры

Метод менеджера СоздатьНаборЗаписей() создаст вам пустой набор записей. Об этом вы можете прочитать в синтакс-помощнике (рисунок 5.113).

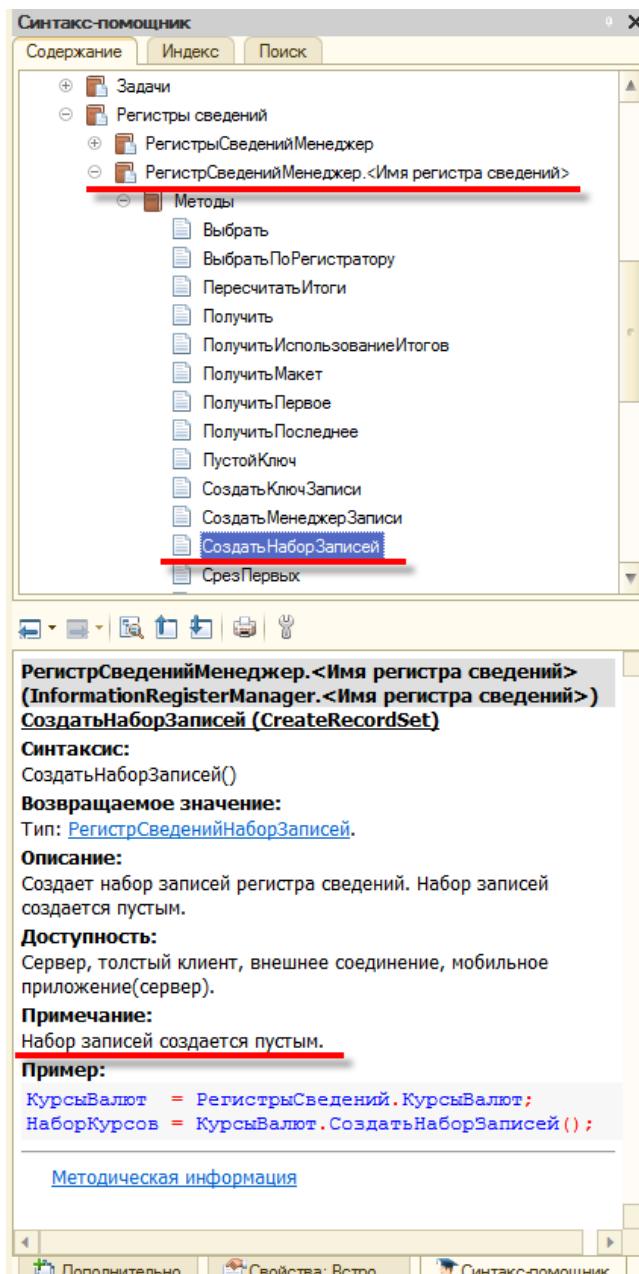


Рисунок 5.113. Метод «СоздатьНаборЗаписей()»

Посмотрите, как выглядит набор записей «вживую». Обновите конфигурацию базы данных, поставьте точку останова в конце процедуры *ПриЗаписи()* и запустите 1С:Предприятие в режиме отладки.

Запишите любой из существующих документов. После того как платформа остановится, посмотрите значение переменной *НаборЗаписей* (рисунок 5.114).

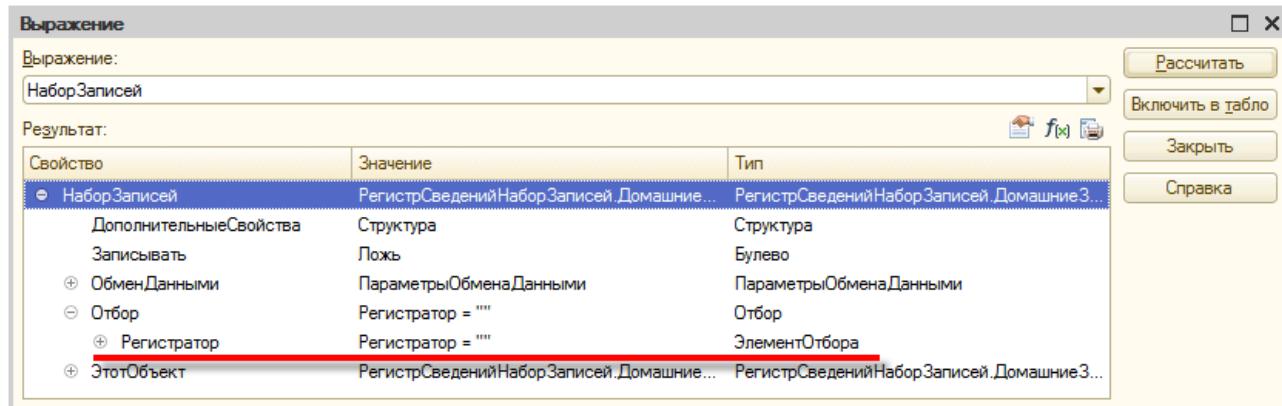


Рисунок 5.114. Набор записей

Здесь вы увидите, что у него есть свойство *Отбор*, которое является коллекцией значений. И в этой коллекции уже есть один элемент с именем *Регистратор*. Этот элемент платформа добавила самостоятельно, потому что знает, что регистр подчиняется регистратору. А раз так, то отбор должен быть установлен по регистратору и никак иначе.

Заметка

Если бы регистр был независимым, то вам нужно было бы самостоятельно выбрать поля, по которым осуществляется отбор. Например, по каким-нибудь его измерениям.

Раз элемент отбора уже есть, вам осталось только установить правильное его значение. Как это сделать, вы можете посмотреть в синтакс-помощнике. Нажав на возвращаемое значение *РегистрСведенийНаборЗаписей* (рисунок 5.111), вы перейдёте к набору записей. Через его свойство *Отбор* — к объекту *Отбор*. Этот объект является коллекцией, состоящей из объектов *ЭлементОтбора*. А у элемента отбора есть метод *Установить()* (рисунок 5.115).

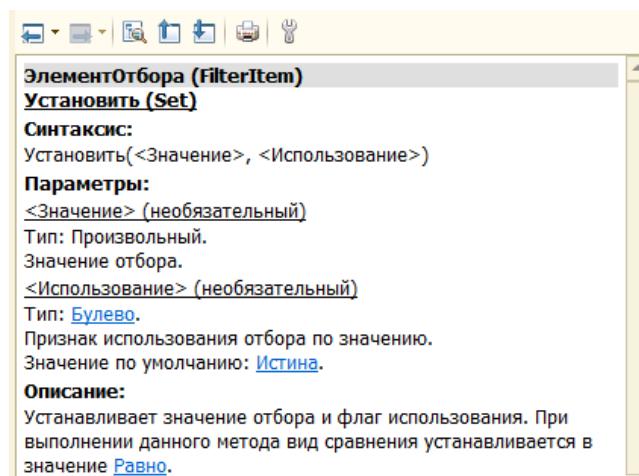


Рисунок 5.115. Метод «Установить()»

Поэтому следующей строчкой вы так и пишете (листинг 5.7).

Листинг 5.7. Установить отбор по регистратору

```
// Регистр ДомашниеЗадания.
НаборЗаписей = РегистрыСведений.ДомашниеЗадания.СоздатьНаборЗаписей();
НаборЗаписей.Отбор.Регистратор.Установить(Ссылка);
```

Здесь Ссылка это свойство документа. Того документа, в котором вы сейчас находитесь.

Теперь нужно обойти табличную часть документа и заполнить набор записями. Как обойти табличную часть, вы знаете. Не забудьте, что записи нужно добавлять только тогда, когда какое-то домашнее задание существует. Если в строке нет домашнего задания, запись добавлять не надо (листинг 5.8).

Листинг 5.8. Обход табличной части документа

```
// Регистр ДомашниеЗадания.  
НаборЗаписей = РегистрыСведений.ДомашниеЗадания.СоздатьНаборЗаписей();  
НаборЗаписей.Отбор.Регистратор.Установить(Ссылка);
```

```
Для Каждого СтрокаТабличнойЧасти Из Уроки Цикл  
Если ЗначениеЗаполнено(СтрокаТАбличнойЧасти.ДомашнееЗадание) Тогда
```

```
КонецЕсли;
```

```
КонецЦикла;
```

Как добавить запись в набор? Набор записей — это тоже коллекция значений, которая состоит из отдельных записей. Если вы посмотрите его описание в синтакс-помощнике то увидите, что у набора записей есть метод *Добавить()*. Он добавит пустую запись (листинг 5.9).

Листинг 5.9. Добавление записи в набор

```
// Регистр ДомашниеЗадания.  
НаборЗаписей = РегистрыСведений.ДомашниеЗадания.СоздатьНаборЗаписей();  
НаборЗаписей.Отбор.Регистратор.Установить(Ссылка);
```

```
Для Каждого СтрокаТабличнойЧасти Из Уроки Цикл  
Если ЗначениеЗаполнено(СтрокаТАбличнойЧасти.ДомашнееЗадание) Тогда  
НоваяЗапись = НаборЗаписей.Добавить();
```

```
КонецЕсли;
```

```
КонецЦикла;
```

Теперь вам нужно заполнить поля этой записи. Какие у неё есть поля, можете посмотреть в синтакс-помощнике или, для разнообразия, прямо в модуле в режиме отладки (рисунок 5.116). Для этого не забудьте поставить точку останова на конец инструкции *Если*. И в том документе, который вы будете записывать в режиме 1С:Предприятие, ввести какое-нибудь домашнее задание по одному из предметов.

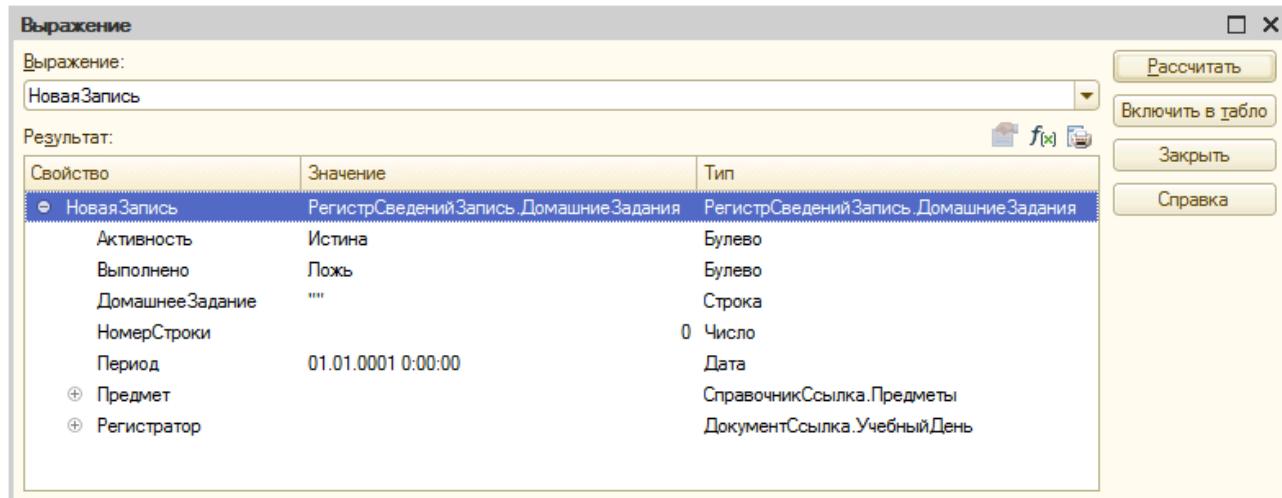


Рисунок 5.116. Поля записи

Поля *Активность* и *НомерСтроки* — «служебные», их платформа заполнит сама. Поля *Регистратор* платформа тоже заполнит сама — она возьмёт то значение, которое вы установили в отборе. *Предмет*, *ДомашнееЗадание* и *Выполнено* — это те данные, которые вы хотите записать в регистр. Остается поле *Период*.

В это поле вам нужно будет записать дату документа. Причём, поскольку у вас не может быть нескольких документов в один день, удобнее будет записать дату начала дня, то есть с нулевым временем.

Так и напишите (листинг 5.10).

Листинг 5.10. Заполнение полей записи

```
// Регистр ДомашниеЗадания.
НаборЗаписей = РегистрыСведений.ДомашниеЗадания.СоздатьНаборЗаписей();
НаборЗаписей.Отбор.Регистратор.Установить(Ссылка);

Для Каждого СтрокаТабличнойЧасти Из Уроки Цикл
Если ЗначениеЗаполнено(СтрокаТабличнойЧасти.ДомашнееЗадание) Тогда
    НоваяЗапись = НаборЗаписей.Добавить();
    НоваяЗапись.Период      = НачалоДня(Дата);
    НоваяЗапись.Предмет     = СтрокаТабличнойЧасти.Предмет;
    НоваяЗапись.ДомашнееЗадание = СтрокаТабличнойЧасти.ДомашнееЗадание;
    НоваяЗапись.Выполнено   = СтрокаТабличнойЧасти.Выполнено;

КонецЕсли;

КонецЦикла;
```

И последнее действие, которое нужно выполнить, это запись набора данных в базу данных. Делается это, как вы догадываетесь, очень просто (листинг 5.11).

Листинг 5.11. Запись набора данных

Процедура ПриЗаписи(Отказ)

```
// Регистр ДомашнеЗадания.
НаборЗаписей = РегистрыСведений.ДомашнеЗадания.СоздатьНаборЗаписей();
НаборЗаписей.Отбор.Регистратор.Установить(Ссылка);

Для Каждого СтрокаТабличнойЧасти Из Уроки Цикл
Если ЗначениеЗаполнено(СтрокаТабличнойЧасти.ДомашнееЗадание) Тогда
    НоваяЗапись = НаборЗаписей.Добавить();
    НоваяЗапись.Период      = НачалоДня(Дата);
    НоваяЗапись.Предмет     = СтрокаТабличнойЧасти.Предмет;
    НоваяЗапись.ДомашнееЗадание = СтрокаТабличнойЧасти.ДомашнееЗадание;
    НоваяЗапись.Выполнено     = СтрокаТабличнойЧасти.Выполнено;

КонецЕсли;

КонецЦикла;

НаборЗаписей.Записать();
```

КонецПроцедуры

Теперь удалите точку останова и запустите 1С:Предприятие в режиме отладки. Добавьте несколько домашних заданий и посмотрите, что запишется в регистр.

Я буду добавлять домашние задания в демонстрационную базу, а вы создайте несколько учебных дней в своей базе на следующую неделю. И добавляйте в них.

В понедельник запишите задания по математике и музыке (рисунок 5.117).

№	Предмет	Учитель, Кабинет	Домашнее задание	Выполнено	Оценка	Комментарий учителя
1	Кл. час	Рыбакова А. Е., 311		<input type="checkbox"/>		
2	Кл. час	Рыбакова А. Е., 311		<input type="checkbox"/>		
3	Английский язык	Николаева Т. Я., 401		<input type="checkbox"/>		
4	Музыка	Евсеева О. Ю., 131	Выучить текст гимна школы.	<input type="checkbox"/>		
5	Математика	Рыбакова А. Е., 311	Задачи 68, 73 и 74.	<input type="checkbox"/>		
6	Русский язык	Авдеева В. М., 409		<input type="checkbox"/>		

Рисунок 5.117. Домашние задания на понедельник

В среду запишите задания по английскому, русскому языку и литературе (рисунок 5.118).

Учебный день 20150909 от 09.09.2015 12:00:00

Провести и закрыть **Записать** **Провести** **Еще**

Дата: 09.09.2015 12:00:00

Добавить **↑** **↓** **Еще**

N	Предмет	Учитель, Кабинет	Домашнее задание	Выполнено	Оценка	Комментарий учителя
1	Математика	Рыбакова А. Е., 311		<input type="checkbox"/>		
2	Английский язык	Николаева Т. Я., 401	Сделать упражнения 4 и 5. Подготовиться к чтению по ролям.	<input type="checkbox"/>		
3	Природоведение	Филиппова Л. В., 220		<input type="checkbox"/>		
4	Природоведение	Филиппова Л. В., 220		<input type="checkbox"/>		
5	Русский язык	Авдеева В. М., 409	Найти и записать 5 пословиц о языке.	<input type="checkbox"/>		
6	Литература	Авдеева В. М., 409	Прочитать статью стр. 3 - 5. Подготовить выразительное чтение песни "Варяг".	<input type="checkbox"/>		

Рисунок 5.118. Домашние задания на среду

А в пятницу запишите домашнее задание по информатике (рисунок 5.119).

Учебный день 20150911 от 11.09.2015 12:00:00

Провести и закрыть **Записать** **Провести** **Еще**

Дата: 11.09.2015 12:00:00

Добавить **↑** **↓** **Еще**

N	Предмет	Учитель, Кабинет	Домашнее задание	Выполнено	Оценка	Комментарий учителя
1	Литература	Авдеева В. М., 409		<input type="checkbox"/>		
2	Математика	Рыбакова А. Е., 311		<input type="checkbox"/>		
3	История	Герасимова Е. С., ...		<input type="checkbox"/>		
4	Литература	Авдеева В. М., 409		<input type="checkbox"/>		
5	Физ. культура	Крюков В. В., 223		<input type="checkbox"/>		
6	Информатика	Давыдова А. В., 418	Стр. 5 - 9 читать. Упражнения 2 - 5.	<input type="checkbox"/>		

Рисунок 5.119. Домашнее задание на пятницу

Теперь посмотрите, что записалось в регистр *Домашние Задания* (рисунок 5.120).

Период	Регистратор	Номер строки	Предмет	Домашнее задание	Выполнено
07.09.2015	Учебный день 20150907 от 07.09.2015 18:15:20	1	Музыка	Выучить текст гимна школы.	
07.09.2015	Учебный день 20150907 от 07.09.2015 18:15:20	2	Математика	Задачи 68, 73 и 74.	
09.09.2015	Учебный день 20150909 от 09.09.2015 12:00:00	1	Английский язык	Сделать упражнения 4 и 5. Подготовиться к ...	
09.09.2015	Учебный день 20150909 от 09.09.2015 12:00:00	3	Литература	Прочитать статью стр. 3 - 5. Подготовить вы...	
09.09.2015	Учебный день 20150909 от 09.09.2015 12:00:00	2	Русский язык	Найти и записать 5 пословиц о языке.	
11.09.2015	Учебный день 20150911 от 11.09.2015 12:00:00	1	Информатика	Стр. 5 - 9 читать. Упражнения 2 - 5.	

Рисунок 5.120. Регистр «Домашние Задания»

Отлично! Это именно то, что вы и хотели!

5.5.4 Работа с регистрами в модуле документа

Теперь вернитесь в конфигуратор, чтобы посмотреть на один очень интересный факт.

Раскройте процедуру проведения и посмотрите, например, на то, как записываются данные в регистр *Прошедшие Занятия*. Сравните это с тем, что написали вы (листинг 5.12).

Листинг 5.12. Разные способы записи движений

Процедура ОбработкаПроведения(Отказ, Режим)

```

Движения.Оценки.Записывать = Истина;
Движения.ПрошедшиеЗанятия.Записывать = Истина;
Для Каждого ТекСтрокаУроки Из Уроки Цикл

    // Регистр Оценки
    Если ЗначениеЗаполнено(ТекСтрокаУроки.Оценка) Тогда
        Движение = Движения.Оценки.Добавить();
        Движение.Период = Дата;
        Движение.Предмет = ТекСтрокаУроки.Предмет;
        Движение.НомерУрока = ТекСтрокаУроки.НомерСтрочки;
        Движение.Оценка = ТекСтрокаУроки.Оценка;

    КонецЕсли;

    // Регистр ПрошедшиеЗанятия
    Движение = Движения.ПрошедшиеЗанятия.Добавить();
    Движение.Период = Дата;
    Движение.Предмет = ТекСтрокаУроки.Предмет;
    Движение.КоличествоУроков = 1;

КонецЦикла;

```

КонецПроцедуры

Процедура ПриЗаписи(Отказ)

```

// Регистр ДомашниеЗадания.
НаборЗаписей = РегистрыСведений.ДомашниеЗадания.СоздатьНаборЗаписей();
НаборЗаписей.Отбор.Регистратор.Установить(Ссылка);

Для Каждого СтрокаТабличнойЧасти Из Уроки Цикл
    Если ЗначениеЗаполнено(СтрокаТабличнойЧасти.ДомашнееЗадание) Тогда
        НоваяЗапись = НаборЗаписей.Добавить();
        НоваяЗапись.Период = НачалоДня(Дата);
        НоваяЗапись.Предмет = СтрокаТабличнойЧасти.Предмет;
        НоваяЗапись.ДомашнееЗадание = СтрокаТабличнойЧасти.ДомашнееЗадание;
        НоваяЗапись.Выполнено = СтрокаТабличнойЧасти.Выполнено;

    КонецЕсли;

КонецЦикла;

НаборЗаписей.Записать();

```

КонецПроцедуры

В обработке проведения, которую вы создавали с помощью конструктора, почему-то не создаются и не записываются никакие наборы данных. Хотя средняя часть алгоритма очень похожа. В обработке проведения добавляется какое-то движение и присваиваются значения его полям.

Здесь всё очень просто. Когда вы находитесь в модуле документа, платформа знает,

по каким регистрам документ может делать движения. Вы указывали это в дереве конфигурации. Поэтому платформа предоставляет вам «сервис», облегчающий вашу работу.

В свойстве документа *Движения* она уже хранит для вас пустые наборы данных для каждого из регистров, по которым вы можете захотеть записывать движения. И в этих наборах данных уже установлен нужный отбор.

Вы можете убедиться в этом сами. Установите точку останова, например, в начале процедуры *ПриЗаписи*. Запустите 1С:Предприятие в режиме отладки и запишите любой документ.

Теперь в конфигураторе откройте вычисление выражения и посмотрите, что находится в свойстве *Движения* (рисунок 5.121).

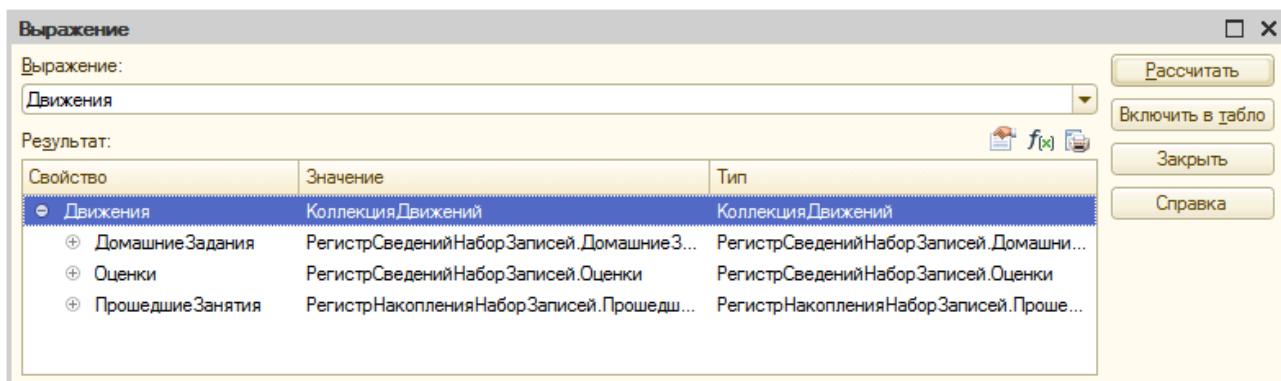


Рисунок 5.121. Наборы записей в свойстве «Движения»

Вы увидите три набора записей, соответствующие трём регистрам, по которым этот документ может делать движения. Причём в каждом из них уже установлен отбор по регистратору. Например, в наборе для регистра *ДомашниеЗадания* (рисунок 5.122).

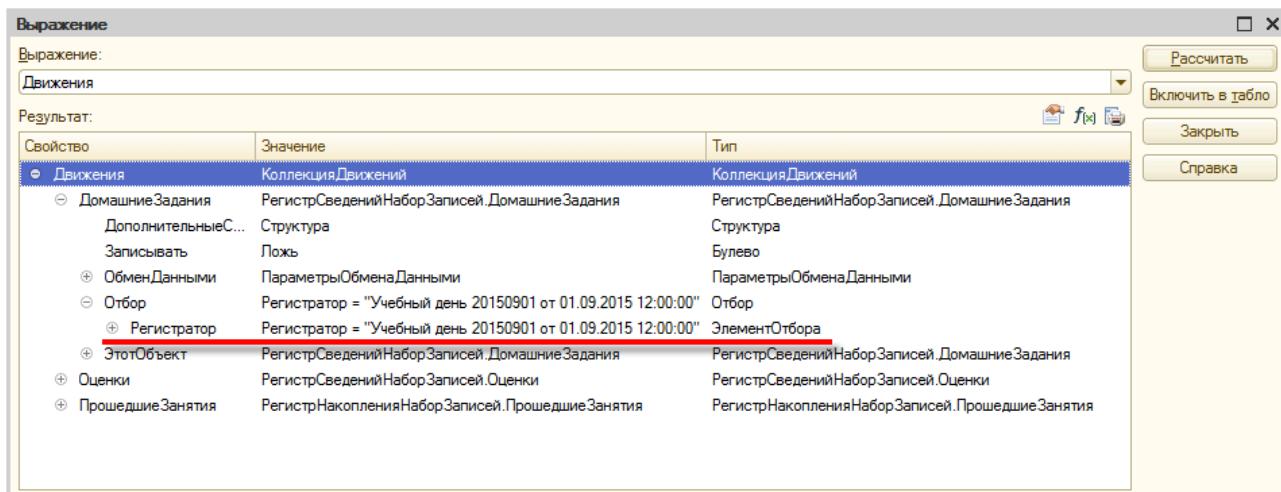


Рисунок 5.122. Установлен отбор по регистратору

Таким образом, если вы используете эти наборы данных, то всё, что вам нужно сделать, это просто добавить в них те записи, которые вам нужны.

К чему я рассказываю всё это? А к тому, что тот алгоритм, который вы написали в процедуре *ПриЗаписи*, вы можете использовать в любом модуле конфигурации. Именно так и записываются данные в регистр.

Но если вы находитесь в модуле документа, вы можете поступить проще и использовать готовые наборы данных, которые есть в свойстве *Движения* (листинг 5.13).

Листинг 5.13. Два способа работы с движениями в модуле документа

```
// Регистр ДомашниеЗадания.
НаборЗаписей = РегистрыСведений.ДомашниеЗадания.СоздатьНаборЗаписей();
НаборЗаписей.Отбор.Регистратор.Установить(Ссылка);

Для Каждого СтрокаТабличнойЧасти Из Уроки Цикл
Если ЗначениеЗаполнено(СтрокаТабличнойЧасти.ДомашнееЗадание) Тогда
    НоваяЗапись = НаборЗаписей.Добавить();
    НоваяЗапись.Период      = НачалоДня(Дата);
    НоваяЗапись.Предмет     = СтрокаТабличнойЧасти.Предмет;
    НоваяЗапись.ДомашнееЗадание = СтрокаТабличнойЧасти.ДомашнееЗадание;
    НоваяЗапись.Выполнено     = СтрокаТабличнойЧасти.Выполнено;

КонецЕсли;

КонецЦикла;

НаборЗаписей.Записать();

// Вариант с использованием коллекции движений документа.
Для Каждого СтрокаТабличнойЧасти Из Уроки Цикл
Если ЗначениеЗаполнено(СтрокаТабличнойЧасти.ДомашнееЗадание) Тогда
    НоваяЗапись = Движения.ДомашниеЗадания.Добавить();
    НоваяЗапись.Период      = НачалоДня(Дата);
    НоваяЗапись.Предмет     = СтрокаТабличнойЧасти.Предмет;
    НоваяЗапись.ДомашнееЗадание = СтрокаТабличнойЧасти.ДомашнееЗадание;
    НоваяЗапись.Выполнено     = СтрокаТабличнойЧасти.Выполнено;

КонецЕсли;

КонецЦикла;

Движения.ДомашниеЗадания .Записать();
```

Оба варианта абсолютно одинаковые, но второй на две строчки короче.

Заметка

Если вы находитесь в процедуре проведения документа, то наборы, содержащиеся в свойстве *Движения*, вам даже не нужно самим записывать. Их запишет платформа. Чтобы она это сделала, в начале процедуры проведения для каждого набора записей есть такая инструкция:

Движения.ПрошедшиеЗанятия.Записывать = Истина;

Почему сделано именно так — это сложная тема. Она совсем не для этой книги. Поэтому просто запомните, что такая строка необходима, если вы работаете с наборами, которые содержатся в свойстве *Движения*.

Информационная база

Прикладное решение, которое должно получиться у вас к этому моменту, содержится в демонстрационной базе «08 РегистрДомашниеЗадания.dt». Как её подключить, написано в разделе А.1 «Как подключить демонстрационную базу» на странице 543.

Глава 6

Язык запросов

Не читайте эту главу!

Если вы знаете:

- почему встроенного языка недостаточно;
- почему вместо ссылки видно наименование;
- где хранится табличная часть;
- зачем нужна консоль запросов;
- как выбрать все уроки, которые проводятся на втором этаже;

смело переходите к главе [7 «Планировщик»](#) на странице [472](#).

Информационная база

Прикладное решение, которое должно получиться у вас к этому моменту, содержится в демонстрационной базе «08 РегистрДомашниеЗадания.dt». Как её подключить, написано в разделе [A.1 «Как подключить демонстрационную базу»](#) на странице [543](#).

В предыдущей главе вы слегка отдалились от мелких деталей системы и освоили несколько крупных «веток» дерева конфигурации: регистры и отчёты.

Сейчас вы снова углубитесь в самую «гущу» 1С:Предприятия. Аналогично тому, как вы это делали при изучении встроенного языка (рисунок [6.1](#)).



Рисунок 6.1. Сейчас вы здесь

Только теперь вы познакомитесь с другим языком, который есть в системе. Этот язык

называется **язык запросов**.

6.1 Чем язык запросов отличается от встроенного языка

Дело в том, что система 1С:Предприятие поддерживает два способа доступа к данным, хранящимся в базе данных: *объектный* и *табличный*.

Объектный означает, что доступ к данным осуществляется как к объектам. Как к объектам встроенного языка. С этим способом вы уже знакомы. Используя этот способ, вы обращаетесь к некоторой совокупности данных, находящихся в базе данных, как к единому целому.

Например, если вы хотите знать, какие занятия проходят в некоторый день, вы обращаетесь к объекту документа. И он получает из базы все данные, которые относятся к этому документу. Может быть, все данные вам и не нужны, но объект документа получит их все. Вы сможете их посмотреть, сможете изменить. Сможете записать обратно в базу данных. При этом произойдёт всё то же самое, что и при интерактивной работе пользователя. Будут вызваны все события, отработают все назначенные вами обработчики. Об этом я рассказывал в разделе 3.11.7 «**События объектов**» на странице 344.

Но что, если вам нужны не все данные документа, а только некоторые? Например, вам нужно знать только то, какой урок первый в этот день. Или даже не в этот день, а для всех дней на этой неделе?

Такие данные вы сможете получить с помощью языка запросов. Язык запросов как раз реализует второй способ работы с данными — табличный. Он позволяет обращаться к отдельным полям таблиц независимо от того, каким объектам принадлежат эти данные.

В результате вы будете иметь только некоторые части данных от многих объектов. Но именно те части, которые вам нужны. Например, только первые уроки, но от многих документов. По этой причине с помощью языка запросов вы можете только читать те данные, которые есть в базе данных. Изменить и записать их вы не можете.

Зато язык запросов очень хорошо подходит для получения информации из базы данных по сложным условиям: отбор, группировка, сортировка, объединение нескольких выборок в одну, расчёт итогов и тому подобное.

Примечание

Если вам нужно записывать новые данные, изменять существующие данные, удалять данные, вы делаете это с помощью встроенного языка. Если нужно только читать данные, причём по разным сложным условиям и максимально быстро, вы используете язык запросов.

6.2 Хранение объектных данных

На примере регистра сведений и регистра накопления вы уже знаете, что данные 1С: Предприятия хранятся в базе данных в таблицах.

Самое интересное, что таким образом хранятся вообще все данные 1С:Предприятия. И объектные, и необъектные. То есть информация, содержащаяся в документах и справочниках, тоже хранится в таблицах. Посмотрите, как они устроены.

Например, справочник Учителя хранится в одной таблице. Каждому учителю в этой таблице соответствует одна запись (рисунок 6.2).

Ссылка	Код	Наименование	...
Ссылка1	000000007	Авдеева В. М.	...
Ссылка2	000000005	Герасимова Е. С.	...
Ссылка3	000000004	Давыдова А. В.	...
...

Рисунок 6.2. Справочник «Учителя»

У каждого элемента справочника есть уникальная ссылка. Это значение хранится в отдельном поле. Для простоты эти значения тут обозначены словом *Ссылка* и номером.

Поля этой таблицы соответствуют тем реквизитам (стандартным и созданным вами), которые есть у справочника (рисунок 6.3).

Ссылка	Код	Наименование	...
Ссылка1	000000007	Авдеева В. М.	...
Ссылка2	000000005	Герасимова Е. С.	...
Ссылка3	000000004	Давыдова А. В.	...
...

Стандартные реквизиты:

- Ссылка
- Код
- Наименование
 - Владелец
 - Родитель
 - Это Группа
- Пометка Удаления
- Предопределенный
- Имя Предопределенных Данных

Рисунок 6.3. Набор полей соответствует реквизитам

Справочник *Предметы* тоже хранится в одной таблице. Он выглядит так (рисунок 6.4).

Ссылка	Код	Кабинет	Учитель	...
Ссылка21	Английский язык	Ссылка37	Ссылка7	...
Ссылка22	ИЗО	Ссылка31	СсылкаБ	...
Ссылка23	Информатика	Ссылка39	Ссылка3	...
...

Рисунок 6.4. Справочник «Предметы»

У него в таблице нет поля *Наименование*, так как при создании этого справочника вы сказали, что длина наименования будет равна нулю. Зато есть поля *Кабинет* и *Учитель*. Они появились потому, что эти реквизиты вы создали в конфигураторе (рисунок 6.5).

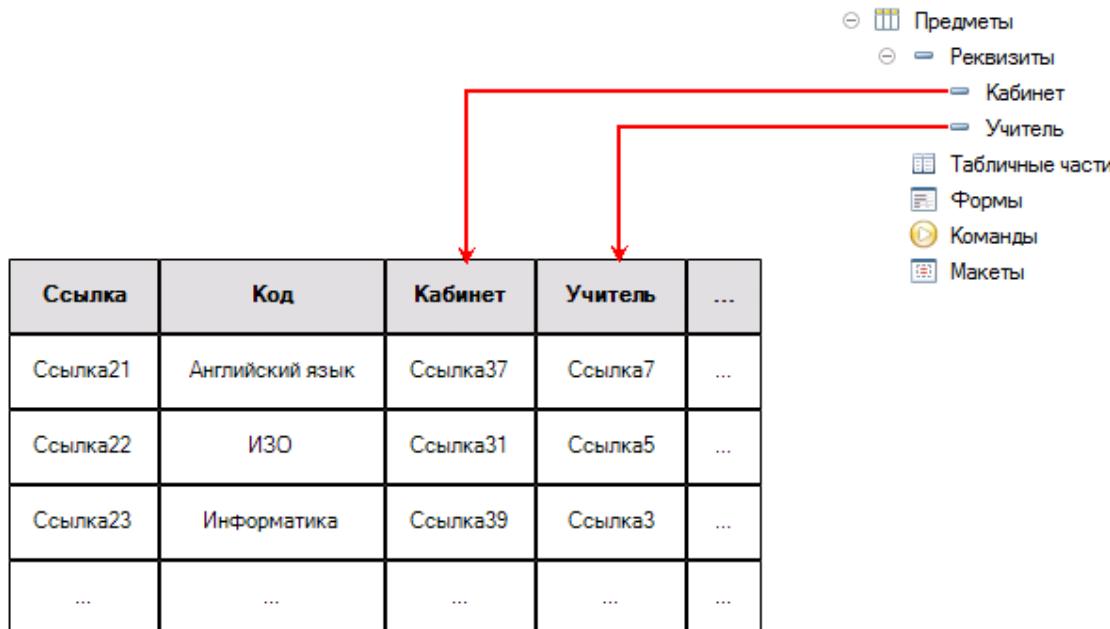


Рисунок 6.5. Реквизиты, созданные в конфигураторе, — это тоже поля таблицы

Для реквизитов *Кабинет* и *Учитель* вы выбирали тип *СправочникСсылка.<Имя справочника>*. И в этих полях действительно хранятся ссылки.

Но как же тогда получается, что в режиме 1С:Предприятие вы видите в этих полях номер кабинета и фамилию учителя (рисунок 6.6)?

Название предмета ↓	Кабинет	Учитель
— Английский язык	401	Николаева Т. Я.
— ИЗО	101	Казаков К. Д.
— Информатика	418	Давыдова А. В.
— История	127	Герасимова Е....
— Кл. час	311	Рыбакова А. Е.
— Литература	409	Авдеева В. М.
— Математика	311	Рыбакова А. Е.
— Музыка	131	Евсеева О. Ю.
— Природоведение	220	Филиппова Л. В.
— Русский язык	409	Авдеева В. М.
— Физ. культура	223	Крюков В. В.

Рисунок 6.6. Справочник «Предметы» в режиме 1С:Предприятие

Когда платформа хочет показать список предметов, происходит следующее. Она видит, что в поле *Учитель* находится ссылка. Она находит в базе данных элемент с такой ссылкой и вместо ссылки показывает то, что указано в объекте конфигурации.

А в объекте конфигурации вы указали, что основное представление справочника Учителя должно быть в виде наименования. Значит, платформа и показывает вам наименование учителя (рисунок 6.7).

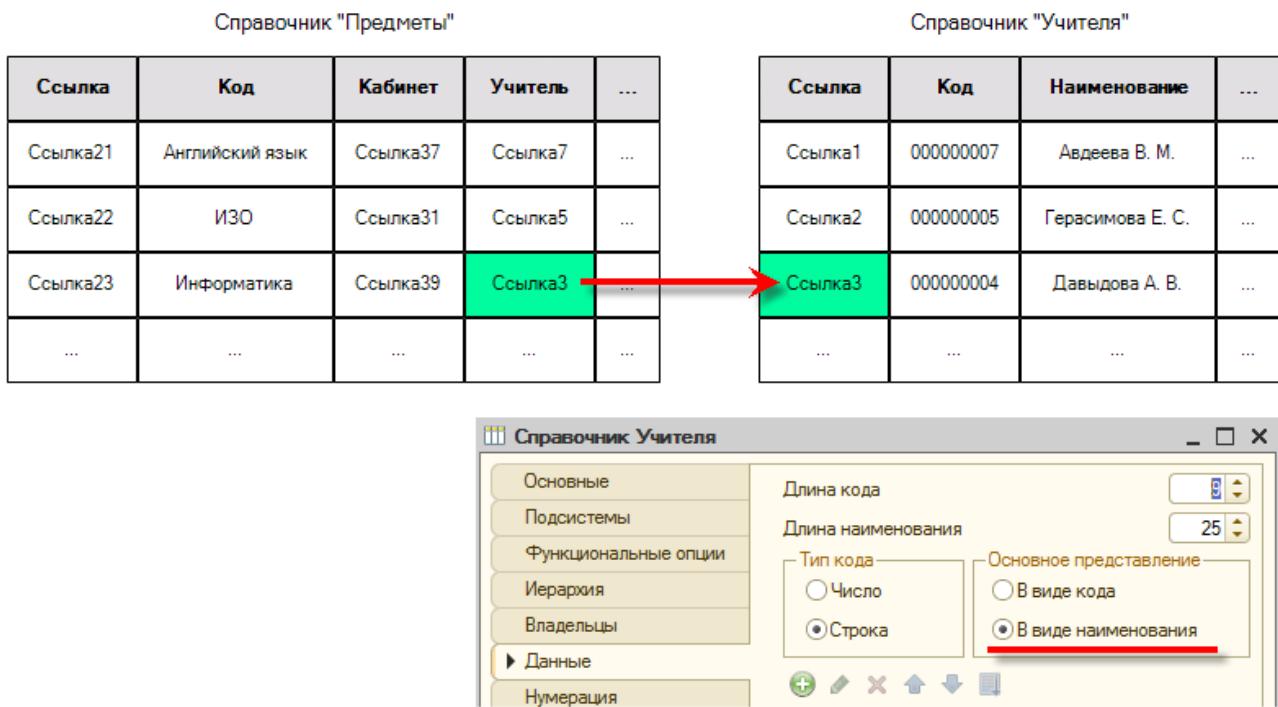


Рисунок 6.7. Отображение основного представления ссылочного значения

Теперь посмотрите на то, как хранятся данные документа *УчебныйДень*. Тут всё немного сложнее.

Документ *УчебныйДень* хранится уже в двух таблицах. Потому что у него есть табличная часть. Она хранится в отдельной таблице (рисунок 6.8).

Документ "УчебныйДень" Основная таблица					Документ "УчебныйДень" Табличная часть					
Ссылка	Номер	Дата	Проведен	...	Ссылка	Номер строки	Предмет	Домашнее задание	Оценка	Комментарий учителя
Ссылка51	20150901	01.09.2015 12:00:00	ИСТИНА	...	Ссылка51	1	Ссылка25			
Ссылка52	20150902	02.09.2015 12:00:00	ИСТИНА	...	Ссылка51	2	Ссылка25			
Ссылка53	20150903	03.09.2015 12:00:00	ЛОЖЬ	...	Ссылка51	3	Ссылка21		5	
...	Ссылка51	4	Ссылка28		4	
					Ссылка51	5	Ссылка27		5	
					Ссылка51	6	Ссылка210		4	
					Ссылка52	1	Ссылка27		4	
					Ссылка52	2	Ссылка21		3	
					Ссылка52	3	Ссылка29		5	
				

Рисунок 6.8. Таблицы документа «УчебныйДень»

Как платформа узнаёт, где во второй таблице строки, которые принадлежат документу из первой таблицы? Опять по ссылке. Каждый документ имеет уникальную ссылку. И точно такая же ссылка есть у его строк табличной части.

Когда в режиме 1С:Предприятие вы открываете документ, платформа «достаёт» строки его табличной части из второй таблицы именно по той ссылке, которая есть у самого документа (рисунок 6.9).

Учебный день 20150901 от 01.09.2015 12:00:00

N	Предмет	Учитель, Кабинет	Домашнее задание	Оценка	Комментарий учителя
1	Кл. час	Рыбакова А. Е., 311			
2	Кл. час	Рыбакова А. Е., 311			
3	Английский язык	Николаева Т. Я., 401		5	
4	Музыка	Евсеева О. Ю., 131		4	
5	Математика	Рыбакова А. Е., 311		5	
6	Русский язык	Авдеева В. М., 409		4	

Рисунок 6.9. Документ «УчебныйДень» в режиме 1С:Предприятие

6.3 Таблицы запросов

Система 1С:Предприятие может использовать разные системы управления базами данных (СУБД) для хранения своих данных. В них таблицы и поля имеют имена, которые вам ничего не скажут об их назначении. Эти имена нужны только самой СУБД.

С другой стороны, вам было бы очень удобно в языке запросов указывать те названия, которые есть у вас в конфигураторе. А платформа пусть сама разбирается, как эта таблица или это поле называется в конкретной СУБД.

Поэтому с помощью языка запросов вы не можете обратиться непосредственно к той таблице, которая есть в базе данных. Вы обращаетесь к некоторому «образу» этой таблицы, который вам предоставляет платформа. К одной из *таблиц запросов*. Любая таблица запроса имеет понятные вам имена и все поля, которые есть в настоящей таблице (рисунок 6.10).

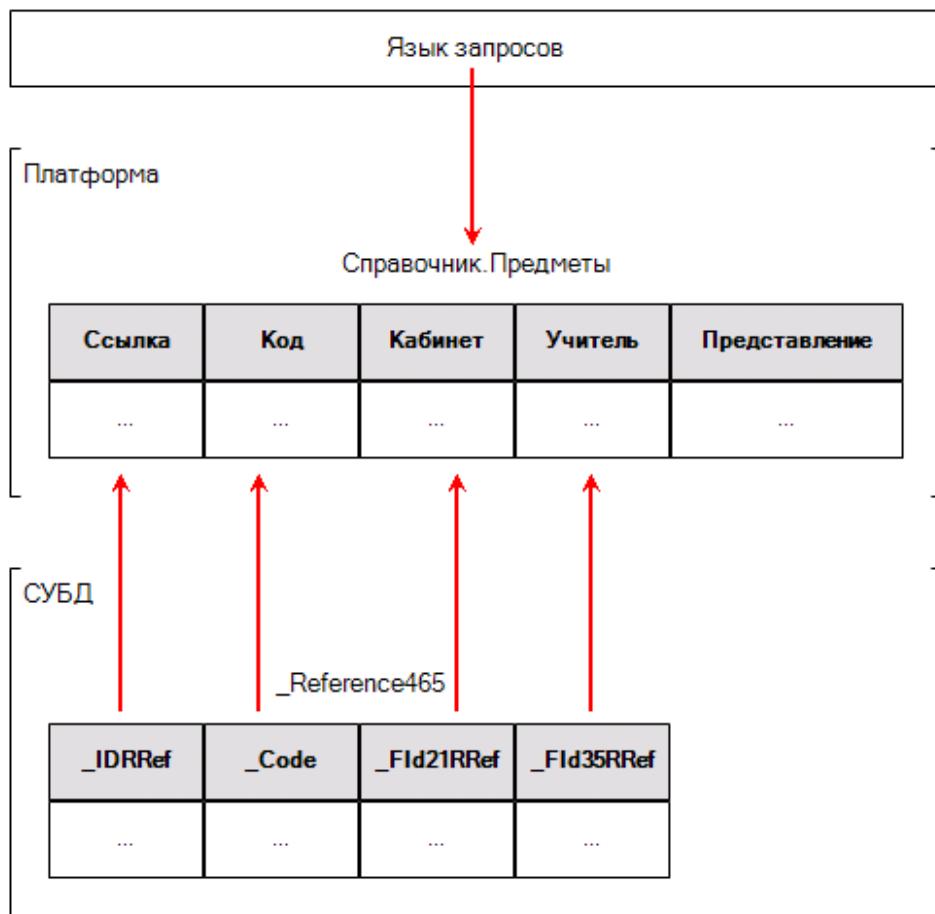


Рисунок 6.10. Физическая таблица и таблица запроса

Но кроме этого таблица запроса может содержать поля, которых нет в физической таблице. Например, поле *Представление*. Платформа сама добавляет для вас это поле, потому что оно может вам понадобиться. Это такой «сервис» от платформы.

Такие поля, которых нет на самом деле в физических таблицах, называются *виртуальными*. Среди таблиц запросов есть даже целые виртуальные таблицы, которых не существует на самом деле в базе данных.

Но сейчас вам не нужно сильно вникать в это. Просто знайте, что это такое, если слово «виртуальный» встретится вам в описании или в документации.

Все таблицы запросов перечислены и описаны в синтакс-помощнике. Вы найдёте их в ветке *Работа с запросами — Таблицы запросов* (рисунок 6.11).

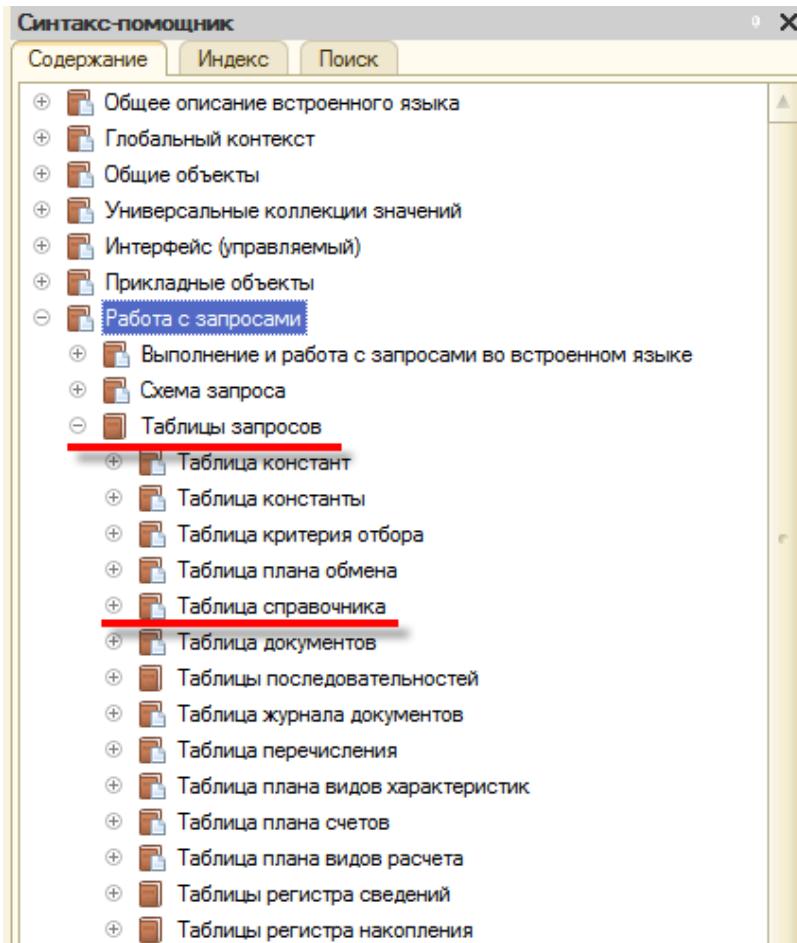


Рисунок 6.11. Таблицы запросов

Подробнее

Подробнее вы можете прочитать про таблицы запросов в документации «[Руководство разработчика. 8.3. Раздел 8.1 Источники данных \(таблицы\) запросов](#)».

6.4 Консоль запросов

Для знакомства с запросами вы будете использовать специальную обработку, которая называется *Консоль запросов*. Это отдельный файл, который вы можете запустить в прикладном решении в режиме 1С:Предприятие. С помощью этой обработки прямо на «живых» данных вы можете попробовать составить тот или иной запрос и сразу посмотреть на его результаты.

Когда результат запроса будет именно такой, какой вам нужен, вы просто скопируете текст этого запроса из обработки в конфигурацию. Такой приём позволяет значительно сократить время, особенно начинающим разработчикам.

Консоль запросов представляет собой файл «КонсольЗапросов.ерф». Этот файл содержится в архиве вместе с демонстрационными конфигурациями к этой книге. Где скачать этот архив, написано в разделе [А.1 «Как подключить демонстрационную базу»](#) на странице [543](#).

Запустите 1С:Предприятие в режиме отладки и выполните команду главного меню *Файл – Открыть*. Выберите файл консоли запросов. Платформа откроет обработку (рисунок [6.12](#)).

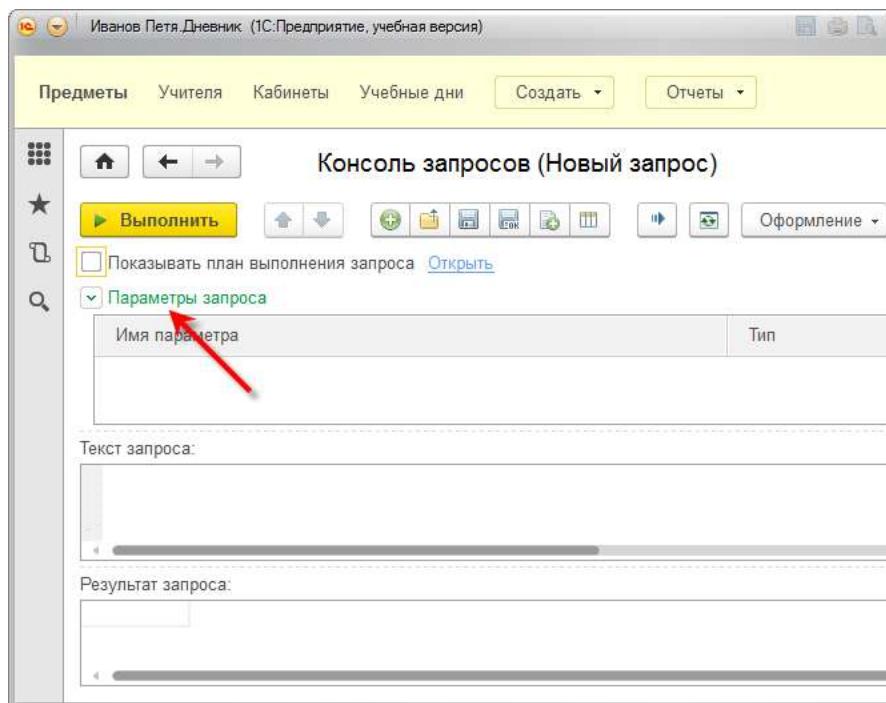


Рисунок 6.12. Консоль запросов

Сразу нажмите на зелёный заголовок *Параметры запроса*, чтобы скрыть верхнее поле. Оно нам не понадобится.

А с остальными полями вы будете работать следующим образом (рисунок 6.13).

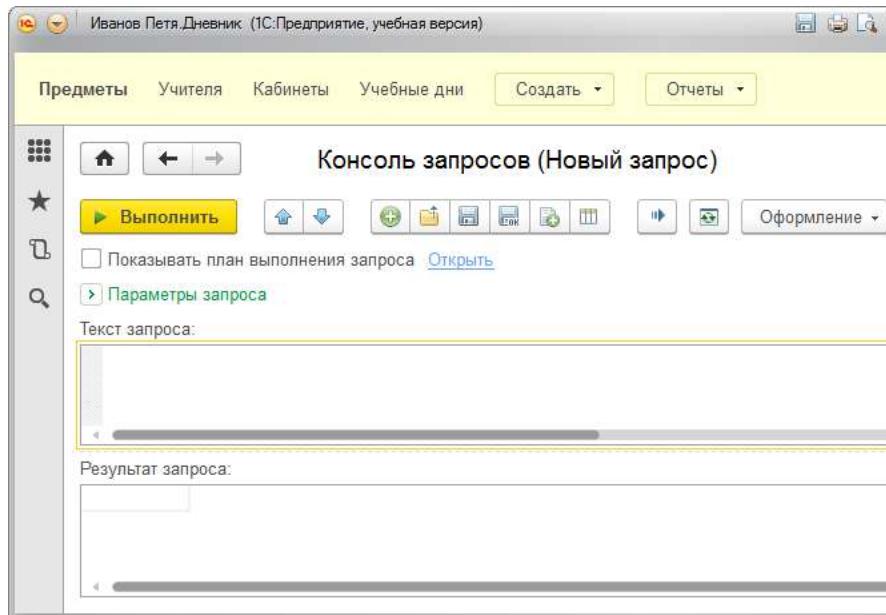


Рисунок 6.13. Консоль запросов, готовая для выполнения упражнений

В поле Текст запроса вы напишете текст, нажмёте кнопку Выполнить и в нижнем поле увидите результат. И так столько раз, сколько вам захочется. Ничего сложного.

6.5 Текст запроса

Текст запроса состоит обычно из нескольких предложений. Выглядит он, например, так, как показано в листинге 6.1.

Листинг 6.1. Текст запроса состоит из предложений

```
"ВЫБРАТЬ
| Предметы.Ссылка КАК УчебныйДень,      // Предложение ВЫБРАТЬ
| Предметы.Кабинет КАК Кабинет,          //
| Предметы.Учитель КАК Учитель           //
| ИЗ
| Справочник.Предметы КАК Предметы       // Предложение ИЗ
| ГДЕ
| Предметы.Кабинет.Код >= 200            // Предложение ГДЕ
| Предметы.Кабинет.Код < 300              //
|
| УПОРЯДОЧИТЬ ПО
| Предметы.Кабинет.Код"                  // Предложение УПОРЯДОЧИТЬ ПО
```

Подробнее

В этом листинге показаны не все предложения языка запросов. Только те, которые понадобятся вам для знакомства. Подробнее вы можете прочитать про язык запросов в документации «Руководство разработчика. 8.3. Раздел 8.4 Язык запросов».

Поскольку главная задача запроса — выбрать данные из базы, то и смысл его предложений очень простой. Предложение *ИЗ* описывает, из какой таблицы нужно выбрать данные. А предложение *ВЫБРАТЬ* описывает те поля, которые надо выбрать.

ВЫБРАТЬ, ИЗ, ГДЕ, УПОРЯДОЧИТЬ ПО — это ключевые слова языка запросов. Их принято писать прописными буквами.

В самом простом виде, когда вы не знаете, какие поля хотите получить, вы можете вместо списка полей написать символ **«*»**. Тогда запрос выберет вам все поля, которые есть в таблице. Попробуйте.

Напишите такой текст запроса (рисунок 6.14).

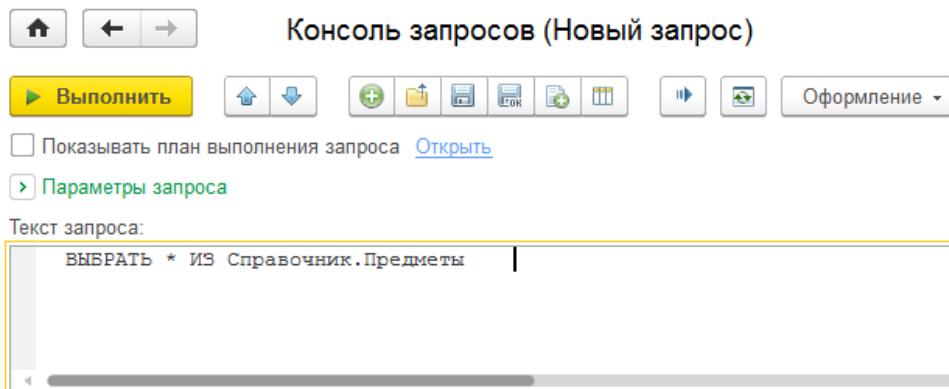


Рисунок 6.14. Выбрать все поля из таблицы

Нажмите *Выполнить*, и в нижнем поле вы увидите результат (рисунок 6.15).

Результат запроса (количество строк = 11, время выполнения = 0,002 с):

Запрос: Справочник.Предметы (Записей в результате: 11)						
Ссылка	ВерсияДанных	ПометкаУдаления	Код	Кабинет	Учитель	Предопределение
Музыка	AAAAAQAAAAAA=	Нет	Музыка	131	Евсеева О. Ю.	Нет
ИЗО	AAAAAQAAAAAE=	Нет	ИЗО	101	Казаков К. Д.	Нет
Английский язык	AAAAAQAAAAAI=	Нет	Английский язык	401	Николаева Т. Я.	Нет
Информатика	AAAAAQAAAAAM=	Нет	Информатика	418	Давыдова А. В.	Нет
История	AAAAAQAAAAQ=	Нет	История	127	Герасимова Е. С.	Нет
Кл. час	AAAAAQAAAAAU=	Нет	Кл. час	311	Рыбакова А. Е.	Нет
Литература	AAAAAQAAAAAY=	Нет	Литература	409	Авдеева В. М.	Нет
Математика	AAAAAQAAAAAc=	Нет	Математика	311	Рыбакова А. Е.	Нет
Природоведение	AAAAAQAAAAAg=	Нет	Природоведение	220	Филипова Л. В.	Нет
Русский язык	AAAAAQAAAAAk=	Нет	Русский язык	409	Авдеева В. М.	Нет
Физ. культура	AAAAAQAAAAO=	Нет	Физ. культура	223	Крюков В. В.	Нет

Рисунок 6.15. Таблица справочника «Предметы»

На этот рисунок поместились не все поля таблицы, но вам это и не важно. Потому что, как правило, нужны только некоторые поля.

Например, чтобы знать расписание, нужны лишь поля *Ссылка* и *Кабинет*. Напишите их вместо звёздочки через запятую (рисунок 6.16).

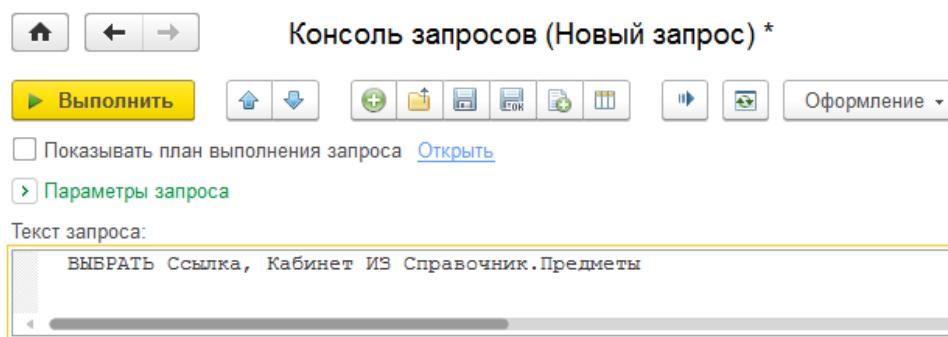


Рисунок 6.16. Выбрать два поля из таблицы

В результате вы получите таблицу, состоящую только из двух полей: *Ссылка* и *Кабинет* (рисунок 6.17).

Результат запроса (количество строк = 11, время	
Запрос: Справочник.Предметы	
Ссылка	Кабинет
Музыка	131
ИЗО	101
Английский язык	401
Информатика	418
История	127
Кл. час	311
Литература	409
Математика	311
Природоведение	220
Русский язык	409
Физ. культура	223

Рисунок 6.17. Поля «Ссылка» и «Кабинет»

Текст запроса, как вы убедились, можно писать вручную. Но в первое время гораздо удобнее пользоваться *конструктором запроса*. Вы уже сталкивались с ним раньше, когда разрабатывали отчёты.

В консоли запросов вы можете вызвать конструктор из контекстного меню в поле Текст запроса (рисунок 6.18).

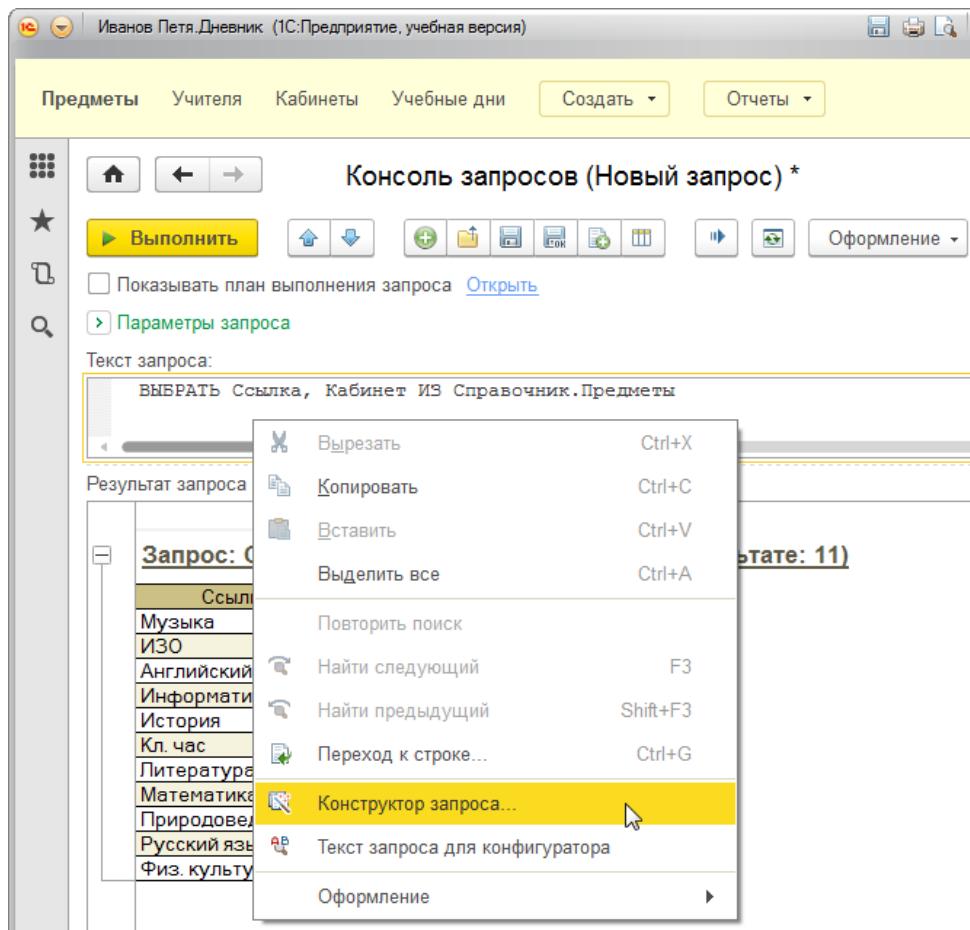


Рисунок 6.18. Вызов конструктора запроса

Если поле Текст запроса пусто, конструктор откроет вам пустой запрос. А если вы уже написали какой-то текст запроса, конструктор «подхватит» его и покажет (рисунок 6.19).

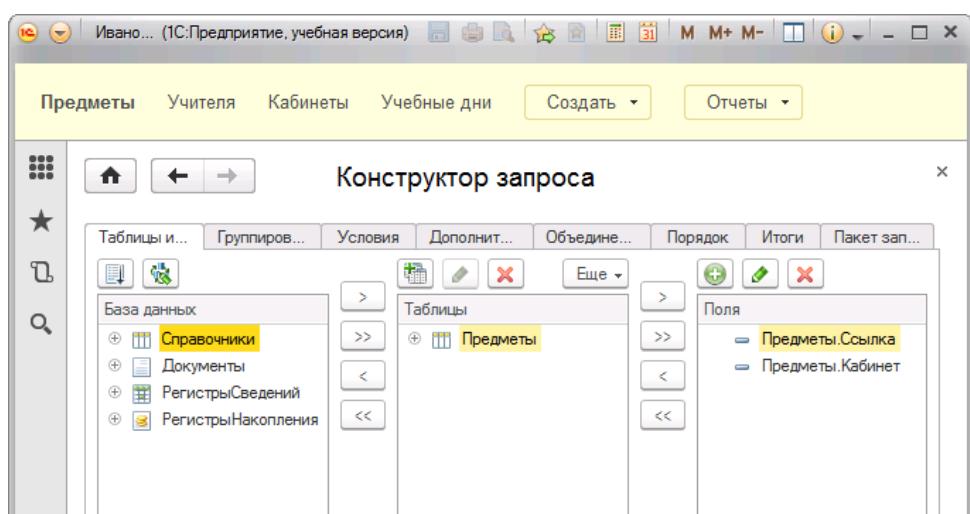


Рисунок 6.19. Конструктор запроса подхватит ваш запрос

В поле База данных находятся доступные вам таблицы запросов. То есть те таблицы запросов, к которым вы можете обратиться. В поле Таблицы находится та таблица, из

которой вы выбираете данные. То есть та таблица, которую вы указали в предложении *ИЗ*. В поле *Поля* находятся поля, которые вы указали в предложении *ВЫБРАТЬ*.

Конструктор запроса не просто позволяет вам с помощью мыши составить текст запроса, но и делает это синтаксически правильно. То есть он не позволит вам допустить ошибку.

Обратной стороной этого является то, что конструктор запроса формирует более сложный текст, чем, например, могли бы написать вы. В этом вы можете легко убедиться, если нажмёте сейчас в конструкторе на кнопку *OK*.

В вашем запросе появятся дополнительные элементы (рисунок 6.20).

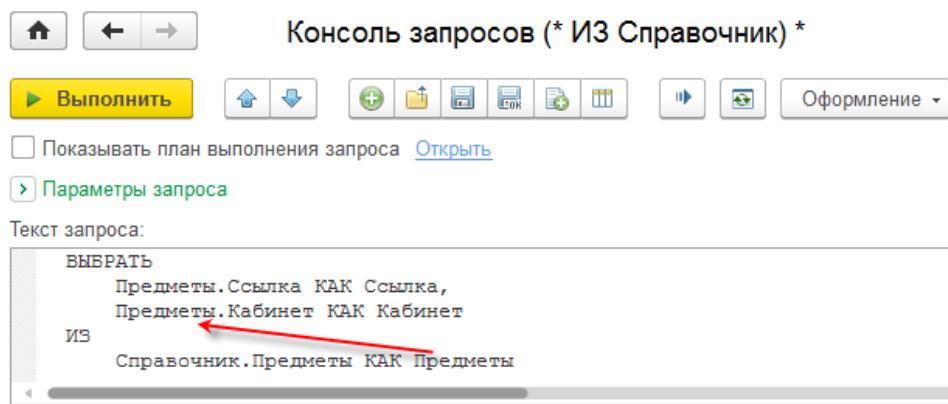


Рисунок 6.20. Псевдонимы источника и полей

Вы видите, что в тексте появилось новое ключевое слово *КАК*. С помощью этого слова задаются *псевдонимы* для источников и полей выборки. Источников у запроса может быть не один, а несколько. Чтобы каждый раз не писать их длинные имена через точку, удобнее задать источнику псевдоним и дальше, в тексте запроса, обращаться уже к этому псевдониму. Например, на рисунке 6.20 так используется псевдоним источника *Предметы*.

Использование псевдонимов не влияет на состав получаемых данных, а просто упрощает запись. Вы можете в этом убедиться, если выполните этот запрос. Результат не поменяется.

Псевдонимы можно задавать и *полям выборки*, то есть тем полям, которые перечислены в предложении *ВЫБРАТЬ*. Смысл их использования такой же, как и у псевдонимов источников. А кроме этого они определяют то, как будут называться колонки в результирующей таблице.

Например, первая колонка сейчас называется у вас *Ссылка*, и это непонятно. Вы можете изменить псевдоним поля *Ссылка* на *Предмет* (рисунок 6.21).

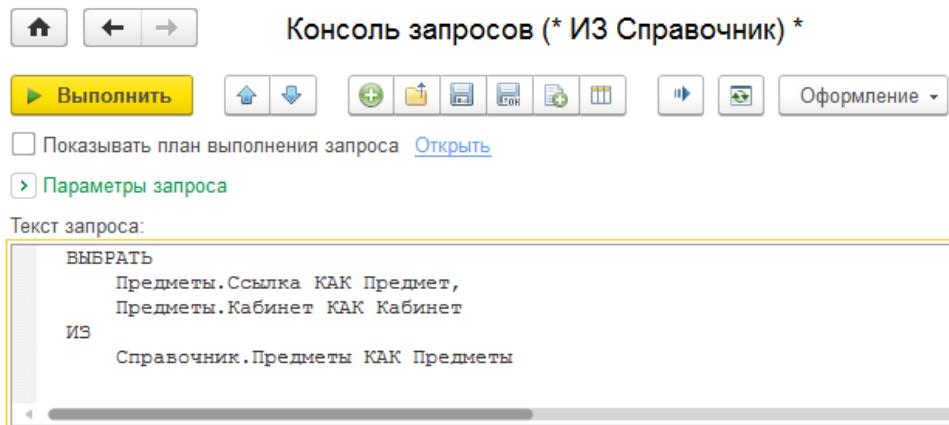


Рисунок 6.21. Изменение псевдонима поля выборки

Результат такого запроса будет более понятным и удобным (рисунок 6.22).

Результат запроса (количество строк = 11, время	
<u>Запрос: Справочник.Предметы</u>	
Предмет	Кабинет
Музыка	131
ИЗО	101
Английский язык	401
Информатика	418
История	127
Кл. час	311
Литература	409
Математика	311
Природоведение	220
Русский язык	409
Физ. культура	223

Рисунок 6.22. Псевдоним поля в результате запроса

Чтобы упростить знакомство, все следующие упражнения вы будете делать с помощью конструктора запроса. Но, как вы понимаете, всё то же самое можно написать и «вручную».

Откройте свой запрос в конструкторе. Сейчас у вас выбрано два поля: *Предмет* и *Кабинет* (рисунок 6.23).

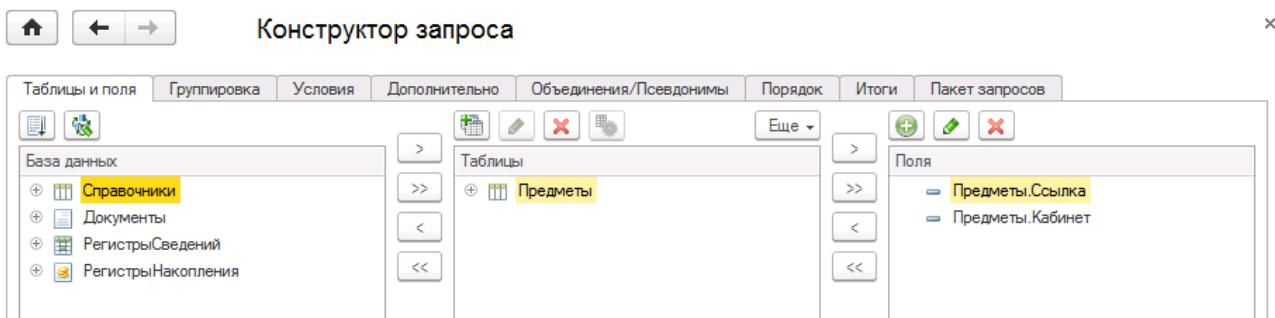


Рисунок 6.23. Выбраны поля «Ссылка» и «Кабинет»

Вместе с ними вам хотелось бы видеть и другие поля, например поле Учитель. Для этого достаточно раскрыть таблицу *Предметы* и перетащить поле Учитель мышью. Или просто дважды щёлкнуть по нему (рисунок 6.24).

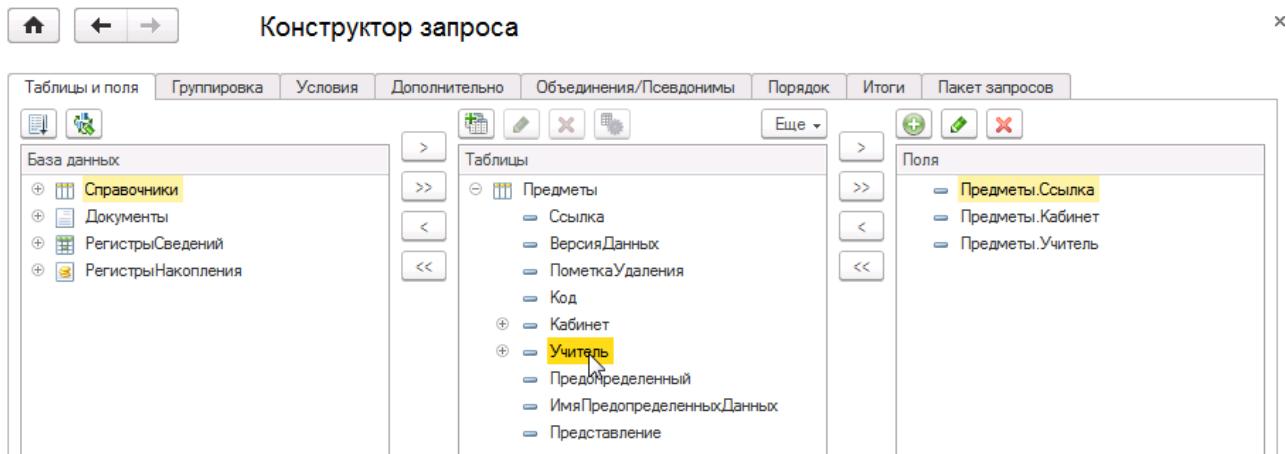


Рисунок 6.24. Добавление поля «Учитель»

На закладке *Объединения/Псевдонимы* вы можете сразу же задать собственный псевдоним для этого поля (рисунок 6.25).

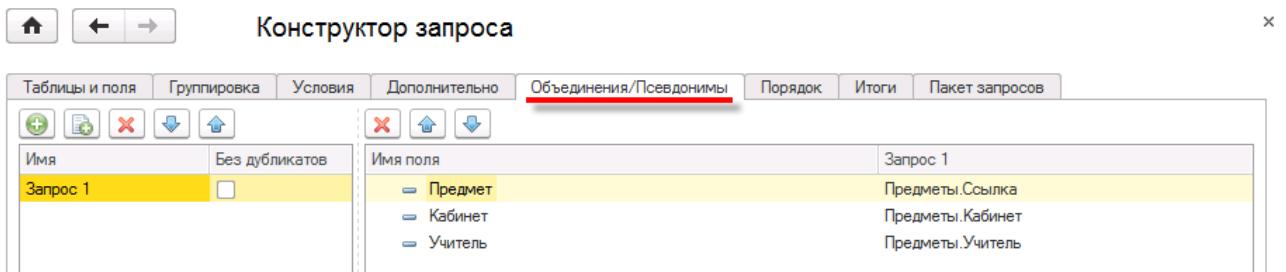


Рисунок 6.25. Псевдонимы полей в конструкторе запроса

В вашем случае сейчас это не нужно, стандартный псевдоним Учитель вполне понятен. Поэтому можете нажать *OK* в конструкторе и выполнить получившийся запрос. В вашей таблице появится третье поле — Учитель (рисунок 6.26).

Текст запроса:

```

ВЫБРАТЬ
    Предметы.Ссылка КАК Предмет,
    Предметы.Кабинет КАК Кабинет,
    Предметы.Учитель КАК Учитель
ИЗ
    Справочник.Предметы КАК Предметы
  
```

Результат запроса (количество строк = 11, время выполнения = 0,

Запрос: Справочник.Предметы (Записей в		
Предмет	Кабинет	Учитель
Музыка	131	Евсеева О. Ю.
ИЗО	101	Казаков К. Д.
Английский язык	401	Николаева Т. Я.
Информатика	418	Давыдова А. В.
История	127	Герасимова Е. С.
Кл. час	311	Рыбакова А. Е.
Литература	409	Авдеева В. М.
Математика	311	Рыбакова А. Е.
Природоведение	220	Филиппова Л. В.
Русский язык	409	Авдеева В. М.
Физ. культура	223	Крюков В. В.

Рисунок 6.26. Три поля в результате запроса

Что некрасиво в этой таблице? Все записи находятся в беспорядке. Это неудобно. Отсортируйте таблицу по полю Учитель.

Для этого откройте свой запрос в конструкторе и перейдите на закладку Порядок. Здесь перетащите или дважды щёлкните по полю Учитель (рисунок 6.27).

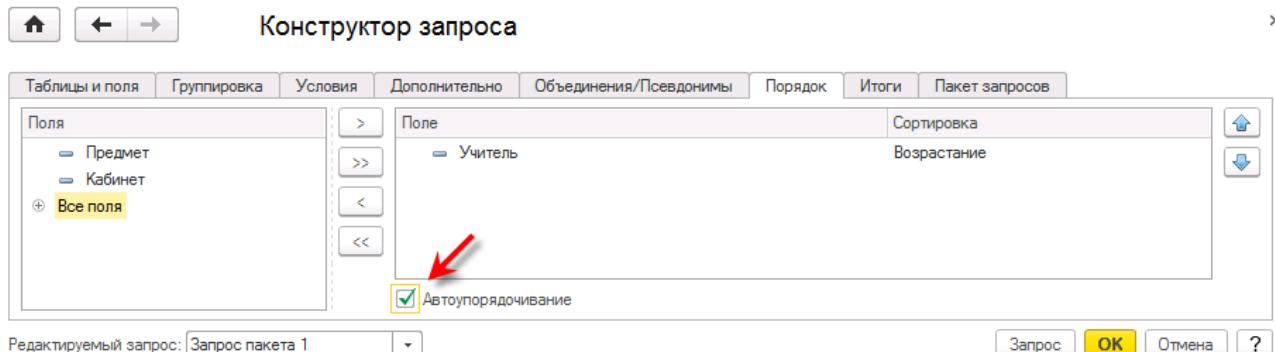


Рисунок 6.27. Автоупорядочивание по ссылочному полю

Платформа предложит сортировать по возрастанию поля Учитель, и это хорошо. Но вспомните: поле Учитель — это ссылка на справочник Учителя. Что значит отсортировать ссылку по возрастанию? Непонятно.

Поэтому в тех случаях, когда вы хотите отсортировать результат запроса по ссылочному полю, устанавливайте всегда флажок Автоупорядочивание (рисунок 6.27). Тогда платформа отсортирует результат по тому полю, которое является основным представлением для этой ссылки. А для справочника Учителя основным представлением является поле Наименование (рисунок 6.28).

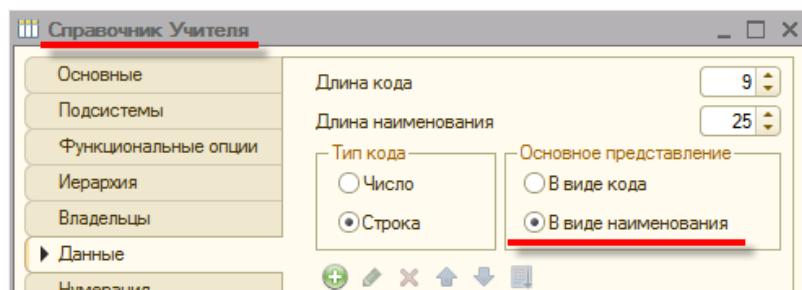


Рисунок 6.28. Наименование — основное представление справочника «Учителя»

Нажмите *OK* в конструкторе, выполните запрос. Действительно, результат будет отсортирован по наименованию учителей (рисунок 6.29).

Текст запроса:

ВЫБРАТЬ
Предметы.Ссылка КАК Предмет,
Предметы.Кабинет КАК Кабинет,
Предметы.Учитель КАК Учитель
ИЗ
Справочник.Предметы КАК Предметы

Результат запроса (количество строк = 11, время выполнения = 0,

Предмет	Кабинет	Учитель
Литература	409	Авдеева В. М.
Русский язык	409	Авдеева В. М.
История	127	Герасимова Е. С.
Информатика	418	Давыдова А. В.
Музыка	131	Евсеева О. Ю.
ИЗО	101	Казаков К. Д.
Физ. культура	223	Крюков В. В.
Английский язык	401	Николаева Т. Я.
Кл. час	311	Рыбакова А. Е.
Математика	311	Рыбакова А. Е.
Природоведение	220	Филипова Л. В.

Рисунок 6.29. Сортировка по наименованию учителей

Автоупорядочивание — это такой дополнительный сервис, который позволяет вам не задумываться о том, каким именно образом нужно сортировать ссылочные поля. Вы можете его использовать, а можете не использовать.

Например, если вам нужно отсортировать эту таблицу по номерам кабинетов, то вы можете в явном виде указать то поле, по которому нужно выполнять сортировку. Тем более что для справочника *Кабинеты* и нет никаких других вариантов.

Откройте запрос в конструкторе, перейдите на закладку *Порядок*. Удалите сортировку по полю *Учитель*, дважды щёлкнув на ней мышью. Снимите флажок *Автоупорядочивание*.

Из ветки *Все поля* выберите код кабинета и добавьте его в поля сортировки (рисунок 6.30).

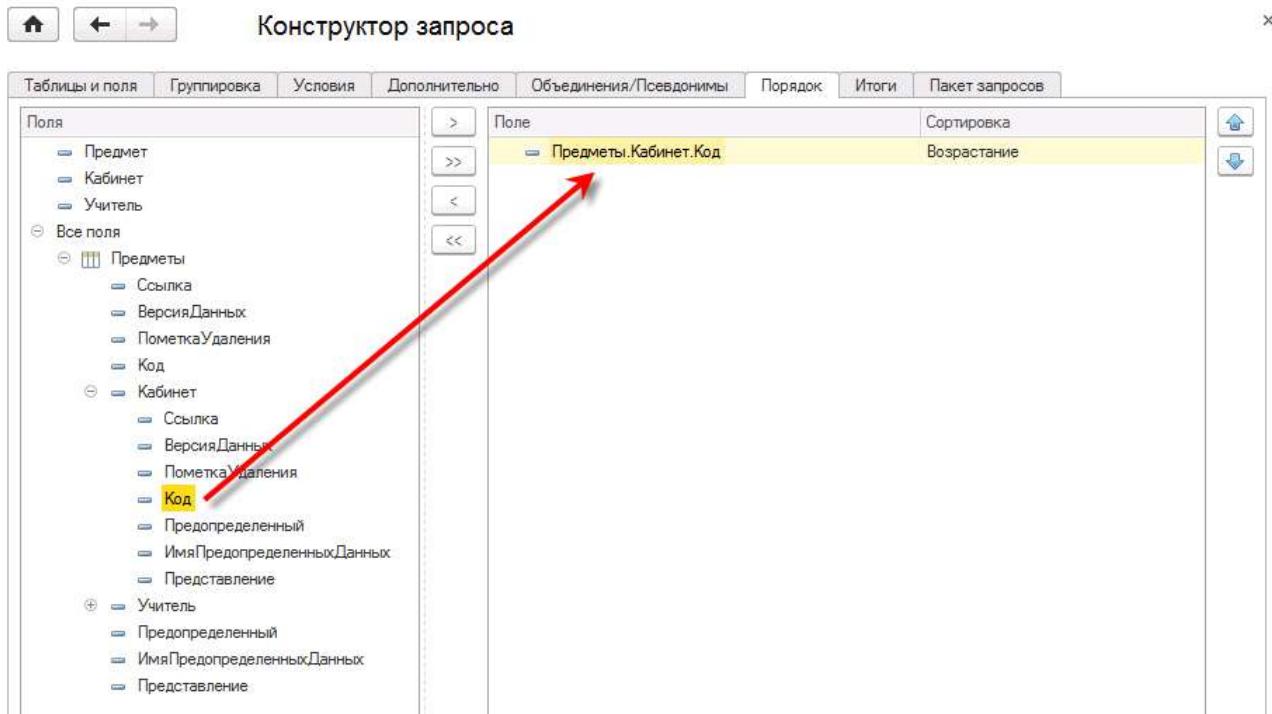


Рисунок 6.30. Сортировка по реквизиту ссылочного поля

Закройте конструктор и выполните получившийся запрос. Таблица будет отсортирована по номерам кабинетов (рисунок 6.31).

```

Текст запроса:
ВЫБРАТЬ
    Предметы.Ссылка КАК Предмет,
    Предметы.Кабинет КАК Кабинет,
    Предметы.Учитель КАК Учитель
ИЗ
    Справочник.Предметы КАК Предметы

УПОРЯДОЧИТЬ ПО
    Предметы.Кабинет.Код
  
```

Результат запроса (количество строк = 11, время выполнения = 0,001 с):

Запрос: Справочник.Предметы (Записей в резул)		
Предмет	Кабинет	Учитель
ИЗО	101	Казаков К. Д.
История	127	Герасимова Е. С.
Музыка	131	Евсеева О. Ю.
Природоведение	220	Филиппова Л. В.
Физ. культура	223	Крюков В. В.
Кл. час	311	Рыбакова А. Е.
Математика	311	Рыбакова А. Е.
Английский язык	401	Николаева Т. Я.
Литература	409	Авдеева В. М.
Русский язык	409	Авдеева В. М.
Информатика	418	Давыдова А. В.

Рисунок 6.31. Сортировка по номерам кабинетов

Задачи, когда нужно выбрать все записи из таблицы, встречаются нечасто. Гораздо чаще возникает необходимость выбрать лишь часть данных. Например, только те предметы, занятия по которым проходят на четвёртом этаже.

Номера всех кабинетов на четвёртом этаже начинаются с цифры 4, поэтому найти такие кабинеты просто. Их номер должен быть больше либо равен 400.

А написать это условие вы можете на закладке *Условия* конструктора запросов. Откройте конструктор, добавьте поле *Код* (рисунок 6.32).

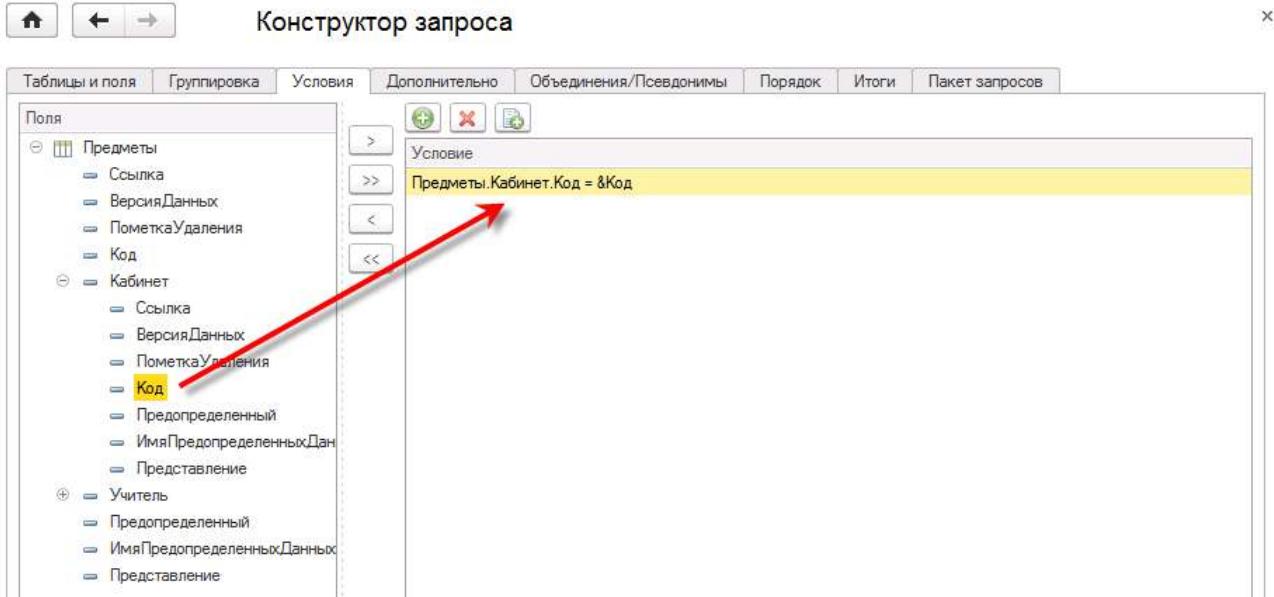


Рисунок 6.32. Добавление поля в условия

А теперь вручную отредактируйте то условие, которое добавила платформа (рисунок 6.33).

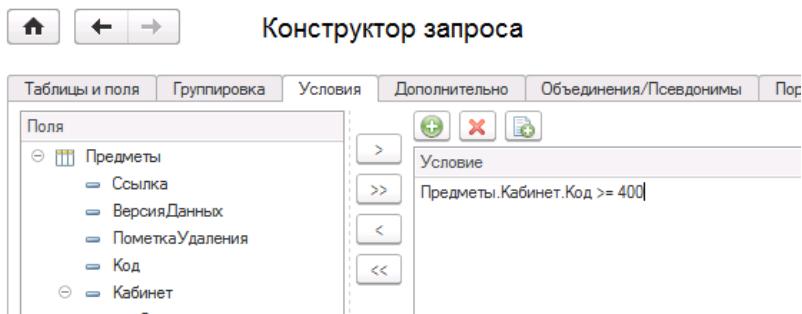


Рисунок 6.33. Условие отбора записей

Закройте конструктор, выполните отчёт. В таблице останутся только те предметы, занятия по которым проходят на 4-м этаже (рисунок 6.34).

Текст запроса:

```

ВЫБРАТЬ
    Предметы.Ссылка КАК Предмет,
    Предметы.Кабинет КАК Кабинет,
    Предметы.Учитель КАК Учитель
ИЗ
    Справочник.Предметы КАК Предметы
ГДЕ
    Предметы.Кабинет.Код >= 400

УПОРЯДОЧИТЬ ПО
    Предметы.Кабинет.Код

```

Результат запроса (количество строк = 4, время выполнения = 0 с):

Предмет	Кабинет	Учитель
Английский язык	401	Николаева Т. Я.
Литература	409	Авдеева В. М.
Русский язык	409	Авдеева В. М.
Информатика	418	Давыдова А. В.

Рисунок 6.34. Уроки, которые проводятся на 4-м этаже

Аналогичным образом, если вы захотите узнать, какие занятия проходят на втором этаже, вам нужно будет изменить условие отбора записей (рисунок 6.35).

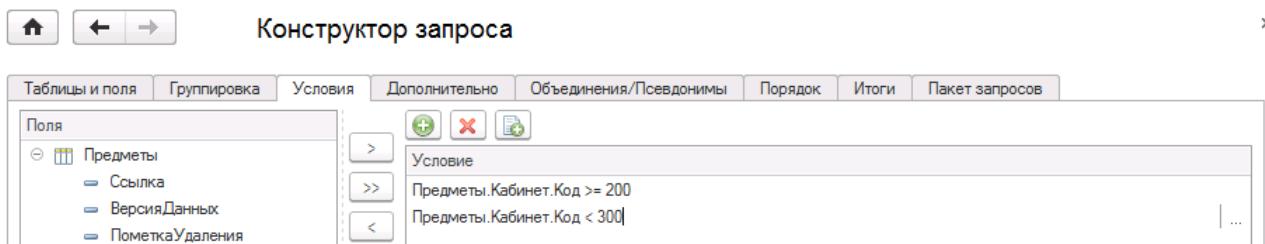


Рисунок 6.35. Объединение условий отбора

Отдельные условия, написанные в конструкторе, объединяются по И. В результате вы получите только те уроки, которые проходят на втором этаже (рисунок 6.36).

Текст запроса:

```

ВЫБРАТЬ
Предметы.Ссылка КАК Предмет,
Предметы.Кабинет КАК Кабинет,
Предметы.Учитель КАК Учитель
ИЗ
Справочник.Предметы КАК Предметы
ГДЕ
Предметы.Кабинет.Код >= 200
И Предметы.Кабинет.Код < 300
УПОРЯДОЧИТЬ ПО
Предметы.Кабинет.Код

```

Результат запроса (количество строк = 2, время выполнения = 0 с):

Предмет	Кабинет	Учитель
Природоведение	220	Филиппова Л. В.
Физ. культура	223	Крюков В. В.

Рисунок 6.36. Уроки, которые проводятся на 2-м этаже

Если условия отбора вам нужно объединить по *ИЛИ* или более сложным образом, вы можете сделать это вручную в тексте запроса.

Для задач, которые вы будете решать дальше, этих знаний языка запросов вам вполне достаточно. Осталось показать только один полезный приём.

После того как в консоли запросов вы создали такой запрос, как вам нужно, вы можете скопировать его текст в тот модуль, в котором собираетесь его использовать. Для вставки в программу текст запроса должен быть специальным образом оформлен.

Получить его в таком виде вы можете с помощью команды контекстного меню Текст запроса для конфигуратора (рисунок 6.37).

Текст запроса:

```

ВЫБРАТЬ
Предметы.Ссылка КАК Предмет,
Предметы.Кабинет КАК Кабинет,
Предметы.Учитель КАК Учитель
ИЗ
Справочник.Предметы КАК Предметы
ГДЕ
Предметы.Кабинет.Код >= 200
И Предметы.Кабинет.Код < 300
УПОРЯДОЧИТЬ ПО
Предметы.Кабинет.Код

```

Результат запроса (количество строк = 2, время выполнения = 0 с):

Предмет	Кабинет	Учитель
Природоведение	220	Филиппова Л. В.
Физ. культура	223	Крюков В. В.

Контекстное меню с выделенным пунктом «Текст запроса для конфигуратора».

Рисунок 6.37. Получить текст запроса для конфигуратора

Текст будет оформлен в виде строкового литерала (он заключён в кавычки). А для

переноса строк внутри литерала используется специальный символ « | », который зарезервирован во встроенным языке (рисунок 6.38).

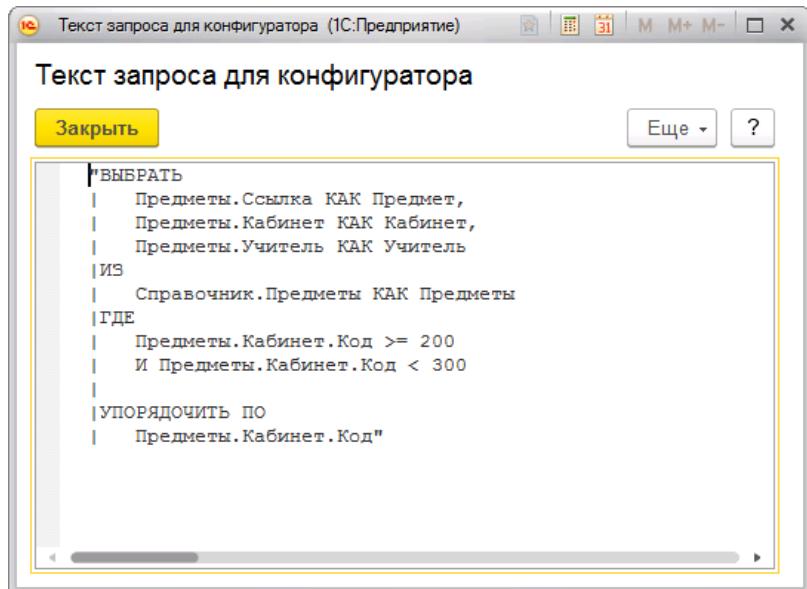


Рисунок 6.38. Текст запроса для вставки в конфигуратор

Куда вставлять этот текст и как его использовать, вы узнаете уже в следующей главе.

В следующей главе вы будете работать в конфигураторе, поэтому сеанс 1С:Предприятия можете закрыть. Сделать это нужно следующим образом: сначала закройте консоль запросов (не сохраняя изменения), а затем — окно приложения (рисунок 6.39).

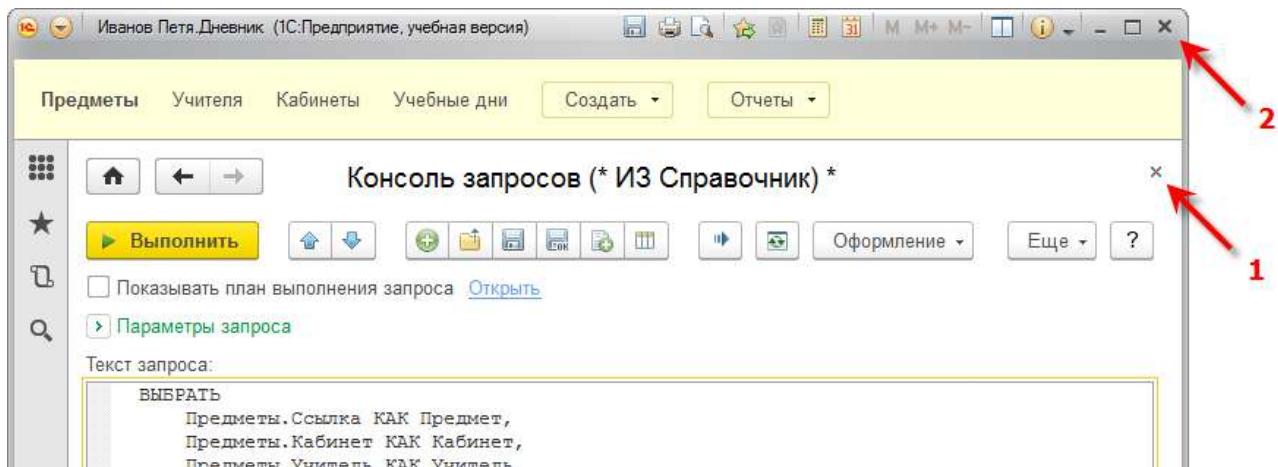


Рисунок 6.39. Порядок закрытия консоли запросов

Если вы будете закрывать другим способом, ничего страшного не случится. Просто вы можете увидеть несколько неожиданных для вас сообщений.

Задание 6.1

Задания выполняйте в консоли запросов. Выведите все кабинеты по возрастанию их номеров. Задайте понятное название для поля, в котором будет показан номер документа.

Задание 6.2

Выведите список учителей в алфавитном порядке. Задайте понятное название для поля, которое вы будете выводить.

Задание 6.3

Из регистра сведений *Домашние Задания* получите список всех невыполненных заданий. Отсортируйте его в порядке учебных дней и в порядке уроков.

Подсказка 1. Таблицы регистра *СрезПервых* и *СрезПоследних* использовать не нужно.

Подсказка 2. В тексте запроса тоже можно использовать литералы. Например, литералы значений *Булево* записываются так же, как и во встроенном языке.

Совет

Если вы хотите изучить язык запросов подробнее, научиться создавать сложные запросы, вам поможет книга Е. Ю. Хрусталевой «[Язык запросов 1С:Предприятия 8](#)» ([информация о книге](#)).



Рисунок 6.40. Книга «Язык запросов 1С:Предприятия 8»

Глава 7

Планировщик

Не читайте эту главу!

Если вы знаете:

- зачем нужен планировщик;
- как заполнить его данными;
- чем назначаемые обработчики событий отличаются от фиксированных;
- как показать только будние дни;
- как сделать, чтобы планировщик автоматически обновлял свои данные;

смело переходите к главе 8 «Доработка интерфейса» на странице 529.

Информационная база

Прикладное решение, которое должно получиться у вас к этому моменту, содержится в демонстрационной базе «08 РегистрДомашнеЗадания.dt». Как её подключить, написано в разделе А.1 «Как подключить демонстрационную базу» на странице 543.

Вот вы и подошли к предпоследней главе. В ней вы в полной мере сможете применить все знания, которые получили раньше.

По традиции, чтобы не потерять ориентацию в 1С:Предприятии, вспомните прогулку по лесу, с которой начинается эта книга. В предыдущей главе вы занимались языком запросов, то есть рассматривали 1С:Предприятие максимально подробно, на уровне отдельных листьев.

В этой главе ваше местонахождение не поменяется (рисунок 7.1).



Рисунок 7.1. Сейчас вы здесь

Вы продолжите модифицировать свой электронный дневник на максимально подробном уровне, который допускает система. На уровне встроенного языка и языка запросов.

7.1 Планировщик

В качестве итогового упражнения, которое закрепит ваши знания, вы сделаете очень полезный и удобный инструмент. Он будет показывать всё ваше расписание занятий в виде таблицы. В колонках будут дни недели, а в строках — часы занятий (рисунок 7.2).

	7 сентября 2015 г.	8 сентября 2015 г.	9 сентября 2015 г.	10 сентября 2015 г.	11 сентября 2015 г.
08:00	08:00	08:00	08:00	08:00	08:00
08:30 Кл. час	08:30 Кл. час	08:30 Математика	08:30 Математика	08:30 Литература	
09:00	09:00	09:00	09:00	09:00	
09:30 Кл. час	09:30 Кл. час	09:30 Английский язык	09:30 Музыка	09:30 Математика	
10:00	10:00	10:00	10:00	10:00	
10:30 Английский язык	10:30 Английский язык	10:30 Природоведение	10:30 Физ. культура	10:30 История	
11:00	11:00	11:00	11:00	11:00	
11:30 Музыка	11:30 Музыка	11:30 Природоведение	11:30 Математика	11:30 Литература	
12:00	12:00	12:00	12:00	12:00	
12:35 Математика	12:35 Математика	12:35 Русский язык	12:35 Английский язык	12:35 Физ. культура	
13:00	13:00	13:00	13:00	13:00	
13:30 Русский язык	13:30 Русский язык	13:30 Литература	13:30 ИЗО	13:30 Информатика	
14:00	14:00	14:00	14:00	14:00	

Рисунок 7.2. Расписание занятий

Для создания такого расписания вам понадобится объект встроенного языка, который называется *Планировщик*. Описание его типа и тех типов, которые с ним связаны, вы можете найти в синтакс-помощнике в ветке *Общие объекты — Планировщик* (рисунок 7.3).

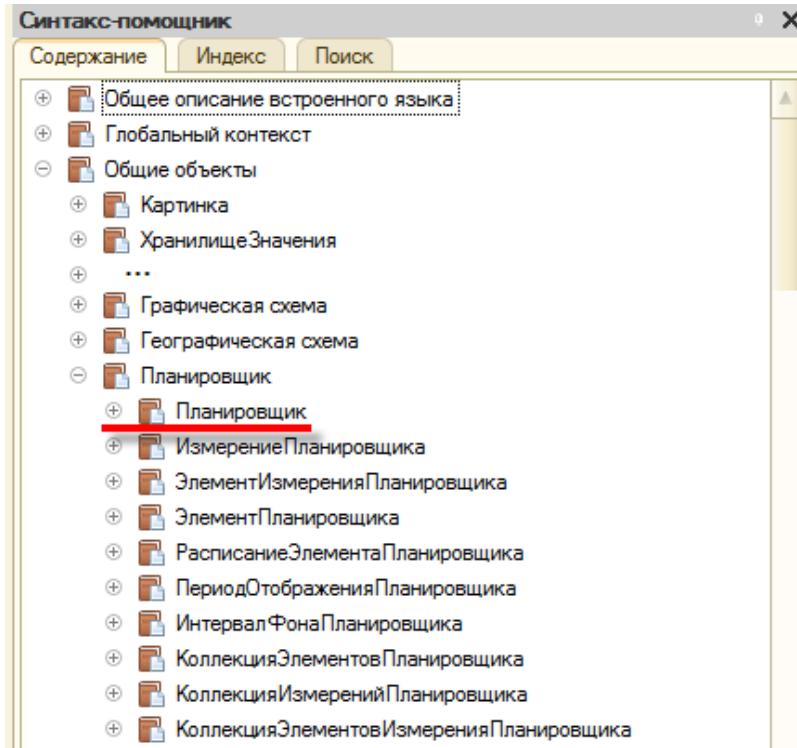


Рисунок 7.3. Тип «Планировщик» в синтакс-помощнике

Планировщик не хранит свои данные в базе данных. Он существует только в оперативной памяти компьютера в то время, пока работает ваше приложение.

Использовать его очень просто. Планировщик вы размещаете в форме. Потом заполняете его данными. Эти данные форма показывает в удобном и красивом виде. На рисунке 7.2 специально показан простейший вариант. У вас он будет красивее и удобнее.

Какие данные может показать планировщик? Любые данные, у которых есть начало и есть конец на оси времени. Например, ваши уроки. У каждого урока есть начало и есть конец. Поэтому ваша задача будет заключаться в том, чтобы прочитать из базы данных и записать в планировщик все ваши уроки. Они, как вы знаете, содержатся в табличной части документов *УчебныйДень*.

Планировщик вы создадите удобным и красивым не сразу, а постепенно. По шагам. Это поможет вам понять его устройство и возможности его изменения.

А начнёте вы с того, что создадите форму, в которой будет размещаться планировщик.

7.2 Создание формы и размещение в ней планировщика

Расписание занятий, которое вы собираетесь создать, является самостоятельным инструментом в вашей конфигурации. Оно будет использовать данные документов *УчебныйДень*, но нельзя сказать, что оно будет являться частью этих документов.

Поэтому для расписания вы создадите общую форму, не «привязанную» к какому-либо объекту конфигурации.

В конфигураторе раскройте ветку *Общие* и добавьте форму в ветку *Общие формы*. Платформа откроет *конструктор общих форм* (рисунок 7.4).

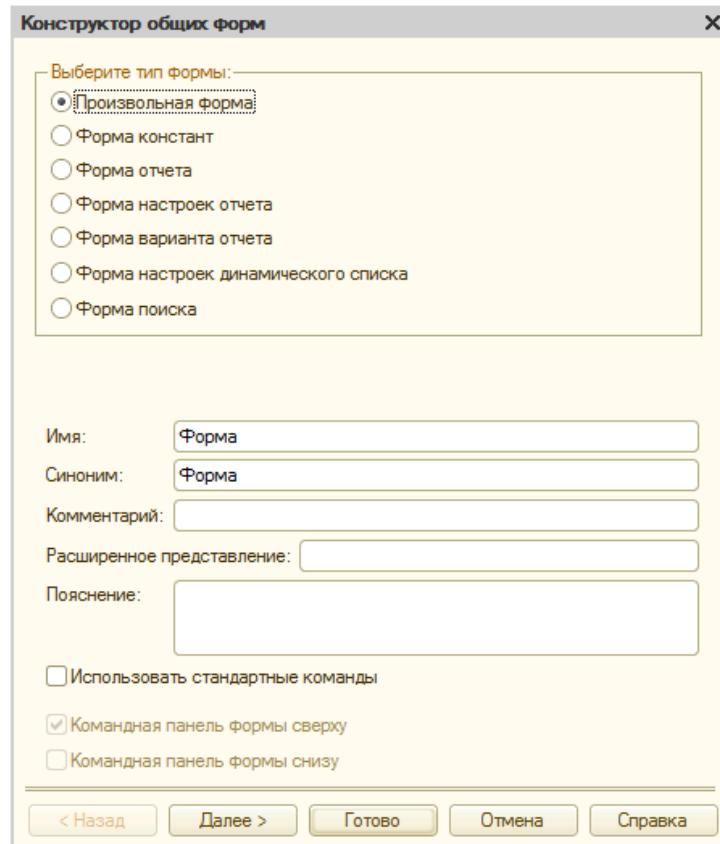


Рисунок 7.4. Конструктор общих форм

Некоторые общие формы могут иметь заранее определённое назначение (форма констант, форма поиска и так далее). Поэтому для них существуют отдельные варианты конструирования.

Ваша форма будет самая что ни на есть произвольная. Поэтому просто нажмите *Готово*. Платформа сконструирует вам совершенно пустую форму, в которой есть только командная панель (рисунок 7.5).

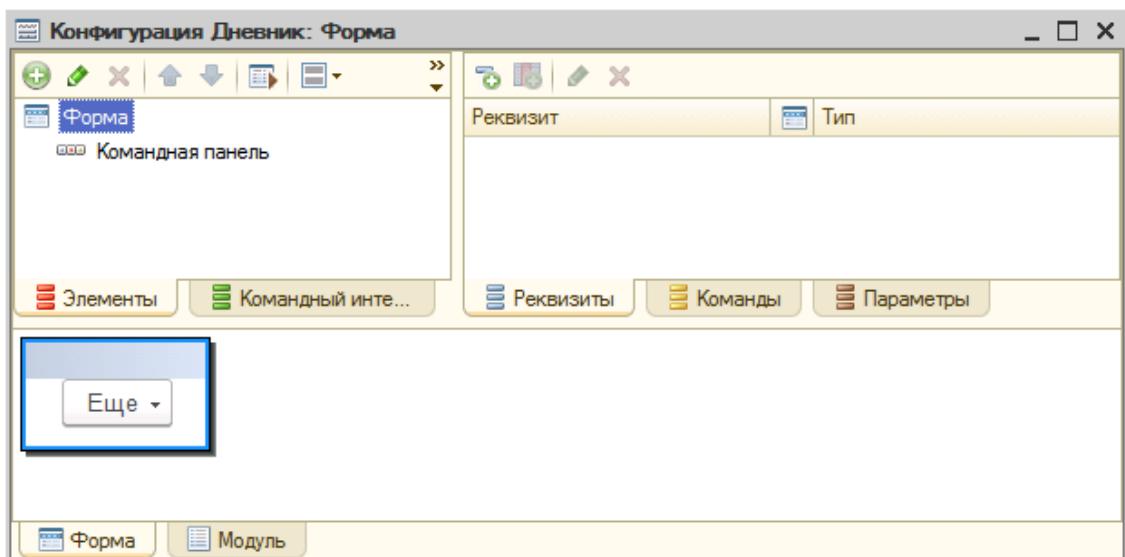


Рисунок 7.5. Пустая форма

Пользоваться редактором формы вы умеете. Если подзабыли, можете освежить свои знания в разделе 2.14.2 «Редактор формы» на странице 147.

Все данные, с которыми может работать форма, находятся в её реквизитах. Поэтому добавьте реквизит формы, назовите его *Планировщик*. Выберите для него тип *Планировщик*. Для этого вам придётся несколько раз прокрутить выпадающий список с типами (рисунок 7.6).

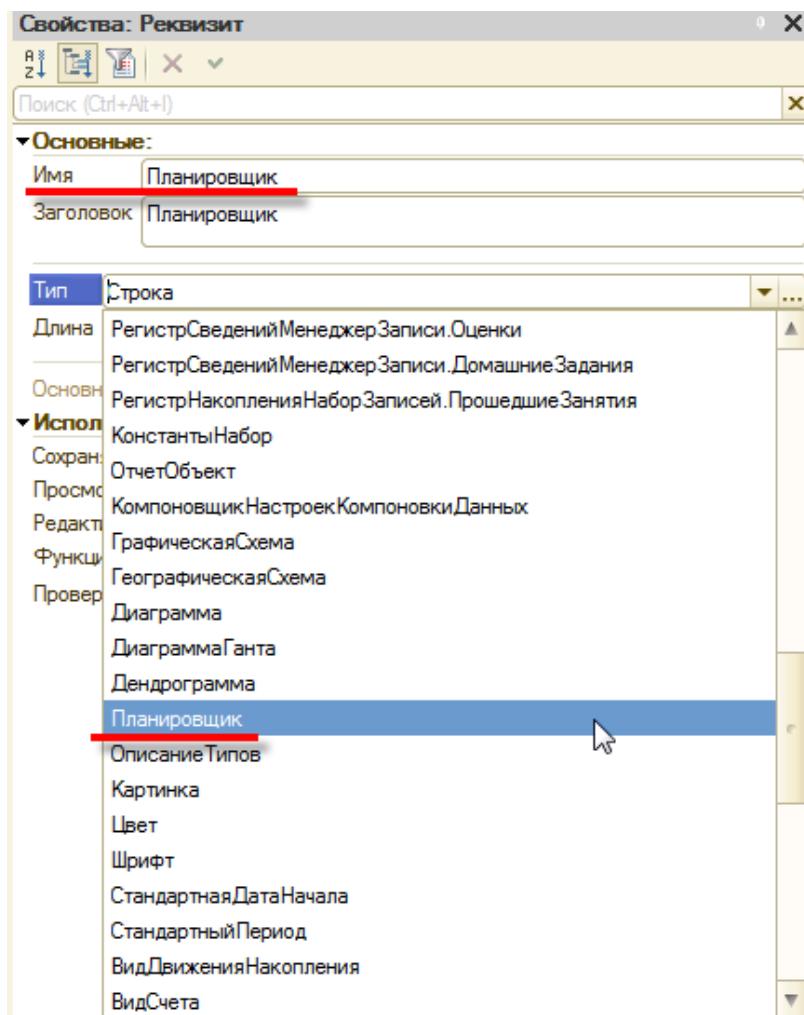


Рисунок 7.6. Тип реквизита — «Планировщик»

Теперь перетащите этот реквизит в дерево элементов формы, и условное изображение планировщика появится в окне предварительного просмотра формы (рисунок 7.7).

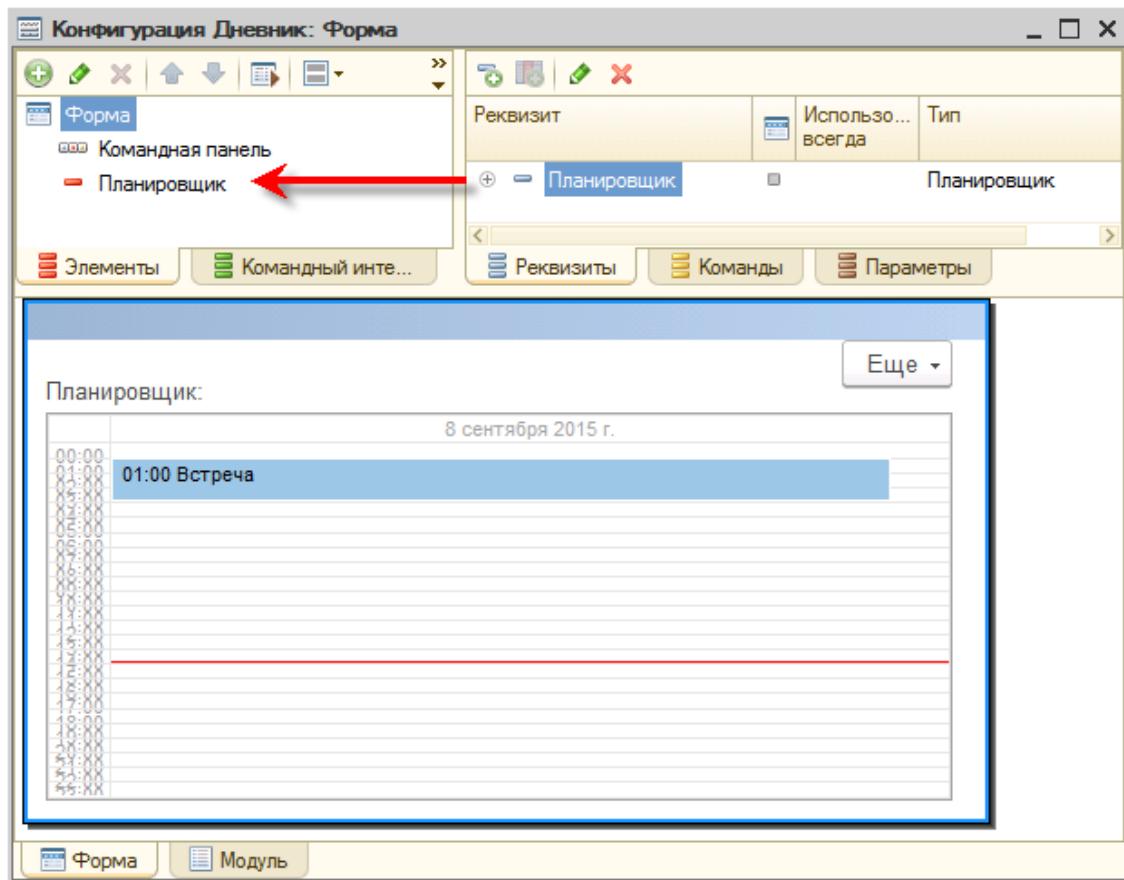


Рисунок 7.7. Планировщик в форме

Всё, форма готова.

Стандартно платформа не размещает общие формы в интерфейсе прикладного решения. Поэтому выделите форму в дереве конфигурации, а в панели свойств установите флажок *Использовать стандартные команды*. После этого ваша форма в режиме 1С:Предприятие появится в группе *Сервис*.

Заодно задайте и её имя — *Планировщик* (рисунок 7.8).

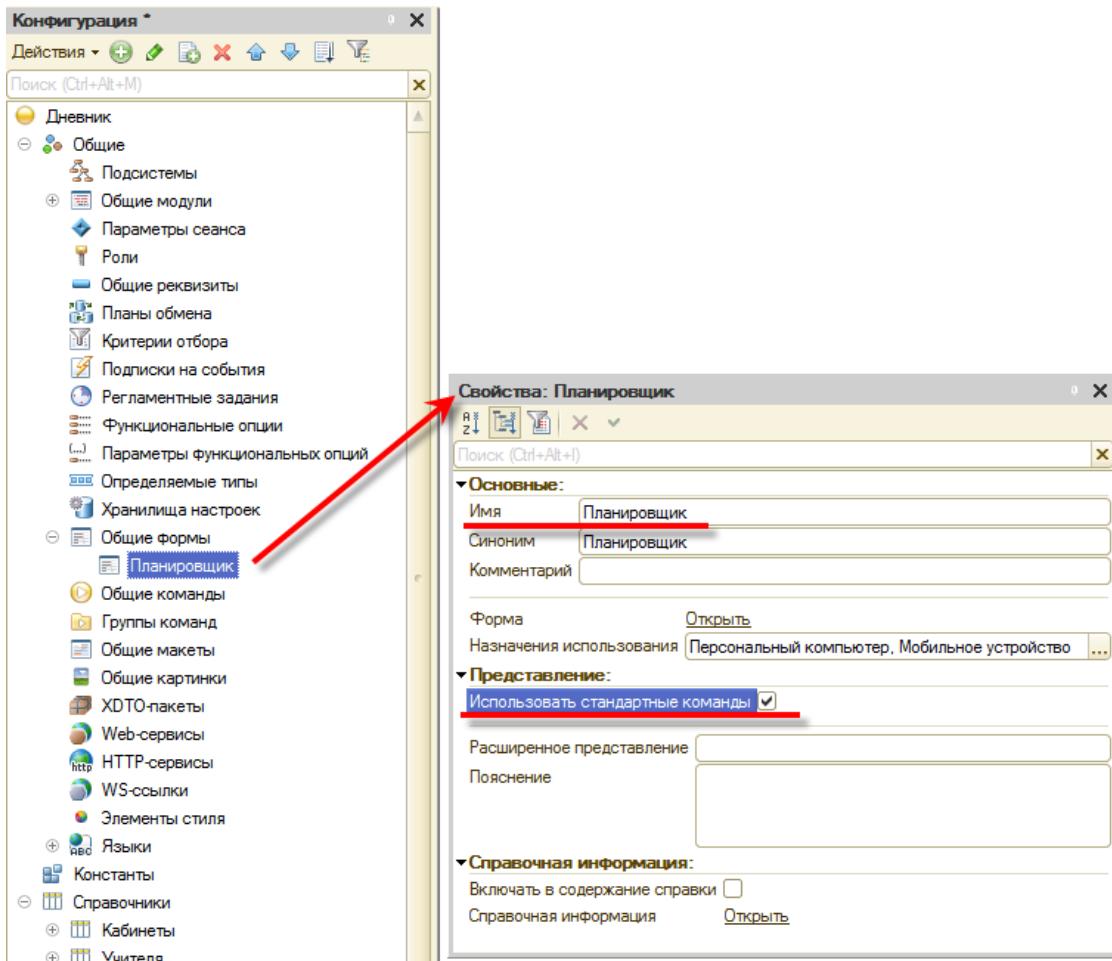


Рисунок 7.8. Свойства формы

Посмотрите, как она выглядит и откуда вызывается в вашем приложении. Запустите 1С:Предприятие в режиме отладки, в группе *Сервис* нажмите *Планировщик*.

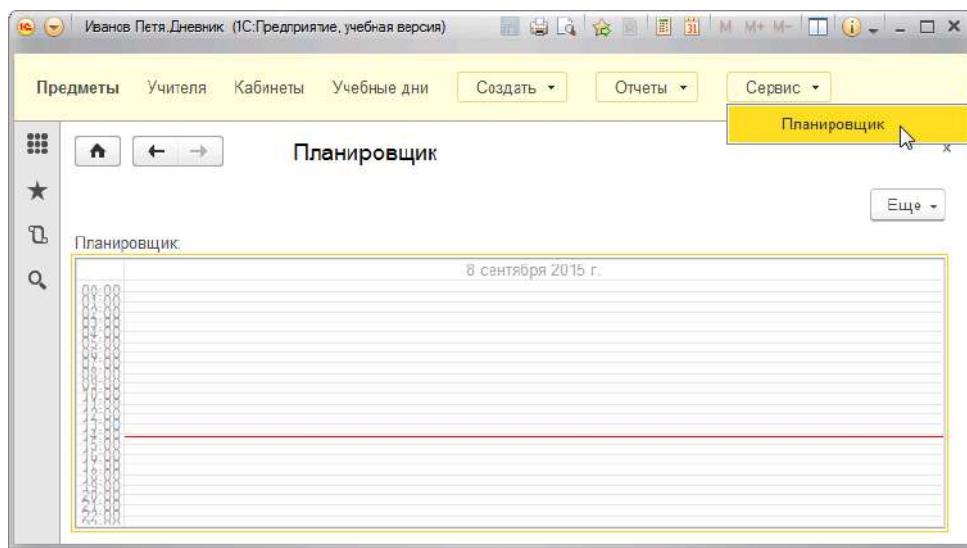


Рисунок 7.9. Форма в режиме 1С:Предприятие

Форма в 1С:Предприятии — это не просто набор визуальных элементов, которые вы видите на экране. Это ещё и объект встроенного языка, который обеспечивает их работу. То есть, когда вы нажали команду *Планировщик*, платформа создала объект встроенного языка *УправляемаяФорма* и показала его на экране.

Как и большинство объектов встроенного языка, форма имеет определённый жизненный цикл и набор событий, которые с этим связаны. Описание типов, связанных с управляемой формой, вы можете найти в синтакс-помощнике в ветке *Интерфейс (управляемый)* — *Управляемая форма* (рисунок 7.10).

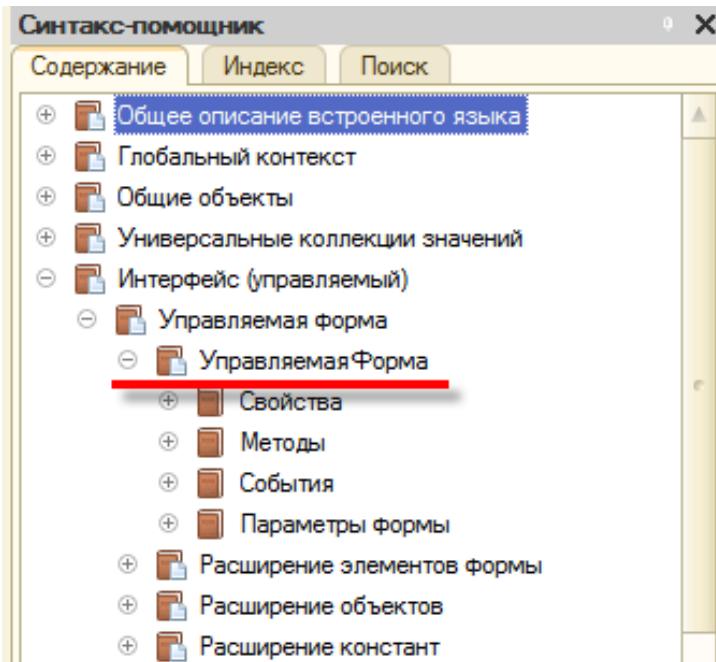


Рисунок 7.10. Тип «УправляемаяФорма»

Сейчас, когда форма уже создана, вам нужно решить, в какой момент заполнять данными планировщик, который находится внутри неё. Для этого вам понадобится кратко познакомиться с событиями управляемой формы.

7.3 События формы

Вообще, работа с формами — это большая, интересная и увлекательная тема. Но эта книга не может «вместить невместимое». Поэтому о формах я расскажу только кратко.

Жизненный цикл формы всегда начинается на сервере. На сервере форма создаётся, заполняется данными, максимально готовится к тому, чтобы отправиться на клиент (рисунок 7.11).

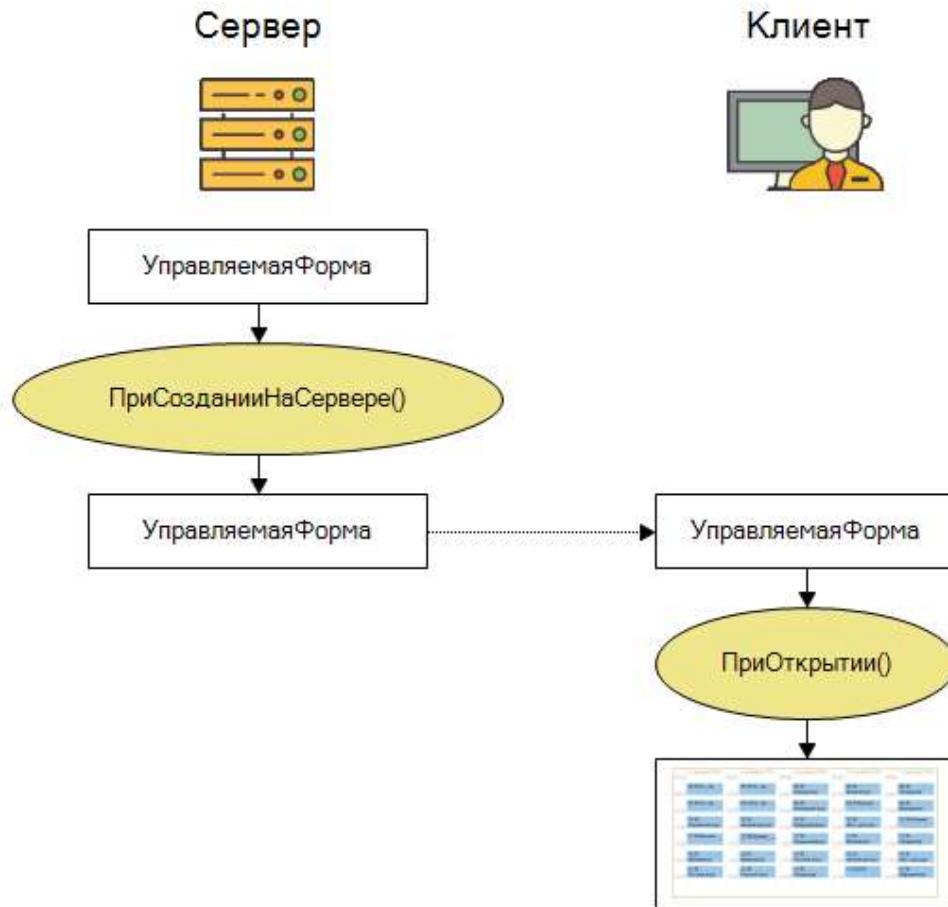


Рисунок 7.11. События при открытии формы

После того как форма оказалась на клиенте, есть возможность настроить какие-то элементы формы, зависящие от клиентского контекста, выдать сообщение пользователю, и тому подобное.

Таким образом, все действия, которые выполняются с формой при её открытии, чётко делятся на две части. Основная часть (заполнение данными, формирование её общего вида) и вспомогательная часть (дополнительная настройка, общение с пользователем).

Основная часть действий выполняется на сервере, вспомогательная может быть выполнена на клиенте. Именно поэтому у формы существуют два события, которые позволяют вам вмешаться в этот процесс. Серверное событие называется *ПриСозданииНаСервере()*, клиентское событие называется *ПриОткрытии()*.

Ваша задача состоит в том, чтобы заполнить планировщик данными. Поэтому, очевидно, выполнять её нужно на сервере, в обработчике события *ПриСозданииНаСервере*.

Подробнее

Подробнее вы можете прочитать про работу с формами в документации «[Руководство разработчика 8.3. Раздел 7.7. Работа с формой из встроенного языка](#)».

7.4 Получение данных из базы

Откройте модуль общей формы *Планировщик* и добавьте обработчик события *ПриСозданииНаСервере* (рисунок 7.12).

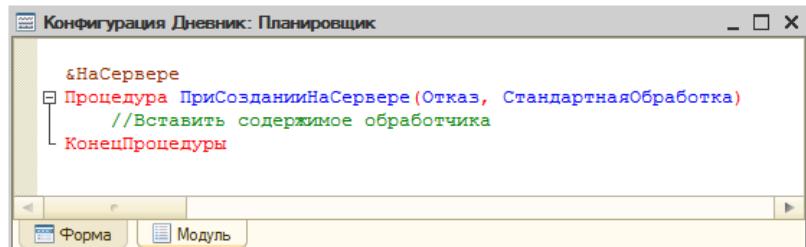


Рисунок 7.12. Заготовка обработчика события «ПриСозданииНаСервере»

Как видите, здесь тоже используются директивы компиляции (`&НаСервере`), с которыми вы уже сталкивались, когда создавали команду заполнения расписания на следующую неделю (смотрите в главе 4 «[Автоматическое заполнение расписания](#)» на странице 356). Форма может «путешествовать» с сервера на клиента, с клиента на сервер. Поэтому в её модуле для каждой процедуры нужно указывать контекст, в котором эта процедура будет исполняться.

Итак, получать данные из базы и заполнять ими планировщик вы будете в обработчике события `ПриСозданииНаСервере`. Но, забегая вперёд, скажу, что делать это вам понадобится не только в момент открытия формы, но и в другие моменты. Позже вы узнаете.

Поэтому алгоритм получения данных и заполнения планировщика лучше сразу же оформить в виде отдельной серверной процедуры. Назовите её `ЗаполнитьПланировщикНаСервере` (рисунок 7.13).

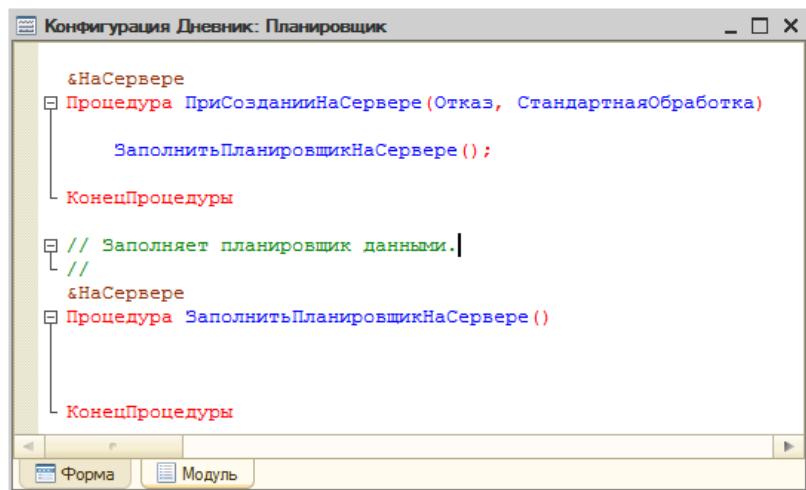


Рисунок 7.13. Процедура для заполнения планировщика

Теперь в этой процедуре вам нужно с помощью запроса получить необходимые данные.

Как составить текст запроса, вы знаете, но как оформить запрос во встроенным языке и обработать его результат — не знаете.

Тут вам поможет *конструктор запроса с обработкой результата*. Он вызывается из контекстного меню в том месте модуля, где вы хотите выполнить запрос (рисунок 7.14).

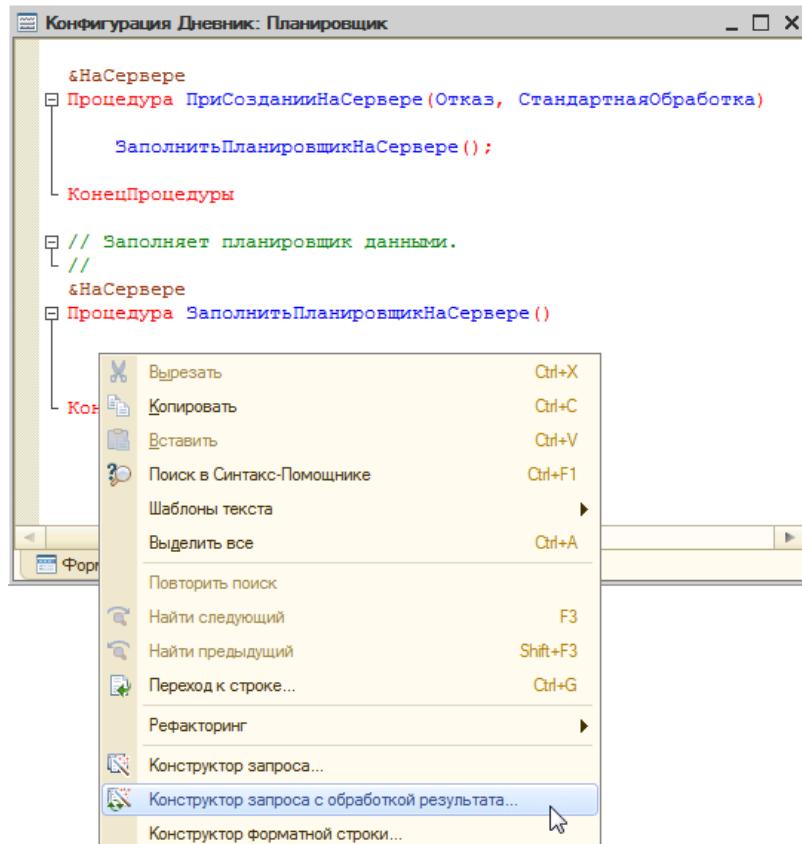


Рисунок 7.14. Вызов конструктора запроса с обработкой результата

Платформа спросит вас, создавать ли новый запрос. Ответьте *Да*.

После этого вы окажетесь на первой закладке конструктора — *Обработка результата* (рисунок 7.15).

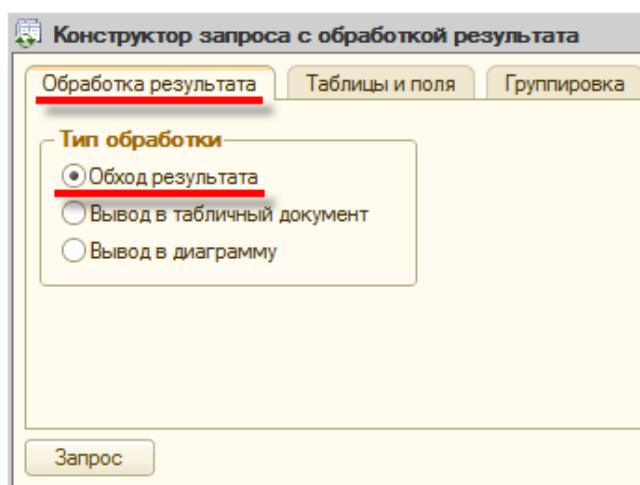


Рисунок 7.15. Закладка «Обработка результата»

Тут есть возможность выбрать вариант, что делать с данными, которые будут прочитаны из базы. Их можно сразу вывести в табличный документ или диаграмму, чтобы показать пользователю. Или вы дальше будете самостоятельно работать с этими данными во встроенным языке (первый вариант).

Первый вариант выбран стандартно, что вам и нужно. Поэтому тут ничего не меняйте и переходите на закладку *Таблицы и поля*. Здесь вы видите уже привычный вам конструктор запросов.

Данные, которые вы хотите показать в планировщике, находятся в табличной части документа УчебныйДень. Как вы помните, табличная часть — это отдельная таблица с данными. Поэтому выберите именно эту таблицу (рисунок 7.16).

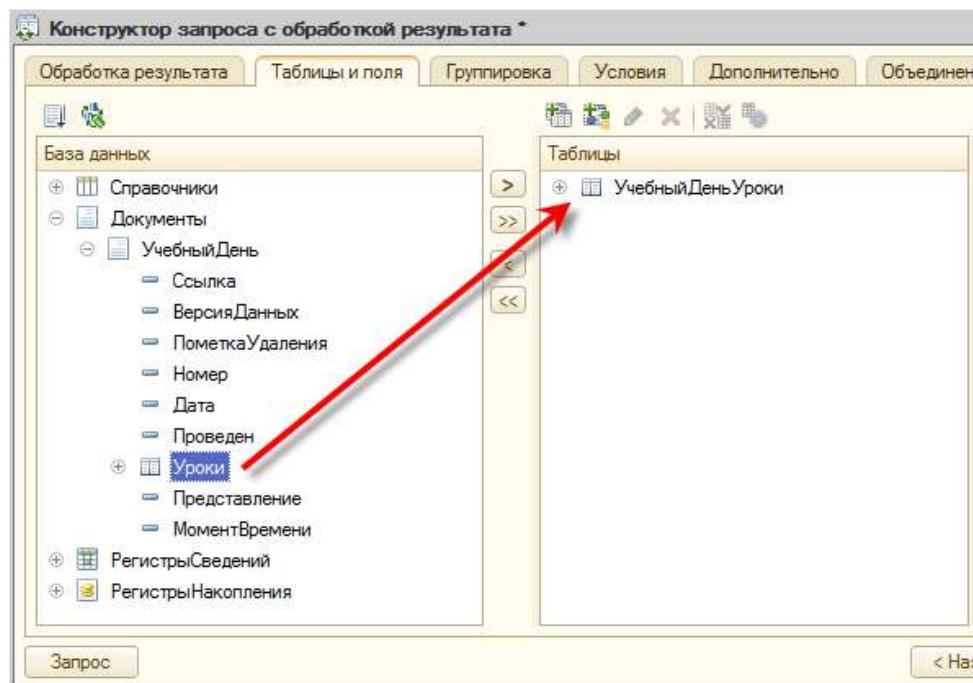


Рисунок 7.16. Табличная часть документа «УчебныйДень»

Что вам понадобится для составления расписания? Дата учебного дня, номер урока и название предмета.

Значит, выберите поля *Ссылка.Дата*, *НомерСтроки* и *Предмет.Представление* (рисунок 7.17).

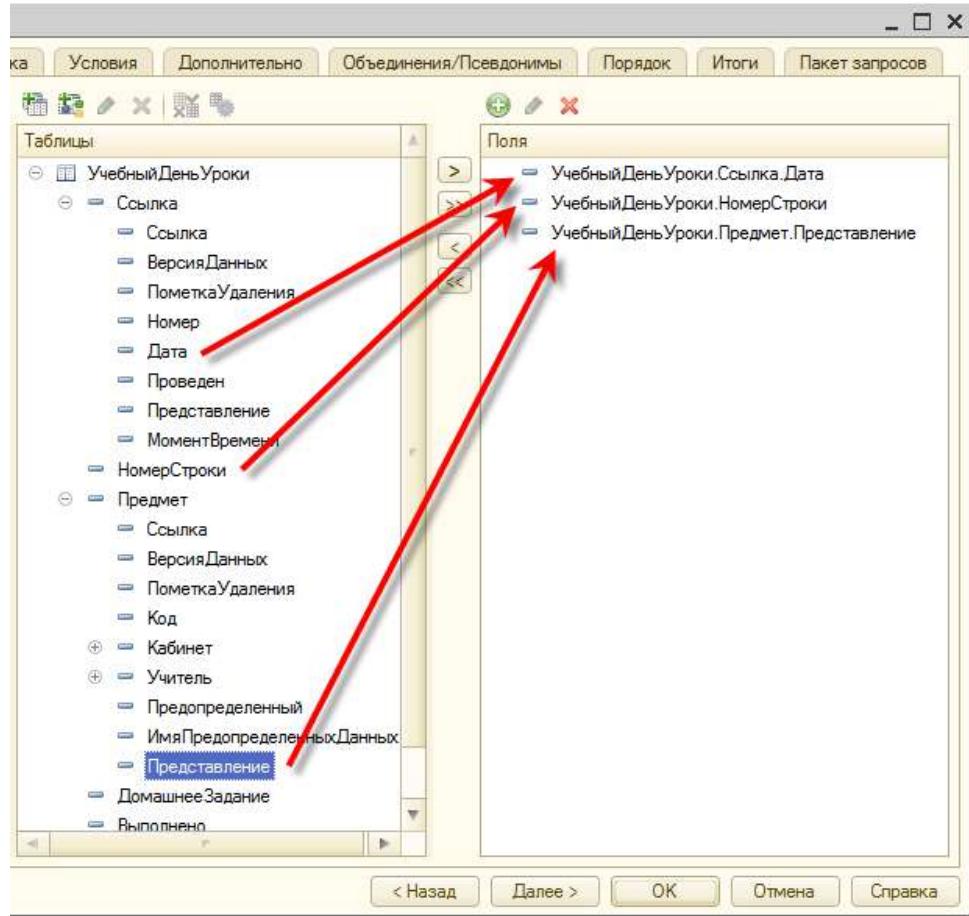


Рисунок 7.17. Выбранные поля

Если вы забыли структуру таблицы, в которой хранится табличная часть (рисунок 6.8), я напомню.

В поле *Ссылка* находится ссылка на документ *УчебныйДень*, которому принадлежат эти записи. Поэтому в поле *Ссылка.Дата* находится дата документа *УчебныйДень*.

Номер строки у вас соответствует порядку уроков в течение дня.

От предмета вы взяли виртуальное поле *Представление*, для того чтобы не зависеть от будущих возможных изменений конфигурации. Сейчас у вас основным представлением предмета является его код. Можно было бы выбрать именно это поле. Но если в будущем вы измените основное представление для этого справочника, то запрос станет работать не совсем правильно. Поэтому лучше взять поле *Представление*, которое платформа сформирует сама. В соответствии с тем, какое основное представление для этого справочника выбрано в дереве конфигурации.

Теперь перейдите на закладку *Объединения/Псевдонимы*. Тут задайте псевдонимы полей выборки, которые будут вам понятны.

Вместо *НомерСтроя* напишите *НомерУрока*, а вместо *ПредметПредставление* — просто *Предмет* (рисунок 7.18).

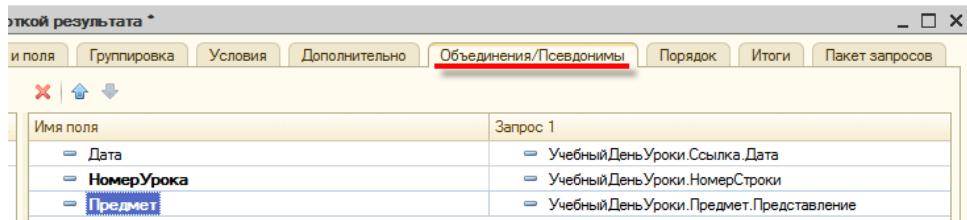


Рисунок 7.18. Псевдонимы полей выборки

В принципе вам всё равно, в какой последовательности будут выбраны данные из базы. Главное — все их перенести в планировщик. Но для порядка всё же перейдите на закладку **Порядок** и укажите, что сначала их нужно отсортировать по дате, а внутри одной и той же даты — по номеру урока (рисунок 7.19).

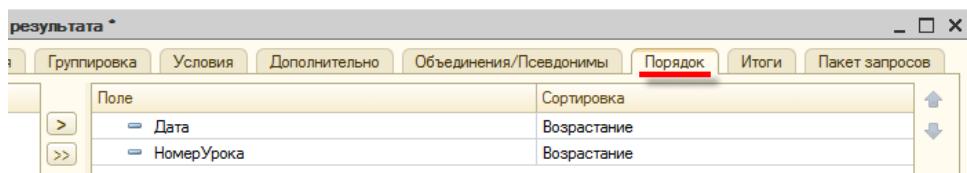


Рисунок 7.19. Порядок полей выборки

Всё, запрос полностью готов, больше ничего не нужно. Нажмите **OK**.

Конструктор вставит в модуль алгоритм формирования, выполнения запроса и обработки его данных (рисунок 7.20).

```
// Заполняет планировщик данными.
// На Сервере
Процедура ЗаполнитьПланировщикНаСервере()

    // {{КОНСТРУКТОР_ЗАПРОСА_С_ОБРАБОТКОЙ_РЕЗУЛЬТАТА
    // Данный фрагмент построен конструктором.
    // При повторном использовании конструктора, внесенные вручную изменения будут утеряны!!!

    Запрос = Новый Запрос;
    Запрос.Текст =
        "ВЫБРАТЬ
        | УчебныйДеньУроки.Ссылка.Дата КАК Дата,
        | УчебныйДеньУроки.НомерСтроки КАК НомерУрока,
        | УчебныйДеньУроки.Предмет.Представление КАК Предмет
        | ИЗ
        |     Документ.УчебныйДень.Уроки КАК УчебныйДеньУроки
        |
        | УПОРЯДОЧИТЬ ПО
        |     Дата,
        |     НомерУрока";

    РезультатЗапроса = Запрос.Выполнить();

    ВыборкаДетальныеЗаписи = РезультатЗапроса.Выбрать();

    Пока ВыборкаДетальныеЗаписи.Следующий() Цикл
        // Вставить обработку выборки ВыборкаДетальныеЗаписи
    КонецЦикла;

    // }}{{КОНСТРУКТОР_ЗАПРОСА_С_ОБРАБОТКОЙ_РЕЗУЛЬТАТА
    |
КонецПроцедуры
```

Рисунок 7.20. Текст, сформированный конструктором

Комментарии в начале и в конце текста вы можете смело удалить. Они носят исключительно информационный характер. После этого у вас останется текст, показанный в листинге 7.1.

Листинг 7.1. Текст, созданный конструктором

```

Запрос = Новый Запрос;
Запрос.Текст = "ВЫБРАТЬ
| УчебныйДеньУроки.Ссылка.Дата КАК Дата,
| УчебныйДеньУроки.НомерСтрочки КАК НомерУрока,
| УчебныйДеньУроки.Предмет.Представление КАК Предмет
| ИЗ
| Документ.УчебныйДень.Уроки КАК УчебныйДеньУроки
|
| УПОРЯДОЧИТЬ ПО
|   Дата,
|   НомерУрока";
РезультатЗапроса = Запрос.Выполнить();
ВыборкаДетальныеЗаписи = РезультатЗапроса.Выбрать();
Пока ВыборкаДетальныеЗаписи.Следующий() Цикл
// Вставить обработку выборки ВыборкаДетальныеЗаписи
КонецЦикла;

```

Сразу, чтобы в дальнейшем не загромождать свою программу длинными именами, измените в двух местах имя переменной *ВыборкаДетальныеЗаписи* на просто *Выборка* (листинг 7.2).

Листинг 7.2. Текст, созданный конструктором

```

Запрос = Новый Запрос;
Запрос.Текст = "ВЫБРАТЬ
| УчебныйДеньУроки.Ссылка.Дата КАК Дата,
| УчебныйДеньУроки.НомерСтрочки КАК НомерУрока,
| УчебныйДеньУроки.Предмет.Представление КАК Предмет
| ИЗ
| Документ.УчебныйДень.Уроки КАК УчебныйДеньУроки
|
| УПОРЯДОЧИТЬ ПО
|   Дата,
|   НомерУрока";
РезультатЗапроса = Запрос.Выполнить();
Выборка = РезультатЗапроса.Выбрать();
Пока Выборка .Следующий() Цикл
// Вставить обработку выборки ВыборкаДетальныеЗаписи
КонецЦикла;

```

Теперь можно познакомиться с тем, как выполняется запрос и как обрабатывать его данные.

Запрос — это объект встроенного языка. Описание его типа вы можете найти в синтакс-помощнике в ветке *Работа с запросами — Выполнение и работа с запросами во встроенным языке* (рисунок 7.21).

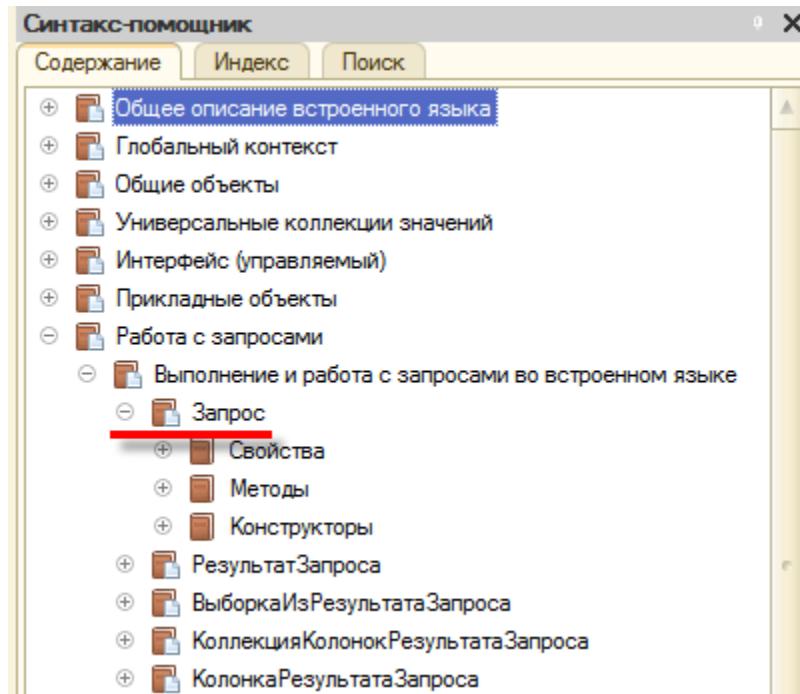


Рисунок 7.21. Тип «Запрос»

Запрос создаётся конструктором. После этого нужно установить его текст, а затем выполнить запрос.

Когда вы делали упражнения с консолью запросов, я показывал вам, как получить текст запроса для вставки в конфигуратор (рисунок 6.38). Именно этот текст и помещается в свойство Текст запроса.

После того как запрос выполнен, вы получаете объект РезультатЗапроса. Этот объект сам по себе практически не используется. Он нужен лишь для того, чтобы решить, каким образом обрабатывать полученные данные дальше.

Способов, по большому счёту, может быть два. Либо вы сразу выгружаете все полученные данные в таблицу и работаете с этой таблицей. Либо вы постепенно, по очереди, обходите все полученные записи, перебирая их.

Чаще всего используется второй способ, поэтому конструктор методом Выбрать() формирует вам такой объект, ВыборкаИзРезультатаЗапроса, позволяющий обходить полученные записи по очереди.

Есть несколько способов обходить то, что получено запросом. Конструктор и вы используете самый простой из них. Когда записи обходятся одна за другой в том порядке, который есть у них в результате выполнения запроса. Поэтому в методе Выбрать() не указываются никакие параметры.

Выборка обходится очень просто, в цикле Пока. Для этого у неё есть метод Следующий(). Если следующий элемент выборки получен, этот метод возвращает значение Истина. И выполняется тело цикла. Как только выборка закончится, метод вернёт Ложь, и цикл перестанет выполняться.

Что вам доступно внутри цикла, вы можете посмотреть прямо сейчас. Установите точку останова на инструкции КонецЦикла и запустите 1С:Предприятие в режиме отладки.

В конфигураторе посмотрите значение переменной Выборка (рисунок 7.22).

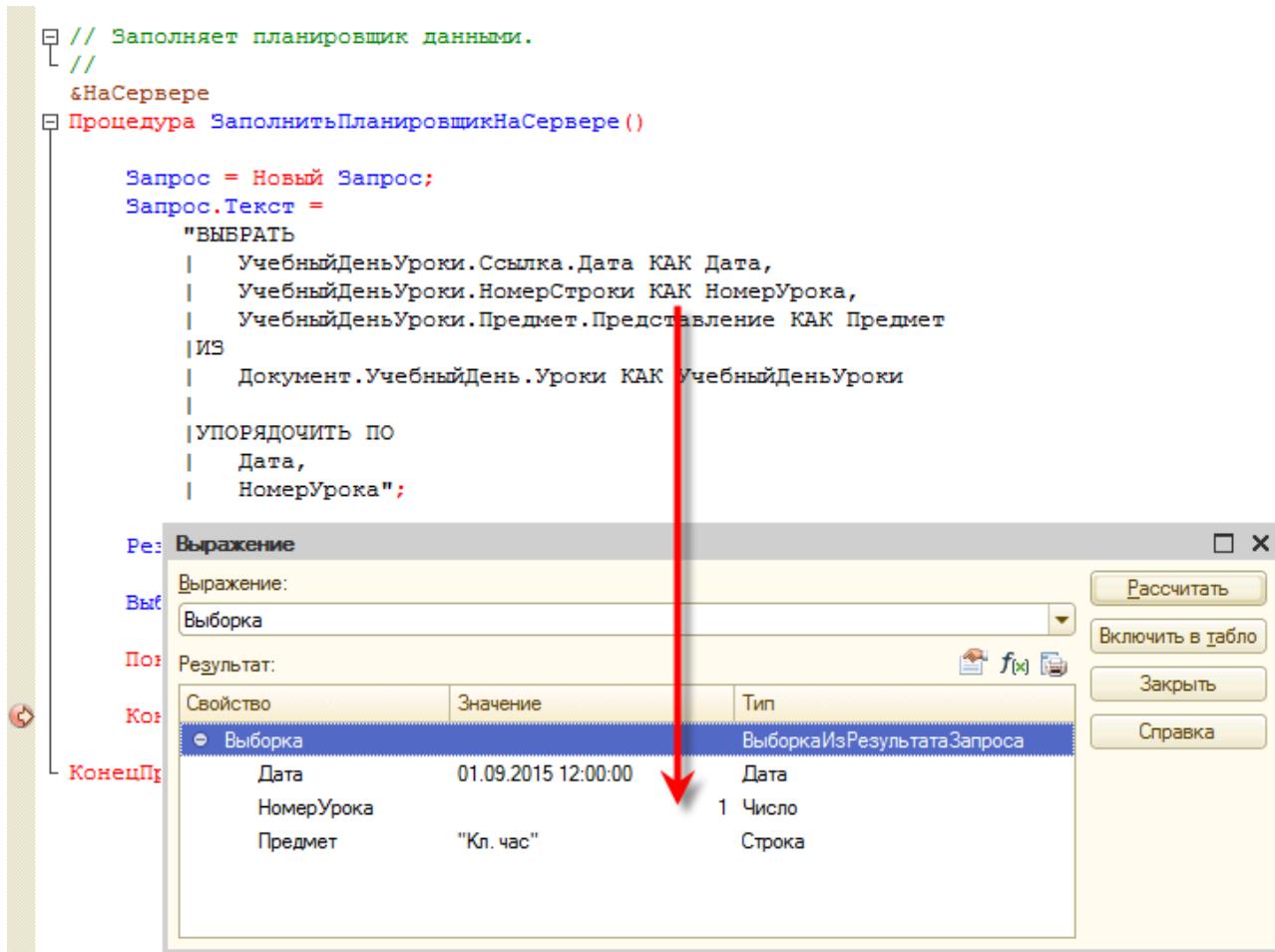


Рисунок 7.22. Поля выборки

На каждом шаге обхода выборки из результата запроса вам доступны значения тех полей, которые вы описали в предложении *ВЫБРАТЬ* запроса. Имена этих полей соответствуют их псевдонимам. Обращаться к ним можно через точку от переменной *Выборка*.

Теперь начните готовить данные для планировщика. В планировщик вы будете заносить уроки, значит, прежде всего нужно вычислить начало урока и конец урока.

Дата урока вам известна. А чтобы получить начало или конец урока, нужно к началу этого дня прибавить столько секунд, сколько их содержится во времени начала или времени окончания.

Эти действия нужно будет выполнять многократно, поэтому лучше вынести их в отдельные процедуры, чтобы не загромождать тело цикла.

Например, дату начала урока вы будете вычислять так (листинг 7.3).

Листинг 7.3. ДатаНачала

```
Запрос = Новый Запрос;
Запрос.Текст = "ВЫБРАТЬ
| УчебныйДеньУроки.Ссылка.Дата КАК Дата,
| УчебныйДеньУроки.НомерСтрочки КАК НомерУрока,
| УчебныйДеньУроки.Предмет.Представление КАК Предмет
| ИЗ
|   Документ.УчебныйДень.Уроки КАК УчебныйДеньУроки
|
| УПОРЯДОЧИТЬ ПО
|   Дата,
|   НомерУрока";
```

```
РезультатЗапроса = Запрос.Выполнить();
Выборка = РезультатЗапроса.Выбрать();
Пока Выборка.Следующий() Цикл
    ДатаНачала = НачалоДня(Выборка.Дата) + НачалоУрока(Выборка.НомерУрока);
КонецЦикла;
```

Вы берёте начало учебного дня и прибавляете к нему количество секунд, которые вам вернёт функция *НачалоУрока()*, которой ещё нет. Чтобы она могла правильно посчитать количество секунд, вы передаёте ей номер урока.

Теперь напишите функцию *НачалоУрока()*. Она будет располагаться в этом же модуле и будет иметь директиву компиляции *&НаСервере* (листинг 7.4).

Листинг 7.4. Функция «НачалоУрока()»

```
&НаСервере
Функция НачалоУрока(НомерУрока)

Если НомерУрока = 1 Тогда
    Возврат 8 * 60 * 60 + 30 * 60;      // 08:30

ИначеЕсли НомерУрока = 2 Тогда
    Возврат 9 * 60 * 60 + 30 * 60;      // 09:30

ИначеЕсли НомерУрока = 3 Тогда
    Возврат 10 * 60 * 60 + 30 * 60;     // 10:30

ИначеЕсли НомерУрока = 4 Тогда
    Возврат 11 * 60 * 60 + 30 * 60;     // 11:30

ИначеЕсли НомерУрока = 5 Тогда
    Возврат 12 * 60 * 60 + 35 * 60;     // 12:35

ИначеЕсли НомерУрока = 6 Тогда
    Возврат 13 * 60 * 60 + 30 * 60;     // 13:30

ИначеЕсли НомерУрока = 7 Тогда
    Возврат 14 * 60 * 60 + 25 * 60;     // 14:25

Иначе
    Возврат 0;

КонецЕсли;
```

КонецФункции

Функция очень простая. В ней «зашито» фиксированное расписание уроков. В «человеческом» виде оно показано в комментариях к каждой ветке условия *Если*.

Первый урок начинается в 8:30. Количество секунд считается так: к 8 часам ($8 * 60 * 60$) прибавляется 30 минут ($30 * 60$).

Теоретически вместо этого выражения можно было бы сразу написать его результат в секундах — 30600. Но если потом понадобится изменить время, сделать это без калькулятора будет очень трудно. А при такой записи достаточно поменять один из элементов формулы.

Для остальных уроков, со 2-го по 7-й, время рассчитывается аналогично. На всякий случай для восьмого и следующих уроков возвращается ноль. Чтобы иметь хоть какую-то определённость в том случае, когда количество уроков окажется больше, чем вы рассчитывали.

Точно таким же образом создайте функцию, которая будет возвращать конец урока. Можете скопировать эту функцию, изменить её имя и время уроков (листинг 7.5).

Листинг 7.5. Функция «КонецУрока()

&НаСервере

Функция КонецУрока(НомерУрока)

```

Если НомерУрока = 1 Тогда
    Возврат 9 * 60 * 60 + 15 * 60;      // 09:15

ИначеЕсли НомерУрока = 2 Тогда
    Возврат 10 * 60 * 60 + 15 * 60;     // 10:15

ИначеЕсли НомерУрока = 3 Тогда
    Возврат 11 * 60 * 60 + 15 * 60;     // 11:15

ИначеЕсли НомерУрока = 4 Тогда
    Возврат 12 * 60 * 60 + 15 * 60;     // 12:15

ИначеЕсли НомерУрока = 5 Тогда
    Возврат 13 * 60 * 60 + 20 * 60;     // 13:20

ИначеЕсли НомерУрока = 6 Тогда
    Возврат 14 * 60 * 60 + 15 * 60;     // 14:15

ИначеЕсли НомерУрока = 7 Тогда
    Возврат 15 * 60 * 60 + 10 * 60;     // 15:10

Иначе
    Возврат 0;

КонецЕсли;

```

КонецФункции

Теперь в цикле, в котором обходится выборка запроса, вы можете вычислить и дату окончания урока (листинг 7.6).

Листинг 7.6. ДатаОкончания

```

Запрос = Новый Запрос;
Запрос.Текст = "ВЫБРАТЬ
    | УчебныйДеньУроки.Ссылка.Дата КАК Дата,
    | УчебныйДеньУроки.НомерСтроки КАК НомерУрока,
    | УчебныйДеньУроки.Предмет.Представление КАК Предмет
    | ИЗ
    |     Документ.УчебныйДень.Уроки КАК УчебныйДеньУроки
    |
    | УПОРЯДОЧИТЬ ПО
    |     Дата,
    |     НомерУрока";

```

РезультатЗапроса = Запрос.Выполнить();

Выборка = РезультатЗапроса.Выбрать();

Пока Выборка.Следующий() Цикл
 ДатаНачала = НачалоДня(Выборка.Дата) + НачалоУрока(Выборка.НомерУрока);
 ДатаОкончания = НачалоДня(Выборка.Дата) + КонецУрока(Выборка.НомерУрока);

КонецЦикла;

Итак, даты начала и окончания у вас есть, название предмета есть. Можно заполнять планировщик.

Сам планировщик находится в реквизите формы. Поэтому к нему можно обращаться прямо по имени этого реквизита — *Планировщик*.

Все данные, которые отображает планировщик, хранятся в его свойстве *Элементы*. Там находится коллекция значений, работа с которой похожа на работу с остальными коллекциями.

Сначала нужно добавить в коллекцию новый элемент, а затем установить его свойства (листинг 7.7).

Листинг 7.7. Добавление элемента в планировщик

```
Запрос = Новый Запрос;
Запрос.Текст = "ВЫБРАТЬ
|   УчебныйДеньУроки.Ссылка.Дата КАК Дата,
|   УчебныйДеньУроки.НомерСтрочки КАК НомерУрока,
|   УчебныйДеньУроки.Предмет.Представление КАК Предмет
|ИЗ
|   Документ.УчебныйДень.Уроки КАК УчебныйДеньУроки
|
|УПОРЯДОЧИТЬ ПО
|   Дата,
|   НомерУрока";
РезультатЗапроса = Запрос.Выполнить();
Выборка = РезультатЗапроса.Выбрать();
Пока Выборка.Следующий() Цикл
    ДатаНачала = НачалоДня(Выборка.Дата) + НачалоУрока(Выборка.НомерУрока);
    ДатаОкончания = НачалоДня(Выборка.Дата) + КонецУрока(Выборка.НомерУрока);
    НовыйЭлемент = Планировщик.Элементы.Добавить(ДатаНачала, ДатаОкончания);
    НовыйЭлемент.Текст = Выборка.Предмет;
КонецЦикла;
```

При добавлении элемента в планировщик нужно обязательно указать его начало и конец на оси времени. А из всех свойств, которые есть у каждого элемента планировщика, вы зададите пока только текст.

Теперь можете запустить 1С:Предприятие в режиме отладки и просмотреть, что получилось (рисунок 7.23).

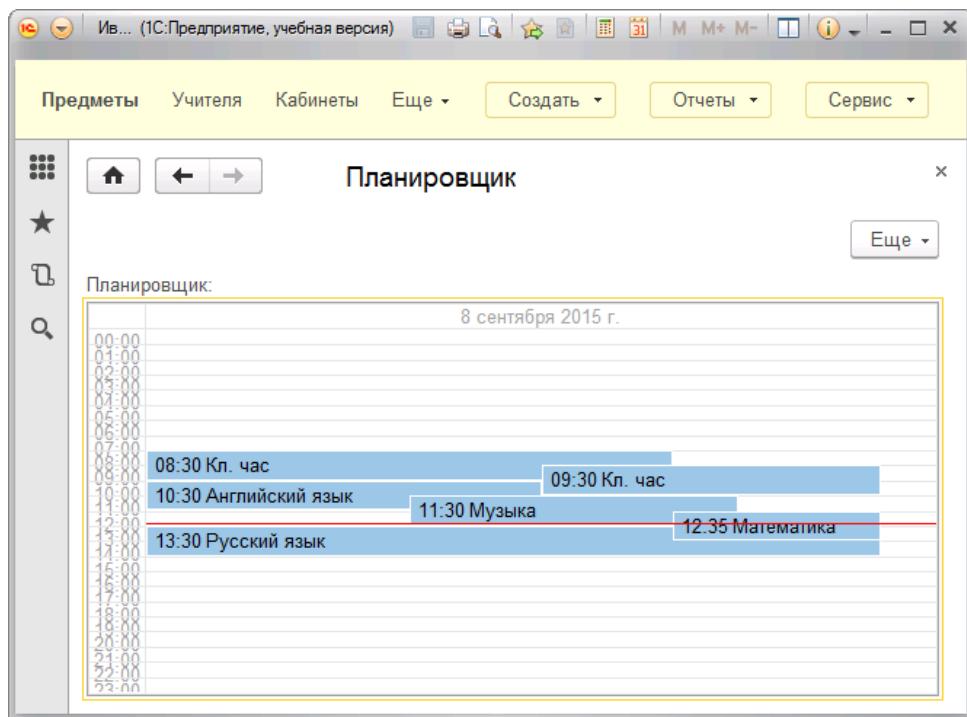


Рисунок 7.23. Планировщик

Пока он выглядит «не очень», но это только начало!

Чтобы вы могли увидеть картинку, похожую на рисунок 7.23, нужно, чтобы в вашей базе данных был документ УчебныйДень для текущего дня. Если такого документа у вас нет, но есть документы за предыдущие дни, вы можете прокрутить планировщик назад. Колесом мыши вверх или клавишей PgUp. Но лучше закройте планировщик, создайте документ для текущего дня и снова откройте планировщик.

7.5 Настройка

Обратите внимание: стандартно планировщик настроен таким образом, что он показывает один день. Вам нужно будет более точно управлять его представлением, поэтому вернитесь в конфигуратор и измените некоторые настройки планировщика.

Настраивать планировщик вы будете уже после того, как заполните его данными. Поэтому откройте процедуру *ПриСозданииНаСервере* и допишите в неё такие строки (листинг 7.8).

Листинг 7.8. Периодичность планировщика в часах

```
&НаСервере
Процедура ПриСозданииНаСервере(Отказ, СтандартнаяОбработка)
```

```
ЗаполнитьПланировщикНаСервере();
// Настроить планировщик.
Планировщик.ЕдиницаПериодическогоВарианта = ТипЕдиницыШкалыВремени.Час;
Планировщик.КратностьПериодическогоВарианта = 24;
```

КонецПроцедуры

Теперь он будет показывать не один день, а двадцать четыре часа.

По сути, это никак не изменит его внешний вид, но зато позволит вам управлять тем, какой промежуток времени показан на экране.

Посмотрите ещё раз на рисунок 7.23. Сейчас день показан с нуля часов до двадцати четырёх. В результате сверху и снизу оказываются большие промежутки, которые ничем не заняты. Хотелось бы их убрать.

И это можно сделать. В этом вам помогут свойства планировщика *ОтступСНачалаПереносаШкалыВремени* и *ОтступСКонцаПереносаШкалыВремени*. Они позволяют отрезать «верхушку» и «конец» отображаемой области. В тех единицах, в которых эта область отображается. А вы только что написали, что отображаться она будет в часах.

Значит, сверху совершенно точно можно «отрезать» восемь часов. Потому что первый урок начинается в 8:30. А снизу можно отрезать 9 часов. То есть ваше расписание будет заканчиваться в 15:00 (листинг 7.9).

Листинг 7.9. Ограничение отображаемой области

```
ЗаполнитьПланировщикНаСервере();  
  
// Настроить планировщик.  
Планировщик.ЕдиницаПериодическогоВарианта = ТипЕдиницыШкалыВремени.Час;  
Планировщик.КратностьПериодическогоВарианта = 24;  
  
Планировщик.ОтступСНачалаПереносаШкалыВремени = 8;  
Планировщик.ОтступСКонцаПереносаШкалыВремени = 9;
```

Запустите 1С:Предприятие в режиме отладки и посмотрите, как это выглядит (рисунок 7.24).

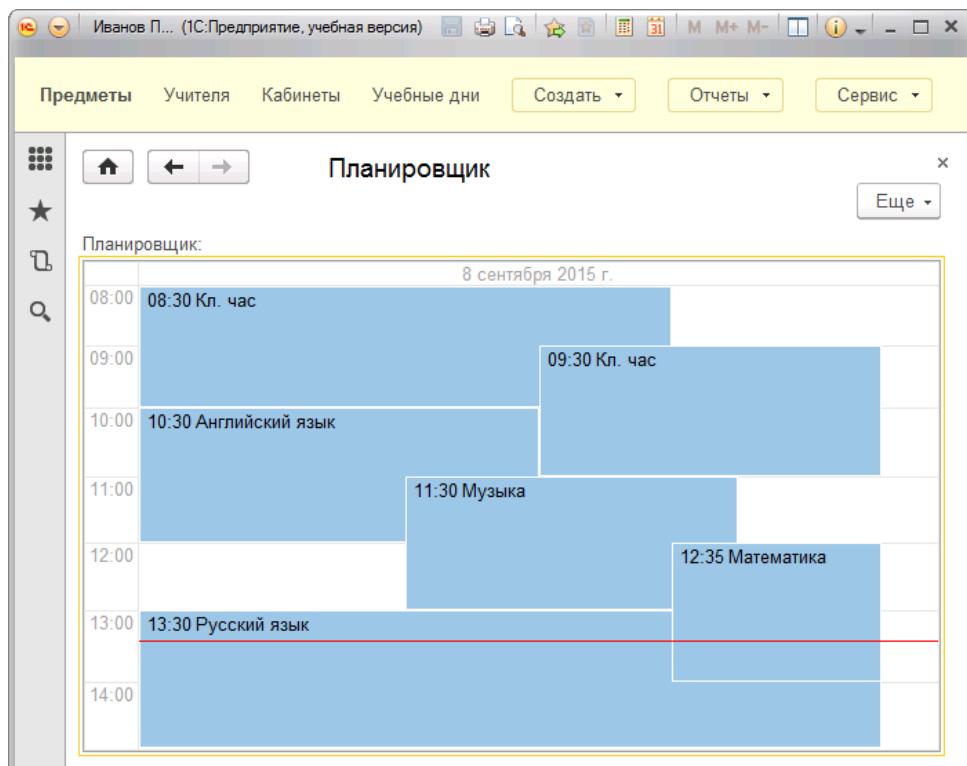


Рисунок 7.24. Планировщик

Определённо стало лучше. Но всё равно есть странность.

Обратите внимание, что элементы планировщика, синие прямоугольники, притянуты к делениям шкалы времени. Например, первый урок начинается в 8:30, но он притянут вверх, к 8:00. Избавьтесь от этого.

Вернитесь в конфигуратор и установите свойство планировщика *ВыравниватьГраницыЭлементовПоШкалеВремени* (листинг 7.10).

Листинг 7.10. Выравнивание границ

```
ЗаполнитьПланировщикНаСервере();  
  
// Настроить планировщик.  
Планировщик.ЕдиницаПериодическогоВарианта = ТипЕдиницыШкалыВремени.Час;  
Планировщик.КратностьПериодическогоВарианта = 24;  
  
Планировщик.ОтступСНачалаПереносаШкалыВремени = 8;  
Планировщик.ОтступСКонцаПереносаШкалыВремени = 9;  
  
Планировщик.ВыравниватьГраницыЭлементовПоШкалеВремени = Ложь;
```

Запустите 1С:Предприятие в режиме отладки и посмотрите, что изменилось (рисунок 7.25).

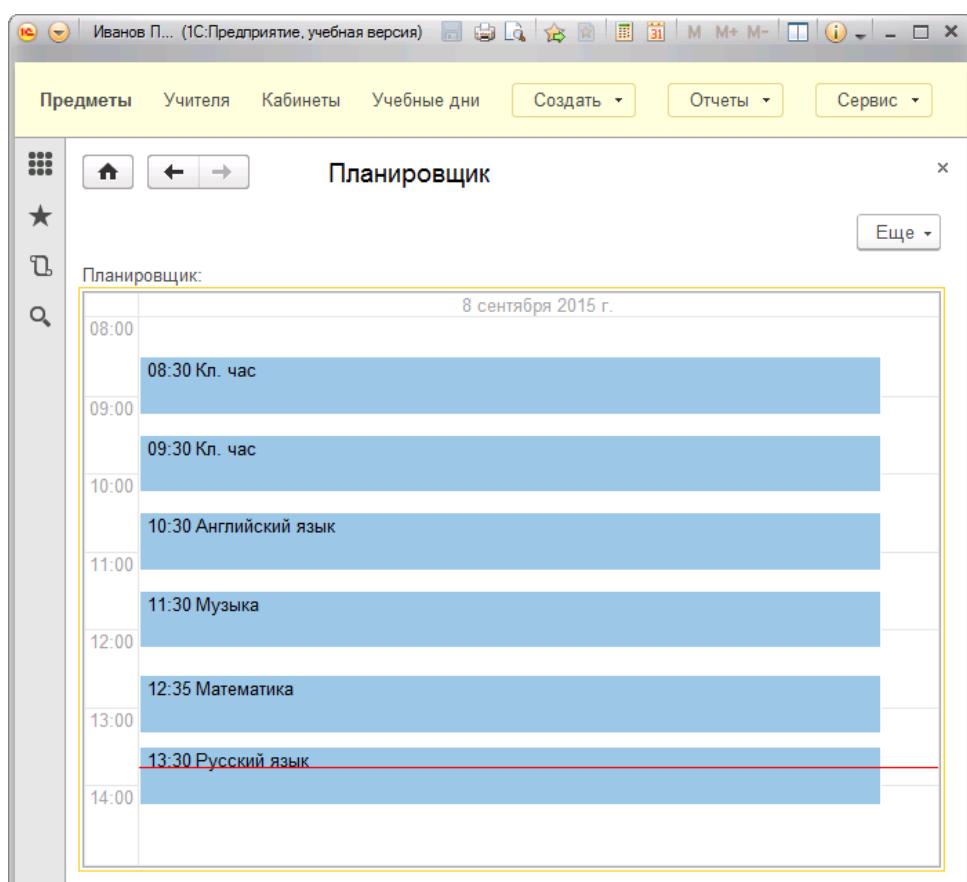


Рисунок 7.25. Планировщик

Так он выглядит гораздо понятнее. Что ещё можно в нём улучшить?

Не очень удобно обозначаются дни. Сейчас непонятно, какой именно день недели на экране: понедельник, вторник, среда? Было бы удобно, чтобы обозначение дня недели находилось рядом с числом.

Это легко поправить. Что-то похожее вы делали в форме документов УчебныйДень (смотрите в разделе 2.14.3 «Изменение формы списка» на страницах 162–165). Вернитесь в конфигуратор и задайте формат заголовков шкалы времени (листинг 7.11).

Листинг 7.11. Формат заголовков шкалы времени

```
ЗаполнитьПланировщикНаСервере();

// Настроить планировщик.
Планировщик.ЕдиницаПериодическогоВарианта = ТипЕдиницыШкалыВремени.Час;
Планировщик.КратностьПериодическогоВарианта = 24;

Планировщик.ОтступСНачалаПереносаШкалыВремени = 8;
Планировщик.ОтступСКонцаПереносаШкалыВремени = 9;

Планировщик.ВыравниватьГраницыЭлементовПоШкалеВремени = Ложь;

Планировщик.ФорматПеренесенныхЗаголовковШкалыВремени = "ДФ='дддд, д ММММ гггг'";
```

Запустите 1С:Предприятие в режиме отладки и посмотрите, что изменилось (рисунок 7.26).

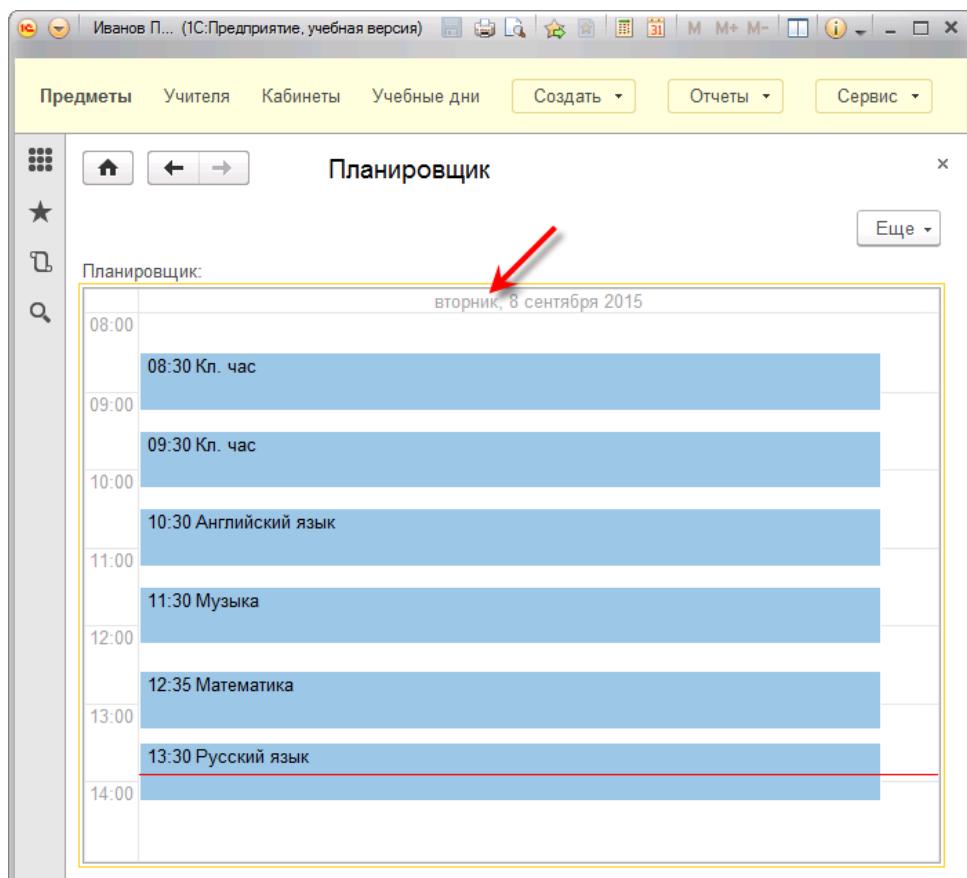


Рисунок 7.26. Планировщик

7.6 Перехват событий

Теперь немного познакомьтесь с тем, как работает планировщик. Если вы нажмёте мышью на любой из элементов, откроется окно быстрого редактирования элемента (рисунок 7.27).

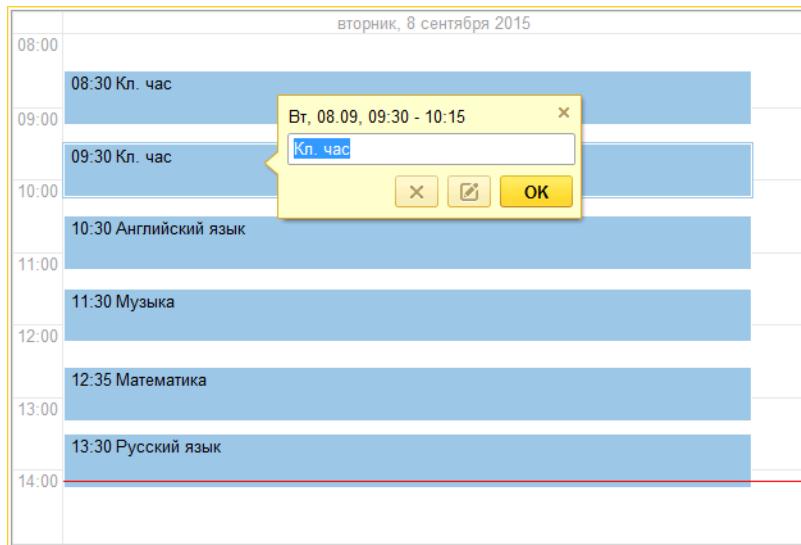


Рисунок 7.27. Окно быстрого редактирования

А если вы дважды щёлкнете на элементе, откроется *форма его редактирования* (рисунок 7.28).

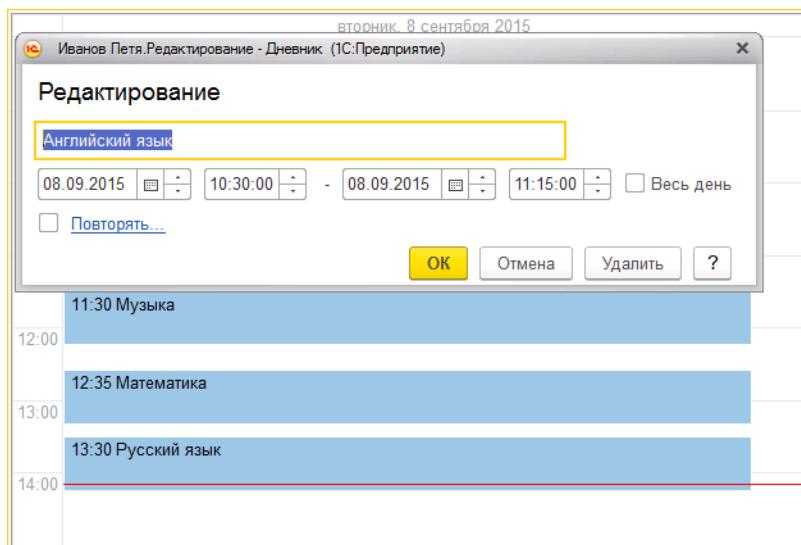


Рисунок 7.28. Форма редактирования

Это удобно, когда вы вручную, интерактивно заполняете планировщик и изменяете его данные. Но у вас другой алгоритм работы.

Заполняется планировщик автоматически, и он должен в точности соответствовать тому, что написано в документах УчебныйДень. Поэтому от быстрого редактирования хотелось бы избавиться совсем. А при двойном щелчке на элементе хотелось бы открывать форму того учебного дня, к которому относится это занятие.

Сделайте это. Перейдите в конфигуратор. В редакторе формы перейдите на закладку *Форма* и посмотрите, какие события есть у элемента формы *Планировщик* (рисунок 7.29).

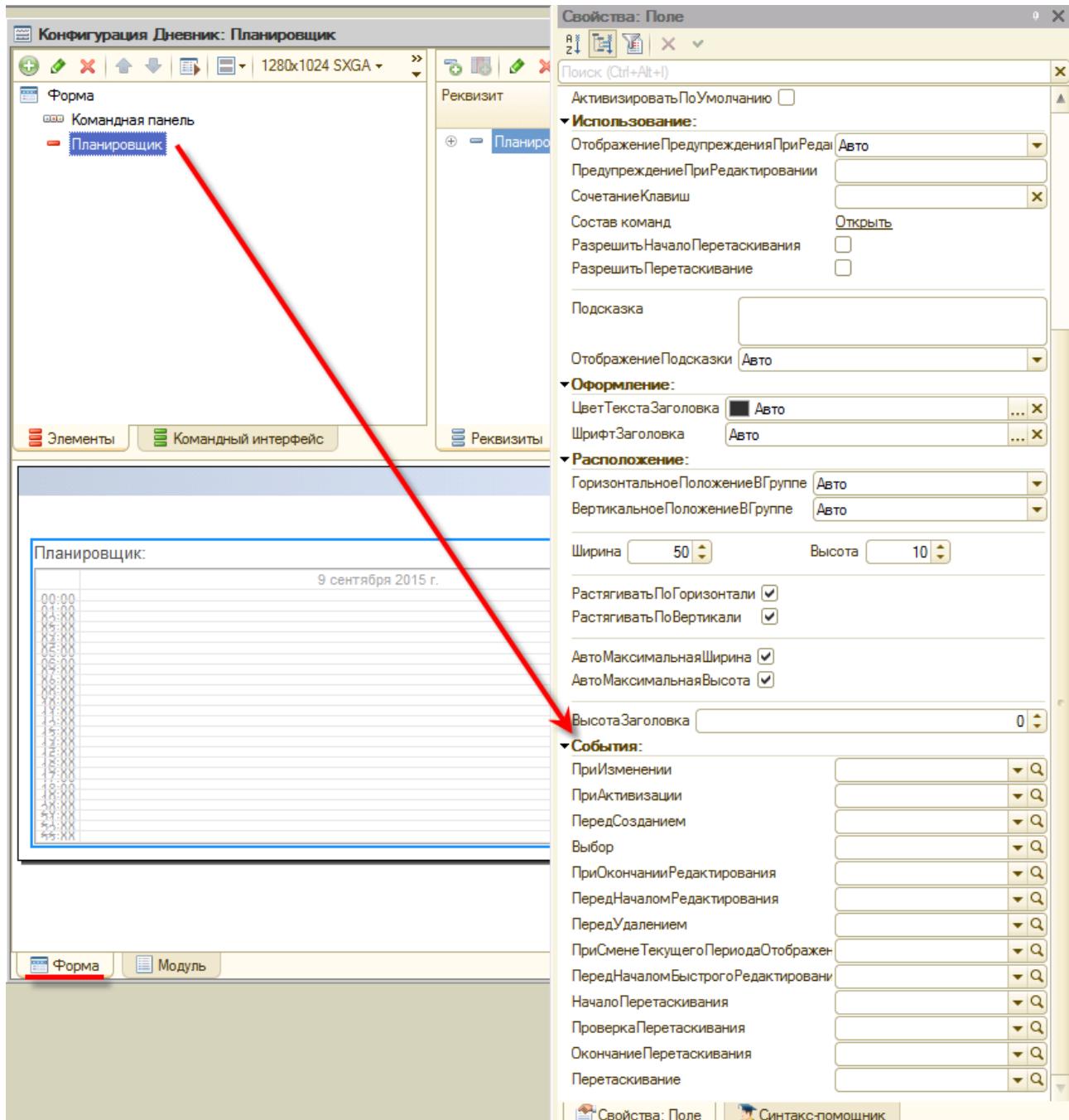


Рисунок 7.29. События поля планировщика

Описание этих же событий вы можете прочитать самостоятельно в синтакс-помощнике в ветке *Интерфейс (управляемый) — Поле формы — Расширение планировщика — События* (рисунок 7.30).

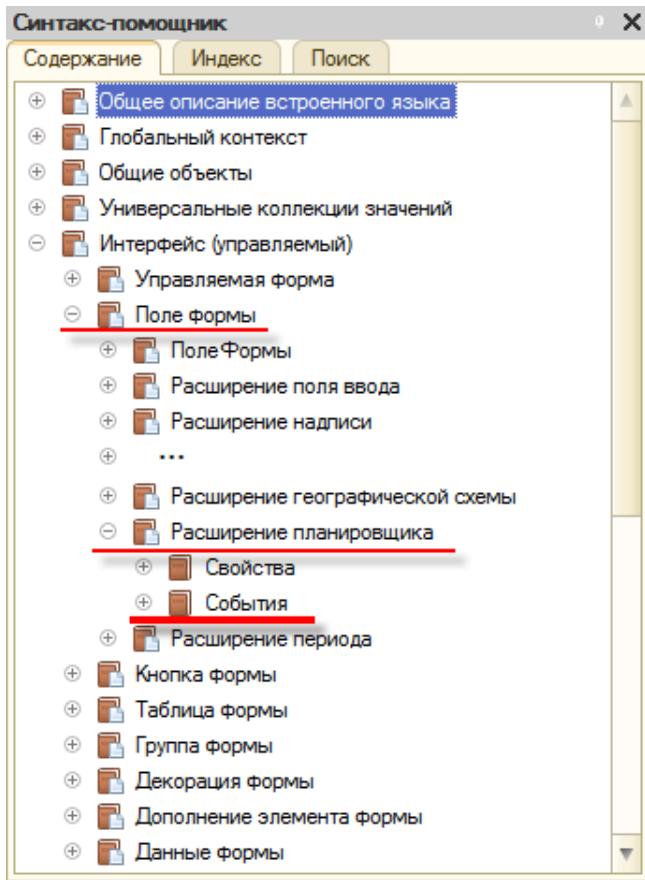


Рисунок 7.30. События расширения поля планировщика

Заметка

В этой книге лишь немного затрагивается программная работа с формами. Но раз вам встретилось слово «расширение», нужно его объяснить.

Дело в том, что в форме могут быть разные элементы: кнопки, поля и другие. У элемента *ПолеФормы* есть набор своих свойств, методов и событий. Они есть у него всегда.

Но поле может отображать разные данные. Например, как в вашем случае, данные планировщика. Тогда к его свойствам, методам и событиям добавляются ещё свойства, методы и события, связанные именно с тем, что внутри поля находятся данные планировщика. Такой набор добавляемых свойств, методов и событий называется *расширение*.

Расширения не являются самостоятельными типами встроенного языка. Но они описаны в синтаксис-помощнике в отдельных ветках. С одной стороны, это отражает внутреннее устройство платформы. А с другой стороны, облегчает поиск в синтаксис-помощнике того, что «добавляется планировщиком».

Итак, вернитесь к редактору формы. Первое, что вам нужно, — избавиться от окна быстрого редактирования. В этом вам поможет событие *ПередНачаломБыстрогоРедактирования*. Нужно создать обработчик этого события.

До сих пор вы имели дело с *фиксированными обработчиками событий*. То есть с обработчиками, у которых имя процедуры, обрабатывающей событие, должно совпадать с именем самого события.

Когда речь идёт об обработчиках событий элементов формы, такой подход не работает.

Вот сейчас вы хотите обработать событие *ПередНачаломБыстрогоРедактирования*. Но написать в модуле формы процедуру с таким именем недостаточно. Потому что у вас в форме может быть не одно, а два поля планировщика. Тогда к какому полю будет относиться этот обработчик?

Поэтому у элементов формы все обработчики *назначаемые*. То есть вы должны сначала выбрать в редакторе формы тот элемент, события которого вас интересуют. А затем в палитре свойств указать, какая процедура будет обрабатывать это событие.

Чтобы упростить вашу работу, платформа предоставляет вам «сервис» по созданию таких обработчиков. В палитре свойств у поля *ПередНачаломБыстрогоРедактирования* нажмите кнопку открытия. Платформа спросит вас, какой обработчик сконструировать (рисунок 7.31).

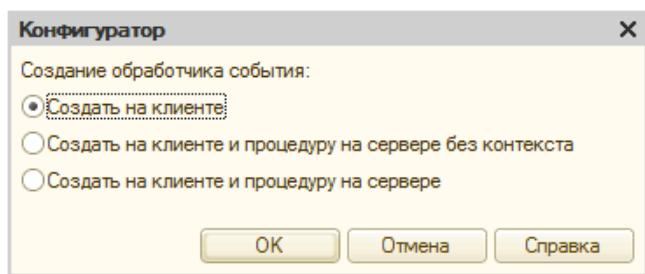


Рисунок 7.31. Создание обработчика

В данном случае вам будет достаточно иметь только клиентский обработчик события, поэтому ничего не меняйте и нажмите *OK*.

Платформа откроет модуль формы и создаст в нём заготовку обработчика события (рисунок 7.32).

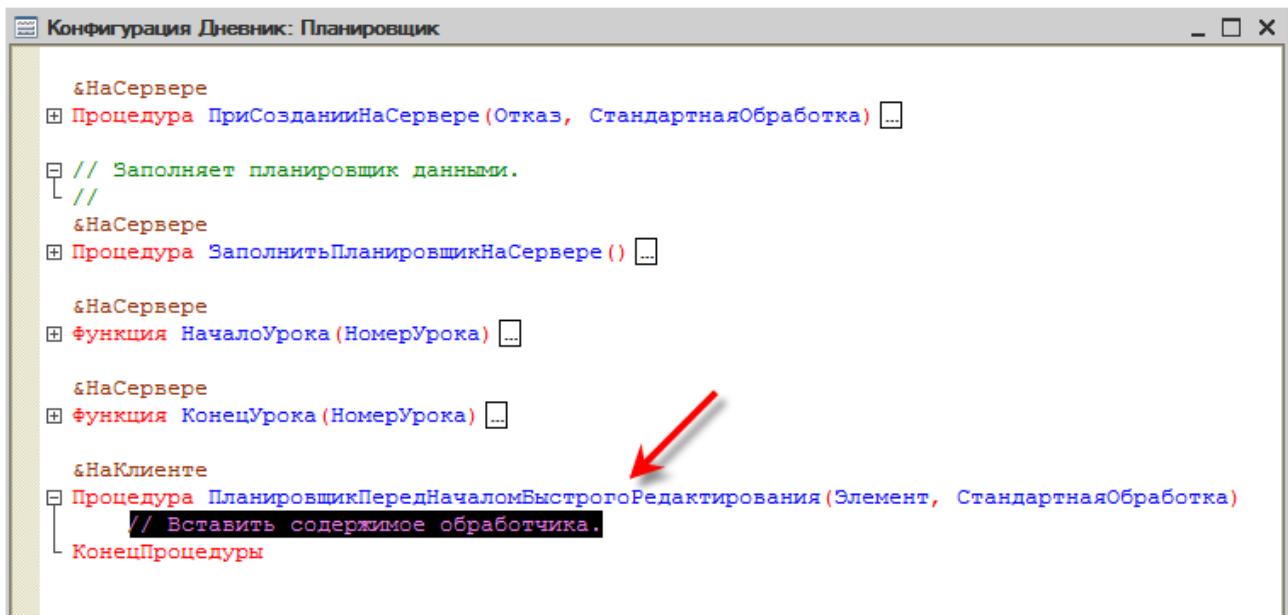


Рисунок 7.32. Заготовка обработчика события

Имя назначаемого обработчика платформа составляет из имени элемента формы и имени события. И одновременно она подставляет эту процедуру в соответствующее свойство элемента управления (рисунок 7.33).

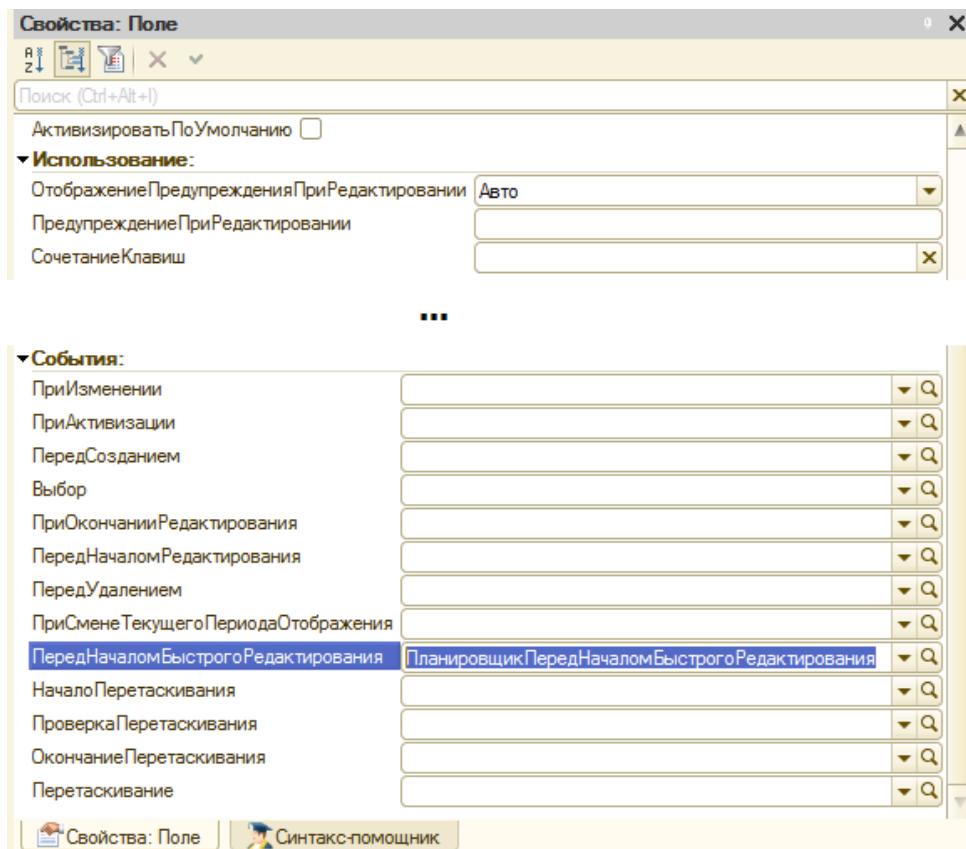


Рисунок 7.33. Имя процедуры в свойстве элемента управления

Таким образом, всё, что вам остаётся, — это написать текст обработчика события.

А текст будет очень простой. Второй параметр этого события, *СтандартнаяОбработка*, как раз управляет тем, будет платформа выполнять свои стандартные действия при наступлении этого события или не будет. А какое стандартное действие платформы на это событие? Правильно, открытие окна быстрого редактирования.

Значит, этому параметру нужно присвоить значение *Ложь* (рисунок 7.34).

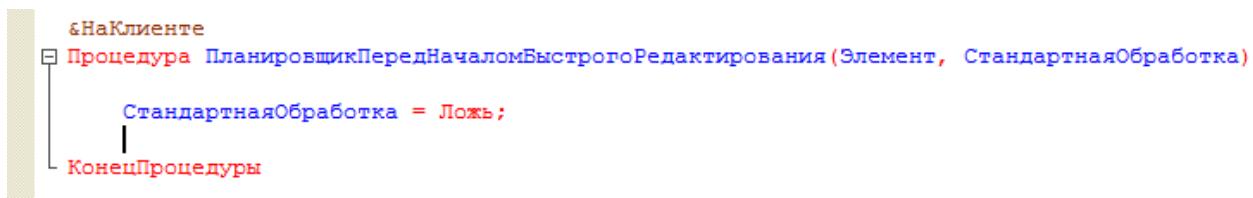


Рисунок 7.34. Отказ от стандартной обработки события

Запустите 1С:Предприятие в режиме отладки. Убедитесь в том, что теперь при одиночном нажатии на элемент планировщика ничего не происходит.

Хорошо, с одним окном вы разобрались. Теперь нужно сделать так, чтобы при двойном нажатии на элементе планировщика открывался учебный день.

Тут будет немного сложнее. Событие, которое отвечает за двойное нажатие, называется *Выбор*. И создать его обработчик не проблема, теперь вы это умеете. Но откуда взять ссылку на учебный день?

Сейчас, когда вы заполняете планировщик, из базы данных вы читаете только дату, номер урока и название предмета. Значит, нужно читать ещё и ссылку на документ *УчебныйДень*. Но это только полдела.

Где в планировщике сохранить эту ссылку? А вот для этого у каждого элемента планировщика есть специальное свойство, которое называется *Значение*. Это свойство как

раз и предназначено для того, чтобы сохранять в нём «источник» данных этого элемента. Например, чтобы к нему можно было перейти из этого элемента планировщика.

Итак, сначала нужно изменить текст вашего запроса в процедуре *ЗаполнитьПланировщикНаСервере*. Сделать это очень просто.

Установите курсор на любое место внутри текста запроса и из контекстного меню вызовите конструктор запроса (рисунок 7.35).

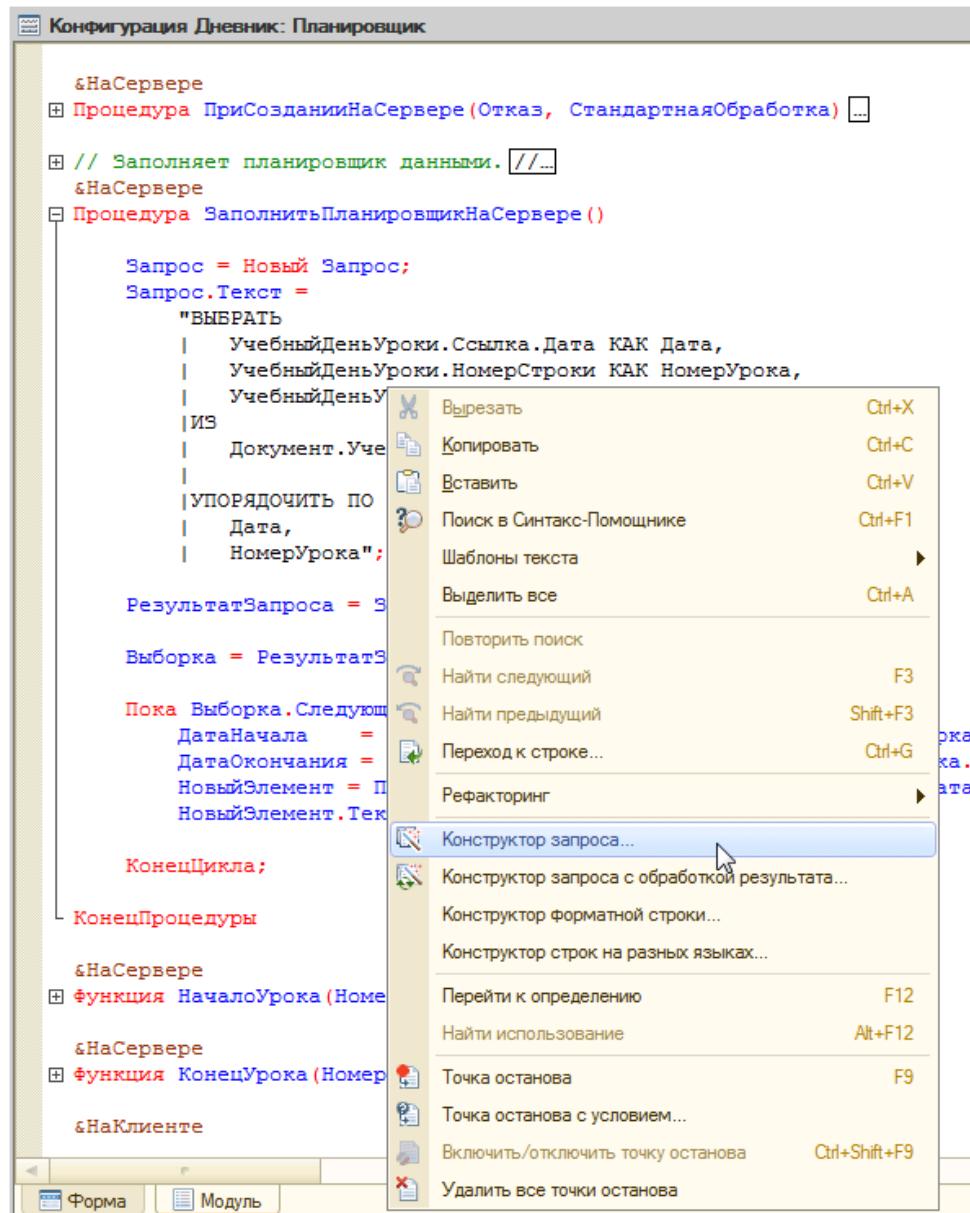


Рисунок 7.35. Вызов конструктора запроса

Платформа «поднимет» текст запроса в конструктор, и вы сможете редактировать его привычным способом.

Выберите из таблицы УчебныйДеньУроки поле Ссылка (это и есть документ УчебныйДень), задайте ему псевдоним УчебныйДень.

Нажмите OK. Изменённый текст запроса платформа вставит обратно (рисунок 7.36).

```

Конфигурация Дневник: Планировщик

&НаСервере
⊕ Процедура ПриСозданииНаСервере (Отказ, СтандартнаяОбработка) [ ]
⊕ // Заполняет планировщик данными. [//...]
&НаСервере
⊖ Процедура ЗаполнитьПланировщикНаСервере ()

Запрос = Новый Запрос;
Запрос.Текст =
    "ВЫБРАТЬ
        | УчебныйДеньУроки.Ссылка.Дата КАК Дата,
        | УчебныйДеньУроки.НомерСтрока КАК НомерУрока,
        | УчебныйДеньУроки.Предмет.Представление КАК Предмет,
        | УчебныйДеньУроки.Ссылка КАК УчебныйДень
    ИЗ
        | Документ.УчебныйДень.Уроки КАК УчебныйДеньУроки
    УПОРЯДОЧИТЬ ПО
        | Дата,
        | НомерУрока";
РезультатЗапроса = Запрос.Выполнить ();
Выборка = РезультатЗапроса.Выбрать ();

```

Рисунок 7.36. Изменённый текст запроса

Теперь вторая часть. Этую ссылку нужно поместить в свойство **Значение** элемента планировщика (листинг 7.12).

Листинг 7.12. Значение элемента планировщика

```

Запрос = Новый Запрос;
Запрос.Текст = "ВЫБРАТЬ
    | УчебныйДеньУроки.Ссылка.Дата КАК Дата,
    | УчебныйДеньУроки.НомерСтрока КАК НомерУрока,
    | УчебныйДеньУроки.Предмет.Представление КАК Предмет,
    | УчебныйДеньУроки.Ссылка КАК УчебныйДень
| ИЗ
    | Документ.УчебныйДень.Уроки КАК УчебныйДеньУроки
|
| УПОРЯДОЧИТЬ ПО
    | Дата,
    | НомерУрока";

РезультатЗапроса = Запрос.Выполнить ();

Выборка = РезультатЗапроса.Выбрать ();

Пока Выборка.Следующий() Цикл
    ДатаНачала      = НачалоДня(Выборка.Дата) + НачалоУрока(Выборка.НомерУрока);
    ДатаОкончания   = НачалоДня(Выборка.Дата) + КонецУрока(Выборка.НомерУрока);
    НовыйЭлемент    = Планировщик.Элементы.Добавить(ДатаНачала, ДатаОкончания);
    НовыйЭлемент.Текст = Выборка.Предмет;
    НовыйЭлемент.Значение = Выборка.УчебныйДень;

КонецЦикла;

```

Теперь можно заняться обработкой события *Выбор*. Создайте обработчик этого события, он тоже будет клиентский.

Прежде всего, в этом событии нужно отказаться от стандартной обработки (листинг 7.13).

Листинг 7.13. Обработчик события «Выбор»

```
СтандартнаяОбработка = Ложь;
```

Затем нужно открыть форму того документа, ссылку на который вы имеете. Делается это так (листинг 7.14).

Листинг 7.14. Открытие формы существующего элемента

```
СтандартнаяОбработка = Ложь;
```

```
ПараметрыФормы = Новый Структура("Ключ", Элемент.ВыделенныеЭлементы[0].Значение);
ОткрытьФорму("Документ.УчебныйДень.ФормаОбъекта", ПараметрыФормы);
```

Чтобы открыть форму, нужно использовать метод глобального контекста *ОткрытьФорму()*.

Нужно сказать, какую именно форму вы хотите открыть. Это указывается в первом параметре метода.

Кроме этого, открывая форму, вы можете захотеть, чтобы она была определённым образом настроена. Например, чтобы она содержала данные какого-то конкретного объекта. Или чтобы в форме списка курсор был установлен сразу на нужную строку. Или ещё что-то.

Для этого у формы есть *параметры*, значения которых вы можете указать при открытии формы.

Как и в случае с полем, параметры есть у самой формы, и есть параметры, которые «добавляются» расширениями. В вашем случае вы открываете форму объекта, и, значит, к параметрам формы добавляются параметры расширения объектов. Среди них есть такой параметр, который называется *Ключ* (рисунок 7.37).

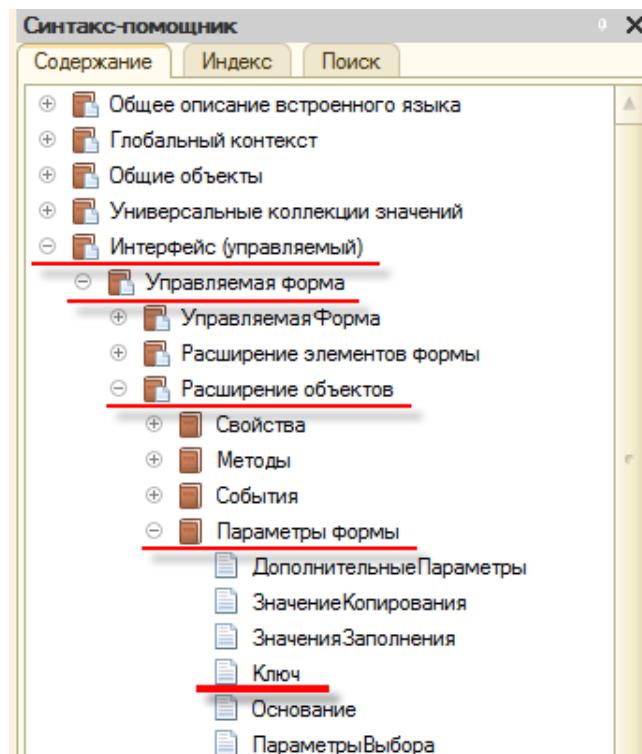


Рисунок 7.37. Параметр «Ключ»

Он как раз и нужен для того, чтобы указать на конкретный объект данных, который должен быть показан в форме.

При открытии формы вы можете захотеть задать несколько параметров. Поэтому все они собираются в объект встроенного языка *Структура*, с которым вы уже знакомы.

Ключ элемента — это имя параметра, а значение элемента — это значение параметра (листинг 7.15).

Листинг 7.15. Структура для параметров формы

```
ПараметрыФормы = Новый Структура("Ключ", Элемент.ВыделенныеЭлементы[0].Значение);
```

Значение ссылки «достаётся» следующим образом. В обработчик события *Выбор* передаётся тот элемент формы, для которого вызвано это событие, *ПолеФормы*. У поля формы есть коллекция *ВыделенныеЭлементы*. В ней находится один элемент, на котором установлен курсор, или несколько элементов поля, которые выделены.

В вашем случае при двойном клике на элементе планировщика в этой коллекции будет один элемент. Тот, на котором выполнено нажатие. Ну, а ссылка на учебный день находится в свойстве *Значение* этого элемента.

Наверняка это сложно воспринимается на слух, поэтому можете посмотреть, как это выглядит на самом деле. Установите точку останова на инструкции *ОткрытьФорму()*. Запустите 1С:Предприятие в режиме отладки, откройте планировщик и дважды щёлкните на любом его элементе.

После этого в конфигураторе посмотрите значение переменной *Элемент* и состав коллекции, находящейся в свойстве *ВыделенныеЭлементы* (рисунок 7.38).

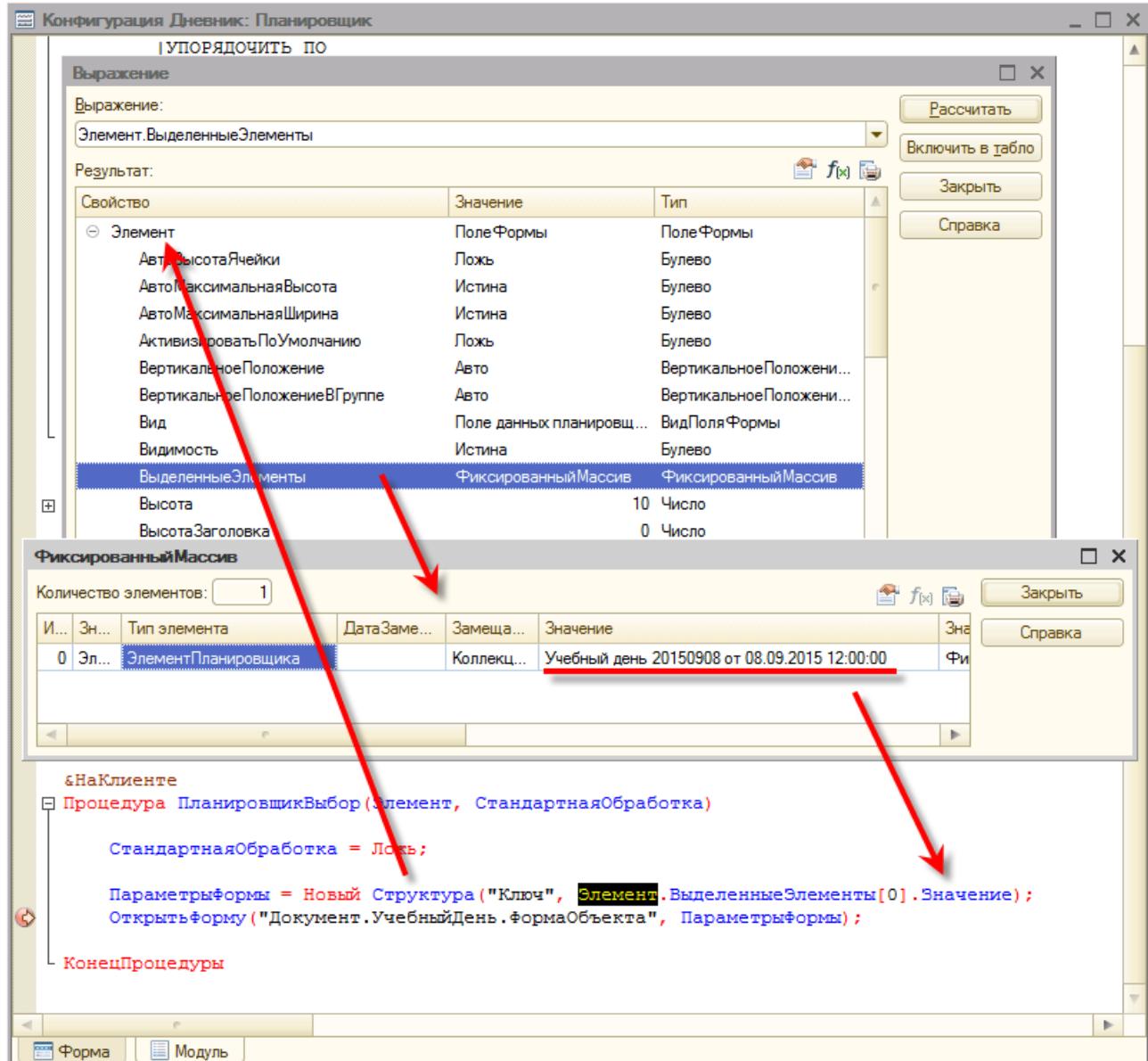


Рисунок 7.38. Выделенные элементы планировщика в конфигураторе

После этого закройте окна вычисления выражений, уберите точку останова и продолжите отладку.

В приложении откроется форма учебного дня. Это то, что вам и было нужно.

Чтобы ещё раз увидеть это уже без остановок, закройте учебный день и снова дважды нажмите на любом элементе планировщика.

7.7 Отображение будних дней

В самом начале я говорил, что в расписании занятий должна отображаться целая неделя. Пока у вас отображается только один день. Сейчас вы это исправите.

У планировщика есть интересное свойство, которым вы пока не пользовались. Планировщику можно указать, какой именно отрезок временной оси нужно отображать. Причём таких отрезков может быть несколько.

Сейчас это свойство у него установлено в стандартное значение, и он отображает только текущий день. Но ему можно сказать, например, что нужно отобразить понедельник,

среду и пятницу. Тогда на экране вы увидите только эти три дня.

С этим же свойством связана другая интересная способность планировщика. Сейчас он отображает у вас 24 часа. Помните, вы настраивали это в начале (листинг 7.16).

Листинг 7.16. Кратность единиц

```
Планировщик.ЕдиницаПериодическогоВарианта = ТипЕдиницыШкалыВремени.Час;
Планировщик.КратностьПериодическогоВарианта = 24;
```

То есть «единицей» отображения является час, а «кратность» — 24.

Так вот, если задать отображаемый интервал планировщика больше, чем «кратность единиц» (в данном случае один день), он начнёт переносить временную шкалу так, чтобы все части отображаемого интервала были видны на экране. В вашем случае он расположит несколько дней рядом друг с другом. Попробуйте.

Отображаемые интервалы содержатся в свойстве планировщика *ТекущиеПериодыОтображения*. Они представляют собой коллекцию значений. Каждое из этих значений — это один интервал, который нужно отобразить.

Сейчас в этой коллекции один элемент, который указывает на текущий день. Вы можете сами убедиться в этом, если поставите точку останова на конец процедуры *ПриСозданииНаСервере* и посмотрите значение этого свойства планировщика (рисунок 7.39).

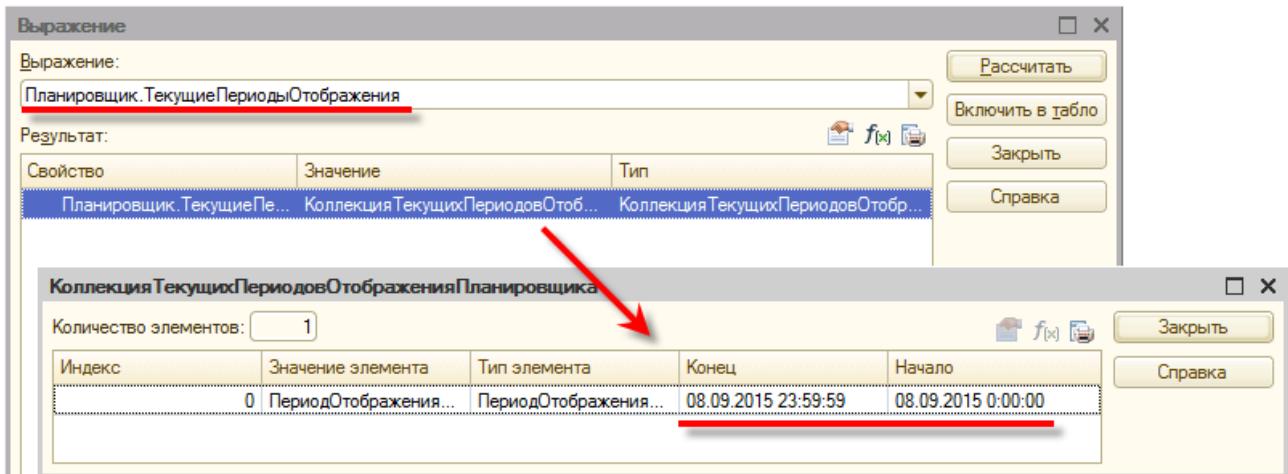


Рисунок 7.39. Текущие периоды отображения

Сделайте следующее. Удалите тот период, который есть в коллекции сейчас, и добавьте в эту коллекцию новый период, который будет длиться с начала по конец текущей недели (листинг 7.17).

Листинг 7.17. Добавление нового периода

```
&НаСервере
Процедура ПриСозданииНаСервере(Отказ, СтандартнаяОбработка)
```

```
ЗаполнитьПланировщикНаСервере();
// Настроить планировщик.
Планировщик.ЕдиницаПериодическогоВарианта = ТипЕдиницыШкалыВремени.Час;
Планировщик.КратностьПериодическогоВарианта = 24;

Планировщик.ОтступСНачалаПереносаШкалыВремени = 8;
Планировщик.ОтступСКонцаПереносаШкалыВремени = 9;

Планировщик.ВыравниватьГраницыЭлементовПоШкалеВремени = Ложь;

Планировщик.ФорматПеренесенныхЗаголовковШкалыВремени = "ДФ='дддд, д ММММ гггг'";

Планировщик.ТекущиеПериодыОтображения.Очистить();
НачалоПериода = НачалоНедели(ТекущаяДата());
КонецПериода = КонецНедели(ТекущаяДата());
Планировщик.ТекущиеПериодыОтображения.Добавить(НачалоПериода, КонецПериода);
```

КонецПроцедуры

Запустите 1С:Предприятие и посмотрите, как теперь выглядит планировщик (рисунок 7.40).

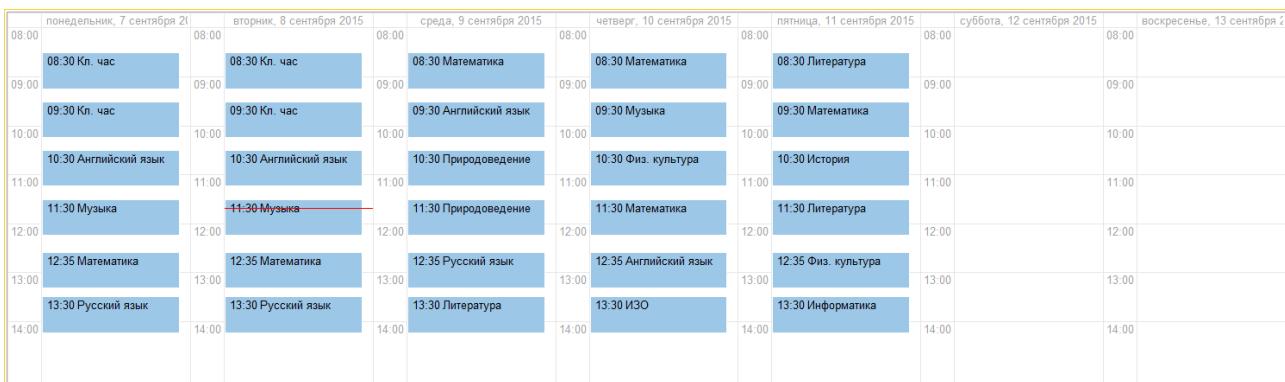


Рисунок 7.40. Неделя в планировщике

Отлично. Это уже почти похоже на то, что хотелось с самого начала. Почти похоже — потому что суббота и воскресенье вас не интересуют. В эти дни нет занятий, а значит, они всегда будут пустые. Хорошо бы их убрать с экрана.

Сделать это просто. От конца недели, который вам известен, нужно отнять количество секунд, которое содержится в двух днях (листинг 7.18).

Листинг 7.18. Отображение рабочих дней текущей недели

&НаСервере

Процедура ПриСозданииНаСервере(Отказ, СтандартнаяОбработка)

```

ЗаполнитьПланировщикНаСервере();
// Настроить планировщик.
Планировщик.ЕдиницаПериодическогоВарианта = ТипЕдиницыШкалыВремени.Час;
Планировщик.КратностьПериодическогоВарианта = 24;

Планировщик.ОтступСНачалаПереносаШкалыВремени = 8;
Планировщик.ОтступСКонцаПереносаШкалыВремени = 9;

Планировщик.ВыравниватьГраницыЭлементовПоШкалеВремени = Ложь;

Планировщик.ФорматПеренесенныхЗаголовковШкалыВремени = "ДФ='дддд, д ММММ гггг'";
Планировщик.ТекущиеПериодыОтображения.Очистить();
НачалоПериода = НачалоНедели(ТекущаяДата());
КонецПериода = КонецНедели(ТекущаяДата()) - 2 * 24 * 60 * 60;
Планировщик.ТекущиеПериодыОтображения.Добавить(НачалоПериода, КонецПериода);

```

КонецПроцедуры

Запустите 1С:Предприятие и посмотрите, как теперь выглядит планировщик (рисунок 7.41).

понедельник, 7 сентября 2015		вторник, 8 сентября 2015		среда, 9 сентября 2015		четверг, 10 сентября 2015		пятница, 11 сентября 2015	
08:00	08:00	08:00	08:00	08:00	08:00	08:00	08:00	08:00	
08:30 Кл. час		08:30 Кл. час		08:30 Математика		08:30 Математика		08:30 Литература	
09:00	09:00	09:00	09:00	09:30 Английский язык		09:30 Музыка		09:30 Математика	
09:30 Кл. час		09:30 Кл. час		09:30 Английский язык		09:30 Музыка		09:30 Математика	
10:00	10:00	10:00	10:00	10:30 Природоведение		10:30 Физ. культура		10:30 История	
10:30 Английский язык		10:30 Английский язык		10:30 Природоведение		10:30 Физ. культура		10:30 История	
11:00	11:00	11:00	11:00	11:30 Природоведение		11:30 Математика		11:30 Литература	
11:30 Музыка		11:30 Музыка		11:30 Природоведение		11:30 Математика		11:30 Литература	
12:00	12:00	12:00	12:00	12:35 Русский язык		12:35 Английский язык		12:35 Физ. культура	
12:35 Математика		12:35 Математика		12:35 Русский язык		12:35 Английский язык		12:35 Физ. культура	
13:00	13:00	13:00	13:00	13:30 Литература		13:30 ИЗО		13:30 Информатика	
13:30 Русский язык		13:30 Русский язык		13:30 Литература		13:30 ИЗО		13:30 Информатика	
14:00	14:00	14:00	14:00						

Рисунок 7.41. Рабочие дни недели в планировщике

Здорово, это то, что и хотелось!

Но есть один нюанс. Попробуйте прокрутить планировщик назад или вперёд. Вы увидите, что вся «красота» тут же ломается и снова появляются пустые субботы и воскресенья (рисунок 7.42).

среда, 2 сентября 2015	четверг, 3 сентября 2015	пятница, 4 сентября 2015	суббота, 5 сентября 2015	воскресенье, 6 сентября 2015
08:00 08:30 Математика	08:00 08:30 Математика	08:00 08:30 Литература	08:00	08:00
09:00 09:30 Английский язык	09:00 09:30 Музыка	09:00 09:30 Математика	09:00	09:00
10:00 10:30 Природоведение	10:00 10:30 Физ. культура	10:00 10:30 История	10:00	10:00
11:00 11:30 Природоведение	11:00 11:30 Математика	11:00 11:30 Литература	11:00	11:00
12:00 12:35 Русский язык	12:00 12:35 Английский язык	12:00 12:35 Физ. культура	12:00	12:00
13:00 13:30 Литература	13:00 13:30 ИЗО	13:00 13:30 Информатика	13:00	13:00
14:00	14:00	14:00	14:00	14:00

Рисунок 7.42. Субботы и воскресенья в планировщике

Это нехорошо, но с этим можно побороться, если знать, в чём проблема. А проблема заключается в том, что после того, как вы прокрутили планировщик вперёд или назад, он самостоятельно формирует период отображения исходя из того, как он это понимает.

Он же не знает, что вы хотите всегда исключать субботы и воскресенья. Но он знает, что вы хотите видеть сразу пять дней. Поэтому он подставляет вам предыдущие пять дней или следующие пять дней. В зависимости от того, в какую сторону вы крутите.

Как с этим бороться? Нужно вместо того периода, который стандартно подставляет разработчик, подставлять свой период, который содержит только дни с понедельника по пятницу. И у планировщика как раз есть событие, которое происходит при его прокручивании. Называется оно *ПриСменеТекущегоПериодаОтображения*.

Перейдите в конфигуратор и создайте клиентский обработчик этого события. Прежде всего в этом обработчике нужно отказаться от стандартной обработки. Потому что теперь вы сами будете делать то, что нужно (листинг 7.19).

Листинг 7.19. Обработчик «ПриСменеТекущегоПериодаОтображения»

```
&НаКлиенте
Процедура ПланировщикПриСменеТекущегоПериодаОтображения(Элемент, ТекущиеПериодыОтображения,
СтандартнаяОбработка)
```

```
СтандартнаяОбработка = Ложь;
```

```
КонецПроцедуры
```

Теперь обратите внимание на второй параметр этого обработчика, который называется *ТекущиеПериодыОтображения*. В этом параметре содержится тот набор периодов, который планировщик собрался вам показывать, после того как вы его прокрутите. То есть в вашем случае там будут предыдущие пять дней или следующие пять дней.

Что нужно сделать? Нужно вместо этого периода подставить свой. Но как его вычислить? Тут есть хитрость.

За основу для вычислений можно взять тот период, который вам собрался показывать планировщик. Вернее, конец этого периода.

Почему именно конец? Да потому, что конец всегда будет находиться в предыдущей или следующей неделе. А вот начало — не всегда.

Если вы крутите назад, то будут показаны дни прошлой недели со среды по воскресенье. А вот если вы крутите вперёд, то началом будет суббота этой недели, а концом — среда следующей недели.

Итак, удаляйте период, который собрался показывать планировщик, и добавляйте свой (листинг 7.20).

Листинг 7.20. Свой период отображения при прокручивании планировщика

&НаКлиенте

Процедура ПланировщикПриСменеТекущегоПериодаОтображения(Элемент, ТекущиеПериодыОтображения, СтандартнаяОбработка)

```
СтандартнаяОбработка = Ложь;
```

```
НовыйПериод = ТекущиеПериодыОтображения[0];
```

```
Планировщик.ТекущиеПериодыОтображения.Очистить();
```

```
НачалоПериода = НачалоНедели(НовыйПериод.Конец);
```

```
КонецПериода = КонецНедели(НовыйПериод.Конец) - 2 * 24 * 60 * 60;
```

```
Планировщик.ТекущиеПериодыОтображения.Добавить(НачалоПериода, КонецПериода);
```

КонецПроцедуры

Запустите 1С:Предприятие в режиме отладки и посмотрите теперь, как прокручивается планировщик. Теперь он прокручивается так, как надо, показывая только будние дни.

И раз уж теперь можно его прокручивать вперёд и назад, сделайте так, чтобы текущая неделя, отображаемая планировщиком, была хорошо видна.

Напишите её в заголовке формы, а надпись *Планировщик*, которая сейчас над полем, уберите (рисунок 7.43).

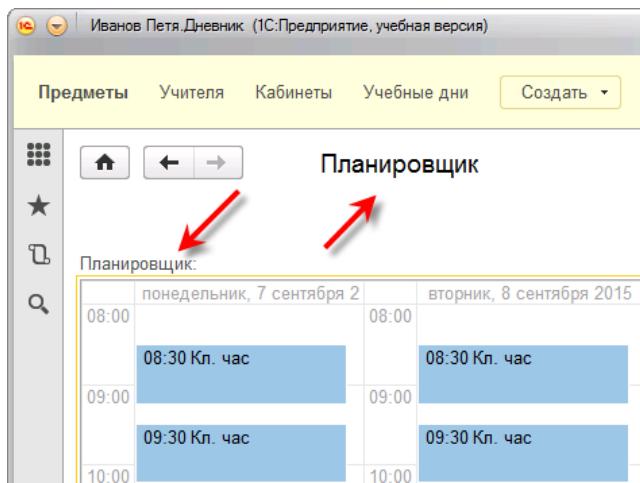


Рисунок 7.43. Заголовок формы и надпись над полем планировщика

Также можно убрать командную панель формы, потому что в ней сейчас только кнопка *Ещё*, а для вашего планировщика она не нужна (рисунок 7.44).

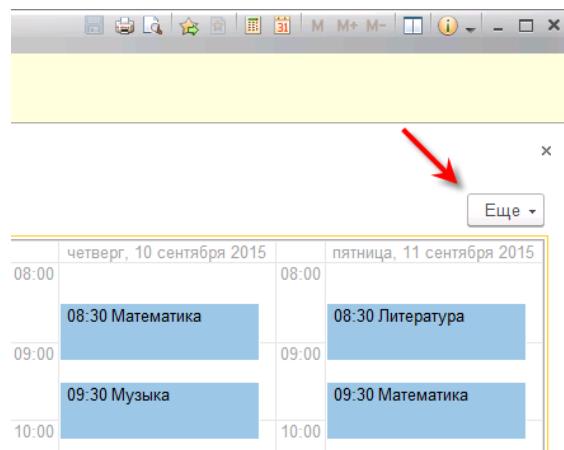


Рисунок 7.44. Командная панель формы

Перейдите в конфигуратор и откройте свойства элемента формы *Планировщик*. Его свойство *ПоложениеЗаголовка* установите в значение *Нет*.

После этого откройте свойства командной панели и сбросьте флаг у свойства *Автозаполнение*.

Теперь откройте свойства корневого элемента *Форма* и сбросьте флаг у свойства *Автозаголовок*.

Заголовок формы вы будете формировать самостоятельно. Каждый раз, как меняется период отображения.

Первый раз это нужно сделать, когда форма создаётся, при открытии формы на сервере (листинг 7.21).

Листинг 7.21. Заголовок формы при создании формы

```
&НаСервере
Процедура ПриСозданииНаСервере(Отказ, СтандартнаяОбработка)

ЗаполнитьПланировщикНаСервере();
// Настроить планировщик.
Планировщик.ЕдиницаПериодическогоВарианта = ТипЕдиницыШкалыВремени.Час;
Планировщик.КратностьПериодическогоВарианта = 24;

Планировщик.ОтступСНачалаПереносаШкалыВремени = 8;
Планировщик.ОтступСКонцаПереносаШкалыВремени = 9;

Планировщик.ВыравниватьГраницыЭлементовПоШкалеВремени = Ложь;
Планировщик.ФорматПеренесенныхЗаголовковШкалыВремени = "ДФ='дддд, д ММММ гггг'";
Планировщик.ТекущиеПериодыОтображения.Очистить();
НачалоПериода = НачалоНедели(ТекущаяДата());
КонецПериода = КонецНедели(ТекущаяДата()) - 2 * 24 * 60 * 60;
Планировщик.ТекущиеПериодыОтображения.Добавить(НачалоПериода, КонецПериода);

Заголовок = "Расписание занятий " + ПредставлениеПериода(НачалоПериода, КонецПериода);
КонецПроцедуры
```

Заголовок формы будет состоять из надписи *Расписание занятий* и периода, который отображается. Для его формирования в виде строки можете воспользоваться готовой

функцией глобального контекста `ПредставлениеПериода()`. Ей нужно передать только начало и конец периода в виде даты.

И ещё один раз, когда нужно установить заголовок формы, — это момент прокрутки планировщика. То есть в обработчике события `ПриСменеТекущегоПериодаОтображения` нужно добавить такую же строку (листинг 7.22).

Листинг 7.22. Заголовок формы при прокручивании планировщика

&НаКлиенте

Процедура ПланировщикПриСменеТекущегоПериодаОтображения(Элемент, ТекущиеПериодыОтображения, СтандартнаяОбработка)

СтандартнаяОбработка = Ложь;

НовыйПериод = ТекущиеПериодыОтображения[0];

Планировщик.ТекущиеПериодыОтображения.Очистить();

НачалоПериода = НачалоНедели(НовыйПериод.Конец);

КонецПериода = КонецНедели(НовыйПериод.Конец) - 2 * 24 * 60 * 60;

Планировщик.ТекущиеПериодыОтображения.Добавить(НачалоПериода, КонецПериода);

Заголовок = "Расписание занятий " + ПредставлениеПериода(НачалоПериода, КонецПериода);

КонецПроцедуры

Запустите 1С:Предприятие в режиме отладки и посмотрите теперь, как выглядит и как прокручивается планировщик (рисунок 7.45).



Рисунок 7.45. Отображаемый период в заголовке формы

7.8 Отметки оценок и домашние задания

Теперь ваше расписание занятий очень похоже на тот вариант, который был в самом начале этой главы на рисунке 7.2. Но нет предела совершенству!

Помните, я говорил, что на рисунке 7.2 специально показан простейший вариант. А у вас он будет красивее и удобнее. Поэтому смотрите, что ещё можно улучшить в этом варианте расписания.

Сейчас все занятия отображаются у вас прямоугольниками одинакового цвета. А горизонтальная красная линия показывает текущее время компьютера. Благодаря этому вы понимаете, где находится «точка актуальности» вашего расписания.

Но если вы начнёте прокручивать расписание вперёд или назад, то ориентироваться станет сложнее. Понять, в какой части расписания вы находитесь (в будущей или в прошлой), вы сможете только по заголовку формы. Это не очень удобно.

Уроки, которые уже прошли, и уроки, которые ещё будут, отличаются принципиально. Когда вы просматриваете прошедшие уроки, вас интересует такая информация, как полученные оценки или комментарии учителя. А когда вы просматриваете будущие уроки, вас интересует совсем другая информация. Например, задано что-то по этому уроку или нет.

Поэтому первое, что вы сделаете, — измените внешний вид тех уроков, которые только планируются. Сейчас все уроки имеют стандартный цвет, и у них нет рамки. Для будущих уроков вы сделаете рамку и закрасите их белым цветом.

Откройте в конфигураторе процедуру, в которой вы заполняете планировщик данными, *ЗаполнитьПланировщикНаСервере()*. В цикле, после того как вы задали текст и значение элемента планировщика, добавьте инструкцию *Если*. Она будет анализировать, где находится дата начала урока: в прошлом или в будущем (листинг 7.23).

Листинг 7.23. Анализ даты урока

```

&НаСервере
Процедура ЗаполнитьПланировщикНаСервере()
{
    Запрос = Новый Запрос;
    Запрос.Текст = "ВЫБРАТЬ
        | УчебныйДеньУроки.Ссылка.Дата КАК Дата,
        | УчебныйДеньУроки.НомерСтроки КАК НомерУрока,
        | УчебныйДеньУроки.Предмет.Представление КАК Предмет,
        | УчебныйДеньУроки.Ссылка КАК УчебныйДень
    | ИЗ
        | Документ.УчебныйДень.Уроки КАК УчебныйДеньУроки
    |
    | УПОРЯДОЧИТЬ ПО
        | Дата,
        | НомерУрока";
}

РезультатЗапроса = Запрос.Выполнить();

Выборка = РезультатЗапроса.Выбрать();

Пока Выборка.Следующий() Цикл
    ДатаНачала = НачалоДня(Выборка.Дата) + НачалоУрока(Выборка.НомерУрока);
    ДатаОкончания = НачалоДня(Выборка.Дата) + КонецУрока(Выборка.НомерУрока);
    НовыйЭлемент = Планировщик.Элементы.Добавить(ДатаНачала, ДатаОкончания);
    НовыйЭлемент.Текст = Выборка.Предмет;
    НовыйЭлемент.Значение = Выборка.УчебныйДень;

    Если ДатаНачала >= ТекущаяДата() Тогда // будущие занятия
        Иначе // прошедшие занятия
    КонецЕсли;

КонецЦикла;
}

```

Сразу напишите обе ветки условия, потому что позже вам понадобится и та, и другая.

Прошедшие занятия будут иметь у вас стандартное оформление. Поэтому в ветке *Иначе* ничего писать не надо. А будущим занятиям нужно задать цвет рамки и цвет фона (листинг 7.24).

Листинг 7.24. Цвет фона и рамка для будущих уроков

```
Запрос = Новый Запрос;
Запрос.Текст = "ВЫБРАТЬ
|   УчебныйДеньУроки.Ссылка.Дата КАК Дата,
|   УчебныйДеньУроки.НомерСтрочки КАК НомерУрока,
|   УчебныйДеньУроки.Предмет.Представление КАК Предмет,
|   УчебныйДеньУроки.Ссылка КАК УчебныйДень
|ИЗ
|   Документ.УчебныйДень.Уроки КАК УчебныйДеньУроки
|
|УПОРЯДОЧИТЬ ПО
|   Дата,
|   НомерУрока";
РезультатЗапроса = Запрос.Выполнить();
Выборка = РезультатЗапроса.Выбрать();

Пока Выборка.Следующий() Цикл
    ДатаНачала = НачалоДня(Выборка.Дата) + НачалоУрока(Выборка.НомерУрока);
    ДатаОкончания = НачалоДня(Выборка.Дата) + КонецУрока(Выборка.НомерУрока);
    НовыйЭлемент = Планировщик.Элементы.Добавить(ДатаНачала, ДатаОкончания);
    НовыйЭлемент.Текст = Выборка.Предмет;
    НовыйЭлемент.Значение = Выборка.УчебныйДень;

    Если ДатаНачала >= ТекущаяДата() Тогда // будущие занятия
        // Белый цвет фона, рамка черная.
        НовыйЭлемент.ЦветФона = WebЦвета.Белый;
        НовыйЭлемент.ЦветРамки = WebЦвета.Черный;
    Иначе // прошедшие занятия
    КонецЕсли;
КонецЦикла;
```

Здесь для задания цвета вы использовали готовые цвета, которые есть в системном наборе значений *WebЦвета* (рисунок 7.46).

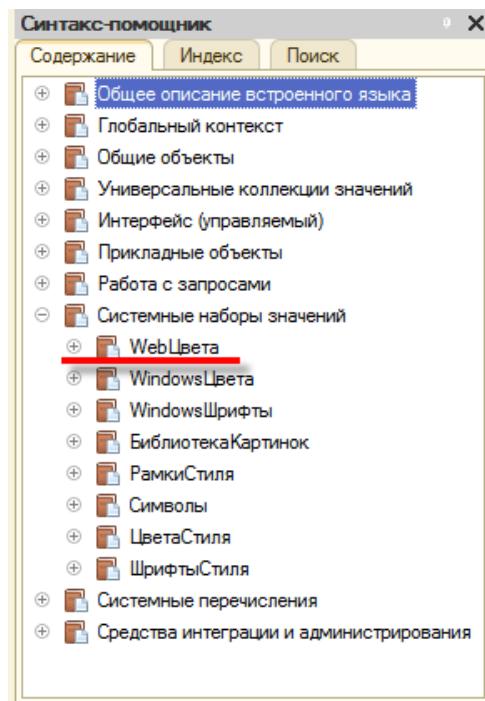


Рисунок 7.46. Системный набор значений «WebЦвета»

Это удобно, но не обязательно. Если вы хотите сконструировать собственный цвет, вы можете воспользоваться объектом *Цвет*. Описание его типа вы найдёте в ветке *Общие объекты — Оформление* синтакс-помощника.

Запустите 1С:Предприятие в режиме отладки и посмотрите теперь, как выглядит планировщик (рисунок 7.47).

понедельник, 7 сентября 2015		вторник, 8 сентября 2015		среда, 9 сентября 2015		четверг, 10 сентября 2015		пятница, 11 сентября 2015	
08:00	08:00	08:00	08:00	08:00	08:00	08:00	08:00	08:00	08:00
08:30 Кл. час		08:30 Кл. час		08:30 Математика		08:30 Математика		08:30 Литература	
09:00		09:00		09:00		09:00		09:00	
09:30 Кл. час		09:30 Кл. час		09:30 Английский язык		09:30 Музыка		09:30 Математика	
10:00		10:00		10:00		10:00		10:00	
10:30 Английский язык		10:30 Английский язык		10:30 Природоведение		10:30 Физ. культура		10:30 История	
11:00		11:00		11:00		11:00		11:00	
11:30 Музыка		11:30 Музыка		11:30 Природоведение		11:30 Математика		11:30 Литература	
12:00		12:00		12:00		12:00		12:00	
12:35 Математика		12:35 Математика		12:35 Русский язык		12:35 Английский язык		12:35 Физ. культура	
13:00		13:00		13:00		13:00		13:00	
13:30 Русский язык		13:30 Русский язык		13:30 Литература		13:30 ИЗО		13:30 Информатика	
14:00		14:00		14:00		14:00		14:00	

Рисунок 7.47. Разное оформление для прошедших и будущих занятий

Теперь, пролистывая планировщик, вы точно не запутаетесь, в какой его части вы находитесь: в прошедшей или в будущей.

Можно улучшать его дальше. Я говорил, что в прошедших уроках вас интересуют полученные оценки. Их можно показать прямо в планировщике. Например, урок, за который получена пятёрка, можно закрасить зелёным цветом. А урок, за который получена тройка, — красным.

Для этого вам понадобится изменить текст запроса, чтобы получать из базы данных ещё и оценку.

Откройте текст запроса в конструкторе и добавьте в выборку поле *Оценка* из табличной части документа. После того как вы закроете конструктор, у вас получится такой текст запроса (листинг 7.25).

Листинг 7.25. Получение оценки из базы данных

```
Запрос.Текст = "ВЫБРАТЬ
| УчебныйДеньУроки.Ссылка.Дата КАК Дата,
| УчебныйДеньУроки.НомерСтрочки КАК НомерУрока,
| УчебныйДеньУроки.Предмет.Представление КАК Предмет,
| УчебныйДеньУроки.Ссылка КАК УчебныйДень,
| УчебныйДеньУроки.Оценка КАК Оценка
|ИЗ
| Документ.УчебныйДень.Уроки КАК УчебныйДеньУроки
|
|УПОРЯДОЧИТЬ ПО
| Дата,
| НомерУрока";
```

Теперь в той ветке условия *Если*, которая относится к прошедшим занятиям, нужно задать цвет фона элементу планировщика. Поскольку цвет будет зависеть от оценки, этот алгоритм лучше вынести в отдельную функцию *ЦветОценки()* (листинг 7.26).

Листинг 7.26. Функция «ЦветОценки()»

&НаСервере

Функция ЦветОценки(Оценка)

```
Если Оценка = 5 Тогда
    Возврат WebЦвета.ВесеннеЗеленый;

ИначеЕсли Оценка = 4 Тогда
    Возврат WebЦвета.Желтый;

ИначеЕсли Оценка = 3 Тогда
    Возврат WebЦвета.Красный;

ИначеЕсли Оценка = 2 Тогда
    Возврат WebЦвета.Серый;

ИначеЕсли Оценка = 1 Тогда
    Возврат WebЦвета.Черный;

Иначе
    Возврат Новый Цвет();

КонецЕсли;
```

КонецФункции

А в инструкции *Если* использовать вызов этой функции (листинг 7.27).

Листинг 7.27. Цвет прошедших занятий

```

Запрос = Новый Запрос;
Запрос.Текст = "ВЫБРАТЬ
| УчебныйДеньУроки.Ссылка.Дата КАК Дата,
| УчебныйДеньУроки.НомерСтрочки КАК НомерУрока,
| УчебныйДеньУроки.Предмет.Представление КАК Предмет,
| УчебныйДеньУроки.Ссылка КАК УчебныйДень,
| УчебныйДеньУроки.Оценка КАК Оценка
| ИЗ
| Документ.УчебныйДень.Уроки КАК УчебныйДеньУроки
|
| УПОРЯДОЧИТЬ ПО
| Дата,
| НомерУрока";
РезультатЗапроса = Запрос.Выполнить();

Выборка = РезультатЗапроса.Выбрать();

Пока Выборка.Следующий() Цикл
    ДатаНачала = НачалоДня(Выборка.Дата) + НачалоУрока(Выборка.НомерУрока);
    ДатаОкончания = НачалоДня(Выборка.Дата) + КонецУрока(Выборка.НомерУрока);
    НовыйЭлемент = Планировщик.Элементы.Добавить(ДатаНачала, ДатаОкончания);
    НовыйЭлемент.Текст = Выборка.Предмет;
    НовыйЭлемент.Значение = Выборка.УчебныйДень;

    Если ДатаНачала >= ТекущаяДата() Тогда // будущие занятия
        // Белый цвет фона, рамка черная.
        НовыйЭлемент.ЦветФона = WebЦвета.Белый;
        НовыйЭлемент.ЦветРамки = WebЦвета.Черный;
    Иначе // прошедшие занятия
        // Цвет фона по оценке, без рамки.
        НовыйЭлемент.ЦветФона = ЦветОценки(Выборка.Оценка);

    КонецЕсли;
КонецЦикла;

```

Запустите 1С:Предприятие в режиме отладки, перейдите на ту неделю, в которой есть проведённые документы с оценками, и посмотрите, как теперь выглядит планировщик (рисунок 7.48).

понедельник, 31 августа 2015		вторник, 1 сентября 2015		среда, 2 сентября 2015		четверг, 3 сентября 2015		пятница, 4 сентября 2015	
08:00	08:00	08:00	08:00	08:00	08:00	08:00	08:00	08:00	
	08:30 Кл. час		08:30 Математика		08:30 Математика		08:30 Литература		
09:00	09:00	09:00	09:30 Английский язык		09:30 Музыка		09:30 Математика		
10:00	10:00	10:00	10:30 Английский язык		10:30 Природоведение		10:30 Физ. культура		
11:00	11:00	11:00	11:30 Музыка		11:30 Природоведение		11:30 Математика		
12:00	12:00	12:00	12:35 Математика		12:35 Русский язык		12:35 Английский язык		
13:00	13:00	13:00	13:30 Русский язык		13:30 Литература		13:30 ИЗО		
14:00	14:00	14:00						13:30 Информатика	

Рисунок 7.48. Оценки, обозначенные цветом

Отлично! Хочется что-нибудь ещё улучшить в планировщике!

Улучшить можно будущие занятия. Про них интересно было бы знать, что задано, а также выполнено это домашние задание или ещё нет.

Для такого улучшения вам снова понадобится дополнительная информация из базы данных. Это поля *ДомашнееЗадание* и *Выполнено*.

Перейдите в конфигуратор и добавьте эти поля в текст запроса. Он примет у вас такой вид (листинг 7.28).

Листинг 7.28. Получение из базы домашнего задания и признака его выполнения

```
Запрос.Текст = "ВЫБРАТЬ
| УчебныйДеньУроки.Ссылка.Дата КАК Дата,
| УчебныйДеньУроки.НомерСтрочки КАК НомерУрока,
| УчебныйДеньУроки.Предмет.Представление КАК Предмет,
| УчебныйДеньУроки.Ссылка КАК УчебныйДень,
| УчебныйДеньУроки.Оценка КАК Оценка,
| УчебныйДеньУроки.ДомашнееЗадание КАК ДомашнееЗадание,
| УчебныйДеньУроки.Выполнено КАК Выполнено
ИЗ
| Документ.УчебныйДень.Уроки КАК УчебныйДеньУроки
|
| УПОРЯДОЧИТЬ ПО
|   Дата,
|   НомерУрока";
```

Цвет, которым нужно закрасить будущий урок, вы тоже будете вычислять по некоторому алгоритму. Этот алгоритм зависит от двух параметров: задано ли что-нибудь и выполнено ли то, что задано. Поэтому его тоже удобнее оформить в виде отдельной функции с двумя параметрами: *Есть* и *Выполнено* (листинг 7.29).

Листинг 7.29. Функция «ЦветДомашнегоЗадания»

```
&НаСервере
Функция ЦветДомашнегоЗадания(Есть, Выполнено)

Если Есть Тогда // есть домашнее задание
    Если Выполнено Тогда
        Возврат WebЦвета.ВесеннеЗеленый; // домашнее задание выполнено

    Иначе
        Возврат WebЦвета.Желтый; // домашнее задание невыполнено

    КонецЕсли;

Иначе
    Возврат WebЦвета.Белый; // нет домашнего задания

КонецЕсли;

КонецФункции
```

Если домашнего задания нет, фон будет белый. У выполненных домашних заданий фон будет зелёный. А те задания, которые ещё предстоит выполнить, будут иметь жёлтый фон.

Использовать эту функцию вы будете там, где устанавливается цвет фона для будущего урока. Теперь вместо белого цвета вы напишете вызов этой функции (листинг 7.30).

Листинг 7.30. Цвет домашнего задания

```

Запрос = Новый Запрос;
Запрос.Текст = "ВЫБРАТЬ
| УчебныйДеньУроки.Ссылка.Дата КАК Дата,
| УчебныйДеньУроки.НомерСтрочки КАК НомерУрока,
| УчебныйДеньУроки.Предмет.Представление КАК Предмет,
| УчебныйДеньУроки.Ссылка КАК УчебныйДень,
| УчебныйДеньУроки.Оценка КАК Оценка,
| УчебныйДеньУроки.ДомашнееЗадание КАК ДомашнееЗадание,
| УчебныйДеньУроки.Выполнено КАК Выполнено
| ИЗ
| Документ.УчебныйДень.Уроки КАК УчебныйДеньУроки
|
| УПОРЯДОЧИТЬ ПО
| | Дата,
| | НомерУрока";
РезультатЗапроса = Запрос.Выполнить();
Выборка = РезультатЗапроса.Выбрать();

Пока Выборка.Следующий() Цикл
    ДатаНачала = НачалоДня(Выборка.Дата) + НачалоУрока(Выборка.НомерУрока);
    ДатаОкончания = НачалоДня(Выборка.Дата) + КонецУрока(Выборка.НомерУрока);
    НовыйЭлемент = Планировщик.Элементы.Добавить(ДатаНачала, ДатаОкончания);
    НовыйЭлемент.Текст = Выборка.Предмет;
    НовыйЭлемент.Значение = Выборка.УчебныйДень;

    Если ДатаНачала >= ТекущаяДата() Тогда // будущие занятия
        // Белый цвет фона, рамка черная.
        ЕстьЗадание = ЗначениеЗаполнено(Выборка.ДомашнееЗадание);
        НовыйЭлемент.ЦветФона = ЦветДомашнегоЗадания(ЕстьЗадание, Выборка.Выполнено);
        НовыйЭлемент.ЦветРамки = WebЦвета.Черный;

    Иначе // прошедшие занятия
        // Цвет фона по оценке, без рамки.
        НовыйЭлемент.ЦветФона = ЦветОценки(Выборка.Оценка);

    КонецЕсли;
КонецЦикла;

```

Есть задание или нет задания, вы определяете с помощью функции глобального контекста `ЗначениеЗаполнено()`. Если в поле `ДомашнееЗадание` «ничего нет» (то есть оно имеет стандартное значение своего типа), эта функция вернёт вам *Ложь*. Если в поле что-то написано, функция вернёт вам значение *Истина*. Стандартным значением для типа `Строка` является пустая строка. То есть строка, не содержащая ни одного символа.

И теперь, раз уж вы отмечаете дни, для которых есть домашнее задание, хорошо бы показать в планировщике и само содержание этого домашнего задания. Его можно добавить к тексту, который вы помещаете в каждый элемент планировщика (листинг 7.31).

Листинг 7.31. Добавление текста домашнего задания

```

Запрос = Новый Запрос;
Запрос.Текст = "ВЫБРАТЬ
| УчебныйДеньУроки.Ссылка.Дата КАК Дата,
| УчебныйДеньУроки.НомерСтрочки КАК НомерУрока,
| УчебныйДеньУроки.Предмет.Представление КАК Предмет,
| УчебныйДеньУроки.Ссылка КАК УчебныйДень,
| УчебныйДеньУроки.Оценка КАК Оценка,
| УчебныйДеньУроки.ДомашнееЗадание КАК ДомашнееЗадание,
| УчебныйДеньУроки.Выполнено КАК Выполнено
| ИЗ
| Документ.УчебныйДень.Уроки КАК УчебныйДеньУроки
|
| УПОРЯДОЧИТЬ ПО
| | Дата,
| | НомерУрока";
РезультатЗапроса = Запрос.Выполнить();
Выборка = РезультатЗапроса.Выбрать();

Пока Выборка.Следующий() Цикл
    ДатаНачала = НачалоДня(Выборка.Дата) + НачалоУрока(Выборка.НомерУрока);
    ДатаОкончания = НачалоДня(Выборка.Дата) + КонецУрока(Выборка.НомерУрока);
    НовыйЭлемент = Планировщик.Элементы.Добавить(ДатаНачала, ДатаОкончания);
    НовыйЭлемент.Текст = Выборка.Предмет + Символы.ПС + Выборка.ДомашнееЗадание;
    НовыйЭлемент.Значение = Выборка.УчебныйДень;

    Если ДатаНачала >= ТекущаяДата() Тогда // будущие занятия
        // Белый цвет фона, рамка черная.
        ЕстьЗадание = ЗначениеВыполнено(Выборка.ДомашнееЗадание);
        НовыйЭлемент.ЦветФона = ЦветДомашнегоЗадания(ЕстьЗадание, Выборка.Выполнено);
        НовыйЭлемент.ЦветРамки = WebЦвета.Черный;

    Иначе // прошедшие занятия
        // Цвет фона по оценке, без рамки.
        НовыйЭлемент.ЦветФона = ЦветОценки(Выборка.Оценка);

    КонецЕсли;
КонецЦикла;

```

Здесь для формирования текста вы используете ещё одно системное перечисление — *Символы*. В нём находится небольшое количество символов, которые могут вам понадобиться при формировании текстов и надписей. В частности, символ, обозначаемый *ПС*. Это символ перевода строки. И используется он тут для того, чтобы название предмета всегда было на отдельной строке и не «сливалось» с текстом домашнего задания.

Здесь специально допущено упрощение, и не выполняется анализ того, есть домашнее задание или нет. В результате такого упрощения к названию предмета всегда будет добавлен служебный символ переноса строки. Будем считать, что для вашей учебной программы это несущественно. Но по-хорошему, конечно, здесь нужно написать инструкцию *Если* и дописывать домашнее задание только тогда, когда оно действительно есть.

Запустите 1С:Предприятие в режиме отладки, раскройте свой электронный дневник

на весь экран и посмотрите, как теперь выглядит расписание занятий (рисунок 7.49).

вторник, 8 сентября 2015	среда, 9 сентября 2015	четверг, 10 сентября 2015
08:00	08:00	08:00
08:30 Кл. час	08:30 Математика	08:30 Математика
09:00	09:00	09:00
09:30 Кл. час	09:30 Английский язык Сделать упражнения 4 и 5. Подготовить ся к чтению по роликам.	09:30 Музыка
10:00	10:00	10:00
10:30 Английский язык	10:30 Природоведение	10:30 Физ. культура
11:00	11:00	11:00
11:30 Музыка	11:30 Природоведение	11:30 Математика
12:00	12:00	12:00
12:30 Математика	12:30 Русский язык Найти и записать 5 пословиц о языке.	12:30 Английский язык
13:00	13:00	13:00
13:30 Русский язык	13:30 Литература Прочитать статью стр. 3 - 5. Подготовить выразительное чтение песни "Варя".	13:30 ИЗО
14:00	14:00	14:00

Рисунок 7.49. Домашние задания в планировщике

Весь экран компьютера не помещается в книгу, поэтому на рисунке 7.49 показана только его часть.

Теперь прямо в планировщике виден текст домашних заданий, и по цвету можно определить, что все эти задания пока ещё не выполнены.

7.9 Обновление данных

Попробуйте поработать с вашим расписанием. Представьте, что вы пришли домой, открыли расписание и увидели, что на завтра есть задания по трём урокам. Допустим, вы сделали задание по русскому языку.

Вы дважды щёлкаете на завтрашнем уроке *Русский язык*. В открывшемся документе отмечаете, что задание по русскому выполнено. Записываете и закрываете этот документ.

Но в планировщике ничего не меняется. Все задания по-прежнему жёлтого цвета.

Если вы сейчас закроете планировщик и откроете его снова, то увидите, что задание по русскому языку стало зелёным (рисунок 7.50).

вторник, 8 сентября 2015	среда, 9 сентября 2015	четверг, 10 сентября 2015
08:00 08:30 Кл. час	08:00 08:30 Математика	08:00 08:30 Математика
09:00	09:00	09:00
09:30 Кл. час	09:30 Английский язык Сделать упражнения 4 и 5. Подготовить ся к чтению по роликам.	09:30 Музыка
10:00	10:00	10:00
10:30 Английский язык	10:30 Природоведение	10:30 Физ. культура
11:00	11:00	11:00
11:30 Музыка	11:30 Природоведение	11:30 Математика
12:00	12:00	12:00
12:35 Математика	12:35 Русский язык: Найти и записать 5 пословиц о языке.	12:35 Английский язык
13:00	13:00	13:00
13:30 Русский язык	13:30 Литература: Прочитать статью стр. 3 - 5. Подготовить выразительное чтение песни "Варяг".	13:30 ИЗО
14:00	14:00	14:00

Рисунок 7.50. Задание по русскому языку выполнено

Это происходит потому, что планировщик заполняет свои данные только один раз. В тот момент, когда создаётся форма. После этого он их только отображает. И он не знает, что в документах что-то изменилось.

Так работать неудобно. Хочется, чтобы планировщик сам обновлял данные, после того как записан или проведён какой-нибудь документ.

Это можно сделать. И, оказывается, сделать это совсем несложно.

В платформе 1С:Предприятие есть специальный механизм, который позволяет разным формам «общаться» друг с другом.

Это очень похоже на ту ситуацию, когда дети занимаются в комнате своими делами, а мама готовит завтрак на кухне. Мама и дети — это три формы, открытые в вашем приложении (рисунок 7.51).



Рисунок 7.51. Мама готовит, дети играют

Когда завтрак готов, мама зовёт детей: «Завтрак готов!» (рисунок 7.52).



Рисунок 7.52. Мама зовёт детей

Ребёнок, который услышал, придёт на кухню и позавтракает. А тот, который не услышал, так и будет заниматься своими делами (рисунок 7.53).



Рисунок 7.53. Один ребёнок услышал, другой — нет

В 1С:Предприятии формы общаются точно так же. Одна форма может послать оповещение всем другим формам, которые открыты в этот момент в вашем приложении.

Другие формы могут «услышать» это оповещение и что-то сделать. А могут просто «не услышать» и пропустить его «мимо ушей».

Как можно использовать этот механизм в вашем расписании занятий?

Вам хочется, чтобы, как только в документе УчебныйДень изменились оценки или домашние задания, это сразу бы видно в расписании. Значит, форма документа должна оповестить форму планировщика, а планировщик по этой команде обновит свои данные.

1С:Предприятие, конечно же, позволяет проанализировать, что именно изменилось в документе. Но у вас всего лишь учебный пример, и загромождать его такими тонкостями и подробностями не хочется. Поэтому вы поступите проще.

Вы не будете анализировать, какие изменения произошли в документе. Главное, что какие бы изменения в нём ни произошли, вы обязательно запишете документ. Иначе ваши изменения просто не сохранятся. И при проведении документа он тоже всегда записывается в базу данных.

Поэтому вы просто перехватите событие записи документа и пошлёте оповещение форме планировщика. И не важно, изменилось в документе что-то или нет. На всякий случай

планировщик обновит свои данные. Да, он выполнит ненужную работу, но хуже ему от этого не станет.

Удобнее всего было бы перехватывать событие записи документа на сервере, сразу после того, как он записан в базу данных. Но, к сожалению, механизм оповещения форм работает только на клиенте. То есть с сервера нельзя послать оповещение формам, которые находятся на клиенте.

К счастью, у управляемой формы есть клиентское событие, которое всегда возникает на клиенте после того, как выполняется запись данных. Оно называется *ПослеЗаписи*. Им вы и воспользуетесь.

Откройте в конфигураторе форму документа *УчебныйДень*. Отсюда вам нужно будет послать оповещение.

Выделите корневой элемент *Форма* и создайте клиентский обработчик события *ПослеЗаписи*. В этом обработчике напишите всего одну строку (листинг 7.32).

Листинг 7.32. Оповещение

```
&НаКлиенте  
Процедура ПослеЗаписи(ПараметрыЗаписи)
```

```
    Оповестить("Учебный день записан");
```

```
КонецПроцедуры
```

Оповестить() — это процедура глобального контекста. Она как раз и рассыпает оповещение всем формам. В самом простом виде при рассылке такого оповещения нужно указать только имя события. Это имя вы придумываете сами, как вам хочется. Главное, чтобы это имя было понятно вам.

Итак, мама позвала детей: «Завтрак готов!»

Теперь нужно сделать так, чтобы у детей были открыты уши, чтобы они могли услышать это оповещение.

Для этого в той форме, которая должна оповещение «услышать», нужно создать обработчик события *ОбработкаОповещения*. Такой формой у вас является общая форма *Планировщик*.

Перейдите в неё и создайте клиентский обработчик этого события.

В этом обработчике нужно сделать две вещи. Сначала нужно узнать, какое именно событие произошло. Для этого вы и давали своему событию имя. Это имя окажется в первом параметре обработчика, *ИмяСобытия* (листинг 7.33).

Листинг 7.33. Анализ имени события

```
&НаКлиенте  
Процедура ОбработкаОповещения(ИмяСобытия, Параметр, Источник)
```

```
    Если ИмяСобытия = "Учебный день записан" Тогда
```

```
        КонецЕсли;
```

```
КонецПроцедуры
```

А теперь нужно обновить данные планировщика. И у вас с самого начала всё для этого есть. Весь алгоритм заполнения планировщика данными находится у вас в отдельной процедуре *ЗаполнитьПланировщикНаСервере()*.

Поэтому всё, что нужно сделать, это удалить старые данные планировщика и снова заполнить его (листинг 7.34).

Листинг 7.34. Обновление данных планировщика

```
&НаКлиенте
Процедура ОбработкаОповещения(ИмяСобытия, Параметр, Источник)

Если ИмяСобытия = "Учебный день записан" Тогда
    Планировщик.Элементы.Очистить();
    ЗаполнитьПланировщикНаСервере();

КонецЕсли;

КонецПроцедуры
```

Вот и всё! Кажется, я дольше объяснял, чем вы это делали.

Запустите 1С:Предприятие в режиме отладки и посмотрите, как это работает.

Сейчас у вас отмечено, что задание по русскому языку выполнено, элемент зелёный. Но это неправда.

Дважды щёлкните на этом элементе, в документе снимите отметку о том, что задание выполнено. Запишите и закройте документ.

Все элементы планировщика снова стали жёлтыми. Сработало оповещение, и планировщик автоматически обновил свои данные.

На этом вы остановитесь. Потому что надо же когда-то остановиться. Сейчас расписание получилось у вас достаточно удобным и функциональным.

Информационная база

Прикладное решение, которое должно получиться у вас к этому моменту, содержится в демонстрационной базе «09 Планировщик.dt». Как её подключить, написано в разделе А.1 «Как подключить демонстрационную базу» на странице 543.

Подробнее

Подробнее вы можете прочитать про планировщик в документации «Руководство разработчика 8.3. Раздел 7.3.3. Планировщик».

Задание 7.1

Заполняйте планировщик не всеми данными, которые есть в базе, а только теми, которые находятся недалеко от текущей даты. Например, за месяц до и за месяц после.

Подсказка. В запросе вам понадобится использовать параметры.

Глава 8

Доработка интерфейса

Информационная база

Прикладное решение, которое должно получиться у вас к этому моменту, содержится в демонстрационной базе «09 Планировщик.dt». Как её подключить, написано в разделе [А.1 «Как подключить демонстрационную базу»](#) на странице [543](#).

Это заключительная глава книги. В ней вы создадите ещё один удобный инструмент для работы с домашними заданиями и окончательно настроите внешний вид вашего приложения.

8.1 Список домашних заданий

Не изменяя традиции, сориентируйтесь в 1С:Предприятии. Сейчас вы отодвинетесь от «подробностей» 1С:Предприятия и займётесь обустройством одной из веток, которая есть в вашем дереве (рисунок 8.1).

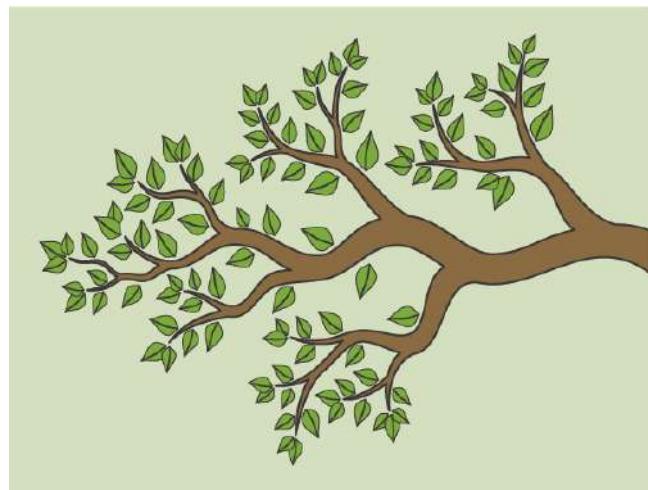


Рисунок 8.1. Сейчас вы здесь

Как вы помните, у вас есть регистр *Домашние Задания*, в котором хранятся задания по всем предметам. Вы научились записывать в него данные.

Теперь вы сделаете простой инструмент, который поможет вам разобраться в том, что задано, и поможет ничего не пропустить. Для этого вы создадите форму списка этого

регистра. А затем, без использования встроенного языка, сделаете в ней несколько доработок.

Итак, перейдите в конфигуратор и создайте форму списка регистра *ДомашниеЗадания*. Сначала уберите из неё поля, которые вам не понадобятся: *Период*, *Регистратор*, *НомерСтроки*, *Выполнено*.

У вас должны остаться только *Предмет* и *ДомашнееЗадание* (рисунок 8.2).

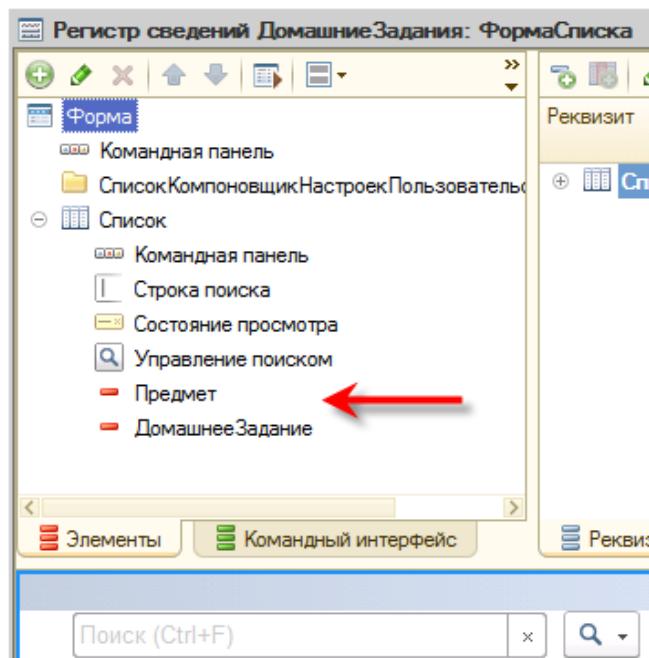


Рисунок 8.2. Поля в форме списка

Дальше, чтобы список был удобным, вы настроите его внешний вид. Все возможности настройки, которыми вы будете сейчас пользоваться, предоставляет источник этих данных. То есть объект *ДинамическийСписок*, который содержится в основном реквизите формы (рисунок 8.3).

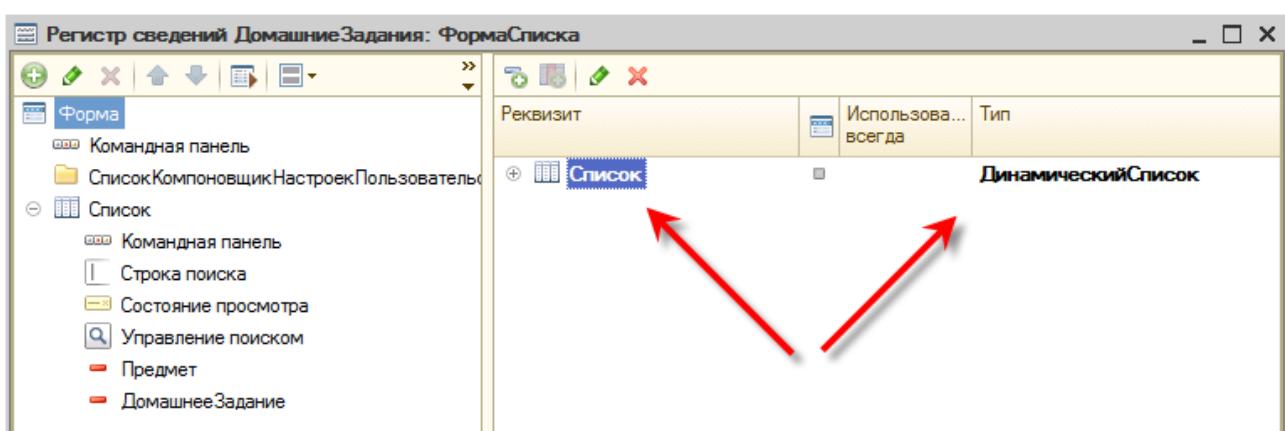


Рисунок 8.3. Источник данных — динамический список

Если интересно, то описание его типа вы можете найти в синтакс-помощнике в ветке *Интерфейс (управляемый) — Данные формы*.

Этот объект используется интересно. У него нет конструктора. Просто потому, что этот объект создаёт всегда сама платформа. Она использует этот объект во всех формах, где есть список элементов.

Динамический список имеет в своей основе систему компоновки данных. Его основная задача — быстро показывать на экране большие списки. Он читает их из базы данных порциями, предоставляет удобные средства поиска и многое другое.

Благодаря тому, что он основан на системе компоновки данных, он имеет очень широкие возможности настройки, вплоть до использования произвольного запроса для получения данных. Задача разработчика и пользователя заключается как раз в настройке этого объекта. Разработчик может настроить его в конфигураторе. Пользователь может настроить его в режиме 1С:Предприятие. Кроме того, аналогичные настройки разработчик может выполнить с помощью встроенного языка.

Вы сейчас настроите этот список в конфигураторе. Для этого нужно выделить реквизит формы *Список* и в палитре свойств открыть настройки этого списка (рисунок 8.4).

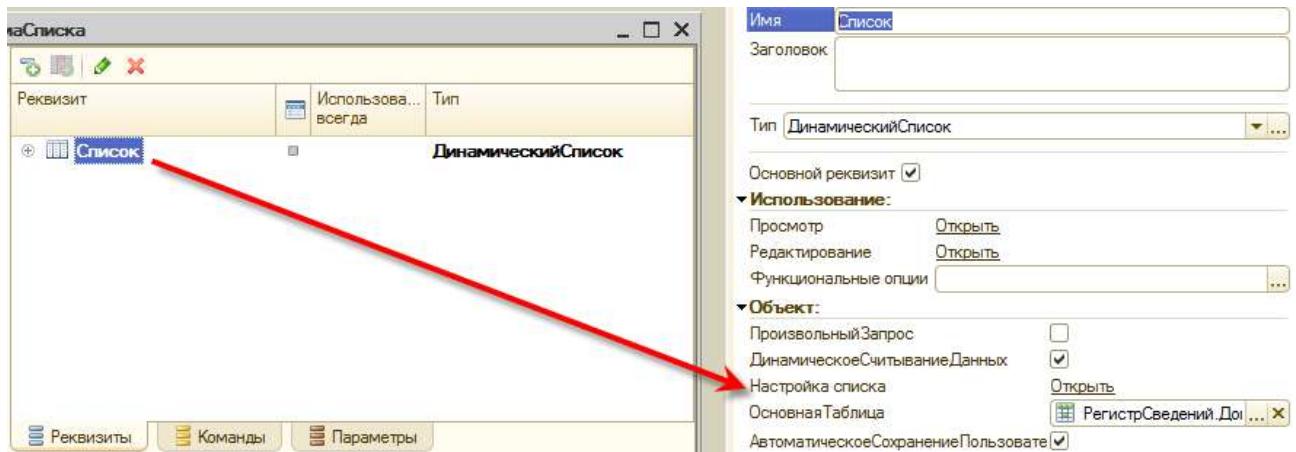


Рисунок 8.4. Открыть настройки динамического списка

Сначала на закладке *Порядок* задайте порядок, в котором будут показаны домашние задания. Сначала они должны быть отсортированы по полю *Период*, а затем по полю *НомерСтроки* (рисунок 8.5).

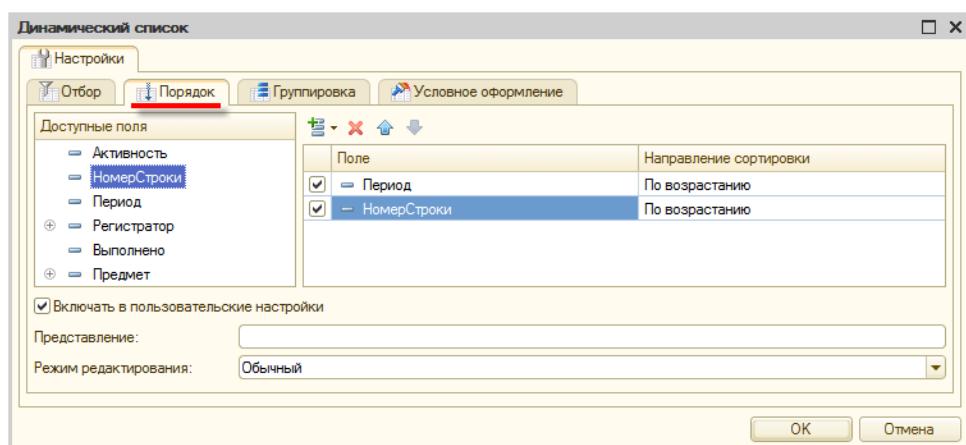


Рисунок 8.5. Порядок

Теперь на закладке *Группировка* укажите, что она будет выполняться по полю *Период* (рисунок 8.6).

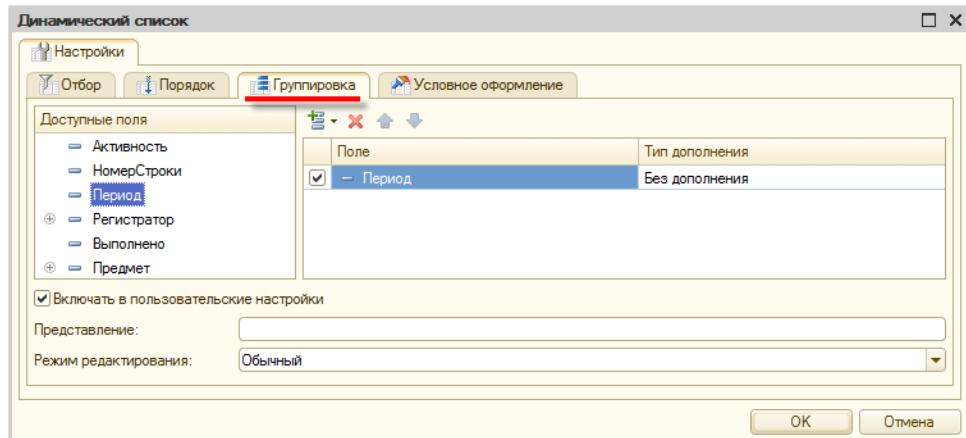


Рисунок 8.6. Группировка

Что это значит, проще показать, чем объяснять.

Чтобы увидеть эту форму в командном интерфейсе, нужно включить команду перехода к ней. Для этого откройте командный интерфейс основного раздела. Если вы забыли, как это делается, посмотрите в разделе 2.9.5 «Объект конфигурации описывает, как будут выглядеть его данные» на страницах 82–83.

Включите команду перехода с списку регистра *Домашние Задания* (рисунок 8.7).

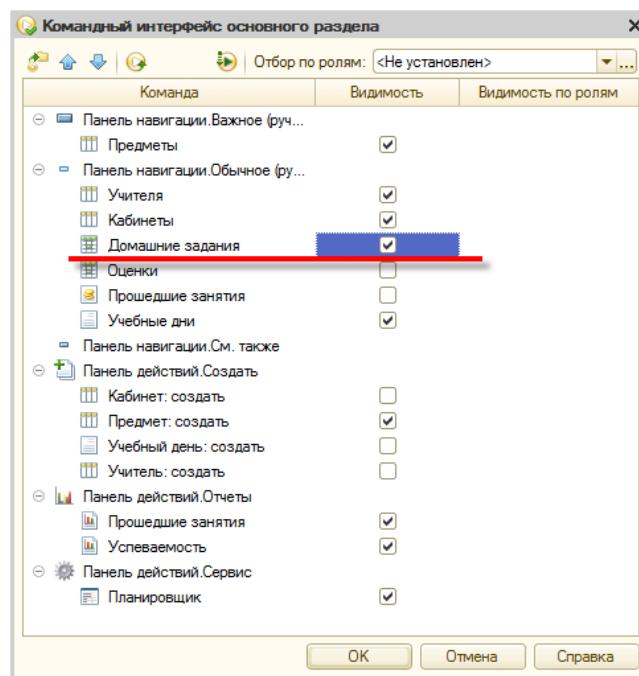


Рисунок 8.7. Команда перехода к списку регистра

Нажмите *OK*, запустите 1С:Предприятие в режиме отладки и откройте список регистра (рисунок 8.8).

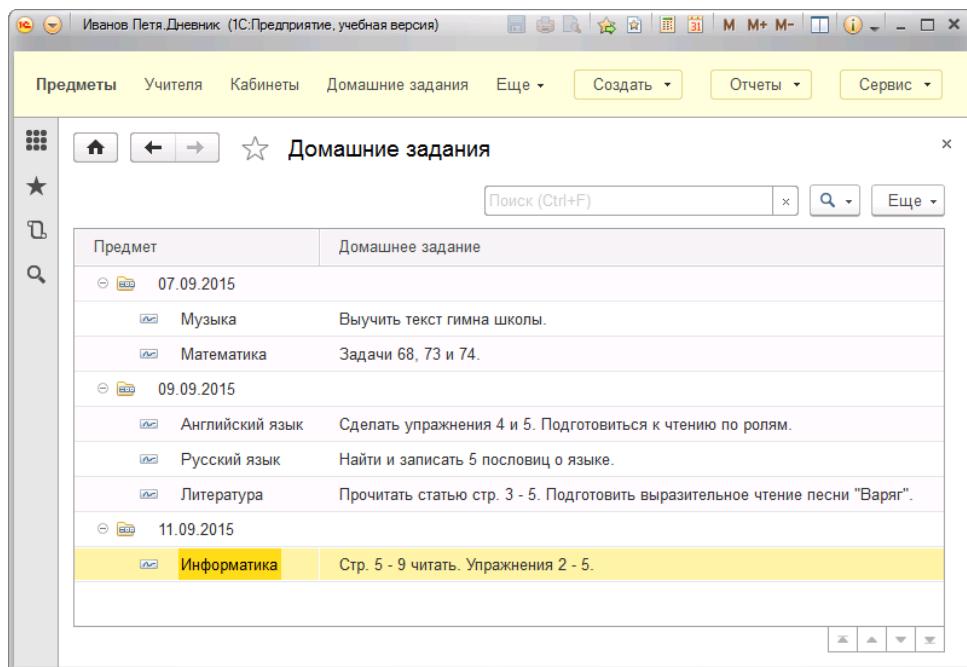


Рисунок 8.8. Список регистра

Вот что значит группировка. Домашние задания собраны в группы по дням. Любую группу вы можете свернуть или развернуть, чтобы вам было удобно.

Вернитесь в конфигуратор и снова откройте настройки динамического списка.

Теперь вас будет интересовать закладка *Условное оформление*. На ней можно задать условия, в соответствии с которыми будет оформлен список. Например, если домашнее задание выполнено, тогда строка с этим заданием будет слегка закрашена серым.

Таких условий может быть много, поэтому вы видите таблицу. Одна строка — одно оформление по условию. Каждый элемент оформления состоит из трёх частей. *Оформление* — это визуальный эффект, который должен быть использован. *Условие* — это условие, при выполнении которого этот визуальный эффект должен быть применён. *Оформляемые поля* — это те поля, к которым вы хотите применить оформление. Если хотите, чтобы были оформлены все поля, которые есть в строке, тут ничего указывать не нужно.

Оформите строки с выполненными домашними заданиями.

Добавьте новый элемент. Сначала задайте оформление (рисунок 8.9).

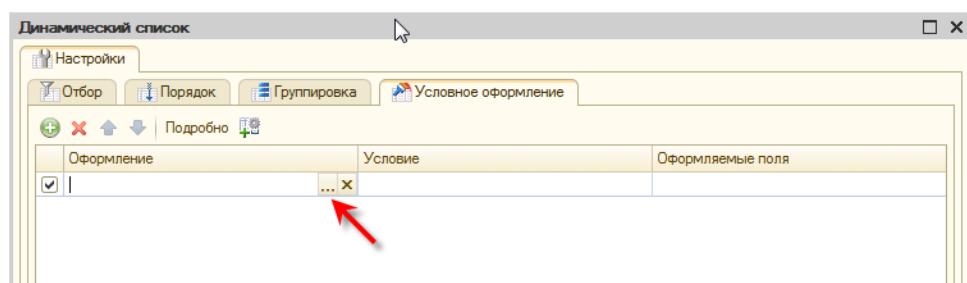


Рисунок 8.9. Оформление

Цвет фона выберите Серебристо-серый (рисунок 8.10).

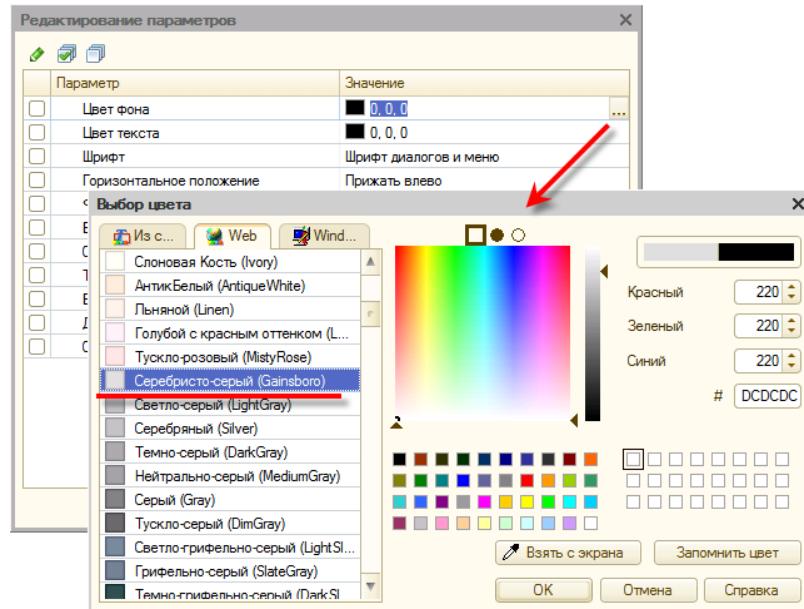


Рисунок 8.10. Цвет фона «Серебристо-серый»

Два раза нажмите *OK*.

После этого задайте условие. Условия могут быть сложными. Поэтому есть возможность добавить один элемент условия или группу, в которой будут содержаться несколько условий. Вам нужен *Новый элемент* (рисунок 8.11).

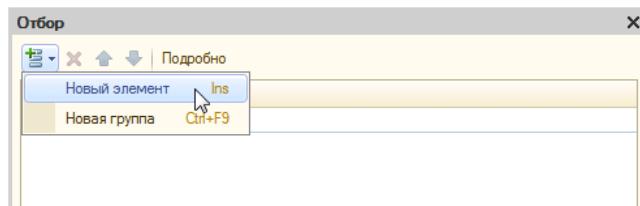


Рисунок 8.11. Добавить новый элемент условия

Условие будет заключаться в том, что в поле *Выполнено* должно находиться значение *Истина*. Именно в этом случае строка будет оформлена (рисунок 8.12).

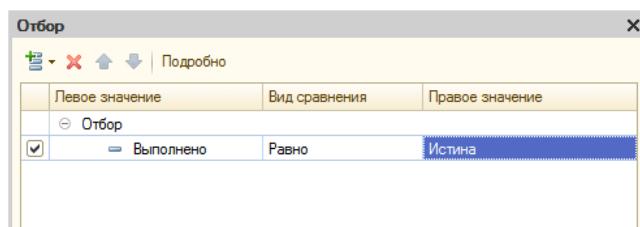


Рисунок 8.12. Условие

Два раза нажмите *OK*.

Запустите 1С:Предприятие в режиме отладки, откройте список регистра. Сейчас у вас нет выполненных домашних заданий, поэтому условное оформление не применяется ни к одной строке (рисунок 8.13).

Предмет	Домашнее задание
07.09.2015	
Музыка	Выучить текст гимна школы.
Математика	Задачи 68, 73 и 74.
09.09.2015	
Английский язык	Сделать упражнения 4 и 5. Подготовиться к чтению по ролям.
Русский язык	Найти и записать 5 пословиц о языке.
Литература	Прочитать статью стр. 3 - 5. Подготовить выразительное чтение песни "Варяг".
11.09.2015	
Информатика	Стр. 5 - 9 читать. Упражнения 2 - 5.

Рисунок 8.13. Нет строк, удовлетворяющих условию оформления

Теперь «выполните» домашнее задание по математике на 7 сентября. Дважды щёлкните по строке, отметьте в документе, что математика выполнена. Запишите и закройте документ.

Установите курсор на первую строку списка (рисунок 8.14).

Предмет	Домашнее задание
07.09.2015	
Музыка	Выучить текст гимна школы.
Математика	Задачи 68, 73 и 74.
09.09.2015	
Английский язык	Сделать упражнения 4 и 5. Подготовиться к чтению по ролям.
Русский язык	Найти и записать 5 пословиц о языке.
Литература	Прочитать статью стр. 3 - 5. Подготовить выразительное чтение песни "Варяг".
11.09.2015	
Информатика	Стр. 5 - 9 читать. Упражнения 2 - 5.

Рисунок 8.14. Оформлена одна строка

Вы увидите, что строка с заданием по математике закрашена серым. Условное оформление сработало.

Почему я попросил вас установить курсор на первую строку? Потому что выделение текущей строки, которое использует платформа, «закрывает» серый фон, который установили вы. И его не видно.

Но это не страшно, потому что сейчас вы сделаете так, чтобы выполненные домашние задания вообще исчезали из этого списка.

Вернитесь в конфигуратор и снова откройте настройки динамического списка.

Теперь вам понадобится самая первая закладка, которая называется *Отбор*. Она определяет, какие данные будут показаны в списке.

Сейчас на ней нет никаких условий, значит, показываются все данные. Вы добавите своё условие, согласно которому должны быть показаны только невыполненные уроки (рисунок 8.15).

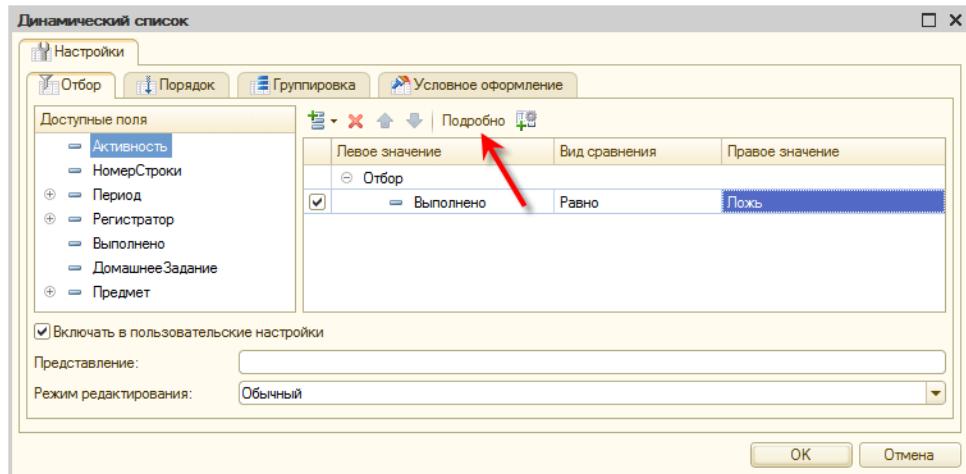


Рисунок 8.15. Отбирать только невыполненные уроки

Теперь вы сделаете так, чтобы этот отбор можно было быстро включать и выключать прямо в режиме 1С:Предприятие. Для этого сначала нужно задать ему представление.

Пока у вас выделена строка с условием, нажмите на кнопку *Подробно* (рисунок 8.5). В появившемся поле *Представление* напишите *Невыполненные* (рисунок 8.16).

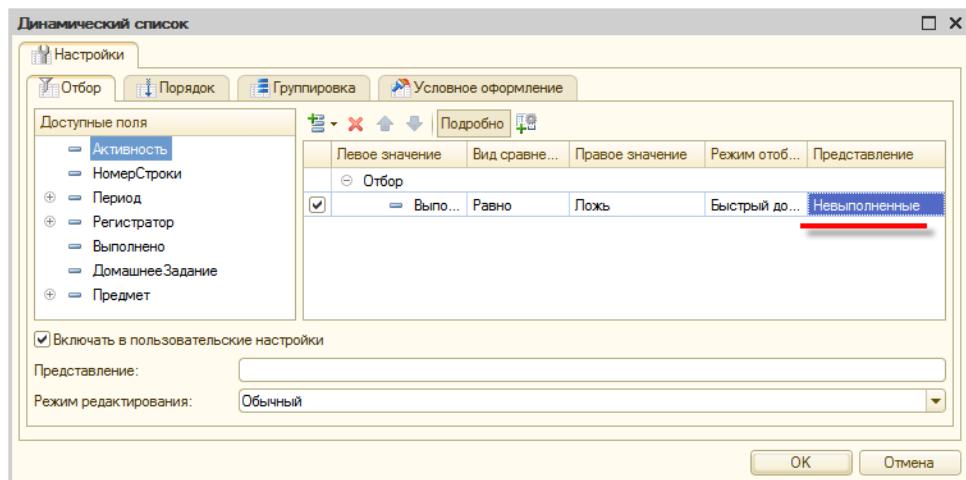


Рисунок 8.16. Представление условия

Теперь нажмите кнопку *Свойства элемента* пользовательских настроек, которая рядом с *Подробно*. В окне настроек установите флажок *Включать в пользовательские настройки* (рисунок 8.17).

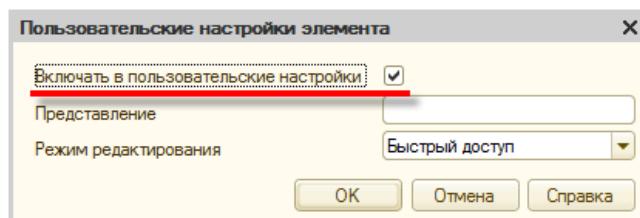


Рисунок 8.17. Включать в пользовательские настройки

Два раза нажмите *OK*. Запустите 1С:Предприятие в режиме отладки и откройте список регистра (рисунок 8.18).

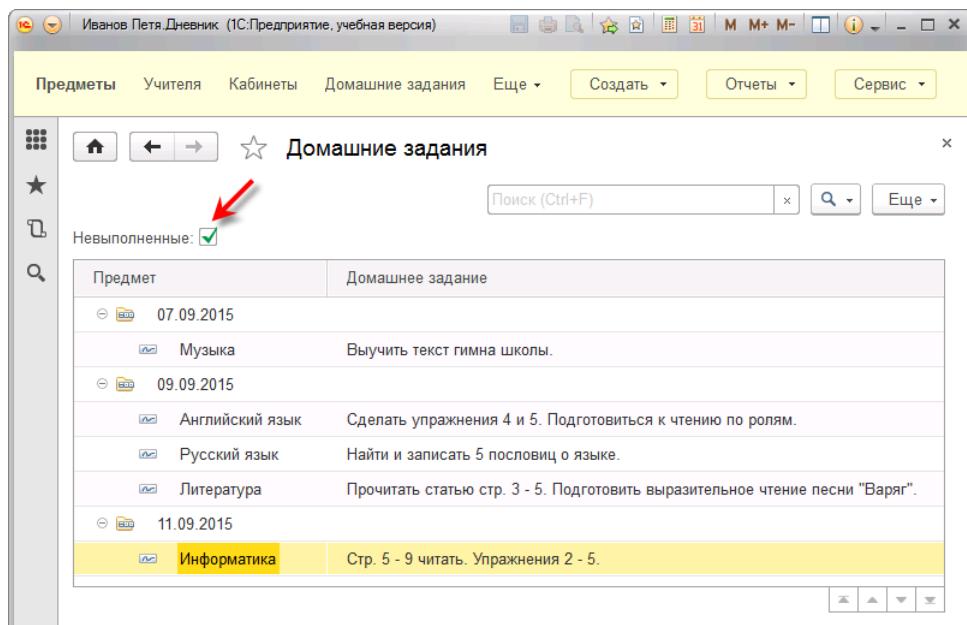


Рисунок 8.18. Список регистра

Во-первых, теперь в нём только невыполненные задания. Во-вторых, появился флажок, который позволяет вам включать и отключать отбор.

«Выполните» домашнее задание по музыке. После того как вы закроете документ, вы увидите, что из списка пропала не только строка с этим заданием, но и вся группировка 7 сентября. Потому что в ней не осталось невыполненных заданий (рисунок 8.19).

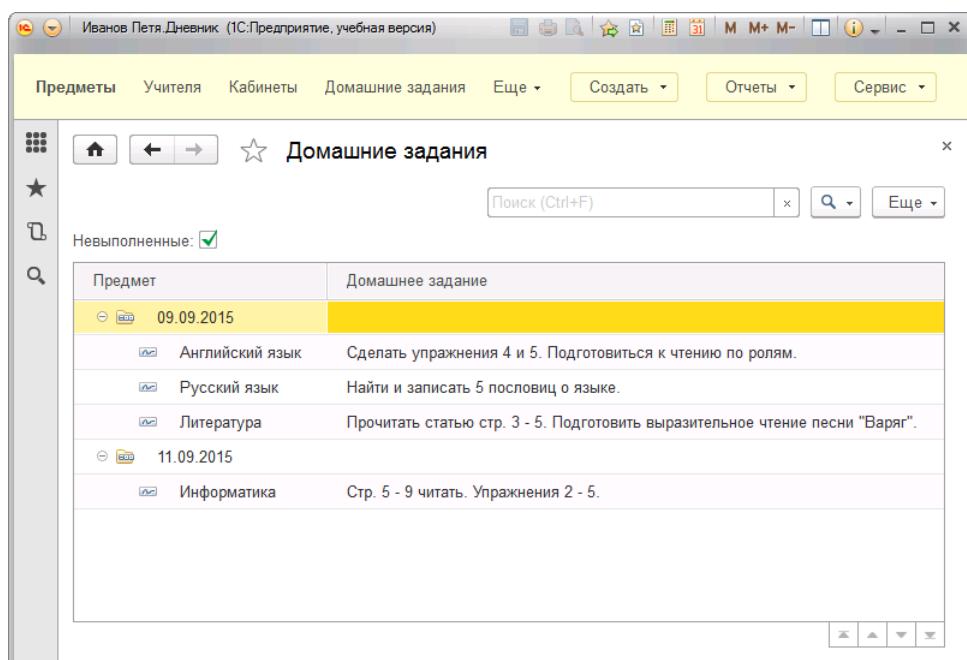


Рисунок 8.19. Группировка «7 сентября» исчезла

Если теперь вы снимете флажок *Невыполненные*, то снова увидите все домашние задания (рисунок 8.20).

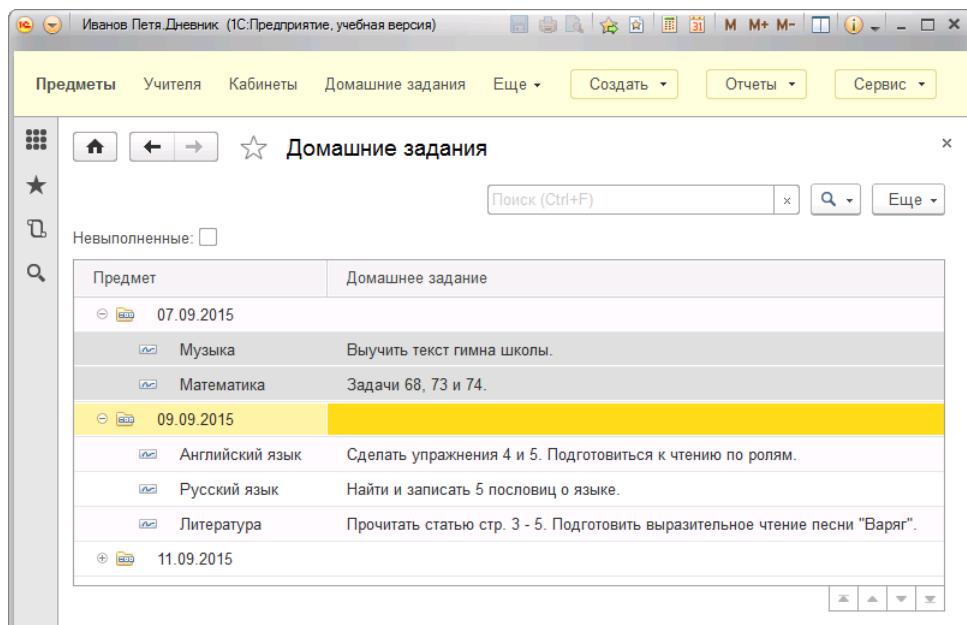


Рисунок 8.20. Все домашние задания

Таким образом список получается очень удобным. Он автоматически показывает только то, что нужно. А если нужно увидеть всё, это выполняется одним нажатием.

Теперь вернитесь в конфигуратор и наведите «окончательную красоту» в списке. Откройте свойства элемента формы *Список*.

Во-первых, сбросьте флажок *Чередование Цветов Строк*. В результате все строки списка будут белыми, без выделения их «через одну».

Во-вторых, свойство *Начальное Отображение Списка* установите в значение *Начало*. Благодаря этому при открытии списка курсор будет всегда на первой строке.

В-третьих, свойство *Начальное Отображение Дерева* установите в значение *Не раскрывать*. В результате список будет открываться со свёрнутыми группами.

Запустите 1C:Предприятие в режиме отладки и откройте список (рисунок 8.21).

Предмет	Домашнее задание
09.09.2015	
11.09.2015	

Рисунок 8.21. Свёрнутые группы списка

Так работать гораздо удобнее. Вы видите, что у вас есть домашние задания на 9 и на 11 сентября. Сначала вы открываете 9 сентября и выполняете эти задания. При этом задания 11 сентября вам не мешают (рисунок 8.22).

Предмет	Домашнее задание
09.09.2015	
Английский язык	Сделать упражнения 4 и 5. Подготовиться к чтению по ролям.
Русский язык	Найти и записать 5 пословиц о языке.
Литература	Прочитать статью стр. 3 - 5. Подготовить выразительное чтение песни "Варяг".
11.09.2015	

Рисунок 8.22. Задания на 9 сентября

Теперь осталось сделать последнюю маленькую доработку. Сейчас в каждой строке списка отображается одна строка домашнего задания. А домашнее задание, если вы помните, может записываться у вас в несколько строк. При такой настройке списка вы не увидите в нём вторую и следующие строки домашнего задания.

Чтобы решить эту проблему, нужно выполнить настройку в двух местах.

Во-первых, в регистре сведений *ДомашниеЗадания*. Нужно открыть свойства его ресурса *ДомашнееЗадание* и установить ему свойство *МногострочныйРежим*. Это приведёт к тому, что в списке регистра домашнее задание будет отображаться в несколько строк.

Во-вторых, в форме списка регистра *ДомашниеЗадания*. Здесь нужно выделить элемент формы *ДомашнееЗадание* и установить его свойство *АвтоВысотаЯчейки*. Это нужно для того, чтобы по высоте раздвигались не все строки списка, а только та строка, в которой находится многострочный текст.

Запустите 1С:Предприятие в режиме отладки и откройте список. Теперь, если попадётся домашнее задание, написанное в несколько строк, вы их увидите в списке (рисунок 8.23).

Предмет	Домашнее задание	
⊖	09.09.2015	
<input checked="" type="checkbox"/> Английский язык	Сделать упражнения 4 и 5. Подготовиться к чтению по ролям.	
<input checked="" type="checkbox"/> Русский язык	Найти и записать 5 пословиц о языке.	
<input checked="" type="checkbox"/> Литература	Прочитать статью стр. 3 - 5. Подготовить выразительное чтение песни "Варяг".	
⊕	11.09.2015	

Рисунок 8.23. Домашнее задание в несколько строк

8.2 Начальная страница

И под конец книги ещё немного отойдите назад, отодвиньтесь от дерева (рисунок 8.24).

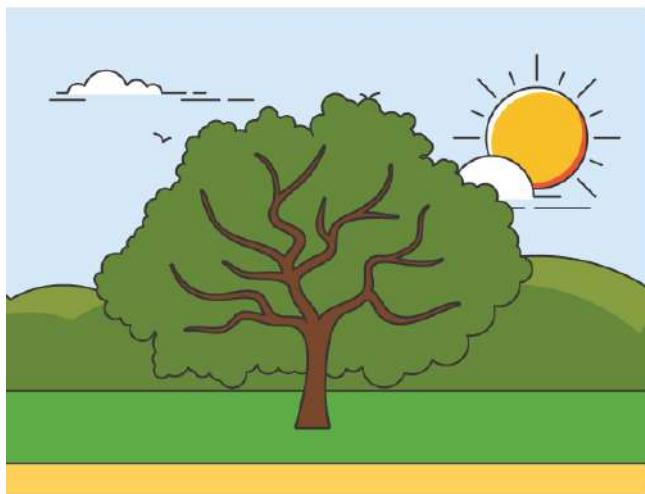


Рисунок 8.24. Сейчас вы здесь

Посмотрите на всё прикладное решение в целом. Создайте ему начальную страницу и удобные команды.

Начальную страницу вы уже редактировали. Если забыли, посмотрите в разделе 2.14.1 «Добавление формы» на страницах 145–146.

Вместо формы списка документов добавьте на неё общую форму *Планировщик*. Поэтому что этот инструмент удобен, и его хочется видеть сразу после открытия вашего электронного дневника. Он даёт общую информацию о том, «как обстоят дела» (рисунок 8.25).

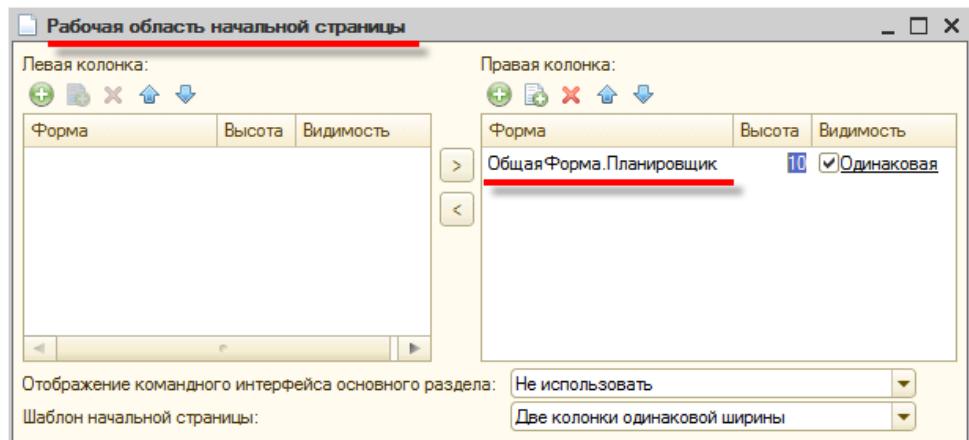


Рисунок 8.25. Планировщик на начальной странице

8.3 Командный интерфейс основного раздела

Теперь наведите порядок в командном интерфейсе основного раздела.

Планировщик в панели Сервис можно отключить. Теперь он у вас на начальной странице.

Список предметов стал теперь не так важен, его можно перенести в группу *Обычное*.

В освободившуюся группу *Важное* логично поместить две команды: *Домашние задания* и *Учебные дни*.

Команду создания предмета можете оставить, а можете отключить. Как вам больше нравится (рисунок 8.26).

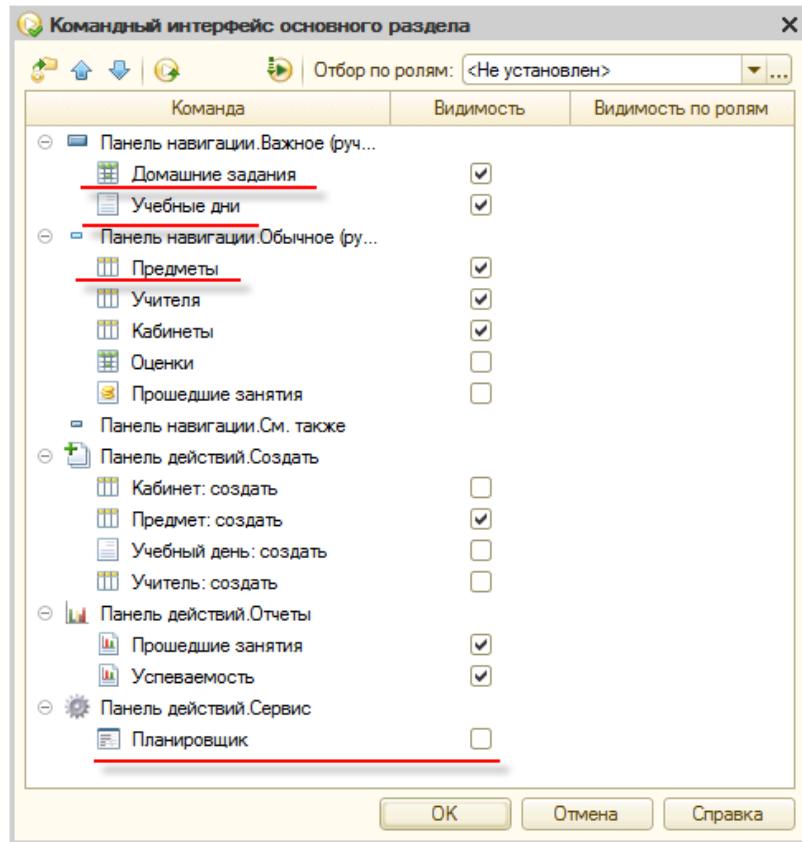


Рисунок 8.26. Командный интерфейс основного раздела

Теперь можете запустить 1С:Предприятие в режиме отладки и посмотреть на окончательный вид вашего электронного дневника (рисунок 8.27).

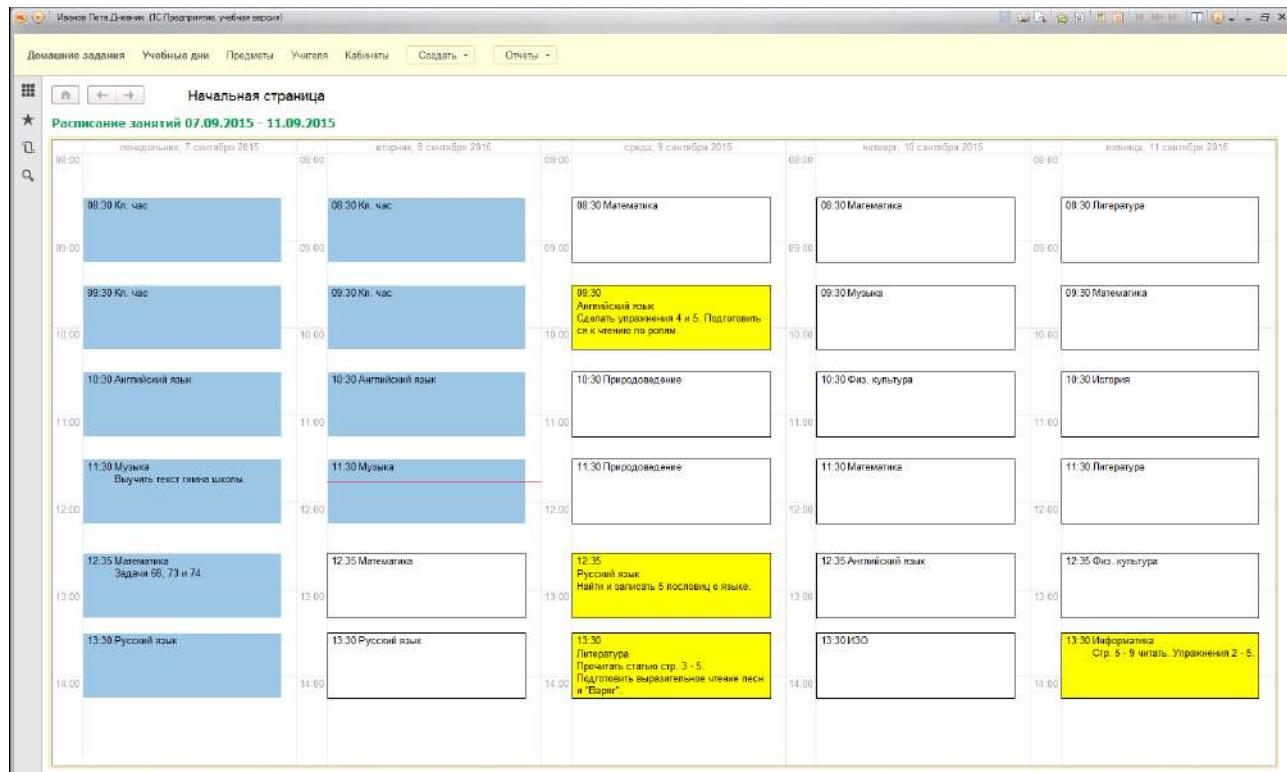


Рисунок 8.27. Окончательный вид электронного дневника

На этом моя книга заканчивается. Надеюсь, что она была для вас интересной и полезной. А если вам хочется «чего-нибудь ещё», можете воспользоваться советом, который находится ниже.

Совет

Если вы хотите получить больше практики, создать более сложное приложение, узнать о других объектах 1С:Предприятия, можете воспользоваться книгой М. Г. Радченко, Е. Ю. Хрусталевой «1С:Предприятие 8.3. Практическое пособие разработчика. Примеры и типовые приёмы» ([информация о книге](#)).



Рисунок 8.28. 1С:Предприятие 8.3. Практическое пособие разработчика

Если в самом начале занятий вы скачивали из Интернета версию для обучения программированию, то эта книга у вас уже есть в электронном виде. Запустите файл autorun.exe и откройте пункт меню *Информационные материалы...* (рисунок 1.42).

Информационная база

Окончательный вариант электронного дневника содержится в демонстрационной базе «10 ОкончательныйВариант.dt». Как её подключить, написано в разделе A.1 «Как подключить демонстрационную базу» на странице 543.

Приложение А

Полезные советы

A.1 Как подключить демонстрационную базу

К книге прилагаются несколько демонстрационных информационных баз. Каждая из них показывает, как должно выглядеть прикладное решение к моменту чтения той или иной главы.

Эти базы будут полезны вам в качестве эталона, с которым вы можете сравнить то, что получается у вас.

Также вы можете использовать эти базы тогда, когда хотите изучить только одну главу книги. Чтобы не выполнять весь пример с самого начала, подключите наиболее подходящую информационную базу и в ней выполняйте дальнейшие примеры.

Все демонстрационные базы собраны в zip-архив, который вы можете скачать по адресу http://its.1c.ru/book_demo. На этой странице найдите название книги и скачайте архив на свой компьютер. Если вы не помните, как разархивировать файлы, можете посмотреть в разделе 1.5.1 «Скачивание дистрибутива» на страницах 29–30.

Каждая демонстрационная база представляет собой файл с расширением «dt». Это файл выгрузки информационной базы. В нём содержатся и конфигурация, и данные. Чтобы из этого файла создать информационную базу, сделайте следующее.

Добавьте новую информационную базу (смотрите в разделе 2.1 «С чего начинается прикладное решение» на страницах 38–41). Назовите её так же, как называется файл «dt». Чтобы вам было понятно назначение этой базы.

Откройте эту новую информационную базу в конфигураторе (смотрите в разделе 2.2 «Список информационных баз» на страницах 42–43) и выполните команду *Администрирование – Загрузить информационную базу...* (рисунок A.1).

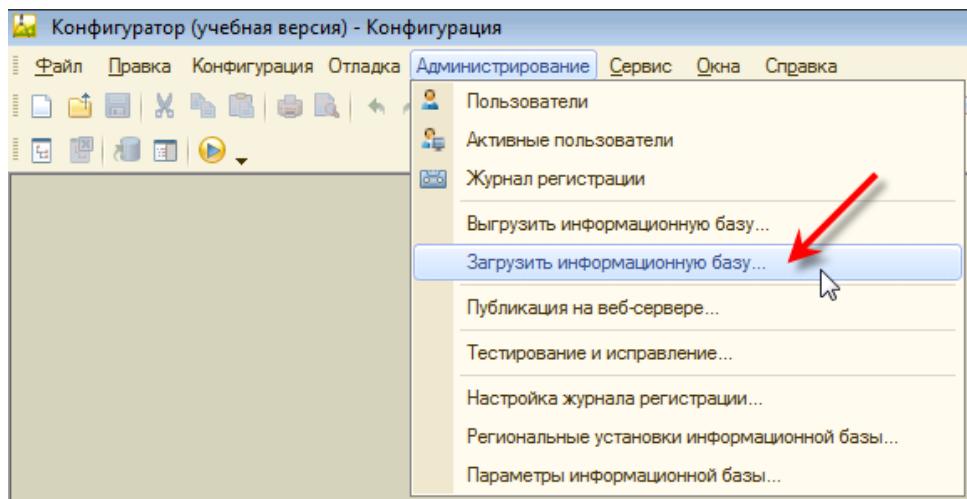


Рисунок А.1. Загрузить информационную базу

Выберите файл выгрузки той информационной базы, который вам нужен.

После этого платформа предупредит вас, на всякий случай, что несохранённые данные могут быть потеряны (рисунок А.2).

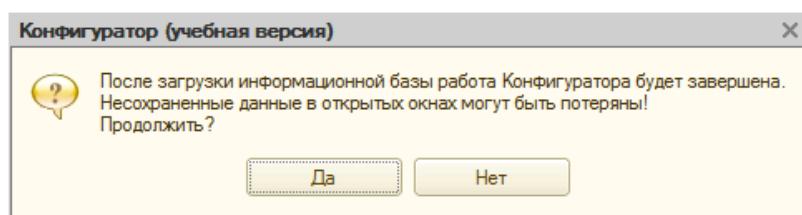


Рисунок А.2. Несохранённые данные могут быть потеряны

В этой базе у вас несохранённых данных нет, поэтому смело нажимайте Да.

После этого платформа загрузит конфигурацию, данные и предложит вам перезапустить конфигуратор (рисунок А.3).

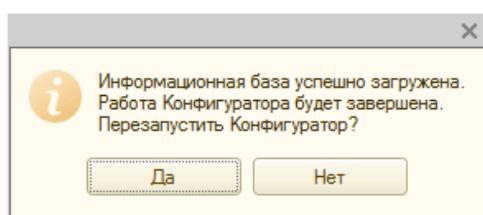


Рисунок А.3. Перезапустить Конфигуратор

Нажмите Да.

После того как конфигуратор перезапустится, выполните команду *Открыть конфигурацию* (смотрите в разделе 2.4 «Дерево объектов конфигурации» на страницах 46–47) и пользуйтесь информационной базой на здоровье!

A.2 Как прочитать сообщение об ошибке

Когда вы пишете текст программы, вы можете допустить ошибку. Платформа, перед тем как начать исполнение вашей программы, проверяет, всё ли написано правильно. Всё ли соответствует тем правилам, которые есть во встроенном языке.

Если платформа находит ошибку, она сообщает об этом. Например, так (рисунок А.4).

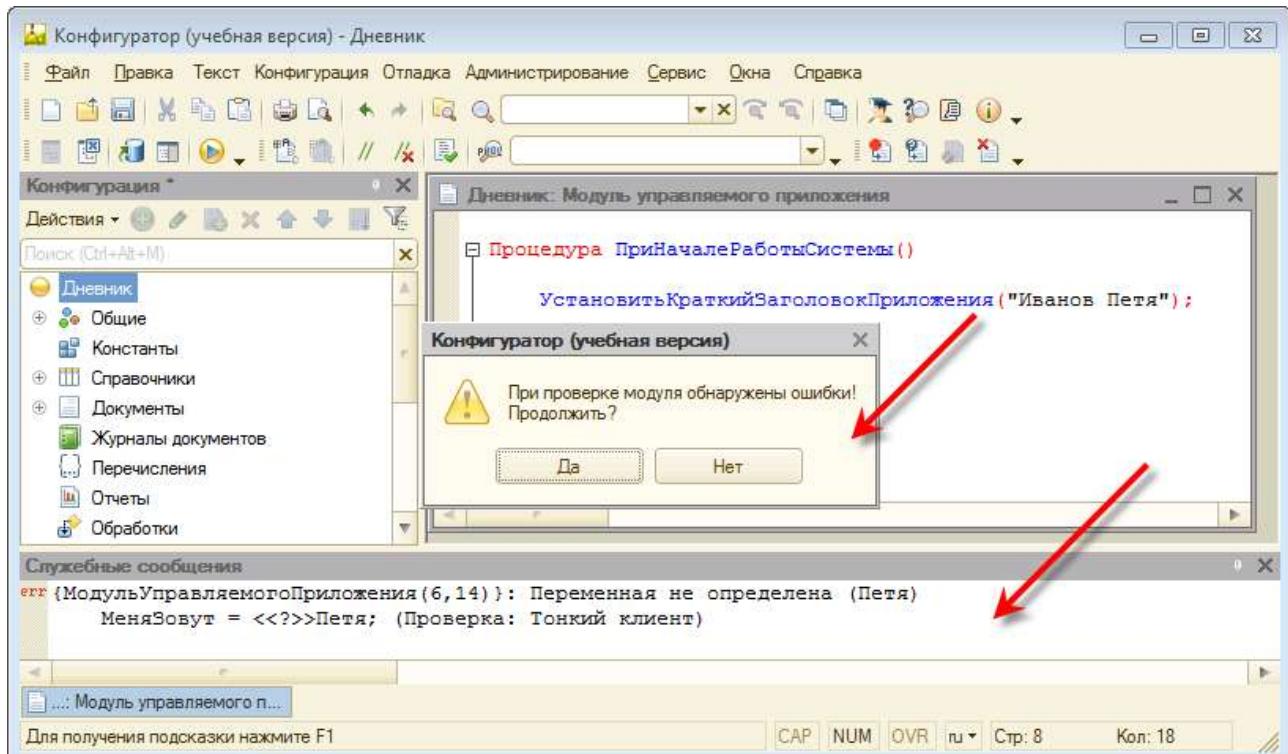


Рисунок А.4. Платформа обнаружила ошибку

В центре появляется сообщение о том, что есть ошибка, и вопрос, продолжить или нет. Чтобы исправить ошибку, нажмите *Нет* и прочитайте, что написано в окне служебных сообщений, которое появляется в нижней части экрана.

В нём всегда указано точное место, в котором платформа обнаружила ошибку. Кроме этого там есть некоторое сообщение. Оно поможет вам понять, в чём заключается ошибка. Также в этом окне платформа показывает инструкцию, которая, по её мнению, является неправильной. И в конце платформа пишет в скобках название режима, в котором выполнялась проверка.

Название режима вам пока не нужно, поэтому на него просто не надо обращать внимания (рисунок А.5).

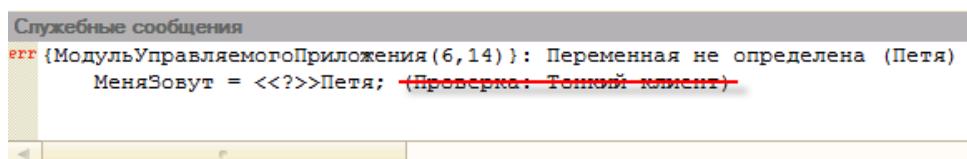


Рисунок А.5. Название режима проверки пока не нужно

Инструкция, в которой обнаружена ошибка, очень полезна. Разработчики, которые уже обладают некоторым опытом, могут понять, в чём ошибка, просто посмотрев на эту инструкцию (рисунок А.6).

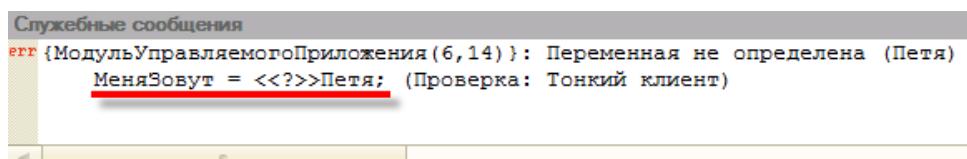


Рисунок А.6. Инструкция, в которой обнаружена ошибка

Символами `<<?>>` в ней отмечено то место, которое вызвало непонимание у платформы.

Но вы так пока не умеете. Поэтому вам нужно пользоваться первой строкой этого сообщения.

Сначала в ней написано имя модуля, в котором обнаружена ошибка (рисунок А.7).

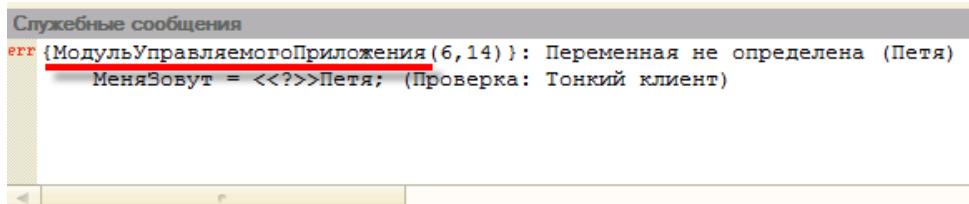


Рисунок А.7. Имя модуля, в котором обнаружена ошибка

В данном случае это *модуль управляемого приложения*. Он у вас один единственный. Но не забывайте, что вообще в конфигурации может использоваться большое количество разных модулей. Поэтому имя модуля — это важная информация.

Дальше в скобках находятся два числа. Первое из них — это номер строки в модуле. Той строки, в которой находится инструкция, вызвавшая непонимание у платформы (рисунок А.8). Строки нумеруются сверху вниз, начиная с единицы.

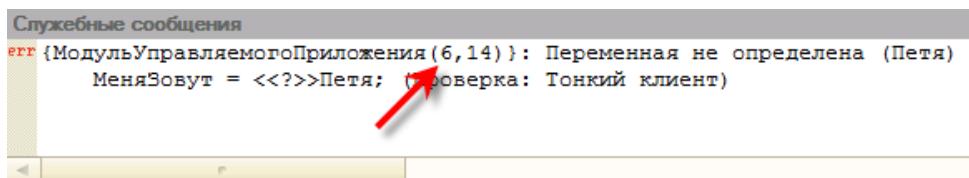


Рисунок А.8. Номер строки с ошибкой

Сейчас у вас модуль совсем маленький, и найти нужную строку не составляет труда. Но обычно модули содержат большое количество строк. Исследовать, например, 645-ю строку вручную совсем неудобно. Поэтому в конфигураторе есть возможность автоматически перейти к нужной строке.

Сейчас, со стандартными настройками, вы её не видите. Чтобы её увидеть, включите панель *Текст*. Для этого нажмите правой кнопкой мыши на пустом месте справа в верхней командной панели (рисунок А.9).

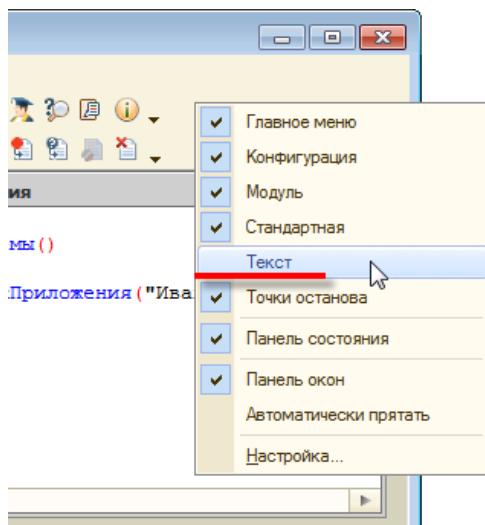


Рисунок А.9. Настройка панелей конфигуратора

В появившемся меню щёлкните по строке *Текст*. В нижней части окна появится новая командная панель (рисунок А.10).

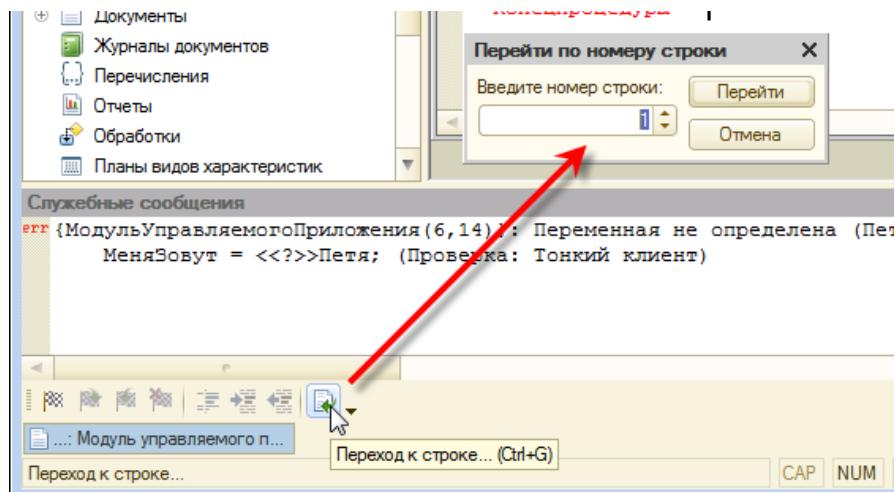


Рисунок А.10. Командная панель «Текст»

В этой панели находятся команды, которые часто используются при работе с текстом. Последняя из этих команд — *Переход к строке...*. Нажмите на неё.

В появившемся окне введите номер нужной вам строки и нажмите *Перейти*. Конфигуратор выделит ту строку, которую вы искали (рисунок А.11).

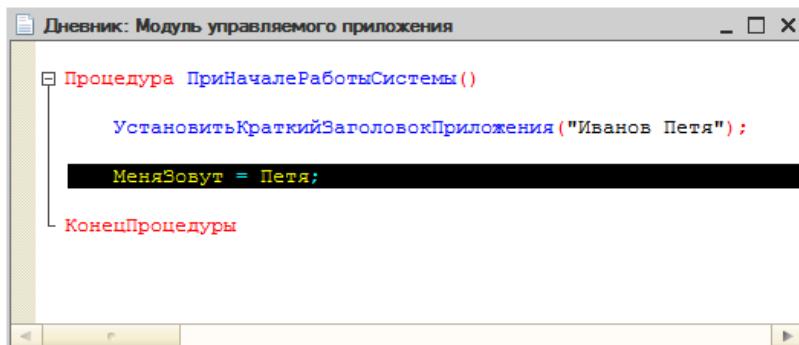


Рисунок А.11. Выделена строка с нужным номером

Теперь вам нужна вторая цифра, которая указана в скобках в сообщении об ошибке. Это номер позиции в этой строке (рисунок А.12).

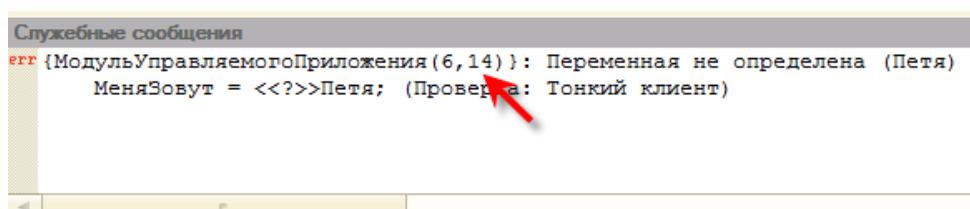


Рисунок А.12. Номер позиции в строке

Здесь всё просто. Позиция, на которой курсор находится в строке, всё время показывается в правом нижнем углу экрана (рисунок А.13).

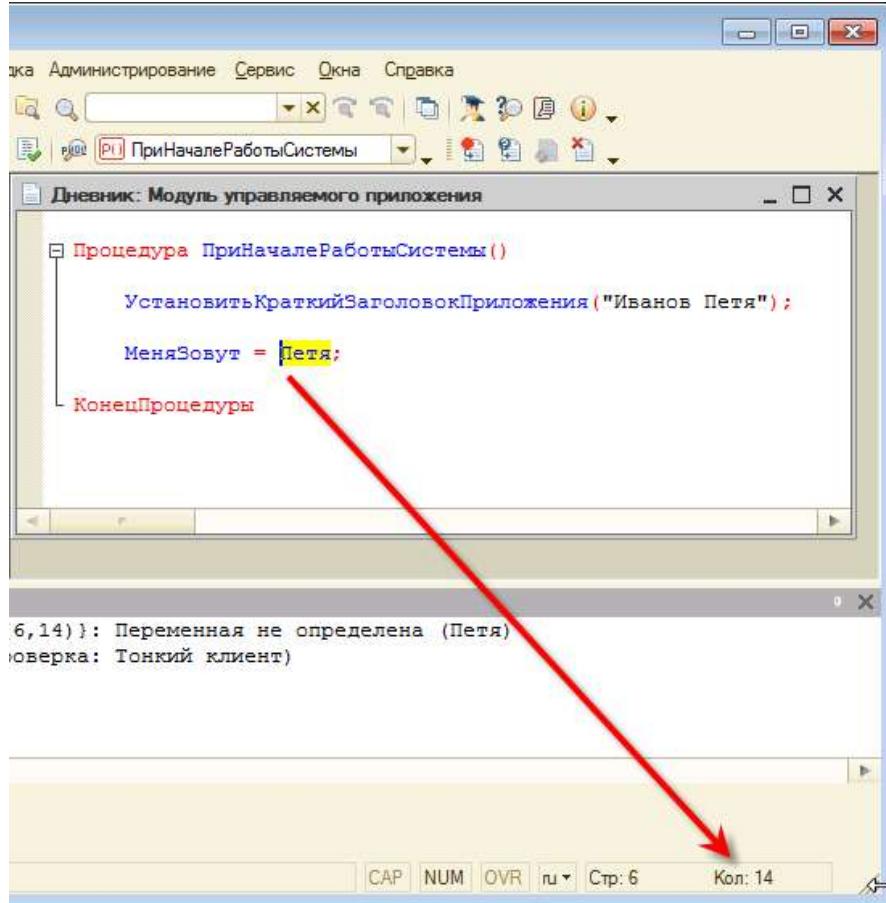


Рисунок А.13. Позиция курсора в строке

Поэтому нужно установить курсор в любое место вашей строки и с помощью стрелок вправо или влево «добраться» до нужной позиции. Если «бежать» далеко, нажмите и удерживайте стрелку. Курсор побежит автоматически.

Таким образом вы добрались до того места, в котором платформа обнаружила ошибку. Тут вам нужно понять, в чём ошибка. В этом поможет сообщение, которое пишет платформа (рисунок А.14).

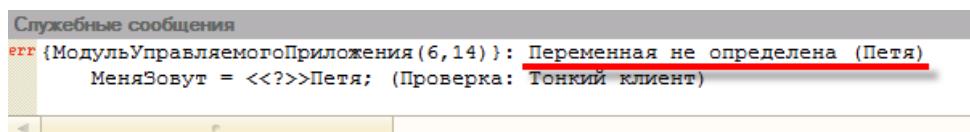


Рисунок А.14. Сообщение об ошибке

Это сообщение чаще всего не говорит, что именно вы сделали не так. Оно говорит о том, какая трудность возникла у платформы. В этом месте трудности у платформы могут возникнуть по разным причинам. И вам нужно понять, какая именно причина является «вашей».

Посмотрите на текст программы (рисунок А.15).

```

Дневник: Модуль управляемого приложения
□ Процедура ПриНачалеРаботыСистемы()
    УстановитьКраткийЗаголовокПриложения ("Иванов Петя");
    МеняЗовут = Петя;
    КонецПроцедуры

```

Рисунок А.15. Текст программы

Платформа думает, что *Петя* — это переменная. Она написала об этом в сообщении. По этой же причине платформы выделила её синим цветом.

А что хотели вы? Вы хотели поместить в *МеняЗовут* значение какой-то переменной *Петя*, которую вы создали раньше? Или вы хотели поместить в *МеняЗовут* ваше имя — «*Петя*»? Наверное, вы хотели поместить ваше имя. То есть значение.

А значения в тексте программы записываются по специальным правилам. Вспомните: они называются литералами. Литерал типа *Строка* всегда должен быть обрамлён кавычками. Вы кавычки не поставили, поэтому платформа вас не поняла.

Если вы поставите кавычки, то и ошибка исчезнет, и платформа раскрасит «*Петя*» чёрным цветом. Что означает: это литерал (рисунок А.16), а не какая-то переменная с именем *Петя*.

```

Дневник: Модуль управляемого приложения
□ Процедура ПриНачалеРаботыСистемы()
    УстановитьКраткийЗаголовокПриложения ("Иванов Петя");
    МеняЗовут = "Петя";
    КонецПроцедуры

```

Рисунок А.16. Правильный текст программы

Примечание

Платформа не может угадать за вас, что вы хотели написать. Она не может сказать вам, например, что «*Петя*» надо взять в кавычки.

Поэтому, прочитав сообщение платформы, всегда нужно ещё и подумать самому. Какую именно ошибку вы допустили, что платформа вас не поняла?

A.3 Как сделать копию рабочей базы

Сделать копию рабочей базы очень просто. Нужно скопировать файл вашей базы в новый каталог и подключить этот каталог как существующую информационную базу.

Путь к каталогу базы данных вы можете посмотреть в списке информационных баз (рисунок А.17).

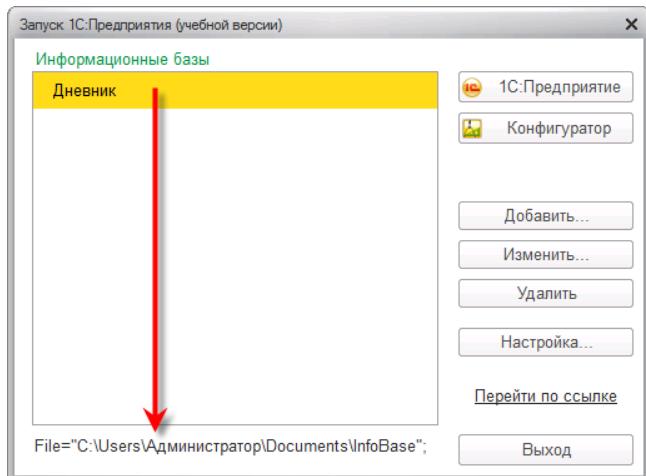


Рисунок А.17. Путь к каталогу базы данных

Этот путь вы можете выделить прямо в этом окне и скопировать (**Ctrl+Insert**). Затем открыть проводник Windows и вставить его в адресную строку (**Shift+Insert**).

Файл базы данных называется *1Cv8.1CD*. Скопируйте его в новый каталог — например, рядом с каталогом существующей базы (рисунок А.18).

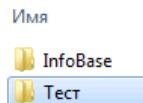


Рисунок А.18. Каталог для копии базы

После этого в списке информационных баз нажмите кнопку *Добавить* (рисунок А.19).

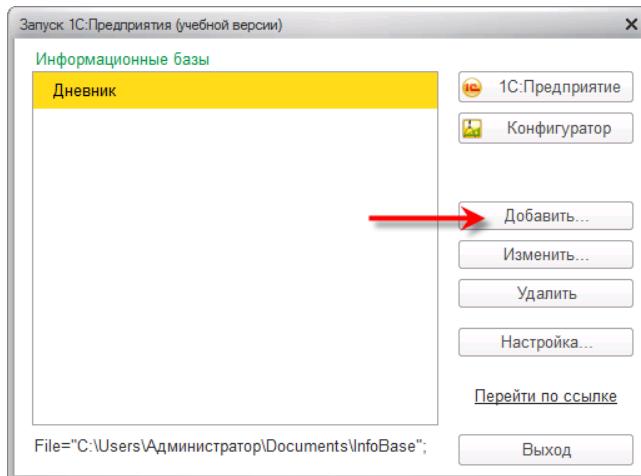


Рисунок А.19. Добавить информационную базу

Отметьте, что вы будете добавлять существующую информационную базу (рисунок А.20).

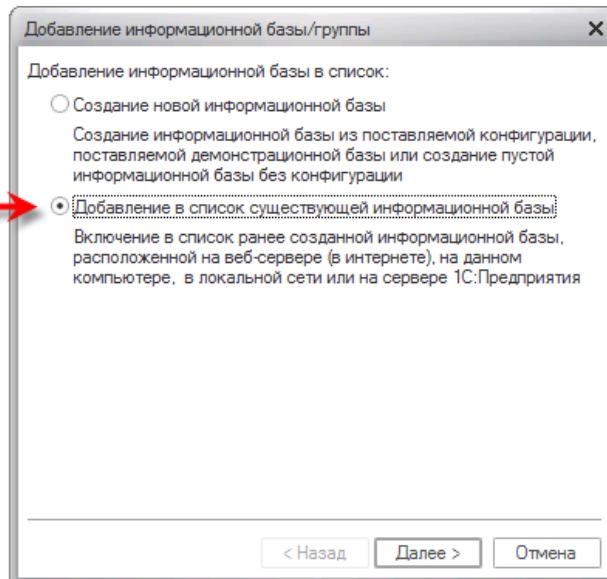


Рисунок А.20. Добавить существующую базу

Задайте название этой базы и укажите путь к тому каталогу, куда вы скопировали файл базы данных (рисунок А.21).

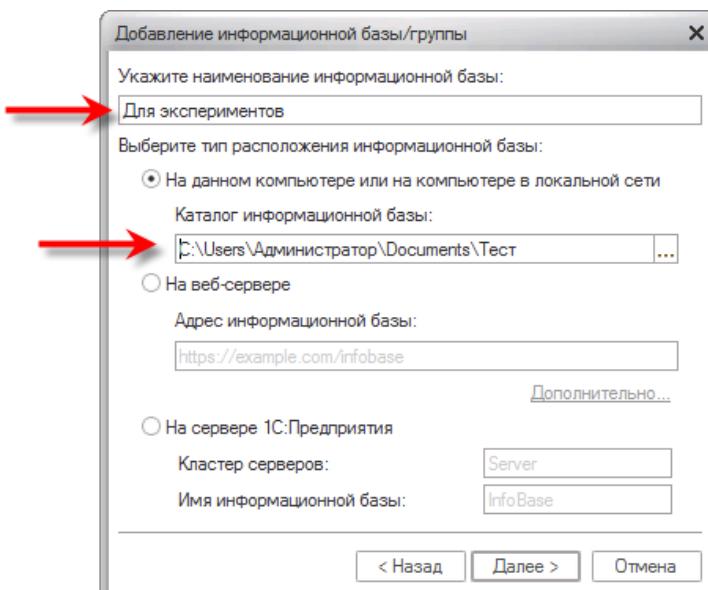


Рисунок А.21. Имя базы и каталог новой базы

После завершения всех шагов копия вашей рабочей базы появится в списке (рисунок А.22).

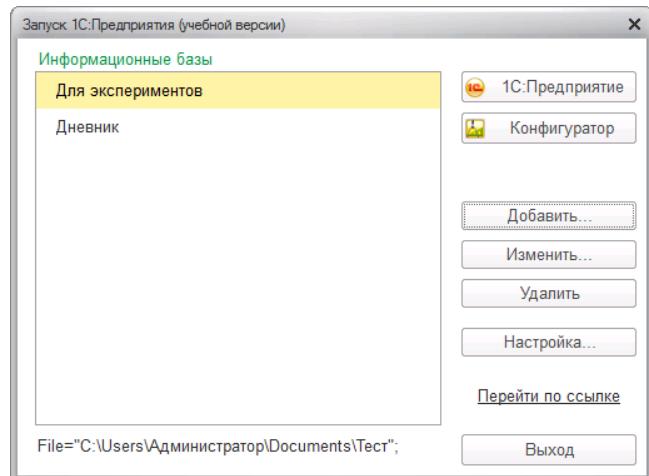


Рисунок А.22. Копия базы в списке информационных баз

Приложение Б

Решения заданий

Б.1 Решение 1.1

Рисунок 1.6. Я на холме, вижу город. Когда я приближаюсь к нему, я начинаю различать отдельные улицы и дома.

Рисунок 1.7. Я в городе, вижу дома, которые стоят вдоль улицы. Когда я приближаюсь к одному из домов, я могу посчитать, сколько в нём этажей. Начинаю различать отдельные окна и балконы.

Рисунок 1.8. Я возле дома, вижу этажи, окна, балконы. Некоторые балконы застеклены, некоторые нет. Я различаю, что есть пустые балконы, а есть балконы, занятые вещами.

Когда я отдаляюсь от дома, я перестаю различать детали отдельных балконов. Я лишь вижу, что в этом подъезде есть балконы, а в другом — нет.

Если отойти ещё дальше, становится сложно определять количество этажей в домах. Можно только сказать, что те дома высокие, а эти — нет.

Если отойти совсем далеко, все дома начинают сливаться. Видно только, что в той стороне находится город, а дальше начинается лесной массив.

Рисунок 1.6. Слова: город, лес, река.

Рисунок 1.7. Слова: улица, дома.

Рисунок 1.8. Слова: этаж, окно, балкон.

Б.2 Решение 1.2

Небо, солнце, облака.

Облака, земля.

Поля и леса, дороги, посёлки, города.

Улицы, дома.

Взлётно-посадочная полоса, аэропорт, машины, люди.

Б.3 Решение 1.3

Небо, море, суши.

На суше есть скалы, лес, большие и маленькие посёлки.

Дома, набережная, порт.

Причал, машины, люди.

Б.4 Решение 2.1

- Представление объекта — *Игра*.
- Расширенное представление объекта — *Компьютерная игра*.
- Представление списка — *Игры*.
- Расширенное представление списка — *Любимые компьютерные игры*.

Ваши представления могут отличаться от этих, но важно сохранить главный принцип.

Представления объекта и списка должны быть короткими, потому что они используются в командном интерфейсе. В нём места немного.

Расширенные представления могут быть длинными и могут полностью пояснить то, что находится в форме объекта или форме списка.

Б.5 Решение 2.2

Справочник *Одноклассники*:

- Представление объекта — *Одноклассник*. Команда должна быть в единственном числе.
- Расширенное представление объекта — *Одноклассник*. Форма элемента — это один одноклассник.
- Представления списка можно не задавать, т. к. для этого вполне подходит синоним — *Одноклассники*. В командном интерфейсе он достаточно короток, а в списке он не требует какого-либо пояснения.

Справочник *СловарьИностранныхСлов*:

- Представление объекта — *Слово*. Команда должна быть короткой. Если других словарей в программе нет, то вполне можно использовать это представление. Если есть и другие словари, то тогда, конечно, нужно сделать так, чтобы их представления объектов отличались друг от друга.
- Расширенное представление объекта — *Иностранное слово*. Лишний раз поясняет, что в этом справочнике хранятся именно иностранные слова.
- Представление списка — *Словарь*. Команда должна быть короткой.
- Расширенное представление списка — *Иностранные слова*. Это то, что вы видите в словаре. Расширенное представление используется ещё и как подсказка у команды перехода к списку. Чтобы её увидеть, нужно подвести курсор к команде *Словарь* и подождать немного. Расширенное представление, заданное именно так, хорошо поясняет, что это за словарь.

Справочник *Ученики*:

- Представление объекта — *Ученик*.
- Расширенное представление объекта — *Ученик*.
- Представление списка можно не задавать, так как для этого вполне подходит синоним.
- Расширенное представление списка — *Список учеников*. Это просто привычка и особенность нашей речи. Про одноклассников мы не говорим «список одноклассников». Мы говорим просто «одноклассники». Или друзья, или коллеги по работе. Потому что это неформальная, неофициальная группа людей. А ученики — это члены учебного коллектива. То есть некоторая официальная, формальная классификация. В таких случаях мы обычно говорим «список учеников», или «список участников соревнований», или «список игроков футбольной команды».

Б.6 Решение 3.1

```
МойРост = 150;  
МойРост = 165;
```

Не забывайте вставлять пробелы перед знаком равенства и после него.

Б.7 Решение 3.2

```
МоеИмя = "Петя";
```

Букву «ё» не используют в именах. Строковый литерал нужно заключать в кавычки.

Б.8 Решение 3.3

```
ДомашнийАдрес = "Можайское ш., д.37, кв.36, г. Одинцово, Московская обл.";
```

Строка может содержать пробелы, знаки препинания и другие символы.

Б.9 Решение 3.4

```
Урок1 = "Музыка";  
Урок2 = "Математика";
```

Можно назвать переменные *ПервыйУрок*, *ВторойУрок*. Нельзя назвать переменные *1Урок*, *2Урок*.

Б.10 Решение 3.5

```
ТелефонныйНомер = 9031234567;  
// или  
ТелефонныйНомер = "(903) 123-45-67";
```

Телефонный номер содержит только цифры. Поэтому можно записать его как число. В то же время для удобства принято выделять в нём код города (оператора) и разделять разряды. В этом случае можно записывать его как строку.

Б.11 Решение 3.6

```
Оценка = 5;  
Оценка = "4+";
```

Одна и та же переменная может хранить значения разных типов. Сначала в ней может находиться число, затем вы можете поместить в неё строку.

Б.12 Решение 3.7

```
СредняяСкорость = 5;
РасстояниеДоШколы = 6;
СколькоМинут = РасстояниеДоШколы / СредняяСкорость * 60;
```

Б.13 Решение 3.8

```
СредняяСкорость = 5;
РасстояниеДоШколы = 6;
СколькоРаз = СредняяСкорость * 24 / РасстояниеДоШколы / 2;
```

Расстояние за сутки *СредняяСкорость* * 24.

Б.14 Решение 3.9

```
СредняяСкорость = 15 ;
РасстояниеДоШколы = 6;
СколькоРаз = СредняяСкорость * 13 / РасстояниеДоШколы / 2;
```

Нужно изменить значение средней скорости и количество часов, которые тратятся на поездку.

Б.15 Решение 3.10

```
// Понедельник
ВПортфеле = 0;
ВзялИзДома = 2;
ВПортфеле = ВПортфеле + ВзялИзДома;

// Вторник
ВзялИзДома = ВзялИзДома + 2;
ВПортфеле = ВПортфеле + ВзялИзДома;

// Среда
ВзялИзДома = ВзялИзДома + 2;
ВПортфеле = ВПортфеле + ВзялИзДома;
```

Б.16 Решение 3.11

```
ДлинаУрока = 45;
ДлинаПеремены = 15;
ДлинаБольшойПеремены = 25;
ВсегоУроков = 6;
ВремяВШколе = ДлинаУрока * ВсегоУроков + ДлинаПеремены * (ВсегоУроков - 2) +
ДлинаБольшойПеремены;
```

Б.17 Решение 3.12

```
СтупенейВМарше = 10;
СтупенейВПодъезде = 5;
Этаж = 7;
СтупенейДоКвартиры = СтупенейВМарше * 2 * (Этаж - 1) + СтупенейВПодъезде;
```

Б.18 Решение 3.13

```
Возраст = "21";
Результат = "Мой возраст " + Возраст + " год";
```

Б.19 Решение 3.14

```
Улица = "Можайское ш.";
НомерДома = "37";
НомерКвартиры = "36";
Город = "Одинцово";
Область = "Московская";
Откуда = Улица + ", д." + НомерДома + ", кв." + НомерКвартиры + ", г. " + Город + ", " +
Область + " обл.;"
```

Б.20 Решение 3.15

```
Результат = '20160902000000';
```

Б.21 Решение 3.16

```
Результат = Дата(2016, 9, 2);
```

Если часы, минуты и секунды равны нулю, их можно не указывать.

Б.22 Решение 3.17

```
ПроизвольнаяДата = ТекущаяДата();
Результат = КонецНедели(ПроизвольнаяДата) + 1;
```

Б.23 Решение 3.18

```
ПроизвольнаяДата = ТекущаяДата();
// Вариант 1
ДевятьУтра = НачалоДня(ПроизвольнаяДата) + 60 * 60 * 9;
// Вариант 2
ДевятьУтра = Дата(Год(ПроизвольнаяДата), Месяц(ПроизвольнаяДата), День(ПроизвольнаяДата),
9, 0, 0);
```

Б.24 Решение 3.19

```
Дата1 = КонецДня(ТекущаяДата());
Дата2 = НачалоДня(ТекущаяДата());
Интервал = Дата1 - Дата2;

// Разделить на часы, в часе 3600 секунд
Часов = Цел(Интервал / 3600);
ОстатокСекунд = Интервал % 3600;

// Разделить на минуты, в минуте 60 секунд
Минут = Цел(ОстатокСекунд / 60);

// Остаток от деления на минуты - это секунды
Секунд = ОстатокСекунд % 60;

Результат = Стока(Часов) + " ч. " + Стока(Минут) + " мин. " + Стока(Секунд) + " с.";
```

Б.25 Решение 3.20

```
ЯИдуНаРаботу = Истина;
ИдетДождь = Истина;
НадоВзятьЗонт = ЯИдуНаРаботу И ИдетДождь;
```

Б.26 Решение 3.21

```
СегодняВыходной= Истина;
ЯБолею = Истина;
ЯНеИдуВШколу = СегодняВыходной ИЛИ ЯБолею;
```

Б.27 Решение 3.22

```
СегодняВыходной = Истина;
СегодняПраздник = Истина;
ХорошаяПогода = Истина;
ЯИдуГулять = (СегодняВыходной ИЛИ СегодняПраздник) И ХорошаяПогода;
```

Б.28 Решение 3.23

```
СегодняСуббота = Истина;
СегодняВоскресенье = Истина;
ЯИдуНаРаботу = НЕ (СегодняСуббота ИЛИ СегодняВоскресенье);
```

Б.29 Решение 3.24

```

НомерДняНедели = 3;

Если НомерДняНедели < 6 Тогда
    ВремяДляИгры = 1;

Иначе
    ВремяДляИгры = 4;

КонецЕсли;

```

Б.30 Решение 3.25

```

НомерДняНедели = 10;

Если НомерДняНедели > 0 И НомерДняНедели < 6 Тогда
    ВремяДляИгры = 1;

ИначеЕсли НомерДняНедели = 6 ИЛИ НомерДняНедели = 7 Тогда
    ВремяДляИгры = 4;

Иначе
    ВремяДляИгры = 0;

КонецЕсли;

```

Б.31 Решение 3.26

```

ЕстьКефир = Истина;
ЕстьРяженка = Истина;
ЕстьМолоко = Истина;
ТекущийЧас = Час(ТекущаяДата());
МояПокупка = "Ничего";

// Супермаркет
Если ТекущийЧас >= 9 И ТекущийЧас < 20 Тогда

    Если ЕстьКефир Тогда
        МояПокупка = "Кефир";

    ИначеЕсли ЕстьРяженка Тогда
        МояПокупка = "Ряженка";

    КонецЕсли;

КонецЕсли;

// Круглосуточный магазин
Если МояПокупка = "Ничего" И ЕстьМолоко Тогда
    МояПокупка = "Молоко";

КонецЕсли;

```

Б.32 Решение 3.27

Для НомерМесяца = 4 По 6 Цикл

Если НомерМесяца = 5 Тогда
Сказать = "5-й месяц 31 день";

Иначе
Сказать = Стока(НомерМесяца) + "-й месяц 30 дней";
КонецЕсли;

КонецЦикла;

Б.33 Решение 3.28

Для Счетчик = 0 По 4 Цикл
Сказать = Стока(5 - Счетчик);

КонецЦикла;

Б.34 Решение 3.29

Процедура ПриНачалеРаботыСистемы()

Результат = ПолучитьМесяц(ТекущаяДата());

КонецПроцедуры

Функция ПолучитьМесяц(ПроизвольнаяДата)

НачалоПериода = НачалоМесяца(ПроизвольнаяДата);
КонецПериода = КонецМесяца(ПроизвольнаяДата);

Возврат ПредставлениеПериода(НачалоПериода, КонецПериода);

КонецФункции

Б.35 Решение 3.30

Процедура ПриНачалеРаботыСистемы()

```
Дата1 = КонецДня(ТекущаяДата());
Дата2 = НачалоДня(ТекущаяДата());
```

```
Результат = ПериодСтрой(Дата2, Дата1);
```

КонецПроцедуры

Функция ПериодСтрой(ДатаНачала, ДатаОкончания)

```
Интервал = ДатаОкончания - ДатаНачала;
```

```
// Разделить на часы, в часе 3600 секунд
```

```
Часов = Цел(Интервал / 3600);
```

```
ОстатокСекунд = Интервал % 3600;
```

```
// Разделить на минуты, в минуте 60 секунд
```

```
Минут = Цел(ОстатокСекунд / 60);
```

```
// Остаток от деления на минуты - это секунды
```

```
Секунд = ОстатокСекунд % 60;
```

```
Возврат Стока(Часов) + " ч. " + Стока(Минут) + " мин. " + Стока(Секунд) + " с.;"
```

КонецФункции

Б.36 Решение 3.31

Процедура ПриНачалоРаботыСистемы()

```
СообщитьМесяц(ТекущаяДата());
```

КонецПроцедуры

Процедура СообщитьМесяц(ПроизвольнаяДата)

```
НачалоПериода = НачалоМесяца(ПроизвольнаяДата);
```

```
КонецПериода = КонецМесяца(ПроизвольнаяДата);
```

```
Результат = ПредставлениеПериода(НачалоПериода, КонецПериода);
```

```
ПоказатьОповещениеПользователя(Результат);
```

КонецПроцедуры

Б.37 Решение 3.32

Процедура ПриНачалеРаботыСистемы()

```
Дата1 = КонецДня(ТекущаяДата());
Дата2 = НачалоДня(ТекущаяДата());
СообщитьПериод(Дата2, Дата1);
```

КонецПроцедуры

Процедура СообщитьПериод(ДатаНачала, ДатаОкончания)

```
Интервал = ДатаОкончания - ДатаНачала;

// Разделить на часы, в часе 3600 секунд
Часов = Цел(Интервал / 3600);
ОстатокСекунд = Интервал % 3600;

// Разделить на минуты, в минуте 60 секунд
Минут = Цел(ОстатокСекунд / 60);

// Остаток от деления на минуты - это секунды
Секунд = ОстатокСекунд % 60;

Результат = Стока(Часов) + " ч. " + Стока(Минут) + " мин. " + Стока(Секунд) + " с./";

ПоказатьОповещениеПользователя(Результат);
```

КонецПроцедуры

Б.38 Решение 3.33

```
Месяцы = Новый Массив();
Месяцы.Добавить("Январь");
Месяцы.Добавить("Февраль");
Месяцы.Добавить("Март");
Месяцы.Добавить("Апрель");
Месяцы.Добавить("Май");
Месяцы.Добавить("Июнь");
Месяцы.Добавить("Июль");
Месяцы.Добавить("Август");
Месяцы.Добавить("Сентябрь");
Месяцы.Добавить("Октябрь");
Месяцы.Добавить("Ноябрь");
Месяцы.Добавить("Декабрь");
```

Б.39 Решение 3.34

Массив		
Индекс	Значение элемента	Тип элемента
0	"Январь"	Строка
1	"Февраль"	Строка
2	"Март"	Строка
3	"Апрель"	Строка
4	"Май"	Строка
5	"Июнь"	Строка
6	"Июль"	Строка
7	"Август"	Строка
8	"Сентябрь"	Строка
9	"Октябрь"	Строка
10	"Ноябрь"	Строка
11	"Декабрь"	Строка

Рисунок Б.1. Содержимое массива в конфигураторе

Б.40 Решение 3.35

```
ИндексИюня = Месяцы.Найти("Июнь");
Месяцы.Вставить(ИндексИюня, "--- Начало лета");

ИндексАвгуста = Месяцы.Найти("Август");
Месяцы.Вставить(ИндексАвгуста + 1, "--- Конец лета");
```

Массив		
Индекс	Значение элемента	Тип элемента
0	"Январь"	Строка
1	"Февраль"	Строка
2	"Март"	Строка
3	"Апрель"	Строка
4	"Май"	Строка
5	"--- Начало лета"	Строка
6	"Июнь"	Строка
7	"Июль"	Строка
8	"Август"	Строка
9	"--- Конец лета"	Строка
10	"Сентябрь"	Строка
11	"Октябрь"	Строка
12	"Ноябрь"	Строка
13	"Декабрь"	Строка

Рисунок Б.2. Летние месяцы, выделенные в массиве

Б.41 Решение 3.36

```
Для Индекс = 0 По Месяцы.ВГраница() Цикл
    НазваниеМесяца = Месяцы.Получить(Индекс);
    Месяцы.Установить(Индекс, НазваниеМесяца + " 2016 г.");
КонецЦикла;
```

Б.42 Решение 3.37

```

СписокУчеников = Новый Массив;
СписокУчеников.Добавить("Сергеев");
СписокУчеников.Добавить("Дмитриев");
СписокУчеников.Добавить("Захаров"); // Александров
СписокУчеников.Добавить("Максимов");

ИндексУченика = СписокУчеников.Найти("Захаров");
Если ИндексУченика <> Неопределено Тогда
    СписокУчеников.Удалить(ИндексУченика);

КонецЕсли;

```

Б.43 Решение 3.38

```

ДниНедели = Новый Массив();
ДниНедели.Добавить("Понедельник");
ДниНедели.Добавить("Вторник");
ДниНедели.Добавить("Среда");
ДниНедели.Добавить("Четверг");
ДниНедели.Добавить("Пятница");
ДниНедели.Добавить("Суббота");
ДниНедели.Добавить("Воскресенье");

ТретийДеньНедели = ДниНедели[2];
ПятыйДеньНедели = ДниНедели[4];

```

Б.44 Решение 3.39

```

ДниНедели = Новый Массив();
ДниНедели.Добавить("Понедельник");
ДниНедели.Добавить("Вторник");
ДниНедели.Добавить("Среда");
ДниНедели.Добавить("Четверг");
ДниНедели.Добавить("Пятница");
ДниНедели.Добавить("Суббота");
ДниНедели.Добавить("Воскресенье");

ДниНедели[5] = ДниНедели[5] + " вых.";
ДниНедели[6] = ДниНедели[6] + " вых.";

```

Б.45 Решение 3.40

```

ДниНедели = Новый Массив();
ДниНедели.Добавить("Понедельник");
ДниНедели.Добавить("Вторник");
ДниНедели.Добавить("Среда");
ДниНедели.Добавить("Четверг");
ДниНедели.Добавить("Пятница");
ДниНедели.Добавить("Суббота");
ДниНедели.Добавить("Воскресенье");

БудниДни = Новый Массив();
Для Каждого ТекущийЭлемент Из ДниНедели Цикл
    Если ТекущийЭлемент = "Суббота" ИЛИ ТекущийЭлемент = "Воскресенье" Тогда
        Продолжить;

    КонецЕсли;

    // Большой и сложный алгоритм
    БудниДни.Добавить(ТекущийЭлемент);

    // ...

КонецЦикла;

```

Б.46 Решение 3.41

```

ДниНедели = Новый Массив();
ДниНедели.Добавить("Понедельник");
ДниНедели.Добавить("Вторник");
ДниНедели.Добавить("Среда");
ДниНедели.Добавить("Четверг");
ДниНедели.Добавить("Пятница");
ДниНедели.Добавить("Суббота");
ДниНедели.Добавить("Воскресенье");

БудниДни = Новый Массив();
Для Каждого ТекущийЭлемент Из ДниНедели Цикл
    Если НЕ (ТекущийЭлемент = "Суббота" ИЛИ ТекущийЭлемент = "Воскресенье") Тогда
        // Простой алгоритм
        БудниДни.Добавить(ТекущийЭлемент);

    КонецЕсли;

КонецЦикла;

```

Б.47 Решение 3.42

```

ДниНедели = Новый Массив();
ДниНедели.Добавить("Понедельник");
ДниНедели.Добавить("Вторник");
ДниНедели.Добавить("Среда");
ДниНедели.Добавить("Четверг");
ДниНедели.Добавить("Пятница");
ДниНедели.Добавить("Суббота");
ДниНедели.Добавить("Воскресенье");

БудниеДни = Новый Массив();
Для Индекс = 0 По ДниНедели.ВГраница() Цикл
    ТекущийЭлемент = ДниНедели[Индекс];
    Если ТекущийЭлемент = "Суббота" Тогда
        Прервать;

    Иначе
        БудниеДни.Добавить(ТекущийЭлемент);

    КонецЕсли;

КонецЦикла;

```

Б.48 Решение 3.43

```

Года = Новый Массив();
Для НомерГода = 2000 По 2020 Цикл
    Года.Добавить(НомерГода);

КонецЦикла;

ВисокосныеГода = Новый Массив();
Для Каждого Год Из Года Цикл

    Если Год % 4 = 0 Тогда
        ВисокосныеГода.Добавить(Год);

    КонецЕсли;

КонецЦикла;

```

В первом случае удобно использовать цикл *Для По*, потому что в нём есть переменная, которая автоматически увеличивается на единицу. Эту переменную можно использовать в качестве номера года.

Во втором случае удобно использовать цикл *Для Каждого*, потому что порядок добавления високосных годов не важен, а важно только, чтобы были добавлены все годы, которые есть.

Индекс	Значение элемента	Тип элемента
0	2 000	Число
1	2 004	Число
2	2 008	Число
3	2 012	Число
4	2 016	Число
5	2 020	Число

Рисунок Б.3. Високосные годы

Б.49 Решение 3.44

```
Адрес = Новый Структура();
Адрес.Вставить("Квартира", 36);
Адрес.Вставить("Дом", 37);
Адрес.Вставить("Улица", "Можайское шоссе");
Адрес.Вставить("Город", "Одинцово");
Адрес.Вставить("Область", "Московская");
```

Б.50 Решение 3.45

Свойство	Значение	Тип
⊕ Адрес[2]	КлючИЗначение	КлючИЗначение
Значение	"Можайское шоссе"	Строка
Ключ	"Улица"	Строка

Рисунок Б.4. Элемент структуры «Улица»

Б.51 Решение 3.46

```
Адрес = Новый Структура();
Адрес.Вставить("Квартира", 36);
Адрес.Вставить("Дом", 37);
Адрес.Вставить("Улица", "Можайское шоссе");
Адрес.Вставить("Город", "Одинцово");
Адрес.Вставить("Область", "Московская");
```

```
НаселенныйПункт = Адрес.Область + " обл., г. " + Адрес.Город;
Квартира = Адрес.Улица + ", д. " + Адрес.Дом + ", кв." + Адрес.Квартира;
```

Б.52 Решение 3.47

```
ДниРождения = Новый Структура("Алексей, Андрей", Дата(2001, 10, 3), Дата(2004, 2, 29));
```

Б.53 Решение 3.48

```

Адрес = Новый Структура();
Адрес.Вставить("Квартира", 36);
Адрес.Вставить("Корпус", 1);
Адрес.Вставить("Дом", 37);
Адрес.Вставить("Улица", "Можайское шоссе");

Корпус = "";
Если Адрес.Свойство("Корпус", Корпус) Тогда
    Квартира = Адрес.Улица + ", д." + Адрес.Дом + " кв." + Корпус +
        ", кв." + Адрес.Квартира;

Иначе
    Квартира = Адрес.Улица + ", д." + Адрес.Дом + " кв." + Адрес.Квартира;

КонецЕсли;

```

Б.54 Решение 3.49

```

Процедура ПриНачалеРаботыСистемы()
    УстановитьКраткийЗаголовокПриложения("Иванов Петя");
    Серверный.УдалитьПоследнийУрок();

КонецПроцедуры

Процедура УдалитьПоследнийУрок() Экспорт
    Выборка = Документы.УчебныйДень.Выбрать();
    Пока Выборка.Следующий() Цикл
        ДокументОбъект = Выборка.ПолучитьОбъект();
        ИндексПоследнегоУрока = ДокументОбъект.Уроки.Количество() - 1;
        ДокументОбъект.Уроки.Удалить(ИндексПоследнегоУрока);

        ДокументОбъект.Записать();

КонецЦикла;

КонецПроцедуры

```

Процедура общего модуля должна быть объявлена с ключевым словом Экспорт.
Чтобы изменить данные, нужно получить объект документа.
После изменения данных объект документа нужно записать в базу данных.

Б.55 Решение 3.50

Процедура ПриНачалеРаботыСистемы()

```
УстановитьКраткийЗаголовокПриложения("Иванов Петя");
```

```
Серверный.УдалитьПервыйУрок();
```

КонецПроцедуры

Процедура УдалитьПервыйУрок() Экспорт

```
// 20150902 отсутствующий номер
ДокументСсылка = Документы.УчебныйДень.НайтиПоНомеру("20150902");
```

```
Если ДокументСсылка.Пустая() Тогда
    Сообщение = Новый СообщениеПользователю();
    Сообщение.Текст = "Документ не найден";
    Сообщение.Сообщить();
```

Иначе

```
    ДокументОбъект = ДокументСсылка.ПолучитьОбъект();
    ДокументОбъект.Уроки.Удалить(0);
```

```
    ДокументОбъект.Записать();
```

КонецЕсли;

КонецПроцедуры

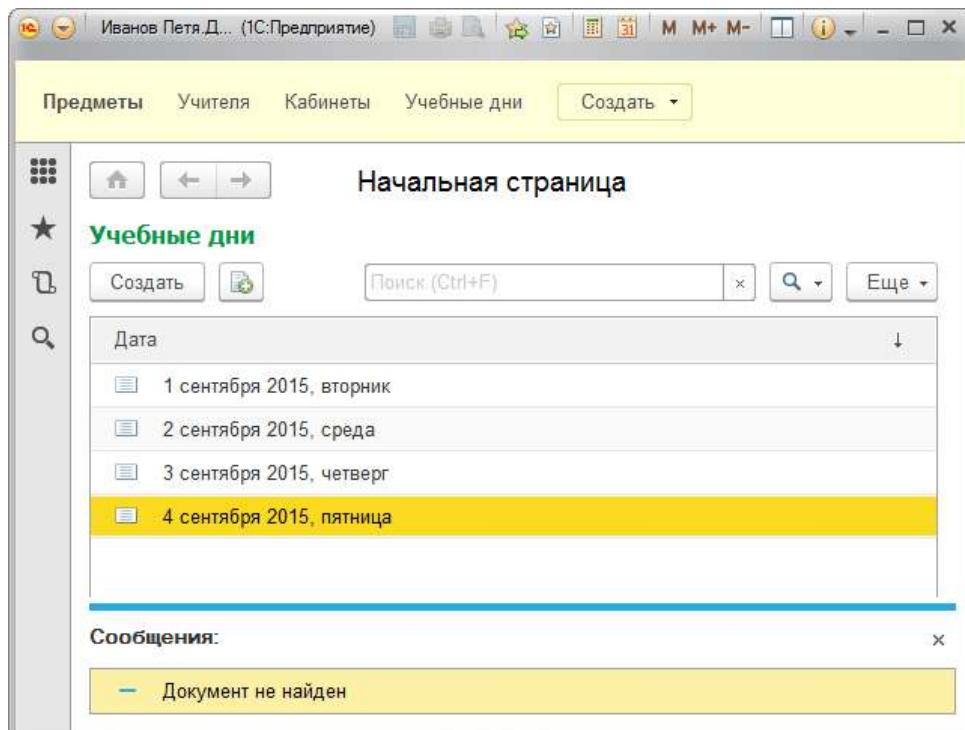


Рисунок Б.5. Сообщение о том, что документ не найден

Б.56 Решение 3.51

Процедура ПриНачалеРаботыСистемы()

```
УстановитьКраткийЗаголовокПриложения("Иванов Петя");
```

```
Серверный.СоздатьНовыйДокумент();
```

КонецПроцедуры

Процедура СоздатьНовыйДокумент() Экспорт

```
ПредметИнформатика = Справочники.Предметы.НайтиПоКоду("Информатика");
```

```
Если НЕ ПредметИнформатика.Пустая() Тогда
```

```
    ДокументОбъект = Документы.УчебныйДень.СоздатьДокумент();
```

```
    ДокументОбъект.Дата = Дата(2015, 9, 8);
```

```
    НоваяСтрока = ДокументОбъект.Уроки.Добавить();
```

```
    НоваяСтрока.Предмет = ПредметИнформатика;
```

```
    НоваяСтрока = ДокументОбъект.Уроки.Добавить();
```

```
    НоваяСтрока.Предмет = ПредметИнформатика;
```

```
    ДокументОбъект.Записать();
```

КонецЕсли;

КонецПроцедуры

Номер документа заполнять не нужно, так как он будет вычислен автоматически в модуле документа в процедуре *ПередЗаписью()*.

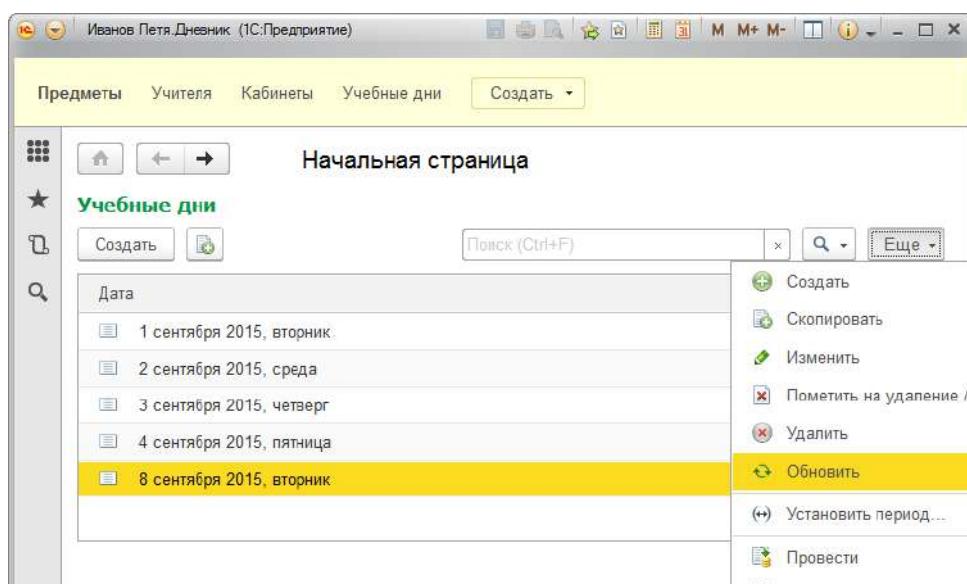


Рисунок Б.6. Команда «Ещё — Обновить»

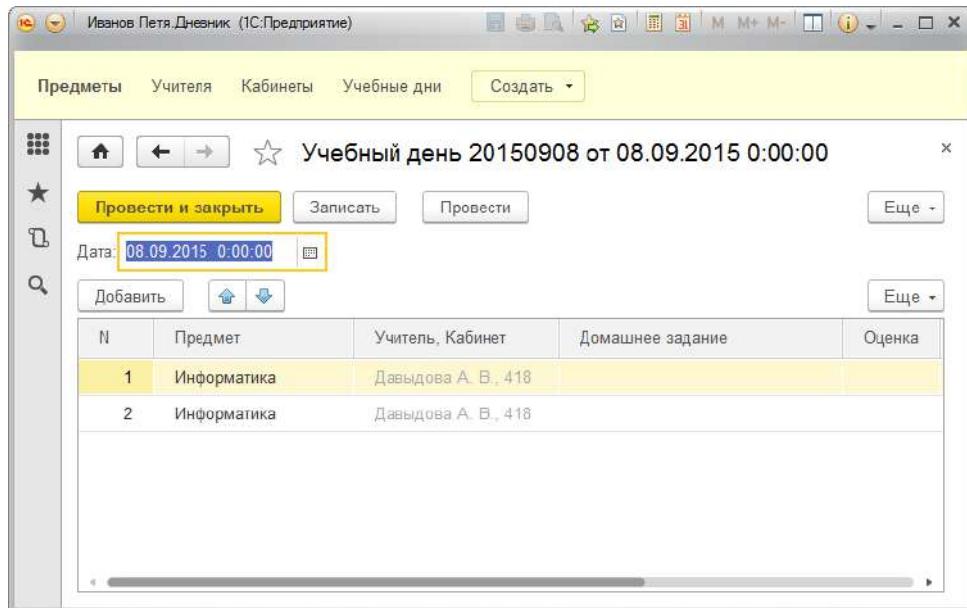


Рисунок Б.7. Новый документ 8 сентября 2015 г.

Б.57 Решение 4.1

&НаКлиенте

Процедура ОбработкаКоманды(ПараметрКоманды, ПараметрыВыполненияКоманды)

```
ЗаполнитьРасписаниеНаСервере( ПараметрКоманды );
ОповеститьОбИзменении(ПараметрКоманды);
```

КонецПроцедуры

&НаСервере

Процедура ЗаполнитьРасписаниеНаСервере(ВыделенныйДень)

```
// Получить начало этой недели.
НачалоЭтойНедели = НачалоНедели( ВыделенныйДень.Дата );
НачалоБудущейНедели = КонецНедели(ТекущаяДатаСеанса()) + 1;
СдвигВСекундах = НачалоБудущейНедели - НачалоЭтойНедели;
```

```
// Перебрать все дни этой недели
Для КоличествоДобавляемыхДней = 0 По 4 Цикл
// ...
```

```
// Вычислить дату и номер для нового документа.
ДатаНового = ДатаСтарого + СдвигВСекундах; // к дате старого прибавить сдвиг
// ...
```

КонецЦикла;

КонецПроцедуры

Б.58 Решение 5.1

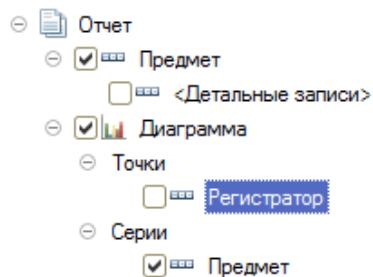


Рисунок Б.8. Новая структура отчёта

Предмет	Оценка
Английский язык	4
Литература	5
Математика	4,5
Музыка	4
Природоведение	4,5
Русский язык	3,5
Итого	4,2

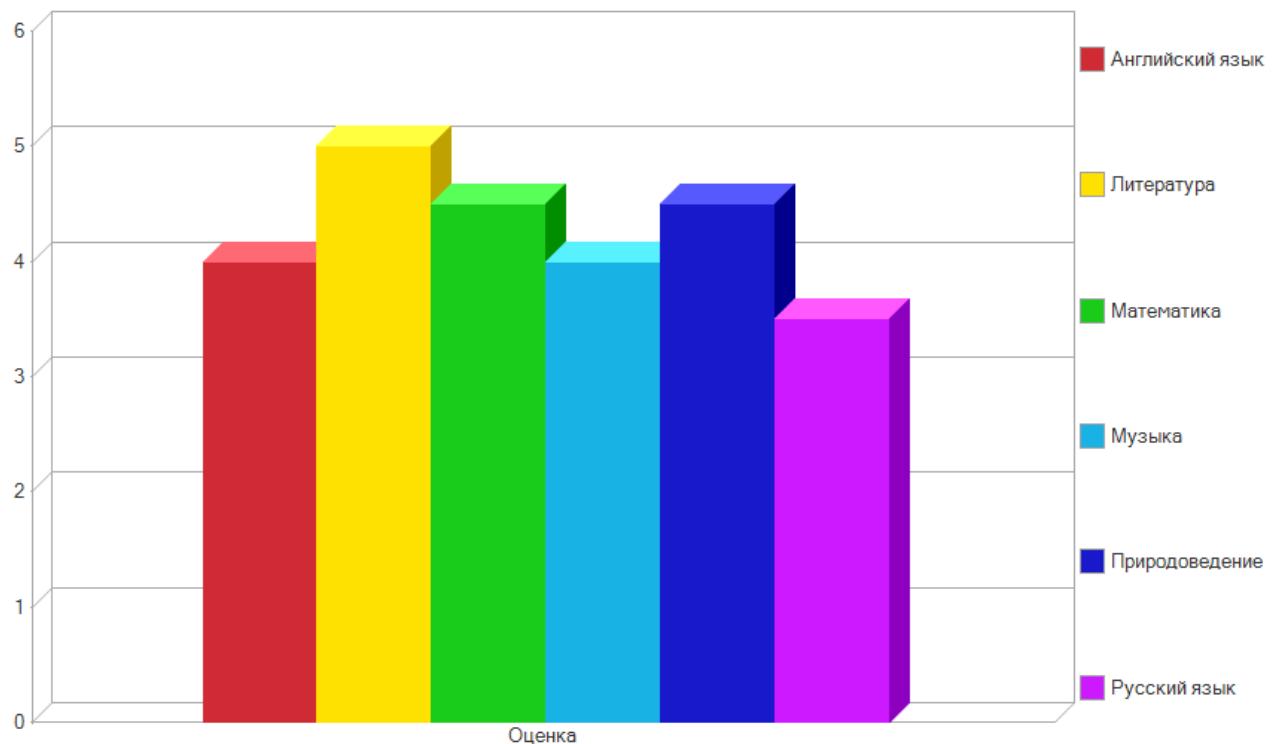


Рисунок Б.9. В таблице и в диаграмме средние оценки по каждому предмету за всё время

Б.59 Решение 5.2

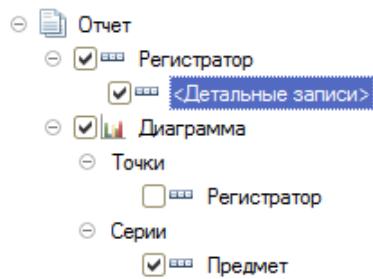


Рисунок Б.10. Новая структура отчёта

Регистратор	Оценка
Предмет	
Учебный день 20150901 от 01.09.2015 12:00:00	4,5
Математика	5
Русский язык	4
Музыка	4
Английский язык	5
Учебный день 20150902 от 02.09.2015 12:00:00	4
Математика	4
Английский язык	3
Природоведение	4
Литература	5
Русский язык	3
Природоведение	5
Итого	4,2

Рисунок Б.11. Оценки по предметам и средние оценки за каждый день

Б.60 Решение 5.3

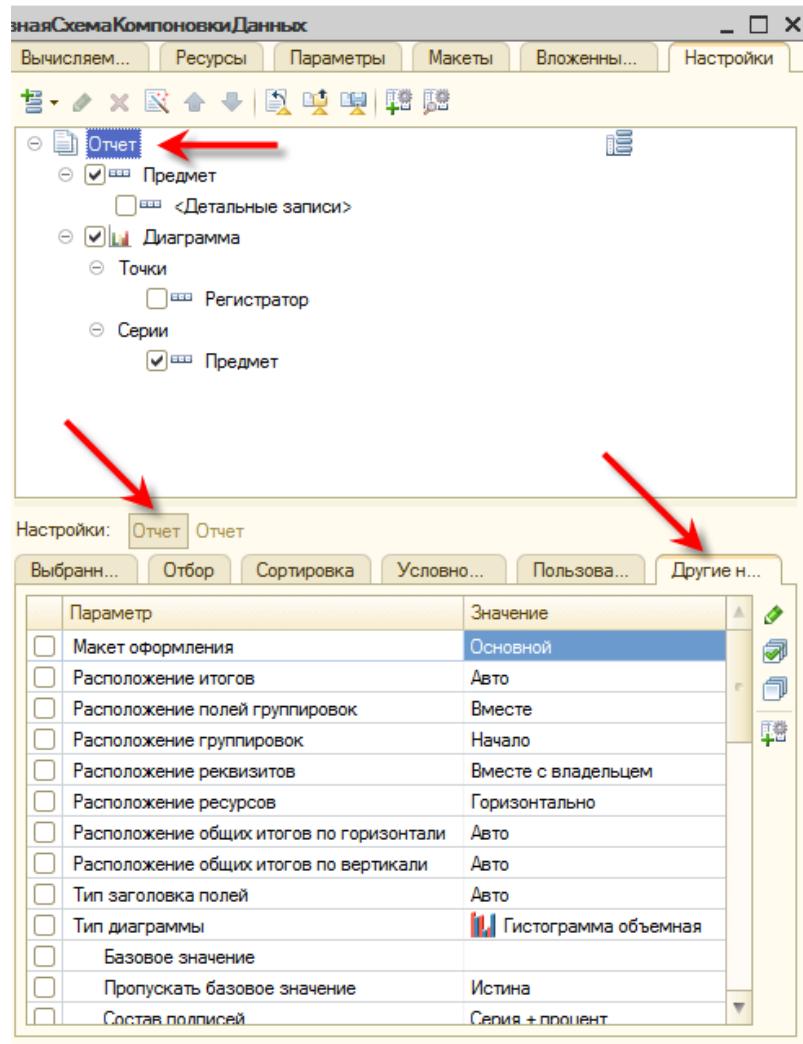


Рисунок Б.12. Другие настройки элемента «Отчёт»

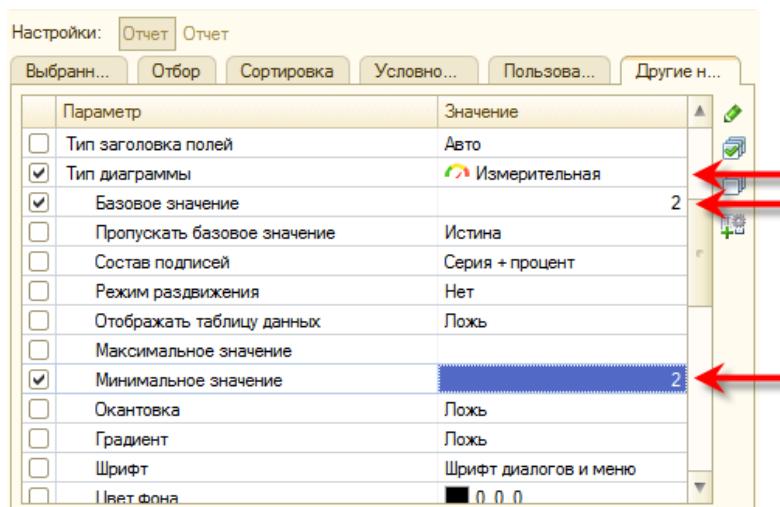


Рисунок Б.13. Тип диаграммы, базовое и минимальное значения

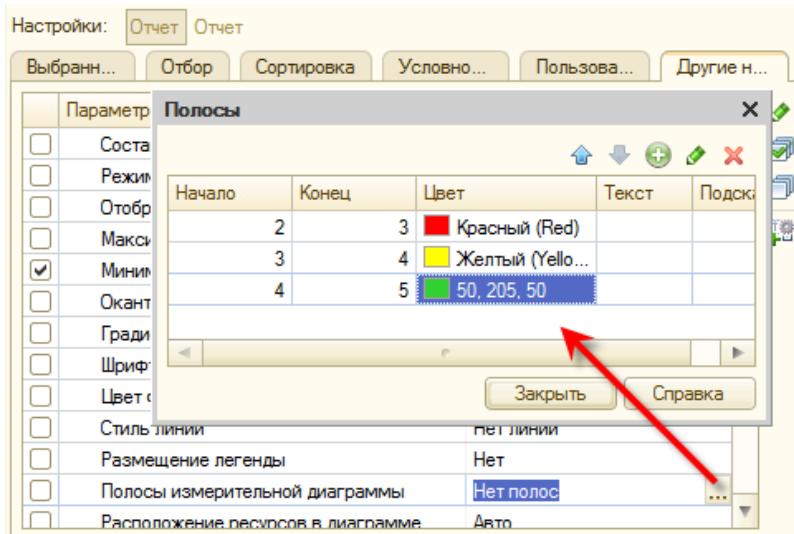


Рисунок Б.14. Полосы измерительной диаграммы

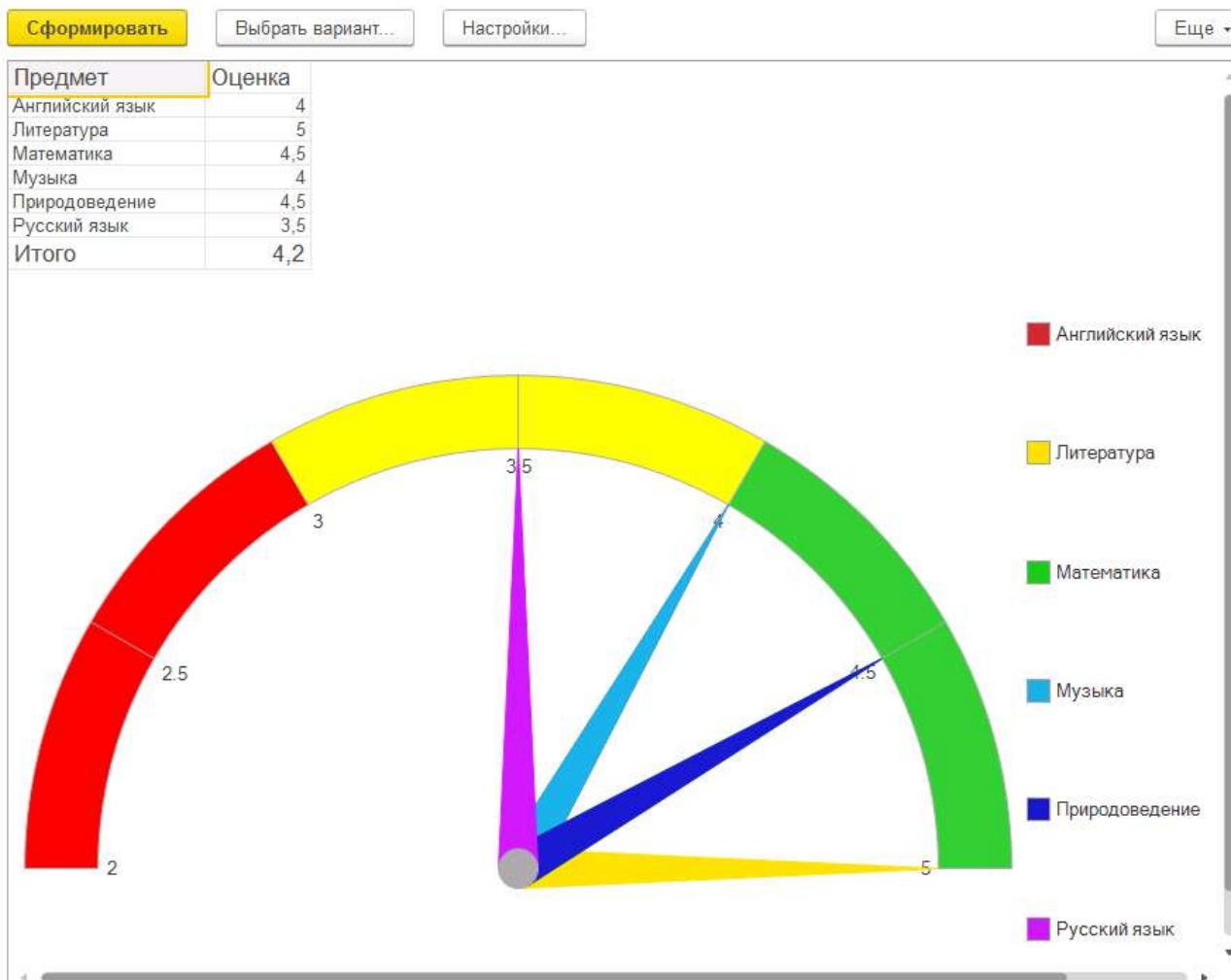


Рисунок Б.15. Измерительная диаграмма

Диаграмма показывает средний балл по каждому предмету. Полосы измерительной диаграммы обозначают диапазоны плохих, средних и хороших оценок. Значения диаграммы отсчитываются не от нуля, а от двойки. Именно для этого вы установили базовое значение и минимальное значение одновременно.

Б.61 Решение 6.1

Текст запроса:

```
ВЫБРАТЬ
    Кабинеты.Код КАК НомерКабинета
ИЗ
    Справочник.Кабинеты КАК Кабинеты
УПОРЯДОЧИТЬ ПО
    Код
```

Результат запроса (количество строк = 9, время выполнения 0.000 с)

НомерКабинета
101
127
131
220
223
311
401
409
418

Рисунок Б.16. Выведите все кабинеты по возрастанию их номеров

Б.62 Решение 6.2

Текст запроса:

```
ВЫБРАТЬ
    Учителя.Наименование КАК ФИО
ИЗ
    Справочник.Учителя КАК Учителя
УПОРЯДОЧИТЬ ПО
    ФИО
```

Результат запроса (количество строк = 9, время выполнения 0.000 с)

ФИО
Авдеева В. М.
Герасимова Е. С.
Давыдова А. В.
Евсеева О. Ю.
Казаков К. Д.
Крюков В. В.
Николаева Т. Я.
Рыбакова А. Е.
Филиппова Л. В.

Рисунок Б.17. Выведите список учителей в алфавитном порядке

Б.63 Решение 6.3

Текст запроса:

```
ВЫБРАТЬ
    ДомашниеЗадания.Период КАК Период,
    ДомашниеЗадания.Предмет КАК Предмет,
    ДомашниеЗадания.ДомашнееЗадание КАК ДомашнееЗадание
ИЗ
    РегистрСведений.ДомашниеЗадания КАК ДомашниеЗадания
ГДЕ
    ДомашниеЗадания.Выполнено = ЛОЖЬ

УПОРЯДОЧИТЬ ПО
    Период,
    ДомашниеЗадания.НомерСтроки
```

Результат запроса (количество строк = 6, время выполнения = 0,02 с):

<u>Запрос: РегистрСведений.ДомашниеЗадания (Записей в результате: 6)</u>			
Период	Предмет	ДомашнееЗадание	
07.09.2015 0:00:00	Музыка	Выучить текст гимна школы.	
07.09.2015 0:00:00	Математика	Задачи 68, 73 и 74.	
09.09.2015 0:00:00	Английский язык	Сделать упражнения 4 и 5. Подготовиться к чтению по ролям.	
09.09.2015 0:00:00	Русский язык	Найти и записать 5 пословиц о языке.	
09.09.2015 0:00:00	Литература	Прочитать статью стр. 3 - 5. Подготовить выразительное чтение песни "Варяг".	
11.09.2015 0:00:00	Информатика	Стр. 5 - 9 читать. Упражнения 2 - 5.	

Рисунок Б.18. Получите список всех невыполненных заданий

Б.64 Решение 7.1

Запрос = Новый Запрос;

Запрос.Текст = "ВЫБРАТЬ

```
| УчебныйДеньУроки.Ссылка.Дата КАК Дата,
| УчебныйДеньУроки.НомерСтроки КАК НомерУрока,
| УчебныйДеньУроки.Предмет.Представление КАК Предмет,
| УчебныйДеньУроки.Ссылка КАК УчебныйДень,
| УчебныйДеньУроки.Оценка КАК Оценка,
| УчебныйДеньУроки.ДомашнееЗадание КАК ДомашнееЗадание,
| УчебныйДеньУроки.Выполнено КАК Выполнено
| ИЗ
| Документ.УчебныйДень.Уроки КАК УчебныйДеньУроки
| ГДЕ
| УчебныйДеньУроки.Ссылка.Дата >= &ДатаНачала
| И УчебныйДеньУроки.Ссылка.Дата <= &ДатаОкончания
|
| УПОРЯДОЧИТЬ ПО
| Дата,
| НомерУрока";
```

Запрос.УстановитьПараметр("ДатаНачала",

НачалоНедели(ДобавитьМесяц(ТекущаяДатаСеанса(), -1)));

Запрос.УстановитьПараметр("ДатаОкончания",

КонецНедели(ДобавитьМесяц(ТекущаяДатаСеанса(), 1)));

РезультатЗапроса = Запрос.Выполнить();

Для описания в тексте запроса условия можно использовать параметры. Их значения нужно установить до выполнения запроса.

Приложение В

Указатель понятий

автогенерируемая форма, 88
автонумерация (свойство), 98
база данных, 310
булево, 231, 232
булевы операции, 232
ввод по строке, 119
ввод по строке (свойство), 100
версия программы, 15
визуальное конструирование, 79
все функции..., 386
встроенный язык, 189
вызов сервера (свойство), 326
выражение, 214
главное меню, 45
группировка, 408
данные, 44, 49, 55
 необъектные, 321
 объектные, 321
дата, 221, 222
дата (свойство), 127
движения документа, 390
движения (свойство), 447
дерево объектов конфигурации, 46
детальные записи, 408
диаграмма, 412
 серии, 413
 точки, 413
диалог выбора цвета, 76
директива компиляции, 356
для каждого цикл, 294
для цикл, 255
документ, 123
если, 240
жёсткий диск, 309
запись, 388

значение, 190
иерархия, 152
измерение (свойство), 392
именованная коллекция, 303
имя (свойство), 62, 127
индекс, 288
инструкция, 195
 для каждого цикл, 294
 для цикл, 255
 если, 240
 пока цикл, 299
 присваивания, 197
 продолжить, 294
интерфейс, 84
 командный, 85
 раздела, 115
пользовательский, 85
программный, 85
 программы, 85
информационная база, 38, 64
исполнение в среде, 17
истина, 231
клиентский контекст, 314
клиентское приложение, 313
кнопка
 выбора, 76
 выпадающего списка, 111
 открытия, 398
 по умолчанию, 137
код (свойство), 94, 100
коллекция значений, 279
 именованная, 303
 массив, 282
 нумерованная, 288
 строковая, 300

- команда, 85, 351
командный интерфейс, 85
командный интерфейс раздела, 115
комментарий, 251
компоновка данных
 конструктор, 398
 настройки, 403
 система, 397
 схема, 397
конкатенация, 219
конструктор
 движений документа, 382
 запроса, 400
 с обработкой результата, 481
 общей формы, 474
 объекта встроенного языка, 281
 схемы компоновки данных, 398
 форматной строки, 162
 формы, 141, 474
контекст, 268
 глобальный, 327
 клиентский, 314
 серверный, 314
контекстная подсказка, 194, 208
контекстное меню, 60
контроль уникальности (свойство), 98
конфигуратор, 20
 параметры, 75
конфигурация, 43
 базы данных, 66
 основная, 65
корень дерева, 48
литерал, 196
 типа булево, 232
 типа дата, 222
ложь, 231
массив, 282
меню
 главное, 45
 контекстное, 60
метод, 281
многострочный режим (свойство), 178
модуль, 187
 менеджера, 346
 общий, 325
 вызов сервера, 326
 сервер, 326
 объекта, 346
управляемого приложения, 187
формы, 188
набор записей, 433
 отбор, 433
назначаемый обработчик событий, 500
наименование (свойство), 94, 100
настройки компоновки данных, 403
начальная страница, 144
необъектные данные, 321
номер (свойство), 127
нумерованная коллекция, 288
область видимости, 268
обработка, 117
обработчик событий
 назначаемый, 500
 фиксированный, 499
общий модуль, 325
объект встроенного языка, 281
объект данных, 80, 321
объект конфигурации, 48
 автонумерация, 98
 ввод по строке, 100
 дата, 127
 движения, 447
 документ, 123
 измерение, 392
 имя, 62, 127
 код, 94, 100
 контроль уникальности, 98
 наименование, 94, 100
 номер, 127
 обработка, 117
 оперативное проведение, 395
 основная форма, 143
 отчёт, 375, 397
 периодичность, 378
 представление объекта, 91
 представление списка, 92
 расширенное представление объекта, 92
 расширенное представление списка, 93
 регистр накопления, 394, 414
 оборотов, 415
 остатков, 415
 регистр сведений, 377, 394
 независимый, 389
 подчинённый регистратору, 389

- режим записи, 378
ресурс, 392
сионим, 62
справочник, 57
табличная часть, 129
удаление движений, 391
объектные данные, 321
окно
 вычисления выражения, 216
 локальных переменных, 203
 редактирования объекта
 конфигурации, 61
оперативная память, 309
оперативное проведение (свойство), 395
операционная система, 11
операция, 214
операция [...], 293
определение функции, 258
основная конфигурация, 65
основная форма (свойство), 143
основной раздел, 86
основной реквизит, 167
отбор, 433
отладчик, 194
отчёт, 375, 397
палитра свойств, 71
панель инструментов, 45, 87
панель функций текущего раздела, 87
параметр
 виртуальной таблицы, 421
 фактический, 261
 формальный, 259
 формы, 504
параметры конфигуратора, 75
переменная, 200
периодичность (свойство), 378
планировщик, 473
платформа, 21
подчинение регистратору, 378, 389
пока цикл, 299
поле, 388
поле выборки, 461
пользовательские настройки, 428
пользовательский интерфейс, 85
пошаговое исполнение, 204
представление, 82, 193
представление объекта (свойство), 91
представление списка (свойство), 92
прикладная программа, 14
прикладное программирование, 14
прикладное программное обеспечение, 14
прикладное решение 1С:Предприятия, 20
прикладной программист, 14
проведение документа, 376, 385, 390
программа, 10
 прикладная, 14
 системная, 13
программирование
 прикладное, 14
 системное, 13
программист
 прикладной, 14
 системный, 13
программное обеспечение
 прикладное, 14
 системное, 13
программный интерфейс, 85
продолжить, 294
псевдоним, 461
рабочая область, 87
раздел, 85
расширенное представление объекта
 (свойство), 92
расширенное представление списка
 (свойство), 93
расширенное редактирование (свойство),
 178
регистр, 374
регистратор, 390
регистр накопления, 394, 414
регистр оборотов, 415
регистр остатков, 415
регистр сведений, 377, 394
регистр сведений, независимый, 389
регистр сведений, подчинённый
 регистратору, 389
редактор
 командного интерфейса, 83
 рабочей области начальной
 страницы, 145
 формы, 147
режим записи (свойство), 378
режим отладки, 74
режим работы
 1С:Предприятие, 43
 Конфигуратор, 43

- режимы работы 1С:Предприятия, 43
реквизит объекта конфигурации, 95
многострочный режим, 178
расширенное редактирование, 178
сионим, 102
стандартный, 96
реквизит формы, 148
основной, 167
ресурс (свойство), 392
свойство, 301
сервер 1С:Предприятия, 313
серверный контекст, 314
сервер (свойство), 326
серии диаграммы, 413
сионим (свойство), 62, 102
синтаксический отступ, 248
синтакс-помощник, 194
система компоновки данных, 397
система программ 1С:Предприятие, 23
система управления базой данных, 313
системная программа, 13
системное программирование, 13
системное программное обеспечение, 13
системный программист, 13
события, 185
события объектов, 344
специализированная среда разработки, 20
список информационных баз, 42
справочник, 57
среда исполнения, 17
среда разработки, 19
специализированная, 20
универсальная, 19
ссылка, 111
ссылка на данные, 58
стандартный реквизит, 96
структура, 300
субд, 313
схема компоновки данных, 397
таблица, 388
таблица запроса, 454
табличная часть (свойство), 129
табличный способ хранения данных, 388
тело функции, 258
технологическая платформа, 21
тип, 192
булево, 231, 232
дата, 221, 222
ссылка, 111
точка останова, 201
точки диаграммы, 413
транзакция, 437
удаление движений (свойство), 391
универсальная среда разработки, 19
的独特性 записей регистра, 392
файл выгрузки информационной базы, 543
фактический параметр, 261
фиксированный обработчик событий, 499
форма, 87
автогенерируемая, 88
конструктор, 141, 474
модуль, 188
объекта, 89
основная, 143
параметр, 504
редактор, 147
 списка, 89
 элемент, 148
формальный параметр, 259
функция, 257
экспорт, 326
элемент формы, 148
язык запросов, 400, 450

Приложение Г

Указатель действий

восстановить положение окна, 112
выбрать цвет, 76
вызвать контекстную подсказку, 208
вычислить выражение, 216
добавить комментарий, 291
добавить начальную страницу, 145
добавить новую информационную базу, 38
добавить объект конфигурации, 60
добавить реквизит справочника, 109
добавить форму, 141
добавить элемент формы, 172
загрузить информационную базу, 543
 зайти в конфигуратор, 43
закомментировать, 291
закрыть прикладное решение, 45
закрыть форму, 90
 запустить 1С:Предприятие, 35
 запустить 1С:Предприятие в режиме отладки, 71
 запустить в пользовательском режиме, 43
 запустить в режиме 1С:Предприятие, 43
 запустить в режиме конфигуратора, 43
 изменить значение переменной в процессе отладки, 242
 изменить значение поля, 61
 найти в палитре свойств, 173
 найти в синтакс-помощнике, 271, 335
 обновить конфигурацию базы данных, 66
 обработать ошибку, 290
 открыть в конфигураторе, 43
 открыть дерево объектов конфигурации, 46
 открыть каталог информационной базы,

310
открыть командный интерфейс основного раздела, 82
открыть конструктор форматной строки, 162, 338
открыть контекстное меню, 60
открыть конфигурацию, 46
открыть локальные переменные, 203
открыть модуль менеджера, 346
открыть модуль объекта, 346
открыть модуль управляемого приложения, 184
открыть модуль формы, 188
открыть окно редактирования объекта конфигурации, 81
открыть рабочую область начальной страницы, 145
открыть стандартные реквизиты, 96
открыть форму для редактирования, 147
перезапустить отладку, 242
перейти к нужной строке модуля, 547
перейти к определению процедуры или функции, 274
перепровести документ, 410
подключить демонстрационную базу, 543
показать реквизит в форме, 170
посмотреть значение выражения, 216
посмотреть значение переменной, 205
посмотреть свойства объекта, 303
посмотреть содержимое коллекции, 287
прочитать сообщение об ошибке, 544
разархивировать, 29
создать обработчик события, 347
сослаться, 58

сохранить изменения конфигурации, 63
удалить элемент коллекции, 297
установить точку останова, 202

шагнуть в, 205
шагнуть из, 275
шагнуть через, 276

Оглавление

Выходные данные	1
Предисловие	2
Благодарности	2
Как работать с книгой	2
Что вы будете уметь	3
Что вы будете делать	4
1 Начало	5
1.1 Воображение	5
1.2 Программа	10
1.3 Как устроено 1С:Предприятие	18
1.4 Зачем нужны прикладные решения 1С:Предприятия	23
1.5 Установка	25
1.5.1 Скачивание дистрибутива	26
1.5.2 Установка платформы 1С:Предприятие 8	31
1.5.3 Как запускать 1С:Предприятие	33
2 Визуальное конструирование	37
2.1 С чего начинается прикладное решение	37
2.2 Список информационных баз	41
2.3 Конфигурация	43
2.4 Дерево объектов конфигурации	45
2.5 Какие объекты конфигурации можно добавлять	52
2.6 Красота, или какой объект выбрать	53
2.7 Данные	55
2.8 Справочник	56
2.9 Кабинеты	59
2.9.1 Информационная база	64
2.9.2 Режим отладки	74
2.9.3 Добавление данных	77
2.9.4 Объект данных	79
2.9.5 Объект конфигурации описывает, как будут выглядеть его данные . .	80
2.9.6 Интерфейс	84
2.9.7 Что такое формы?	88
2.9.8 Представления объекта конфигурации в интерфейсе	91
2.9.9 Наименование и код	94
2.10 Учителя	104
2.11 Предметы	106

2.11.1 Реквизиты	108
2.11.2 Командный интерфейс раздела	115
2.11.3 Ввод по строке	119
2.12 Документ	123
2.13 Учебные дни	125
2.14 Редактирование форм	139
2.14.1 Добавление формы	141
2.14.2 Редактор формы	147
2.14.3 Изменение формы списка	159
2.14.4 Изменение формы объекта	165
3 Встроенный язык	182
3.1 Ваша первая программа — заголовок приложения	183
3.2 События	185
3.3 Модули	187
3.4 Встроенный язык	189
3.5 Значение	189
3.6 Тип	191
3.7 Представление	193
3.8 Где писать примеры и чем пользоваться	194
3.9 Простые типы	194
3.9.1 Почему текст разноцветный	194
3.9.2 Какие бывают инструкции	197
3.9.3 Инструкция присваивания	197
3.9.4 Переменная	199
3.9.5 Точки останова и просмотр значений	201
3.9.6 Изменение значений переменных	207
3.9.7 Контекстная подсказка	208
3.9.8 Выбор имени для переменной	212
3.9.9 Выражение	214
3.9.10 Арифметические операции	218
3.9.11 Операции со строками	219
3.9.12 Тип Дата и операции с датами	221
3.9.13 Тип Булево и логические операции	230
3.9.14 Булевые операции	232
3.9.15 Инструкция Если	239
3.9.16 Красивая программа	248
3.9.17 Инструкция Цикл	253
3.9.18 Функции	256
3.9.19 Контекст и область видимости	264
3.9.20 Процедуры	270
3.9.21 Чтение и отладка процедур и функций	274
3.10 Коллекции значений	278
3.10.1 Объекты встроенного языка	278
3.10.2 Методы, конструкторы	281
3.10.3 Массив	281
3.10.4 Обрабатывайте ошибочные ситуации	290
3.10.5 Используйте операцию [...]	293

3.10.6 Используйте инструкцию Для Каждого Цикл	294
3.10.7 Удаляйте элементы с конца	297
3.10.8 Структура	300
3.11 Прикладные типы	309
3.11.1 База данных	309
3.11.2 Клиент и сервер	312
3.11.3 Прикладные типы	318
3.11.4 Объектные данные	321
3.11.5 Как устроен документ	324
3.11.6 Номер документа УчебныйДень	336
3.11.7 События объектов	344
3.11.8 Установка номера для новых документов	346
4 Автоматическое заполнение расписания	351
5 Регистры и отчёты	373
5.1 Зачем нужны регистры	374
5.2 Что будет в этой главе	377
5.3 Регистр сведений	377
5.3.1 Регистр сведений Оценки	378
5.3.2 Процедура проведения документов	380
5.3.3 Заполнение регистра данными	385
5.3.4 Хранение данных в таблицах	386
5.3.5 Устройство регистра сведений	389
5.3.6 Оперативное проведение	394
5.3.7 Отчёт Успеваемость	395
5.4 Регистр накопления	414
5.4.1 Регистр накопления ПрошедшиеЖанятия	415
5.4.2 Отчёт ПрошедшиеЖанятия	420
5.5 Работа с регистрами из встроенного языка	432
5.5.1 Необъектные данные	432
5.5.2 Регистр сведений ДомашниеЗадания	434
5.5.3 Запись в регистр ДомашниеЗадания	436
5.5.4 Работа с регистрами в модуле документа	445
6 Язык запросов	449
6.1 Чем язык запросов отличается от встроенного языка	450
6.2 Хранение объектных данных	450
6.3 Таблицы запросов	454
6.4 Консоль запросов	456
6.5 Текст запроса	457
7 Планировщик	472
7.1 Планировщик	473
7.2 Создание формы и размещение в ней планировщика	474
7.3 События формы	479
7.4 Получение данных из базы	480
7.5 Настройка	493
7.6 Перехват событий	496

7.7 Отображение будних дней	506
7.8 Отметки оценок и домашние задания	513
7.9 Обновление данных	524
8 Доработка интерфейса	529
8.1 Список домашних заданий	529
8.2 Начальная страница	539
8.3 Командный интерфейс основного раздела	540
A Полезные советы	543
A.1 Как подключить демонстрационную базу	543
A.2 Как прочитать сообщение об ошибке	544
A.3 Как сделать копию рабочей базы	549
Б Решения заданий	553
Б.1 Решение 1.1	553
Б.2 Решение 1.2	553
Б.3 Решение 1.3	553
Б.4 Решение 2.1	554
Б.5 Решение 2.2	554
Б.6 Решение 3.1	555
Б.7 Решение 3.2	555
Б.8 Решение 3.3	555
Б.9 Решение 3.4	555
Б.10 Решение 3.5	555
Б.11 Решение 3.6	555
Б.12 Решение 3.7	556
Б.13 Решение 3.8	556
Б.14 Решение 3.9	556
Б.15 Решение 3.10	556
Б.16 Решение 3.11	556
Б.17 Решение 3.12	557
Б.18 Решение 3.13	557
Б.19 Решение 3.14	557
Б.20 Решение 3.15	557
Б.21 Решение 3.16	557
Б.22 Решение 3.17	557
Б.23 Решение 3.18	557
Б.24 Решение 3.19	558
Б.25 Решение 3.20	558
Б.26 Решение 3.21	558
Б.27 Решение 3.22	558
Б.28 Решение 3.23	558
Б.29 Решение 3.24	559
Б.30 Решение 3.25	559
Б.31 Решение 3.26	559
Б.32 Решение 3.27	560
Б.33 Решение 3.28	560
Б.34 Решение 3.29	560

Б.35 Решение 3.30	561
Б.36 Решение 3.31	561
Б.37 Решение 3.32	562
Б.38 Решение 3.33	562
Б.39 Решение 3.34	563
Б.40 Решение 3.35	563
Б.41 Решение 3.36	563
Б.42 Решение 3.37	564
Б.43 Решение 3.38	564
Б.44 Решение 3.39	564
Б.45 Решение 3.40	565
Б.46 Решение 3.41	565
Б.47 Решение 3.42	566
Б.48 Решение 3.43	566
Б.49 Решение 3.44	567
Б.50 Решение 3.45	567
Б.51 Решение 3.46	567
Б.52 Решение 3.47	567
Б.53 Решение 3.48	568
Б.54 Решение 3.49	568
Б.55 Решение 3.50	569
Б.56 Решение 3.51	570
Б.57 Решение 4.1	571
Б.58 Решение 5.1	572
Б.59 Решение 5.2	573
Б.60 Решение 5.3	574
Б.61 Решение 6.1	576
Б.62 Решение 6.2	576
Б.63 Решение 6.3	577
Б.64 Решение 7.1	577
В Указатель понятий	578
Г Указатель действий	582