Самоучитель

Игорь Ощенко

# Азбука программирования в 1С:Предприятие 8.2

Установка и настройка системы 1С:Предприятие 8.2

Язык программирования 1С:Предприятие 8.2

Язык запросов

Конструирование отчетов

Внешние обработки

Администрирование 1С:Предприятие 8.2

bhv

**Игорь Ощенко**

Самоучитель

# Азбука программирования в 1С:Предприятие 8.2

Санкт-Петербург
«БХВ-Петербург»
2013

УДК 681.3.06
ББК 32.973.26-018.2
О-97

**Ощенко И. А.**

О-97 Азбука программирования в 1С:Предприятие 8.2. — СПб.: БХВ-Петербург, 2013. — 272 с.: ил. — (Самоучитель)

ISBN 978-5-9775-0852-0

Обучение программированию и конфигурированию в 1С:Предприятие 8.2 ведется по принципу «делай как я». Программный код подробно комментируется и разбирается на примерах. Вы научитесь самостоятельно устанавливать систему 1С:Предприятие и подключать к ней любое количество баз данных, дорабатывать и поддерживать работающую конфигурацию, разрабатывать новые отчеты различной сложности, овладеете приемами администрирования системы. Подборка примеров из серии «А как сделать...» содержит типовые приемы программирования и может быть использована читателем в собственных разработках.

*Для начинающих 1С-программистов*

УДК 681.3.06
ББК 32.973.26-018.2

**Группа подготовки издания:**

| | |
|---|---|
| Главный редактор | *Екатерина Кондукова* |
| Зам. главного редактора | *Евгений Рыбаков* |
| Зав. редакцией | *Екатерина Капалыгина* |
| Редактор | *Анна Кузьмина* |
| Компьютерная верстка | *Ольги Сергиенко* |
| Корректор | *Зинаида Дмитриева* |
| Дизайн серии | *Инны Тачиной* |
| Оформление обложки | *Марины Дамбиевой* |

Подписано в печать 31.0712.
Формат 70×100 1/16. Печать офсетная. Усл. печ. л. 21,93.
Тираж 1500 экз. Заказ №
"БХВ-Петербург", 191036, Санкт-Петербург, Гончарная ул., 20.

Первая Академическая типография "Наука"
199034, Санкт-Петербург, 9 линия, 12/28

ISBN 978-5-9775-0852-0

© Ощенко И. А., 2013
© Оформление, издательство "БХВ-Петербург", 2013

# Оглавление

# Введение

В деятельности любого предприятия всегда большую роль играет правильно поставленный учет. Учитываются товары, деньги, договоры, фирмы-поставщики и фирмы-покупатели, сотрудники и многое другое — вся эта информация важна при работе предприятия для правильной оценки и планирования дальнейшей деятельности. В наш век информационных технологий конечно же учет ведется с помощью компьютера. Но для того чтобы компьютерный учет был действительно эффективным, одного компьютера мало, нужна хорошая учетно-аналитическая программа. Можно ведь вести учет в электронных таблицах MS Excel, а можно в специализированной программе, предназначенной для ведения учета. Таких программ, более или менее эффективных, написано немало. Особое место среди них занимают программные продукты компании "1С".

Огромную популярность у пользователей в течение многих лет имела учетная система "1С:Предприятие 7.7". Однако прогресс на месте не стоит, появилась альтернатива — новая, 8-я линейка "1С:Предприятие", которая все более вытесняет своего популярного, но все более устаревающего предшественника. Сменились уже три версии платформы 8 — 8.0, 8.1 и 8.2. Наша книга посвящена системе "1С:Предприятие 8.2" — последней и наиболее удачной версии линейки 8.*x*.

Что же такое "1С:Предприятие 8.2"? Это современный программный комплекс, контролирующий все стадии товарооборота, от производства или поступления товара на склад до его продажи и проведения через бухгалтерские книги. При этом очень важно, что система "1С:Предприятие 8.2" является открытой для доработки и самостоятельной разработки, что позволяет достичь большой гибкости в организации системы учета на предприятии, поскольку можно использовать как типовые программные решения компании "1С", так и дорабатывать их или разрабатывать новые, под специфику работы предприятия.

## Чему обучит книга

Книга поможет читателю, решившему научиться программировать в системе "1С:Предприятие", сделать свои первые шаги на этом увлекательном поприще. Когда-то, когда я сам хотел научиться программировать на языке 1С, такого изобилия книг, посвященных этой системе, еще не было. Много искал, перерыл массу лите-

ратуры, очень часто слишком сложной для новичка, в общем, интересующую информацию добывал по крупицам, из разных источников. Теперь, при написании собственной книги, я пытаюсь сделать ее такой, какую сам искал когда-то, может быть, не слишком академичную и исчерпывающую, зато с упором на наглядность, доходчивость и практические примеры — в общем, азбуку для новичка.

После прочтения читатель усвоит основные принципы разработки в системе "1С:Предприятие 8.2", а именно:

- сможет самостоятельно установить систему "1С:Предприятие 8.2" и подключить базы данных;
- познакомится с синтаксисом языка программирования 1С 8.2;
- сможет дорабатывать и поддерживать конфигурации "1С:Предприятие";
- познакомится с системой компоновки данных и научится разрабатывать отчеты;
- научится работать с отладчиком;
- познакомится со встроенным языком запросов;
- научится писать внешние отчеты и обработки;
- познакомится с администрированием базы 1С, сможет сам назначать права учетным записям пользователей, формировать пользовательские интерфейсы, делать резервные копии базы данных.

## Для кого эта книга

Эта книга для начинающих 1С-программистов или пользователей системы "1С:Предприятие", которые хотят расширить свои знания об этой замечательной системе. Скорее всего, для опытного программиста 1С здесь найдется немного нового, хотя, как справочник, шпаргалка, способ упорядочить свои знания или освежить давно изученное — как знать — возможно, она пригодится и ему.

Именно потому, что читать эту книгу будут начинающие, автор старался излагать свои мысли ясно и предметно, тут же приводя примеры сказанного. Надеюсь, такая форма обучения позволит читателю достаточно быстро уяснить основные принципы разработки и быстро приступить к практике. Книга содержит большое количество иллюстраций.

## Как пользоваться книгой

Читать книгу лучше всего по порядку, начиная с установки, описанной в *главе 1*. В *главе 2* представлены основные операторы и языковые конструкции системы "1С:Предприятие", их практическая ценность при прочтении главы может быть неясна, поскольку результаты работы учебных примеров мы просто выводим на экран. Однако уже в *главе 3* мы столкнемся непосредственно с объектами конфигурации, такими как справочники, документы и регистры, и здесь теория, полученная в *главе 2*, найдет свое применение. Если читатель не планирует работать с языком

запросов или создавать отчеты, то *главу 4* он может пропустить, однако следует помнить, что создание отчетов — одна из наиболее часто встречающихся задач. Также можно было бы пропустить *главу 5*, если бы читатель твердо был уверен, что ему не придется заниматься назначением прав пользователей в системе "1С:Предприятие". Однако программист это должен уметь так же, как и пользоваться отладчиком *(см. главу 6)*.

Мой совет читателю: читать весь материал книги последовательно. Ведь книга уже изначально планировалась, как дающая начинающему программисту самое необходимое. То, что менее важно, было и так пропущено, чтобы не загромождать и не усложнять учебный материал.

Также советую читать книгу за компьютером с установленной системой "1С:Предприятие 8.2" и со специально подключенной конфигурацией "для опытов", которую не жалко испортить, разбирать приведенные примеры на практике, создавать в своей учебной конфигурации такие же, смотреть как они работают. Больше экспериментируйте и импровизируйте. Если в процессе разбора примера возник вопрос "А что будет, если сделать не так, как в примере, а вот так?" — так и сделайте и посмотрите, что получится.

Примеры с пометкой *"А как сделать?"* вполне можно использовать как типовые конструкции для собственных разработок.

ГЛАВА 1

# Установка и первоначальная настройка "1С:Предприятие 8.2"

В жизни многих пользователей программных продуктов нередко наступает такой момент, когда просто использовать имеющийся функционал недостаточно. Меняется работа, меняется занимаемая должность, сфера ответственности, наконец, просто появляется желание узнавать новое и совершенствоваться в выбранной области. Хочется большего: научиться дорабатывать программу или вообще разработать новую под текущие задачи. Для системы "1С:Предприятие" это особо актуально, поскольку она открыта для настройки и доработки и имеет собственный язык программирования. Я предполагаю, что читатель, держащий сейчас в руках эту книгу, — вполне уверенный пользователь системы "1С:Предприятие 8.2", знаком с интерфейсом программы, умеет работать со справочниками, документами, отчетами, но... Но настал момент, когда ему захотелось большего — уметь не только пользоваться программой, но и самостоятельно дорабатывать и конфигурировать ее. Эта книга поможет вам в этом.

Итак, начнем. Работать в программе читатель умеет, а вот как программу установить и подготовить к использованию? Об этом мы узнаем в данной главе.

## Платформа и конфигурация. Что это и зачем они нужны?

Когда пользователь просит программиста "установить 1С", то для первого обычно все просто и понятно. Ему нужна программа, в которой он будет работать, и, как правило, программа эта видится ему как одно целое. На самом деле, все не так просто, поэтому давайте определимся, что же значит "установить 1С"? Что мы, собственно, будем устанавливать?

Представьте себе самую обыкновенную дрель. Она состоит из собственно корпуса с ударно-вращательным механизмом и набора различных насадок разной длины, диаметра и назначения. Так вот, само по себе "1С:Предприятие", так называемая *платформа* — это и есть дрель. Без насадки. А роль насадки играет *конфигурация* — набор форм и программных модулей, которые по необходимости можно видоизменять и дополнять. Это инструмент, подогнанный под задачи конкретно

именно вашей организации, с учетом присущих только ей нюансов. Именно поэтому конфигураций написано много и самых разных.

Для системы "1С:Предприятие" разработано много типовых конфигураций, которые поставляются вместе с ней или могут быть приобретены отдельно. Это "1С:Бухгалтерия", "1С:Управление торговлей", "1С:Управление производственным предприятием", "1С:Управление строительной организацией", "1С:Управление небольшой фирмой", "1С:Зарплата и управление персоналом", "1С:Розница" и многие другие.

Немало конфигураций создано сторонними разработчиками, как партнерами компании "1С", так и отдельными, не связанными с ней программистами, как на основе типовых конфигураций, так и "с нуля".

В общем, при установке мы должны установить как платформу, так и одну или несколько конфигураций. К примеру, нашему пользователю нужны в работе и "1С:Бухгалтерия", и "1С:Управление торговлей", а это уже две разные конфигурации. Платформа же при этом используется одна и та же, и устанавливается она один раз. Конфигураций же вы можете использовать много, при необходимости добавляя новые.

Выяснив, что собой представляют платформа и конфигурация, давайте теперь рассмотрим, что такое *релиз*. Фирма "1С" и ее представители постоянно дорабатывают свои программные продукты. Это связано с различными причинами: найденные ошибки и "глюки", изменение законодательства, требования к программе... Причин много. Вот и появляются обновленные версии платформы и конфигураций. Такие версии и называются *релизами*. Для того чтобы получить информацию о текущих релизах платформы и конфигурации, в режиме Конфигуратора или Предприятия следует зайти в меню **Справка** и выбрать пункт **О программе**. Откроется окно с информацией об используемом программном продукте (рис. 1.1).

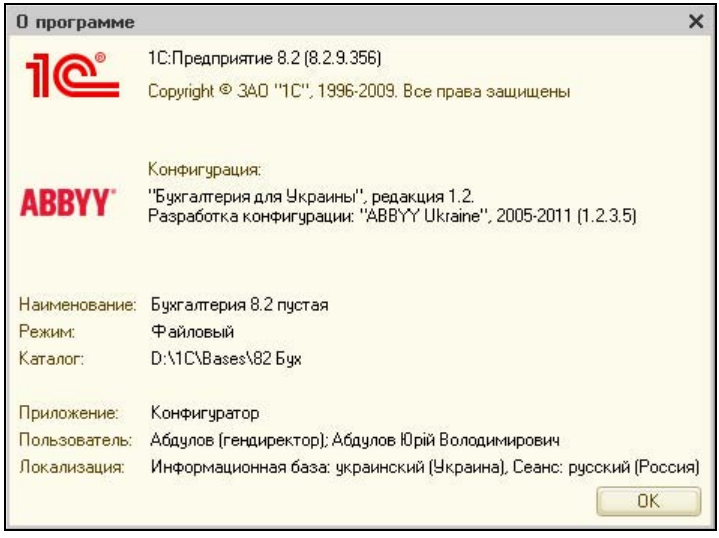



**Рис. 1.1.** Информация об используемом программном продукте фирмы "1С"

В верхней части окна мы видим логотип фирмы-разработчика ("1С"), версию и номер релиза платформы (в примере — 8.2.9.356). Немного ниже расположена информация, касающаяся уже не платформы, а конфигурации (в примере — "Бухгалтерия для Украины", редакция 1.2, разработка компании ABBYY, релиз 1.2.3.5). Еще ниже указаны рабочее название базы данных, путь к ней, информация о пользователе.

В данном случае цифры и есть номер релиза. Релиз платформы 8.2.9.356, а релиз конфигурации — 1.2.3.5.

Теперь, разобравшись с теорией, приступим к установке платформы и конфигурации.

# Установка "1С:Предприятие 8.2"

Хотя эта книга ориентирована на пользователей, работавших с системой "1С:Предприятие", далеко не факт, что они уже устанавливали ее и знают, как это делается. В любом случае, повторить не помешает. Рассмотрим установку пошагово.

Предполагаем, что платформа на нашем компьютере еще не установлена, так что начинаем именно с ее установки.

1. На дисках, которые приобретаются у представителей фирмы "1С", установку можно запустить непосредственно из установочного меню, которое появляется на экране при вставке диска в привод. Но если меню не открылось или вы устанавливаете платформу не с компакт-диска, а, предположим, предварительно переписали содержимое диска на флешку или жесткий диск, то запустить установку платформы можно, дважды щелкнув на файле setup.exe в каталоге с установочными файлами платформы "1С:Предприятие".
2. Запустится мастер установки (рис. 1.2).
3. Нажмем кнопку **Далее**.
4. Появится окно выбора устанавливаемых компонентов (рис. 1.3). В списке компонентов для каждого из них имеется выпадающий список, в котором мы указываем, устанавливать этот компонент, отказаться от его установки или установить частично. На рисунке для установки выбрана собственно платформа "1С:Предприятие" и частично интерфейсы на различных языках (языков интерфейса в поставке платформы много, меня интересовали только два: русский и украинский). Вариант установки визуально мы определяем по виду значка: полная установка компонента выглядит как  , частичная —  , неустанавливаемые компоненты —  . Выберите вариант установки, как на рис. 1.3. Если в дальнейшей работе вдруг понадобятся какие-то компоненты из тех, что мы не установили — это всегда можно будет сделать позже. Сделав выбор, нажмем кнопку **Далее**.
5. Следующее окно — выбор языка интерфейса. В стране, где русский язык не единственный используемый, вполне может понадобиться интерфейс на другом

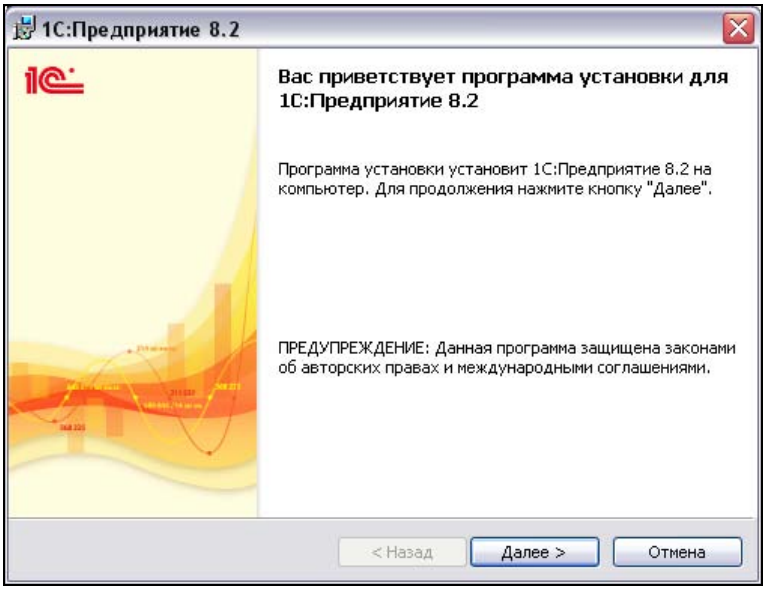



**Рис. 1.2.** Мастер установки платформы "1С:Предприятие"

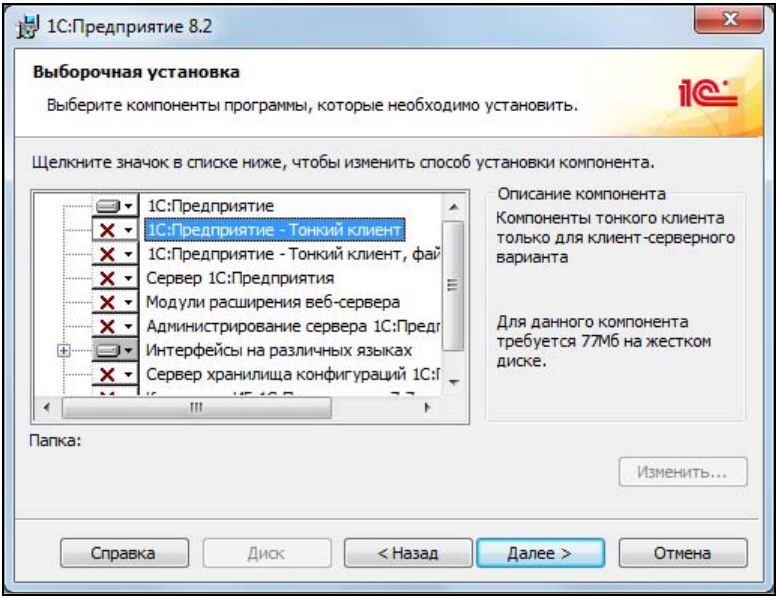



**Рис. 1.3.** Выбираем устанавливаемые компоненты

языке. Украинском, например. Более гибкий вариант — пункт **Системные установки**. Если он выбран, то при запуске платформы "1С:Предприятие" система определяет, какой язык установлен в Windows по умолчанию (посмотреть это можно через апплет **Язык и региональные стандарты** в Панели управления), и этот язык выбирает в качестве языка интерфейса. Именно на этом языке

будут все меню, подсказки и системные сообщения в системе "1С:Предприятие". Разумеется, интерфейс на этом языке должен был быть выбран среди устанавливаемых интерфейсов на предыдущем шаге установки (см. рис. 1.3). Нажмем кнопку **Далее**.

6. На следующем этапе установки можно подтвердить либо отменить введенные данные о пользователе (рис. 1.5). Если мы хотим что-то изменить на предыду-

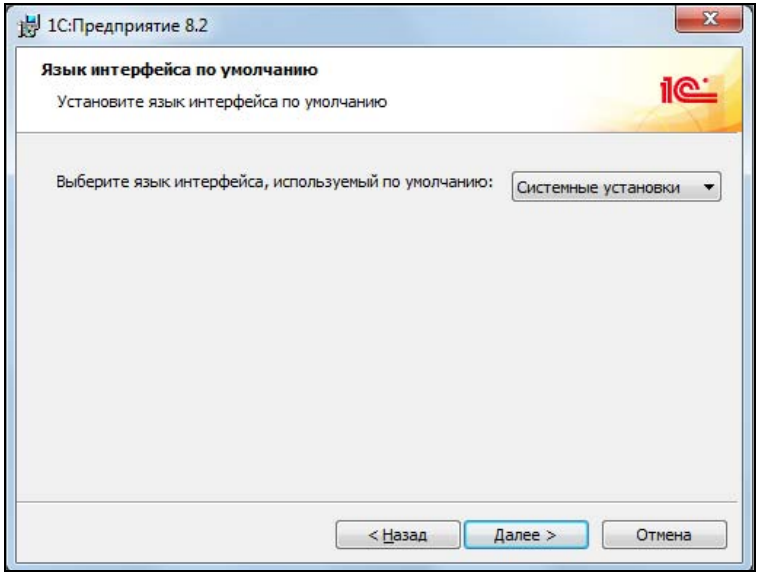



**Рис. 1.4.** Выбор языка интерфейса

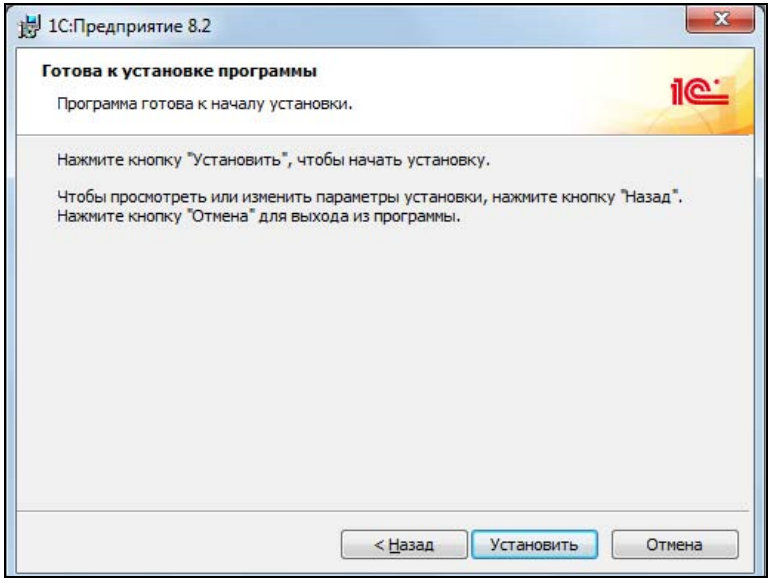



**Рис. 1.5.** Приступаем к установке

щих шагах установки, то можем вернуться, нажав кнопку **Назад**. Если же все верно, нажимаем кнопку **Установить**.

7. Начнется копирование файлов "1С:Предприятие" (рис. 1.6).
8. На следующем шаге нам предлагают установить драйвер защиты (рис. 1.7). Устанавливаем обязательно, ведь этот драйвер отвечает за взаимодействие с ключом защиты — той самой флешкой, прилагающейся к дискам и книгам коробочной поставки "1С:Предприятие". Нажмем кнопку **Далее**.

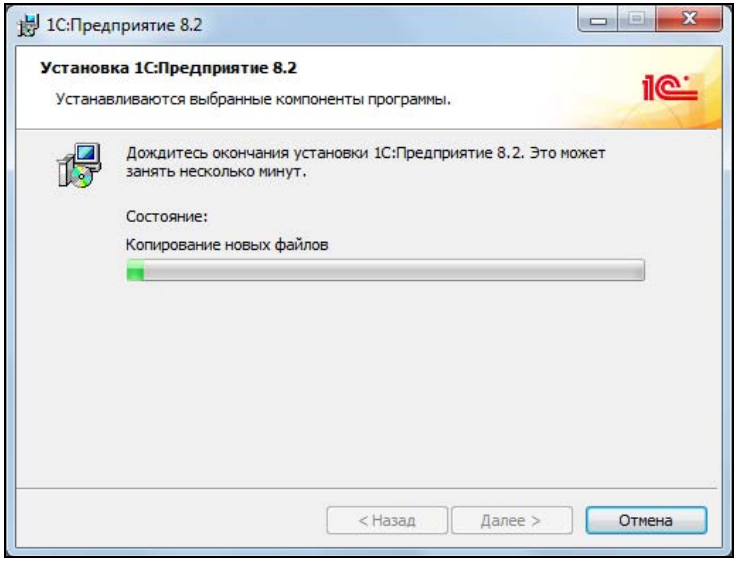



**Рис. 1.6.** Идет процесс установки

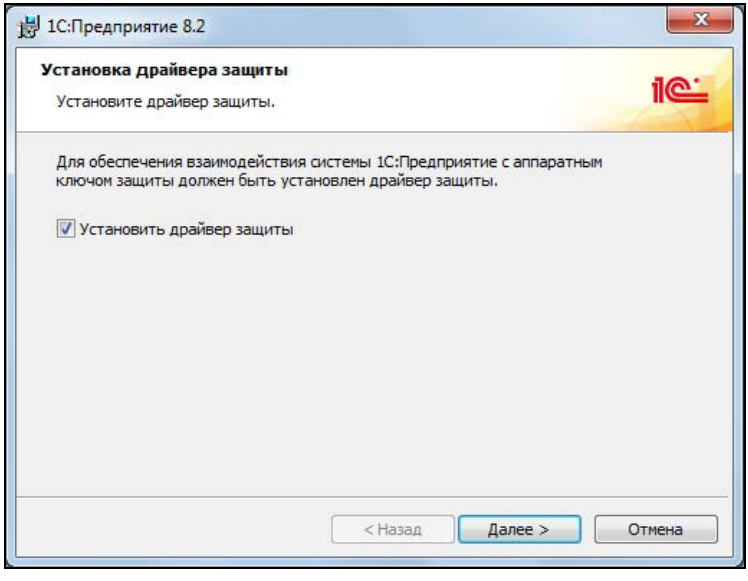



**Рис. 1.7.** Драйвер защиты устанавливаем обязательно

9. Устанавливается драйвер защиты, нас просят немного подождать (рис. 1.8).
10. Итак, установка платформы завершена. Нажмем кнопку **Готово** для завершения работы мастера (рис. 1.9). Файл Readme, который нам предлагают открыть, содержит дополнительную информацию.

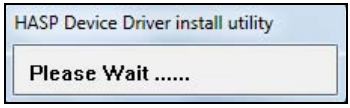



**Рис. 1.8.** Устанавливается драйвер защиты

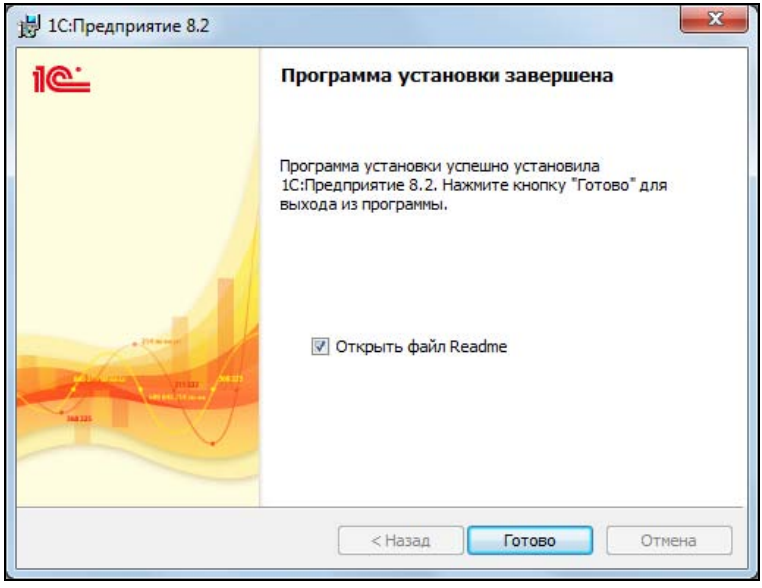



**Рис. 1.9.** Установка платформы завершена

11. Теперь запускаем "1С:Предприятие". Если на этапе установки все было сделано правильно, откроется окно списка информационных баз (рис. 1.10).

    При установке платформы "с нуля" список информационных баз будет пуст. Если же платформа "1С:Предприятие" на компьютере была установлена и вы ее переустановили, то базы данных, бывшие в списке до переустановки, попадут в него и сейчас.
12. Теперь создадим новую, учебную базу, в которой мы в дальнейшем будем экспериментировать и в которой будем писать программный код. Создание новой базы начинается нажатием кнопки **Добавить**. Перед добавлением новой базы (или использованием старой) не забудьте вставить в USB-порт ключ защиты программы, если вы не сделали этого перед установкой. В противном случае будет выдано окно об отсутствии лицензии на программу (рис. 1.11).
13. Если ключ у нас есть, то нажмите кнопку **Нет**. Появится еще одно сообщение об отсутствии лицензии (рис. 1.12), которое закроем кнопкой **OK.**

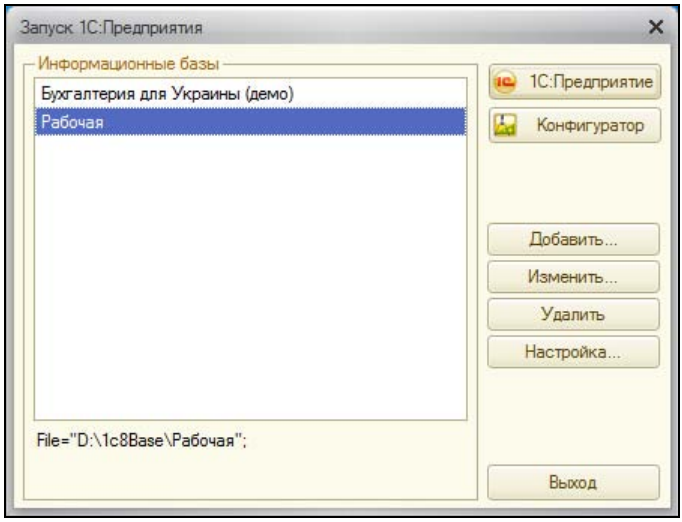



**Рис. 1.10.** Список рабочих баз "1С:Предприятие"

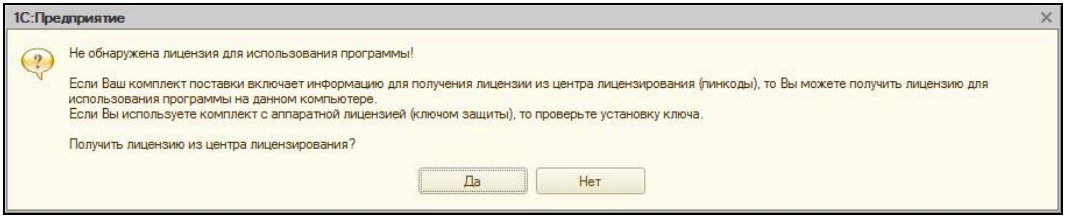



**Рис. 1.11.** Если ключ защиты не вставлен в USB-порт, то бывает так..

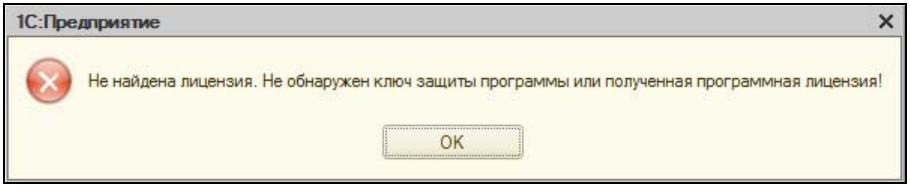



**Рис. 1.12.** ... А затем так

Вставляем ключ защиты. Если ключ уже был вставлен и не определился, не исключено, что он вставлен неплотно или сгорел USB-порт. При нормально работающем USB-порте ключ обычно определяется системой корректно.

14. Если все в порядке, то при нажатии кнопки **Добавить** начинает работать мастер добавления новой информационной базы (рис. 1.13).
15. Нам предлагают либо создать новую базу, либо включить в список ранее созданную. Предположим, мы переустанавливали Windows на компьютере, затем установили "1С:Предприятие", и теперь нам надо подключить наши рабочие базы данных. Или мы получили какую-то готовую базу данных и нам надо

с ней поработать (например, бухгалтер в период отчетности вполне может взять копию рабочей базы, подключить ее и поработать дома), вот тогда мы бы добавляли ее, как существующую. Теперь же наша задача — создать новую базу данных, поэтому оставляем все, как на рис. 1.13, и нажимаем кнопку **Далее**.

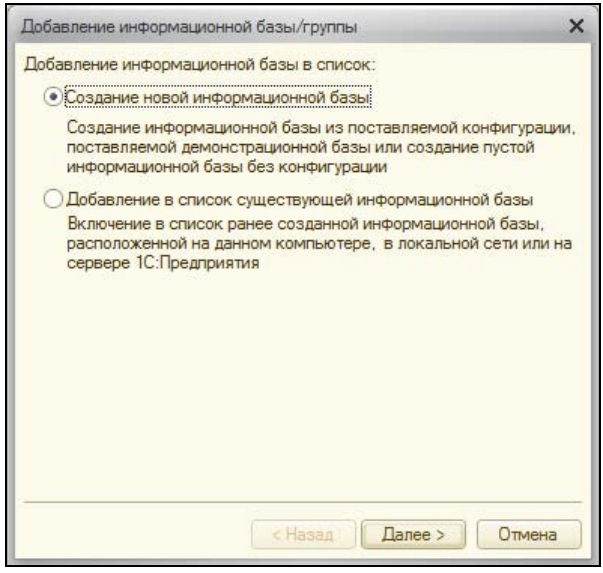



**Рис. 1.13.** Добавляем информационную базу

16. Теперь мы должны выбрать, каким образом мы будем создавать базу данных — из шаблона или просто пустую базу без конфигурации (рис. 1.14).

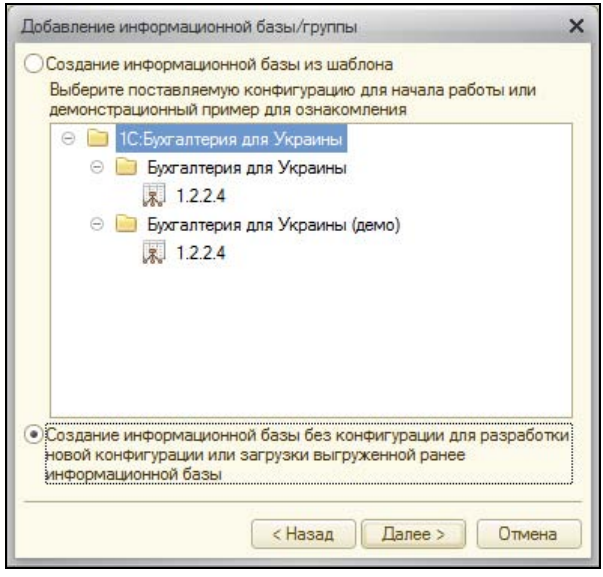



**Рис. 1.14.** Можно создать новую базу данных или на основе шаблона

Что означает "создание из шаблона"? Предположим, на вашем компьютере установлены какие-то типовые конфигурации системы "1С:Предприятие". Например, "Управление Торговлей", "Бухгалтерия", "Управление Производственным предприятием" и т. п. В этом случае мы можем просто и быстро создать новую базу данных сразу на основании одной из этих конфигураций. Мы ведем бухгалтерию нескольких фирм и хотим их вести в разных базах данных? Пожалуйста — устанавливаем "1С:Бухгалтерия" всего один раз, затем создаем новую базу на основе шаблона "1С:Бухгалтерия", а потом просто вносим данные. И так столько раз, сколько у нас будет баз данных. Разумеется, если мы только установили платформу, то список шаблонов у нас будет пуст, никакие конфигурации мы еще не устанавливали. У меня на компьютере уже была установлена "Бухгалтерия для Украины", поэтому она и присутствует в списке шаблонов. Пока что давайте рассмотрим создание пустой базы, без шаблонов (мы ведь условились, что платформу устанавливаем впервые, поэтому и шаблонам взяться пока неоткуда). Так что выбираем переключатель **Создание информационной базы без конфигурации...** и нажимаем кнопку **Далее**. После этого нам предложат дать базе рабочее название (рис. 1.15).

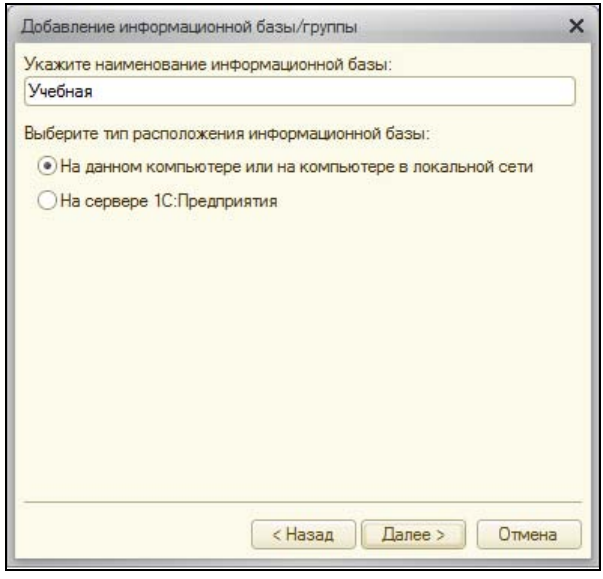



**Рис. 1.15.** Указываем рабочее наименование базы данных и способ ее размещения

17. Название задаем любое, удобное нам. Именно это название будет отображаться в списке информационных баз (см. рис. 1.10.) Местонахождение выбираем **На данном компьютере или на компьютере в локальной сети**. Второй вариант предполагает работу в клиент-серверном варианте, например, под управлением MS SQL Server. Конфигурирование подобных систем мы рассматривать не будем, поскольку по этой теме можно писать отдельную книгу, а наша задача пока что — хорошо уяснить основы. Нажимаем кнопку **Далее**, после чего нам нужно указать местонахождение нашей будущей базы (рис. 1.16).

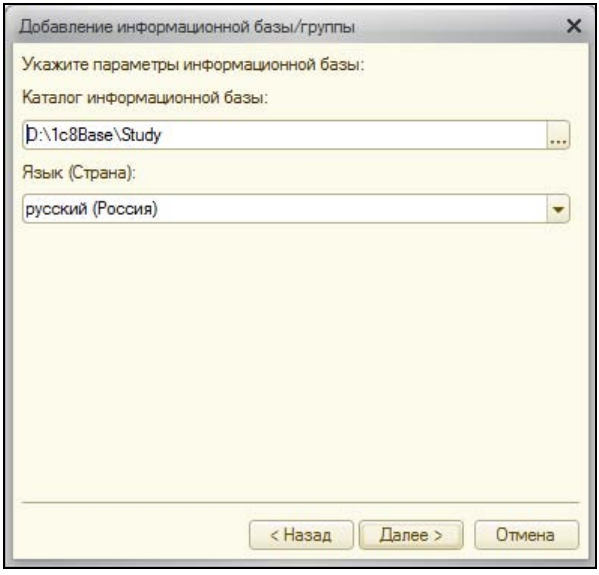



**Рис. 1.16.** Указываем место расположения базы данных

18. Здесь мы прописываем путь к будущей базе данных, а также выбираем язык интерфейса (выбор предоставляется при нескольких установленных языках; если язык только русский, выбирать нам не из чего). Нажимаем кнопку **Далее**, переходя к окну, в котором мы должны задать дополнительные параметры для будущей базы (рис. 1.17).

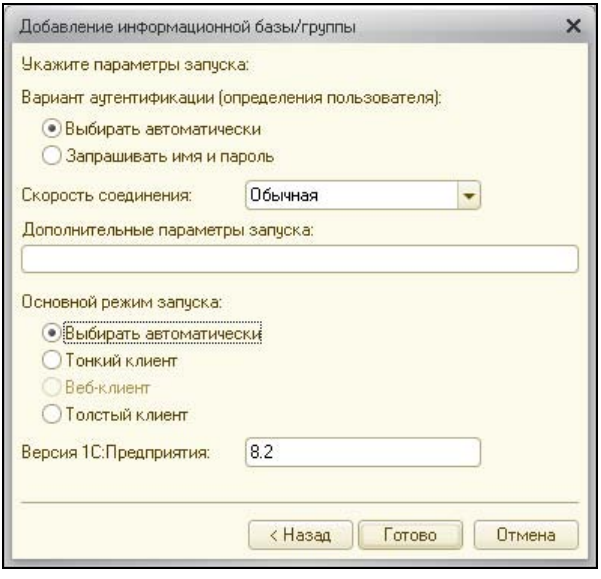



**Рис. 1.17.** Задаем дополнительные параметры для базы данных

19. Здесь мы выбираем вариант аутентификации, скорость соединения, основной режим запуска и задаем дополнительные параметры. Аутентификация, или иными словами — определение пользователя, может проводиться как вводом имени и пароля, так и распознаваться операционной системой. Мы в такие тонкости вдаваться сейчас не будем и оставим все как есть, т. е. **Выбирать автоматически**. Так же и с режимом запуска. Выбор клиента (тонкий, толстый, веб) — это по сути выбор способа работы программного клиента с базой данных. **Толстый клиент** — полный функционал клиентской части; **Тонкий клиент** — облегченный клиент, где все основные расчеты выполняются на стороне сервера; **Веб-клиент** — работа вообще без клиента, здесь клиентом выступает обычный браузер. Режим запуска оставляем так же **Выбирать автоматически.**

    Отдельный интерес представляет поле **Дополнительные параметры запуска**. Например, если прописать в ней **/Execute D:\obrabotka.epf**, то при запуске системы "1С:Предприятие" внешняя обработка obrabotka.epf, находящаяся в корневом каталоге диска D:, будет выполнена автоматически (разумеется, если она существует, в противном случае будет выведено сообщение, что файл не найден).

    Подробнее о дополнительных параметрах запуска мы поговорим в *главе 5*. Пока же оставляем все, как на рис. 1.17, и нажимаем кнопку **Готово**, после чего попадаем в пустое окно Конфигуратора. Откроем нашу конфигурацию через пункт меню **Конфигурация | Открыть конфигурацию**. Появится дерево конфигурации (рис. 1.18).

20. Поскольку на шаге 16 (см. рис. 1.14) мы выбирали вариант установки пустой конфигурации, то и получили пустую конфигурацию. На рис. 1.18 видно, что элементы дерева конфигурации не имеют вложений (слева нет "плюсиков", которыми открывается элемент дерева). В работающей конфигурации есть справочники, документы, отчеты. Все это мы можем увидеть, щелкнув на "плюсике" напротив соответствующей группы данных в дереве. В пустой конфигурации это все предстоит разрабатывать "с нуля" (если стоит такая задача) или загрузить конфигурацию, с которой мы будем работать в дальнейшем.

21. Для того чтобы загрузить конфигурацию из файла, мы должны выбрать пункт меню **Конфигурация | Загрузить конфигурацию из файла**. При этом наша текущая конфигурация (пусть пустая) должна быть уже открыта (см. рис. 1.18), иначе пункт загрузки конфигурации будет неактивен. При выборе загрузки конфигурации нам предложат указать путь к файлу загружаемой конфигурации. Выбираем, как показано на рис. 1.19.

22. Разумеется, в вашем случае местонахождение конфигурации и ее название могут быть иным. Расширение файла конфигурации — cf. Стандартное имя — как показано на рис 1.19 — 1Cv8.cf. Выбираем файл и нажимаем кнопку **Открыть**. Некоторое время происходит загрузка конфигурации, после чего нам предлагают обновить конфигурацию базы данных (рис. 1.20).

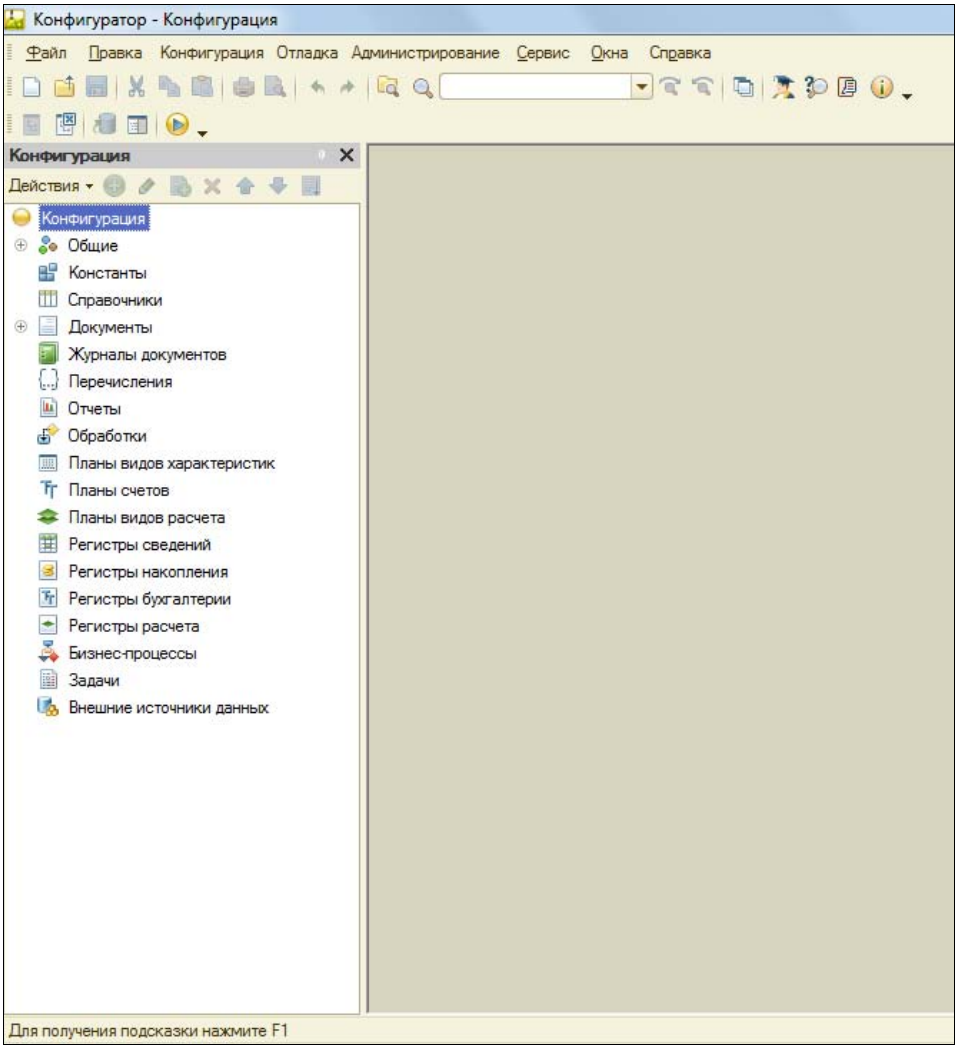



**Рис. 1.18.** Дерево пустой конфигурации

23. Что это значит? В системе "1С:Предприятие 7.7", когда пользователи работали с базой данных, программист не мог вносить правки в конфигурацию, надо было или работать на копии, а потом объединять конфигурации, или выгонять всех из базы на время обновления конфигурации. В отличие от "1С:Предприятие 7.7" в системе "1С:Предприятие 8.2" в информационной базе есть не одна, а две конфигурации: основная конфигурация и конфигурация базы данных. *Основная конфигурация* — аналог конфигурации "1С:Предприятие 7.7", именно ее мы открываем и правим через пункт меню **Конфигурация | Открыть конфигурацию**. *Конфигурация базы данных* — это та конфигурация, с которой работают пользователи. То есть программист может свободно править основную конфигурацию, не тревожа при этом пользователей, работающих в "своей" конфигурации, и обновить конфигурацию базы данных тогда,

когда пользователи завершат работу. Конфигурация базы данных доступна в пункте меню **Конфигурация | Конфигурация базы данных | Открыть конфигурацию БД**, но при этом — только для чтения. Правки производятся путем обновления из основной конфигурации. В нашем случае мы только что обновили основную конфигурацию, и программа запрашивает у нас обновление конфигурации базы данных. Нажимаем кнопку **Да**.

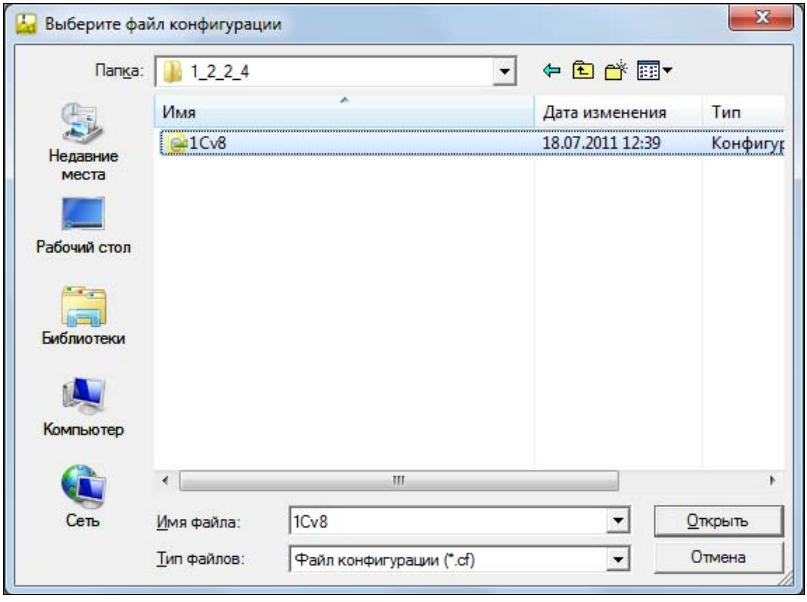



**Рис. 1.19.** Выбор файла конфигурации для загрузки

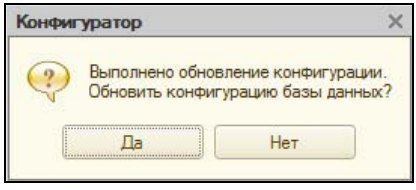



**Рис. 1.20.** Конфигурация загружена, надо обновить конфигурацию базы данных

24. После этого будет выведен список изменений в структуре конфигурации (рис. 1.21).
25. Принимаем изменения кнопкой **Принять**. Теперь если мы откроем конфигурацию, то увидим, что дерево конфигурации заполнено объектами конфигурации (рис. 1.22).

Загрузка конфигурации завершена, и с ней можно работать.

Бывает, что нам нужно обновить рабочую конфигурацию с помощью файла обновления. К примеру, работала фирма какое-то время на типовой конфигурации, вышел новый релиз этой конфигурации. Вы хотите свою рабочую конфигурацию

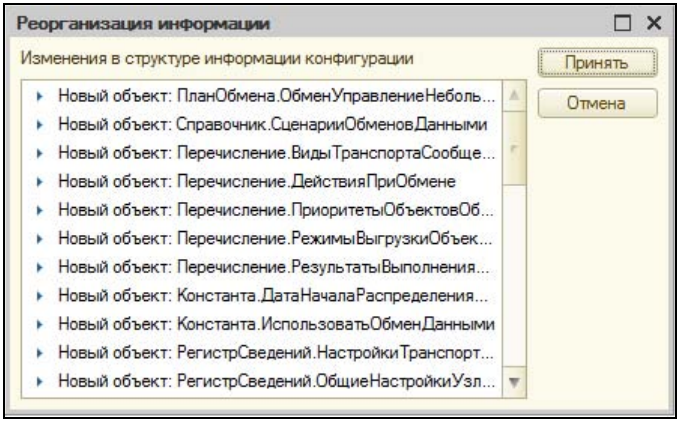



**Рис. 1.21.** Список изменений в базе данных

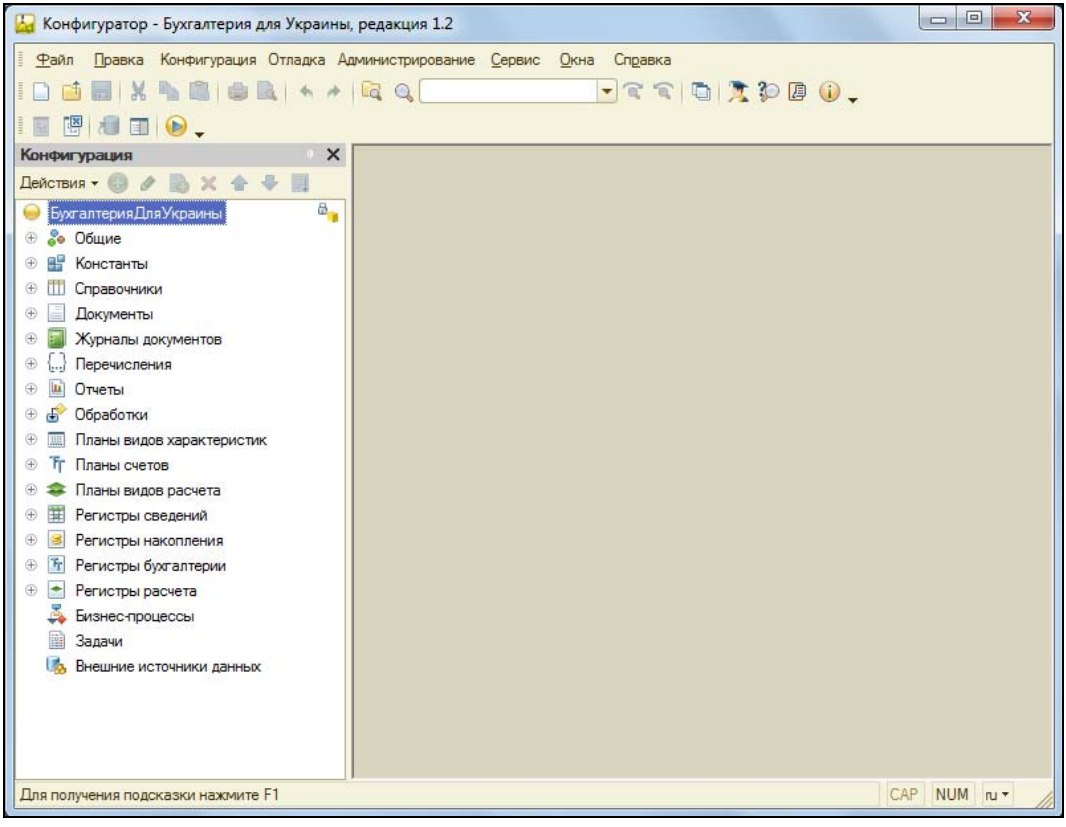



**Рис. 1.22.** Дерево конфигурации после обновления

обновить. И вот тут возникает нюанс: в каком именно виде представлено это обновление? Это может быть *cf-файл конфигурации*, как тот, который мы выбирали на рис. 1.19. Или это может быть *cfu-файл обновления конфигурации*.

В чем между ними разница? Файл с расширением cf — это полноценная конфигурация, которую можно развернуть "с нуля", т. е. с пустой базой данных, а можно объединить этот файл с файлом конфигурации рабочей базы данных (как это делается, мы рассмотрим далее). Файл с расширением cfu — это файл обновления конфигурации, который сам по себе конфигурацией быть не может. Мало того, если обновление не подходит к обновляемой конфигурации (не та версия, например), то обновить конфигурацию не удастся.

Давайте рассмотрим, как обновлять базу данных обоими способами: и собственно обновлением конфигурации, и объединением файлов конфигураций.

# Обновление конфигурации из cfu-файла

1. В режиме Конфигуратора откроем пункт меню **Конфигурация | Поддержка | Обновить конфигурацию**. Запустится мастер обновления конфигурации (рис. 1.23).

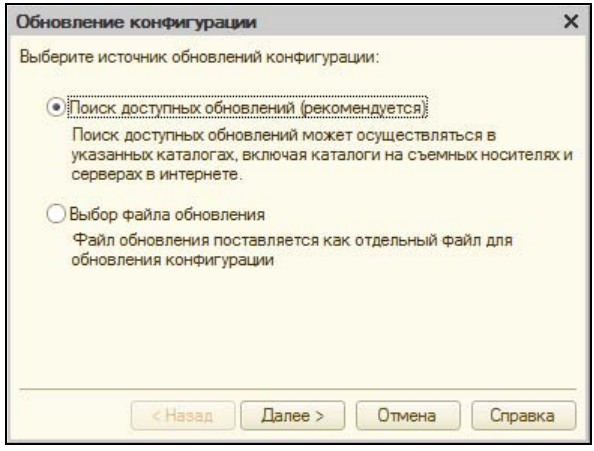



**Рис. 1.23.** Обновление конфигурации

2. Выберем переключатель, позволяющий указать местонахождение файла обновления либо начать поиск этого файла, если его точное местонахождение мы не знаем. Если точное местонахождение файла неизвестно, тогда оставим все так, как на рис. 1.23, и нажмем кнопку **Далее**.
3. Нам будет предложено указать место поиска файла обновления (рис. 1.24).

   По умолчанию поиск осуществляется в каталогах шаблонов и обновлений, а также на съемных носителях (в нашем примере — это дискета в дисководе, которого у меня нет и который не выбран, каталог обновления на CD/DVD-ROM, а также USB-диск, иначе говоря — флешка (u:\)). Вы можете отредактировать

путь к каталогам, в которых будет осуществляться поиск, убрать ненужные или, наоборот, добавить новые кнопкой . В этом случае вам будет предложено прописать путь к файлу обновления, будь то локальный каталог или интернет-страница.

4. Если же мы точно знаем, где находится файл обновления, то на шаге 2 (см. рис. 1.23) установим переключатель **Выбор файла обновления** и нажмем кнопку **Далее**. Откроется окно выбора пути к файлу (рис. 1.25).

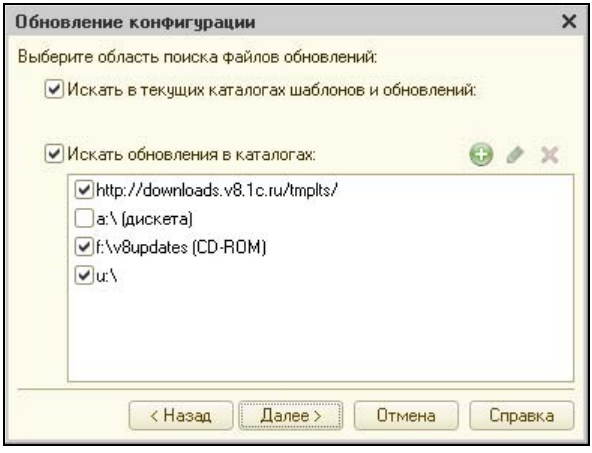



**Рис. 1.24.** Обновление конфигурации

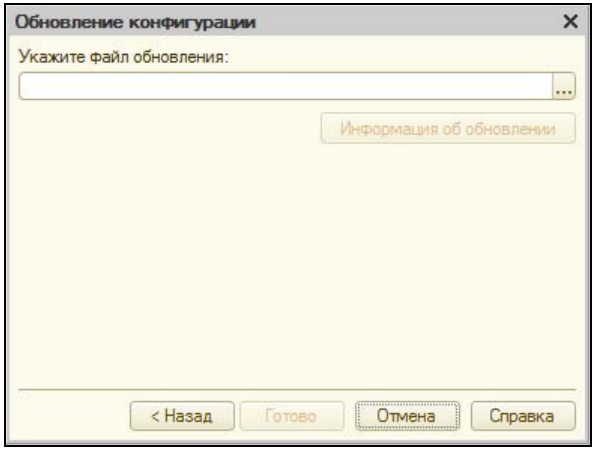



**Рис. 1.25.** Выбор файла обновления конфигурации

5. Выберем файл обновления. Окно мастера с выбранным файлом обновления может выглядеть, к примеру, так, как показано на рис. 1.26.
6. Нажмем кнопку **Готово**. Нам предложат ознакомиться с описанием и порядком обновления конфигурации (рис. 1.27).

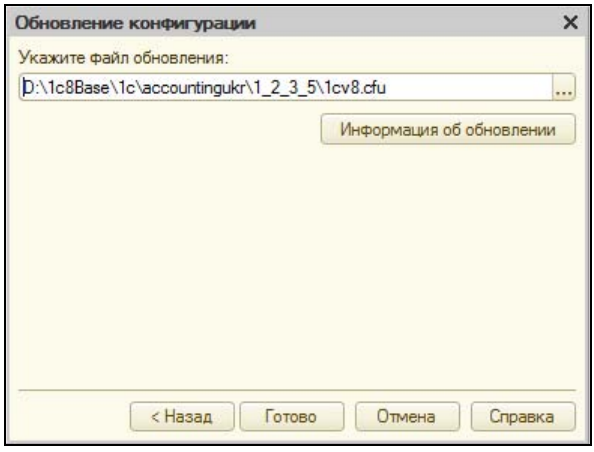



**Рис. 1.26.** Файл обновления конфигурации выбран

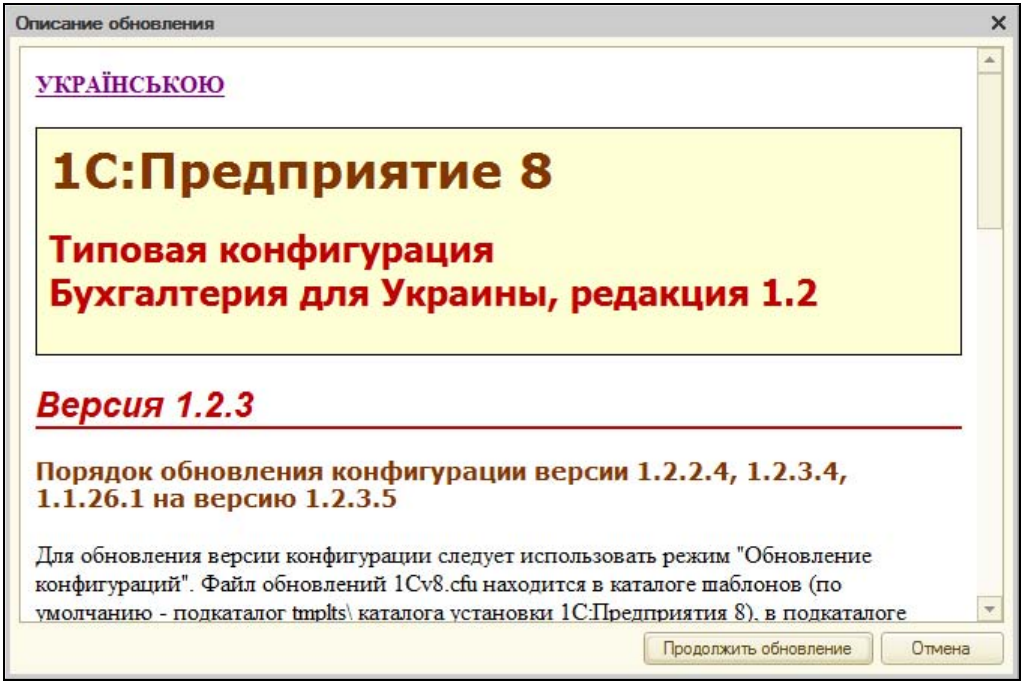



**Рис. 1.27.** Описание обновления

7. Прочитаем описание обновления и нажмем кнопку **Продолжить обновление**. Подготовка к установке обновления конфигурации будет завершена, и мы увидим сводку по произведенному обновлению: какая версия конфигурации была у нас установлена и до какой версии мы ее обновили (рис. 1.28).
8. Нажмем кнопку **OK**, после чего будет произведено обновление конфигурации и конфигурации базы данных (см. рис. 1.20 и 1.21).

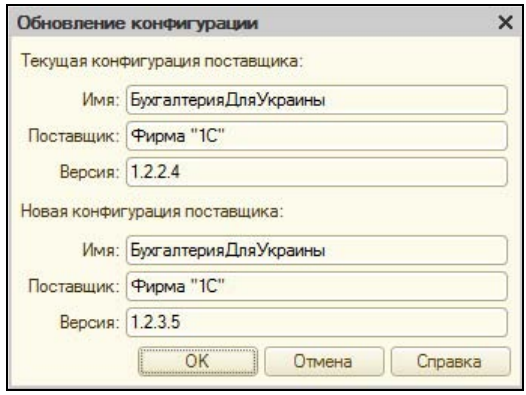



**Рис. 1.28.** Итоги обновления

Теперь давайте рассмотрим порядок обновления конфигурации, если у нас есть не файл обновления, а просто файл конфигурации.

# Обновление конфигурации путем объединения cf-файлов

Файл с расширением cf, в отличие от cfu-файла обновления — полноценный файл конфигурации "1С". Если обновления конфигурации к вам попали в виде такого файла, то способ обновления, рассмотренный в предыдущем разделе, не годится. Обновлять рабочую конфигурацию в этом случае мы будем не путем обновления, а путем сравнения и объединения конфигураций.

Рассмотрим, как это делается.

1. В режиме Конфигуратора открываем пункт меню **Конфигурация | Сравнить, объединить с конфигурацией из файла**. При этом конфигурация в окне Конфигуратора должна быть открыта, в противном случае данный пункт меню будет неактивным до тех пор, пока конфигурация не будет открыта через пункт меню **Конфигурация | Открыть конфигурацию**. Если конфигурации, которые мы объединяем, типовые, то мы можем получить сообщение, как на рис. 1.29.
2. Нас предупреждают о том, что конфигурация типовая и файл, с которым мы производим сравнение и объединение, является ее обновлением. В таких случаях рекомендуется обновлять конфигурацию штатными средствами обновления (в предыдущем разделе мы рассматривали такое обновление). Если же мы все-таки хотим продолжать объединение вручную, нажмем кнопку **Да**. В случае если файл, с которым мы объединяем конфигурацию, не является ее обновлением, окно, показанное на рис. 1.29, открываться не будет.
3. Окно сравнения и объединения конфигурации выглядит так, как показано на рис. 1.30.
4. В этом окне мы видим два дерева конфигурации. То, что слева, принадлежит нашей рабочей конфигурации, то, что справа — файлу конфигурации, с которым

мы производим объединение. При щелчке на плюсике возле любого из объектов объект будет развернут, также будет развернут такой же объект в сравниваемой конфигурации. Обратите внимание на надпись "Запрещено редактирование объекта конфигурации". Это характерно для типовых конфигураций, которые находятся на поддержке у производителя. В чем выражается такая поддержка и как ее, при необходимости, отключить, мы рассмотрим в *разд. "Редактирование объектов конфигурации, поддержка и настройка поддержки" далее в этой главе*.

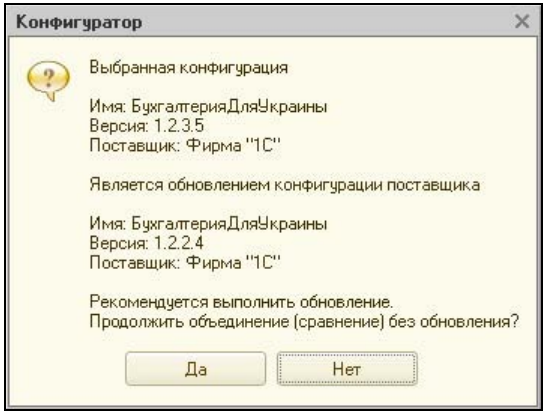



**Рис. 1.29.** Так бывает, если сравнение и объединение производится на типовых конфигурациях

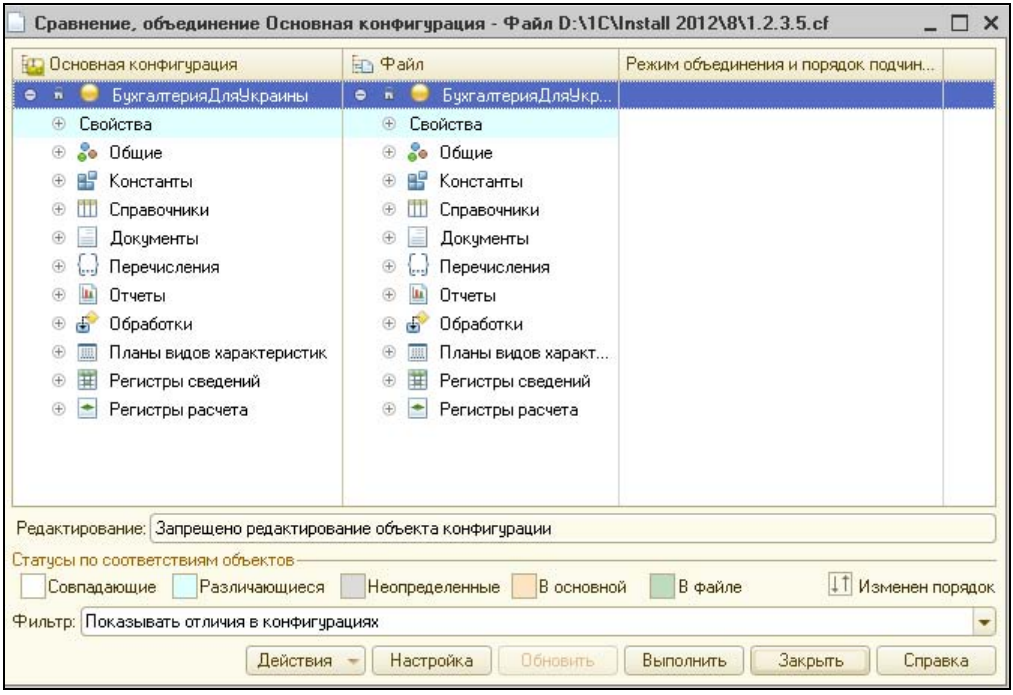



**Рис. 1.30.** Окно сравнения и объединения конфигураций

5. Если развернуть какой-либо из объектов сравниваемых конфигураций, то мы можем увидеть различия, они выделяются в дереве конфигурации цветом. Так, на рис. 1.31 видно, что в конфигурации из файла, по сравнению с рабочей конфигурацией, добавилось несколько новых регистров сведений.

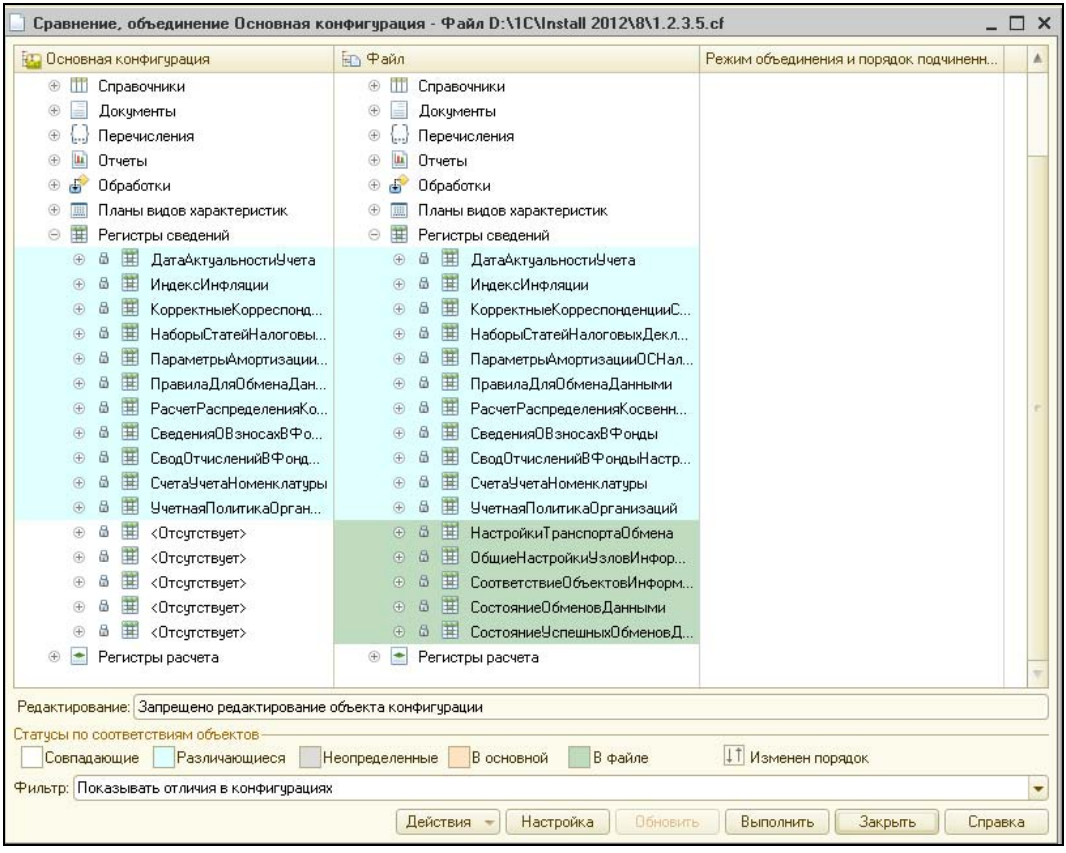



**Рис. 1.31.** Окно сравнения и объединения конфигураций. Различия в дереве конфигурации

6. Пока мы лишь видим различия в конфигурациях, но объединить конфигурации не можем, поскольку возможность редактирования нам закрыта. Поддержку и редактирование объектов конфигурации подробнее мы рассмотрим далее, сейчас же просто включим возможность редактирования конфигурации. Делается это через пункт меню **Конфигурация | Поддержка | Настройка поддержки**. Откроется окно настройки поддержки (рис. 1.32).
7. В этом окне мы должны нажать кнопку **Включить возможность изменения**. Нас предупредят, что это приведет к запрету обновления конфигурации в полностью автоматическом режиме. Но поскольку мы и так собрались конфигурации объединять в ручном режиме (а особенно, если впоследствии будем заниматься изменениями и доработками), то в окне предупреждения нажмем кнопку **Да**. После этого мы можем выбрать, как именно теперь будет работать поддержка конфигурации (рис. 1.33).

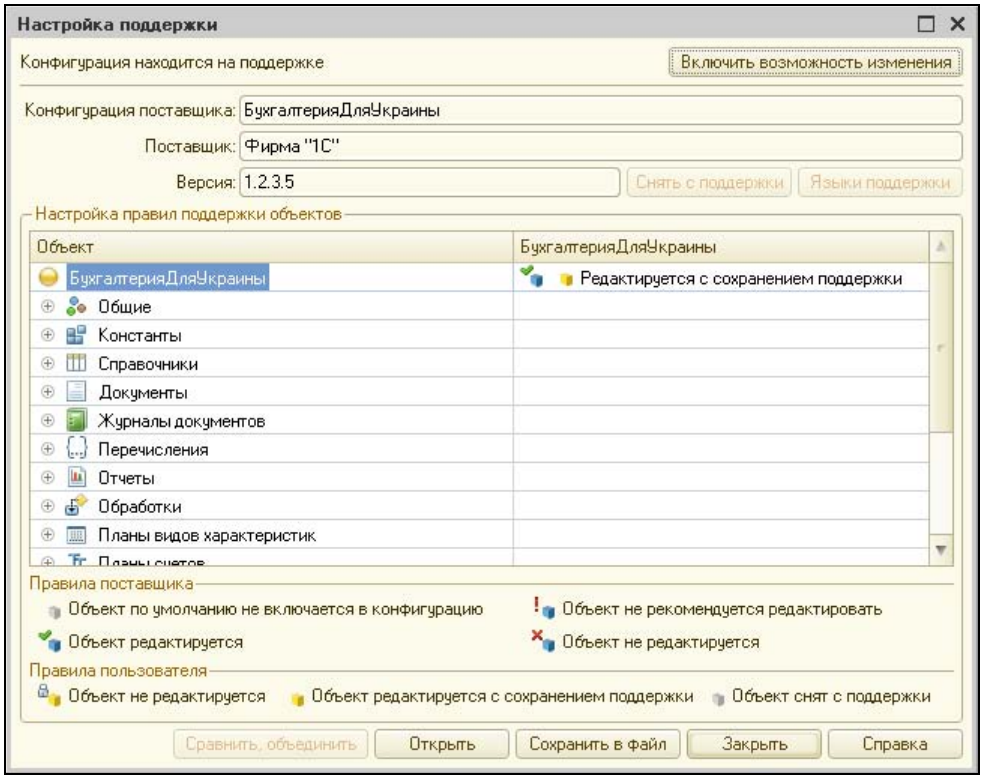



**Рис. 1.32.** Окно сравнения и объединения конфигураций. Различия в дереве конфигурации

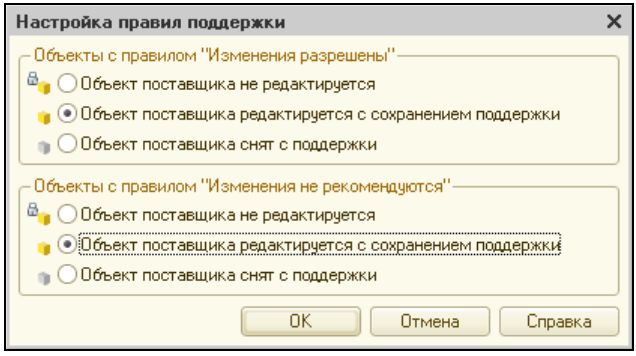



**Рис. 1.33.** Настройка правил поддержки

8. Подробнее правила поддержки мы рассмотрим далее, а пока просто выставим опции такими, как показано на рис. 1.33, и нажмем кнопку **OK**. Этим мы разрешаем редактирование объектов конфигурации, не отказываясь от их поддержки. Если теперь мы откроем пункт меню **Конфигурация | Сравнить, объединить с конфигурацией из файла**, как мы делали это на этапе 1, то увидим, что окно сравнения и объединения конфигураций изменилось. Сравните рис. 1.31 и 1.34.

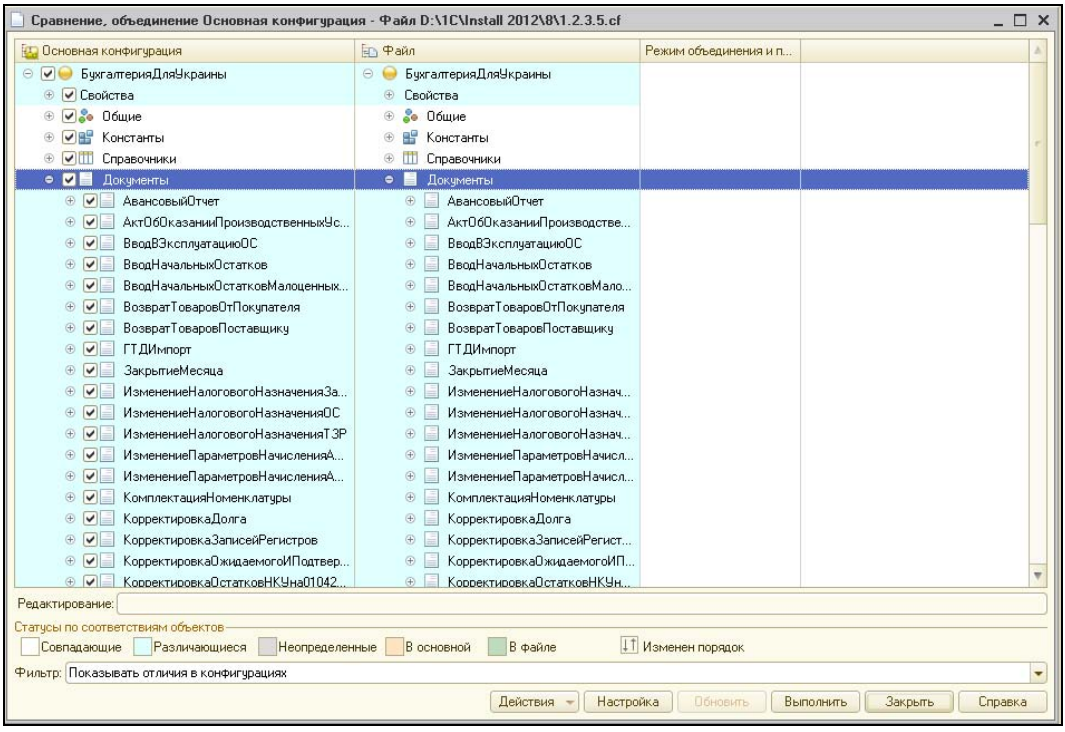



**Рис. 1.34.** Окно сравнения и объединения конфигураций с возможностью редактирования

9. Видно, что по сравнению с рис. 1.31 на рис. 1.34 слева от каждого объекта появилось поле с флажком. Когда флажок установлен, это означает, что данный объект будет обновляться при объединении конфигураций. Таким образом, мы можем гибко настраивать обновление только тех объектов, которые нам нужны. Предположим, программист на отдельной копии конфигурации произвел по требованию заказчика доработки в справочниках номенклатуры и контрагентов. В этом случае, чтобы внести обновления в рабочую конфигурацию, ему в режим сравнения и объединения конфигураций понадобится сбросить флажки напротив всех объектов и установить только напротив справочников номенклатуры и контрагентов. Установка или сброс флажка на вышестоящем иерархически объекте разрешает или запрещает обновление также для объектов, ему подчиненных. Например, если мы указали, что должен обновляться справочник номенклатуры, то будут обновлены также все относящиеся к нему формы, модули и печатные формы. Если мы выбрали вообще всю ветку **Справочники**, то наш выбор будет унаследован всеми справочниками, пока мы прямо не выберем иное (т. е. установим или сбросим флажок для конкретных справочников или элементов справочника). Для каждого из сравниваемых объектов мы можем посмотреть, в чем же они различаются в рабочей конфигурации и в той, с которой мы ее объединяем. Для этого щелкнем правой кнопкой мыши на объекте и выберем пункт **Отчет о сравнении объектов**. Для объектов, у которых есть программные модули, например модули форм справочника или документа, в контекстном

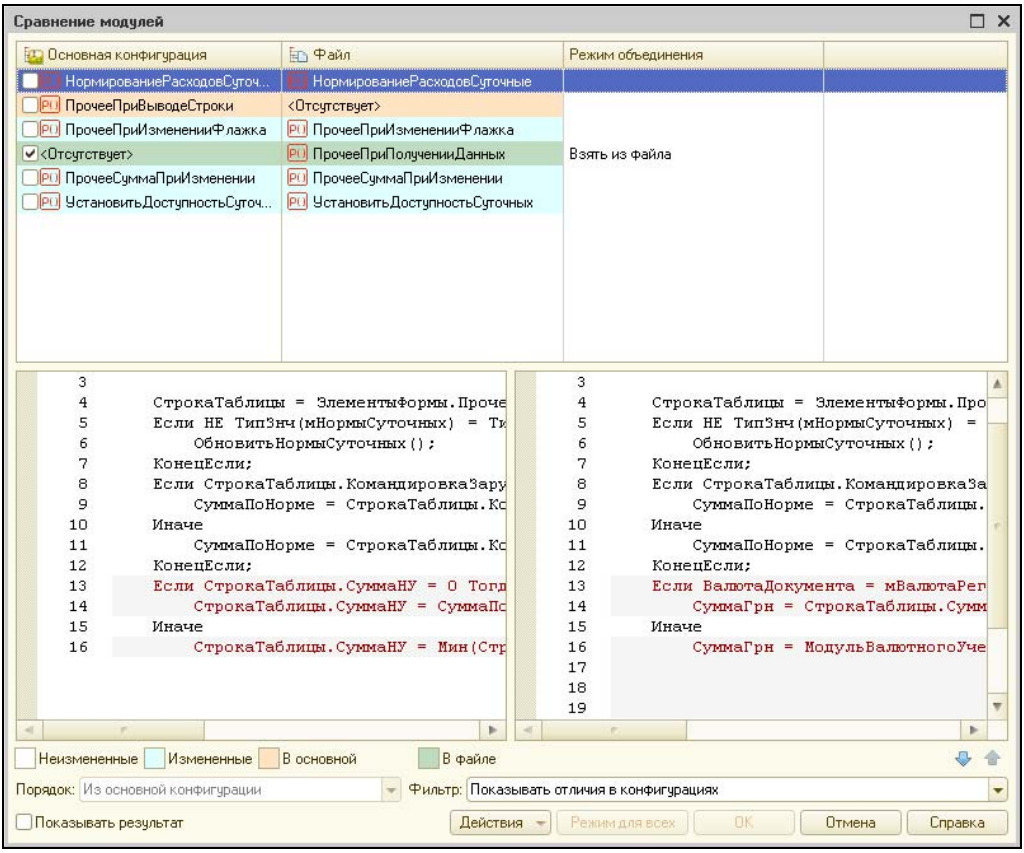



**Рис. 1.35.** Сравнение модулей

меню также появится пункт **Показать различия в модулях**. Типичный пример окна сравнения модулей приведен на рис. 1.35.

10. В приведенном примере мы видим окно программного кода объекта для рабочей конфигурации и той, которую мы с ней объединяем. Отличия выделены серым фоном.

11. После того как мы настроим все параметры обновления, нажмем кнопку **Выполнить** в окне сравнения и объединения конфигураций, после чего конфигурации будут объединены.

Теперь, когда мы научились устанавливать "1С:Предприятие 8.2", устанавливать и обновлять конфигурации, давайте рассмотрим интерфейс Конфигуратора — основной рабочей среды разработчика "1С".

# Конфигуратор

Давайте запустим "1С:Предприятие" в режиме Конфигуратора и ознакомимся с его интерфейсом подробнее. Выбор режима Конфигуратора производится в окне выбора списка информационных баз (см. рис. 1.10) одноименной кнопкой.

Рабочее окно Конфигуратора выглядит так, как показано на рис. 1.36.

Основные элементы окна Конфигуратора:

- *дерево конфигурации* — древовидная структура всех элементов конфигурации: константы, справочники, документы, отчеты и подчиненные им формы, программные модули и печатные формы;
- *меню* — сгруппированные по смыслу команды в верхней части окна Конфигуратора;
- *панели инструментов* — панели с командами-кнопками, расположенные ниже меню, дублирующие наиболее часто используемые команды меню и предназначенные для быстрого доступа к ним;
- *рабочий стол программы* — собственно говоря, основное рабочее поле. Здесь будут располагаться открытые нами формы и программные модули, здесь мы будем заниматься визуальным конструированием и писать программный код;
- *строка состояния* — вспомогательная область с текущей информацией, расположенная в нижней части окна Конфигуратора.

Давайте разберем перечисленные элементы окна Конфигуратора более подробно.

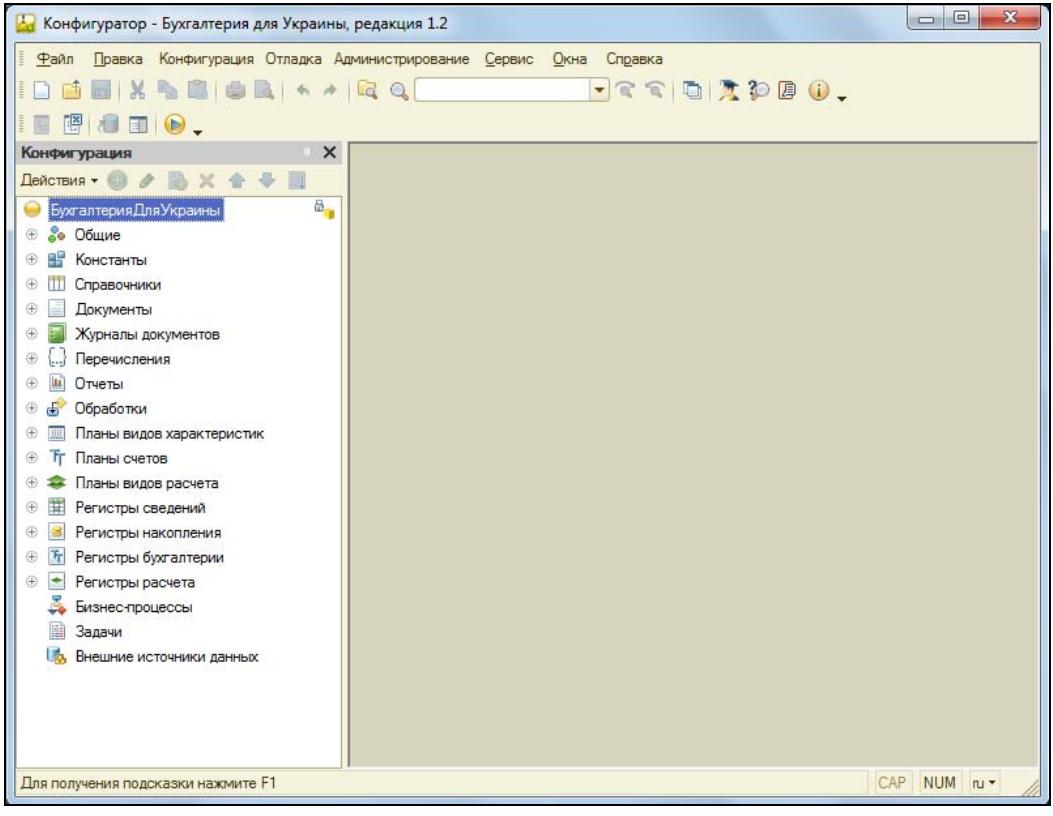



**Рис. 1.36.** Окно Конфигуратора

## Дерево конфигурации

Как уже говорилось ранее, дерево конфигурации открывается через пункт меню **Конфигурация | Открыть конфигурацию.**

Какие же элементы представлены в дереве конфигурации и каково их назначение?

В верхней части дерева — непосредственно сама конфигурация (в выбранном примере — Бухгалтерия для Украины). Ниже по иерархии располагаются элементы, входящие в конфигурацию. Такие элементы называются *метаданными*. Принцип работы с деревом метаданных прост — знак с плюсом означает группу объектов, которую можно раскрыть. Чтобы раскрыть полностью все дерево, можно нажать клавишу <*> на цифровой клавиатуре. Давайте рассмотрим назначение каждого типа метаданных.

- **Свойства конфигурации.** Свойства элемента в самой верхней части дерева метаданных (в нашем случае — "Бухгалтерии для Украины"). Здесь хранятся некоторые общие свойства конфигурации. Для того чтобы посмотреть их или изменить, нужно щелкнуть на объекте правой кнопкой мыши и выбрать в контекстном меню пункт **Свойства** или просто сделать двойной щелчок левой кнопкой на объекте. Впрочем, этот способ просмотра и редактирования свойств используется для всех объектов дерева метаданных. Окно свойств открывается в правой части окна Конфигуратора. Например, окно свойств конфигурации может выглядеть так, как показано на рис. 1.37.

  Содержимое окна свойств для каждого объекта соответственно изменяется, показывая свойства конкретно выбранного объекта. Для конфигурации будут одни свойства, для справочника свойства — уже иные, для поля ввода свойства также будут свои, только располагаться они будут в том же окне свойств.

  Для конфигурации в свойствах задается ее имя и релиз, основной язык работы, способ запуска (обычное приложение или управляемое, проще говоря, работающее через веб-формы), пути к хранилищам настроек и формам отчетов по умолчанию. Здесь же можно внести сведения об авторе, задать заставку, появляющуюся при запуске конфигурации, указать путь к каталогу обновлений и многое другое.

- **Общие.** В этом разделе дерева конфигурации хранятся объекты, так или иначе относящиеся ко всем объектам конфигурации. Так, если в подразделе **Реквизиты** задать общий реквизит, то этот реквизит может быть использован в любом из объектов конфигурации (например, при создании общего реквизита **Примечание** мы можем использовать его в любом из документов конфигурации). К созданным в разделе **Общие** общим модулям можно обратиться из любого модуля разрабатываемой конфигурации (то же касается общих форм и общих макетов — печатных форм), в подразделе **Роли** мы задаем роли всех пользователей и их права, в подразделе **Интерфейсы** — интерфейсы всех пользователей.

- **Константы** предназначены для хранения постоянной и условно-постоянной информации, которая в процессе работы не изменяется или изменяется редко. Главная особенность констант — возможность их многократного использова-

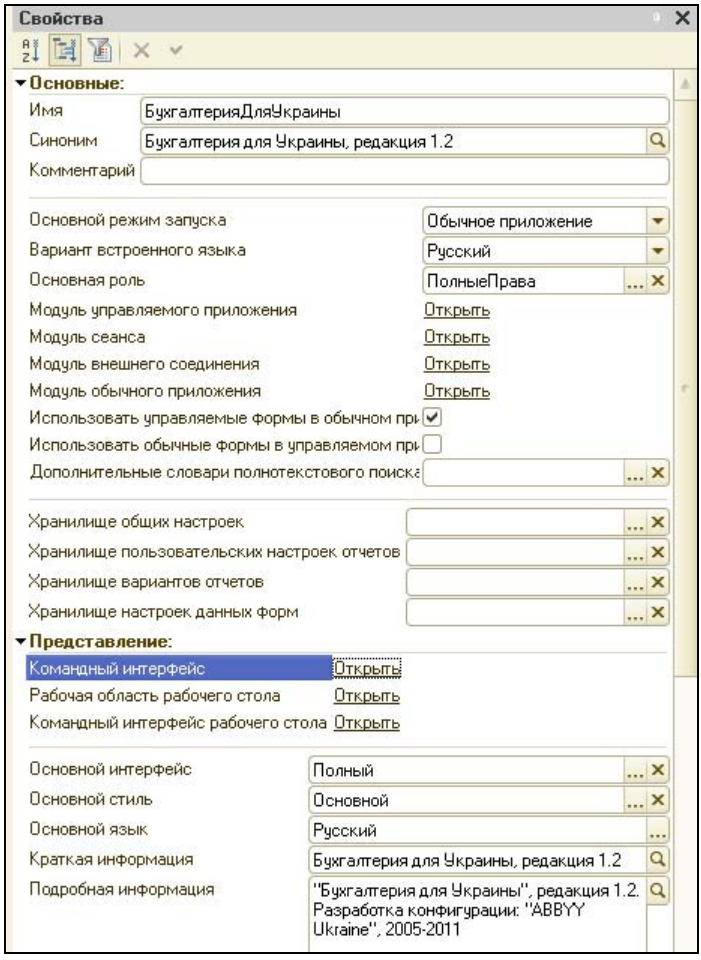



**Рис. 1.37.** Окно свойств конфигурации "Бухгалтерия для Украины"

ния. Приведу пример: пусть на предприятии работает сотрудник, ответственный за составление документов и их подписание. Обычно рядом с местом для подписи на таких документах должна указываться также фамилия этого человека. Фамилию можно прописать в печатной форме документа, а можно записать в константу. Если записывать фамилию в печатную форму, то при смене лица, ответственного за выписку документов, придется править все печатные формы (а их может быть много). Гораздо проще Ф.И.О. ответственного лица записать в константу, а уже ссылку на нее поместить в печатную форму каждого документа. Если нужно изменить фамилию, она правится непосредственно в константе, и на печатных формах будет меняться автоматически, ведь там находится не само значение, а только ссылка на него.

- **Справочники** — это средство для работы со списками однородных элементов данных. При помощи справочников организуется ввод стандартной информации в документы, ее просмотр и изменение. Обычно справочниками являются

списки товаров, организаций, валют, сотрудников и др. Основные поля, по которым уникально характеризуется любая запись в справочнике, — это код и наименование.

- **Документы** — основное средство совершения хозяйственных операций в системе "1С:Предприятие". С их помощью осуществляются все движения товарно-денежных потоков на предприятии, осуществляется ввод первичных данных в систему, их просмотр и корректировка. Приход товаров на склад, перемещение между складами, отгрузка или продажа через кассовый аппарат, поступление денег на расчетный счет или в кассу, списание неликвидов, т. е. вся эта информация вводится в систему посредством документов соответствующего типа (приходных и расходных накладных, перемещений, списаний, банковских выписок, кассовых ордеров и т. п.). Основные поля, по которым уникально характеризуется любой документ, — это его номер и дата.
- **Журналы документов** являются средством для отображения списка документов (по аналогии с реестром). Работая с журналом, пользователь может вводить документы, просматривать, редактировать и удалять. Журналы позволяют сортировать и группировать список документов, просматривать выбранный документ, править его либо удалить. Сами по себе журналы никакой информации не хранят, они лишь отображают списки документов в удобном виде.
- **Перечисления** — это специальные типы данных. Они не представляют собой самостоятельные объекты, как справочники или документы, а используются в комплексе с прочими типами данных: числовыми, текстовыми и т. п. Например, в крупном оптовом магазине формируются накладные к отправке заказчикам. Перед погрузкой товаров по каждой накладной, товар проверяет и пересчитывает контролер или охранник: проверил и сделал в накладной пометку "Проверено". Какое может быть состояние проверки? Либо проверено, либо нет. Если бы нам для чего-либо потребовалось указывать в накладной, прошла она проверку или нет — мы могли бы добавить в документ реквизит **Проверено**, принимающий значения либо "Да", либо "Нет". Вот это и есть перечисление — такой тип данных, который может принимать только одно из заранее определенных значений. В данном случае или "Да", или "Нет".
- **Отчеты** предназначены для выборки определенных пользователем данных за указанный период. Сами по себе отчеты не являются хранимыми в базе данных объектами, содержащими информацию, наподобие справочников или документов. Это всего лишь выборки из подобных объектов, создаваемые динамически. Например, вам нужно отобрать остатки в ценах себестоимости по одному из складов за последний месяц. При запуске соответствующего отчета он выбирает из множества записей в базе данных те, которые соответствуют условиям отбора, и выдает на экран в форме, заданной программистом при проектировании отчета в Конфигураторе.
- **Обработки** — это программный код, предназначенный выполнять заданные программистом действия. Метаданные этого вида схожи с отчетами, однако в отличие от последних могут не только делать выборку данных, но и производить их изменение, в том числе групповые действия над большим количеством

данных. Например, чтобы внести в справочник товаров розничные цены на 20% выше текущих, можно написать обработку, перебирающую все записи справочника и перемножающие соответствующие им розничные цены на 1,2. Обработки бывают *внутренними* и *внешними*. Внутренние являются элементами дерева конфигурации, внешние запускаются из внешних файлов с расширением epf через меню **Файл | Открыть**. Внешние обработки не являются частью конфигурации, а представляют собой внешние программные модули. Понятия "отчет" и "обработка" очень часто пересекаются, внешние отчеты в epf-файлах являются ничем иным, как внешними обработками.

- **Планы видов характеристик** предназначены для хранения информации о характеристиках различных объектов. Например, характеристиками товара могут служить цвет, размер, запах, вкус и т. д. По своей структуре схожи со справочниками.
- **Планы счетов** — совокупность синтетических счетов, предназначенных для хранения и группировки информации о хозяйственной деятельности предприятия. Счета имеют иерархическую структуру и могут разбиваться на неограниченное количество субсчетов (вложенных счетов). Анализ остатков на таких счетах и движений между счетами позволяет получить информацию о деятельности предприятия в денежном выражении и текущем финансовом состоянии.
- **Планы видов расчета** используются в механизме сложных периодических расчетов и служат для описания видов расчета и их взаимного влияния друг на друга.
- **Регистры сведений** — в упрощенном представлении это таблицы, которые позволяют хранить произвольные данные в разрезе нескольких измерений. Информация в регистре сведений хранится в виде записей, каждая из которых содержит значения измерений и соответствующие им значения ресурсов. *Измерения* регистра описывают разрезы, в которых хранится информация, а *ресурсы* регистра непосредственно содержат хранимую информацию.
- **Регистры накопления** — многомерные таблицы, составляющие основу механизма учета движения средств (товаров, денежных средств и т. д.), который позволяет автоматизировать такие направления, как складской учет, взаиморасчеты, планирование. Регистр накопления образует многомерную систему измерений и позволяет "накапливать" числовые данные в разрезе нескольких измерений. Например, в подобных регистрах можно накапливать информацию об остатках товаров в разрезе номенклатуры или склада, или информацию о продажах в разрезе номенклатуры или точек продажи. *Измерения* регистра описывают разрезы, в которых хранится информация, а *ресурсы* регистра непосредственно содержат хранимую информацию.
- **Регистры бухгалтерии** — это многомерные таблицы, использующиеся в бухгалтерском учете и позволяющие вести учет по нескольким планам счетов, а также количественный, суммовой и валютный учет по отдельным разрезам аналитики. По принципу работы схожи с регистрами накопления. *Измерения* регистра описывают разрезы, в которых хранится информация, а *ресурсы* регистра непосредственно содержат хранимую информацию.

- **Регистры расчета** — многомерные таблицы, которые служат для хранения записей о тех или иных видах расчета, а также для хранения промежуточных данных и самих результатов выполненных расчетов. *Измерения* регистра описывают разрезы, в которых хранится информация, а *ресурсы* регистра непосредственно содержат хранимую информацию.
- **Бизнес-процессы** — вид метаданных, предназначенный для описания схем бизнес-процессов.
- **Задачи** предназначены для учета заданий и описывают способ их распределения по исполнителям, с учетом организационной структуры предприятия. Напрямую взаимосвязаны с механизмом безнес-процессов.
- **Внешние источники данных** позволяют работать с внешними базами данных, не основанными на "1С:Предприятие", такими как MS SQL Server или Oracle Database.

Разумеется, программист в своей деятельности далеко не всегда может использовать какие-либо из приведенных объектов метаданных. Например, в работе магазина по продаже продуктов питания вовсе не обязательно задействовать механизм бизнес-процессов и задач. Но если такая потребность возникнет, возможность его использования есть.

## Меню

Меню расположено сразу под заголовком программы и содержит команды, необходимые для работы в программе. Команды объединены в пункты, названия которых и составляют строку главного меню программы. Выбор мышью какого-либо из пунктов главного меню открывает список команд, входящих в данное меню.

Рассмотрим подробнее содержимое меню.

В меню **Файл** находятся стандартные файловые операции **Новый**, **Открыть**, **Сохранить**, **Печать** и т. п. Есть небольшой нюанс, о который нередко спотыкаются начинающие. Пользователь, имеющий навыки работы в программах, типа MS Word, MS Excel и им подобных, для создания нового документа сразу нажимает на пункт меню **Файл** и пытается создать новый документ. И вот здесь важный момент: команда **Файл | Новый** на самом деле *НЕ создает* чаще всего используемые в "1С:Предприятие" документы. В ней можно создать текст (аналог Блокнота), таблицу (аналог MS Excel), HTML-страничку, картинку (встроенный в "1С:Предприятие" аналог стандартной графической программы Paint), и только. Техника работы с основными документами "1С:Предприятие" совсем другая. Так что на деле, большинство пользователей не так уж часто обращается к этому пункту меню. Программисту этот пункт меню необходим чаще. Например, для создания внешних программ-обработок, которые потом могут быть запущены в системе "1С:Предприятие" через пункт меню **Файл | Открыть**. Кроме того, здесь есть очень полезный пункт **Сравнить файлы**. При его выборе откроется окно, в котором нужно указать путь к двум сравниваемым файлам (рис. 1.38).

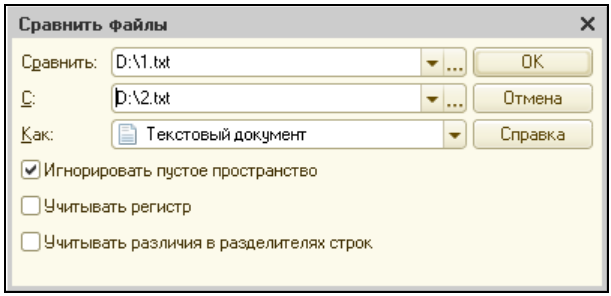



**Рис. 1.38.** Сравнение двух файлов

Дополнительно можно задать игнорирование пустого пространства (для программиста два файла являются идентичными, если в них одинаковый код, но в одном из файлов есть лишняя пустая строка между строками кода) и выбрать правило для регистра букв (для языка 1С регистр значения не имеет, команды `НайтиПоНаименованию` и `найтипонаименованию` будут работать одинаково, поэтому регистр можно не различать).

В раскрывающемся списке **Как** мы можем выбрать, как мы сравниваем два файла. Это может быть не только текст, но и таблица. Для сравнения нажмем кнопку **OK**. Откроется окно сравнения текста двух файлов (рис. 1.39).

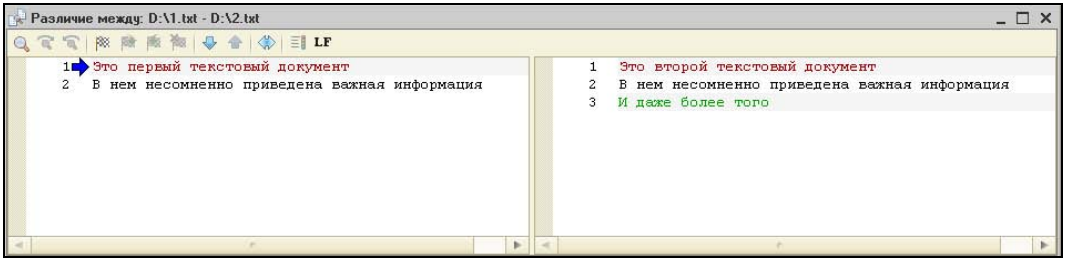



**Рис. 1.39.** Окно сравнения файлов

Как видим, все очень просто. Строки, в которых есть расхождения, выделены серым, удаленные строки напечатаны синим шрифтом, измененные — красным, добавленные — зеленым. Сравнение ведется по правой части окна (что сравниваем) по отношению к левой части окна (с чем сравниваем). Например, в сравнении на рис. 1.39 мы видим, что во втором файле по сравнению с первым добавилась новая строка и имеются расхождения в первой строке.

Полезное практическое применение подобного инструмента — например, сравнение двух внешних обработок, в одной из которых были сделаны доработки. Сравнением файлов эти доработки легко выявляются, тем более что программный код обработки — это по сути тот же самый текст.

Еще в меню **Файл** стоит обратить внимание на пункт **Сохранить копию**. С его помощью можно сохранить копию текущего рабочего контента. Так, если открыта печатная форма (макет) документа, то с помощью данного пункта печатную форму

можно сохранить в файл удобного для дальнейшего просмотра и обработки формата. Например, в формате XLS, XLSX, PDF или HTML.

Если открыт программный код, то пункт **Файл | Сохранить копию** позволит сохранить текст кода в отдельный текстовый файл.

Если же в Конфигураторе вообще нет открытых окон, кроме дерева конфигурации, то пункт **Файл | Сохранить копию** сохранит копию текущей конфигурации в cf-файл конфигурации.

Меню **Правка** содержит стандартные команды работы с буфером обмена, команды **Отменить** и **Вернуть** и команды поиска текста. Отдельно хочу обратить внимание читателя на пункты **Глобальный поиск** и **Глобальная замена**, позволяющие найти и/или заменить нужное слово или выражение в пределах всей конфигурации.

Меню **Конфигурация** содержит команды и вспомогательные подменю для управления конфигурацией. Рассмотрим назначение основных из них.

- **Открыть конфигурацию** — открывает дерево метаданных, аналогично кнопке **Открыть конфигурацию**.
- **Закрыть конфигурацию** — закрывает дерево метаданных, аналогично кнопке **Закрыть конфигурацию**.
- **Сохранить конфигурацию** — сохраняет текущие изменения в конфигурации.
- **Обновить конфигурацию базы данных.** О конфигурации базы данных мы уже говорили ранее. Пока программист ведет разработку в основной конфигурации, пользователи спокойно работают с базой данных. Но когда приходит время внести изменения в то, с чем работают пользователи — вот тут и нужен этот пункт меню. Изменения из основной конфигурации переносятся в конфигурацию базы данных и становятся доступными пользователям.
- **Конфигурация базы данных** — подменю работы с конфигурацией базы данных. В отличие от основной конфигурации, конфигурация базы данных доступна только в режиме чтения и может быть изменена лишь путем обновления из основной конфигурации.
- **Поддержка** — подменю настройки поддержки, с его помощью можно настроить уровень поддержки конфигурации, а также дать доступ на изменение, если элементы дерева конфигурации закрыты от редактирования.
- **Сохранить конфигурацию в файл** — сохраняет конфигурацию в cf-файл. Этот файл является копией рабочей конфигурации, но без данных, хранимых в базе данных. Для того чтобы сохранить не просто файл конфигурации, а всю базу данных вместе с конфигурацией, используется пункт меню **Администрирование | Выгрузить информационную базу**.
- **Загрузить конфигурацию из файла** — в отличие от предыдущего пункта, наоборот загружает конфигурацию из cf-файла.
- **Сравнить и объединить с конфигурацией из файла** — запуск режима сравнения и объединения текущей конфигурации с конфигурацией из файла. Принципы работы режима сравнения и объединения были рассмотрены нами в

*разд. "Обновление конфигурации путем объединения cf-файлов" ранее в этой главе*.

- **Сравнить конфигурации** — режим автоматического сравнения конфигураций. Могут сравниваться основная конфигурация, конфигурация базы данных, конфигурации из cf-файлов.
- **Хранилище конфигурации** — подменю, относящееся к работе с хранилищем конфигурации. *Хранилище конфигурации* — это механизм Конфигуратора, позволяющий нескольким разработчикам одновременно заниматься разработкой одной конфигурации. Предположим, один программист вносит правки в отчет по продажам, другой — правит справочник товаров, третий — дорабатывает документ "Поступление товаров и услуг". В этом случае каждый из этих программистов захватывает из хранилища конкретный объект и работает с ним. Для остальных разработчиков этот объект будет заблокирован, но остальные объекты конфигурации — свободны. По окончании работы разработчики помещают в хранилище измененный объект и разблокируют доступ к нему.
- **Проверка модулей** — запускает механизм поиска ошибок в программных модулях конфигурации.
- **Проверка конфигурации** — более разносторонний способ проверки, включающий в себя не просто поиск ошибок в синтаксисе программного кода, а и поиск некорректных ссылок, неиспользуемых процедур и функций и многое другое.

Меню **Отладка** предназначено для работы с отладчиком программного кода. Работу с отладчиком мы рассмотрим в *главе 6*.

Меню **Администрирование** предоставляет доступ к инструментам контроля за базой данных и функциям администрирования. Рассмотрим содержимое этого меню подробнее.

- **Пользователи** — открывает список пользователей базы данных.
- **Активные пользователи** — открывает список пользователей, активных в настоящий момент (работающих в базе данных).
- **Журнал регистрации** — история работы пользователей за выбранный период. Показано, кто, когда и какие изменения производил. Можно устанавливать фильтры по различным критериям (имя пользователя, событие, вид данных и т. д.).
- **Выгрузить информационную базу** — выгрузка базы данных в dt-файл. Используется для резервного копирования базы данных. Файл конфигурации при этом отдельно выгружать не требуется. Имея dt-файл, вы можете развернуть из него полноценную копию вашей базы данных на момент выгрузки файла.
- **Загрузить информационную базу** — загружает dt-файл базы данных, полностью при этом замещая текущую конфигурацию и базу данных.
- **Публикация на Web-сервере** — этот пункт меню предназначен для организации работы с базой данных через Интернет, через веб-формы.

- **Тестирование и исправление** — тестирует структуру базы данных и сами данные на наличие ошибок и устраняет их. Обычно используется при частичном повреждении информационной базы, наличии "битых" ссылок, т. е. ссылок на несуществующие объекты, для сжатия и реструктуризации таблиц базы данных и пересчета итогов.
- **Настройка журнала регистрации** — здесь можно определить типы информации, которые будут записываться в журнал регистрации.
- **Региональные установки информационной базы** — здесь находятся такие настройки базы данных, как используемый язык, разделители дробной части и разрядов чисел, форматы чисел, даты и времени.
- **Параметры информационной базы** — здесь хранятся дополнительные параметры базы данных, касающиеся защиты информации, такие как минимальная длина пароля пользователя и включение/отключение контроля за сложностью пароля (очень полезно, если вы хотите запретить пользователям вводить пароли вроде "111" или "12345").

Меню **Сервис** выполняет вспомогательные функции. Здесь можно воспользоваться калькулятором, открыть календарь, отсюда же можно запустить "1С:Предприятие" прямо из режима Конфигуратора. Из этого же меню доступны настройки панелей команд и пункт **Параметры**, который позволяет настроить параметры запуска системы "1С:Предприятие", параметры работы с формами, программными модулями, текстом и т. п.

Меню **Окна** служит для управления окнами в системе "1С:Предприятие". Можно изменить расположение открываемых окон (**Каскад**), закрыть текущее окно либо закрыть сразу все открытые окна (**Закрыть все**). При желании отсюда можно закрыть и окно сообщений (хотя быстрее его закрыть стандартно — нажав кнопку с крестиком). При нескольких открытых окнах в этом меню также можно осуществлять переход между ними, выбирая то окно, которое нужно сделать текущим.

В меню **Справка** вы найдете различные справочные материалы, посвященные работе в системе "1С:Предприятие". Также особо следует отметить пункт **О программе**, при выборе которого можно получить сведения об используемой версии "1С:Предприятие", релизе, разработчике конфигурации, рабочем названии и месторасположении базы данных.

## Панели инструментов, или кнопки

Теперь, рассмотрев меню системы "1С:Предприятия", вкратце упомянем о панелях инструментах, или, проще говоря, о кнопках. Вкратце — потому что все возможности, реализуемые кнопками, есть в меню, и мы их уже рассмотрели. Кнопки вынесены для наиболее часто используемых действий: операций с буфером обмена, предварительного просмотра, печати, поиска, перехода в различные режимы работы, работы с пользователями и т. д. Все кнопки снабжены всплывающими подсказками и подходящими по смыслу картинками. Настройка панелей инструментов может производиться не только через меню, но и посредством щелчка правой кнопкой мыши на панели. В появившемся контекстном меню мы можем выбрать панели инструментов, которые будут отображаться (отмечены "птичками"). А при

нажатии пункта **Настройка** мы попадем в окно настройки панелей инструментов (рис. 1.40).

Кнопка **Создать** позволяет создать новую панель инструментов, а с вкладки **Команды** на нужную нам панель инструментов можно перемещать отдельные кнопки.

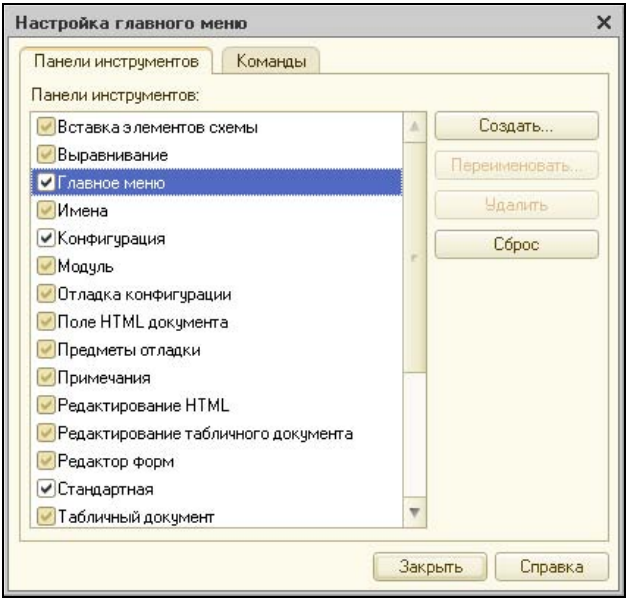



**Рис. 1.40.** Настройка панелей инструментов

# Выгрузка и загрузка данных, копирование базы данных

Чем дольше организация работает со своей базой данных, чем больше информации там накапливается, тем большим ударом была бы частичная или полная потеря этих данных. Сбой на жестком диске, ошибка в конфигурировании или обработке данных (особенно групповой обработке больших массивов данных) могут привести к полной или частичной потере данных, а то и к полной неработоспособности базы данных. Выход — регулярно делать резервные копии, из которых всегда можно восстановить базу данных.

Чтобы выгрузить базу данных (или, попросту говоря, создать ее копию), нужно в меню системы "1С:Предприятие" выполнить пункт **Администрирование | Выгрузить информационную базу** (этот пункт мы с вами уже рассматривали, когда говорили о меню и назначении отдельных команд, туда входящих). Откроется стандартное окно сохранения файла, где мы должны выбрать место, куда будем сохранять dt-файл базы данных, а также задать его имя. Лучше делать резервные копии в специально отведенную для этого папку, имя же можно дать произвольное (разумеется, не меняя расширение файла). По умолчанию файл называется 1Cv8.dt, я же

люблю давать файлам копий имя в виде даты сохранения файла. Например, 250212.dt. Вполне удобно и наглядно.

Чтобы загрузить базу данных из файла, нужно в меню системы "1С:Предприятие" выполнить пункт **Администрирование | Загрузить информационную базу**, после чего в окне выбора файла указать dt-файл, из которого будем загружать. При загрузке конфигурация и данные в базе будут полностью замещены конфигурацией и данными из файла.

Разумеется, и выгрузка, и загрузка базы данных должны производиться тогда, когда пользователи в текущей базе данных не работают, иначе будет выдано предупреждение ("Ошибка исключительной блокировки информационной базы") и выгрузка/загрузка не произойдет.

# Редактирование объектов конфигурации, поддержка и настройка поддержки

В системе "1С:Предприятие 8.2" существуют два режима поддержки конфигураций — полная поддержка конфигурации поставщика и поддержка с возможностью редактирования. В режиме полной поддержки пользователь всегда работает с точной копией конфигурации разработчика. Такой вариант поддержки конфигурации имеет как преимущества, так и недостатки. Главное преимущество — полностью автоматическое обновление конфигурации, недостаток — невозможность изменения и доработки конфигурации (конфигурация доступна в режиме "только чтение"). Полностью автоматическое обновление конфигурации включает в себя две основные особенности:

- *интегрированная проверка версий конфигурации*, т. е. если при обновлении в ручном режиме пользователь может ошибочно загрузить не ту конфигурацию, то в режиме автоматического обновления подобная загрузка "не той" конфигурации невозможна;
- *возможность использовать для обновления cfu-файлы*, которые имеют меньший размер, чем стандартные cf-файлы конфигурации.

Для определения текущего режима поддержки, а также его изменения, следует воспользоваться пунктом меню **Конфигурация | Поддержка | Настройка поддержки**. Открывшееся окно настройки поддержки нам уже знакомо (см. рис. 1.32). В этом окне приведено дерево конфигурации: слева — объекты метаданных, справа — текущее правило поддержки для этого объекта. Для каждого из объектов можно назначить правило поддержки вручную, щелкнув по текущему правилу поддержки правой кнопкой мыши и выбрав в контекстном меню пункт **Установить правило поддержки**. Форму выбора правила поддержки мы рассматривали на рис. 1.33.

Хочу также заметить, что если вы перевели конфигурацию из режима полной поддержки, то обратно в этот режим вы вернуть ее уже не сможете (хотя, если вам нужно заниматься доработками конфигурации, то режим полной поддержки вам все равно не нужен). Если же по какой-то причине вам потребуется удалить из конфигурации какой-либо элемент метаданных, который был в конфигурации изначально, то его сначала придется перевести в режим "Объект поставщика снят с поддержки".

ГЛАВА 2

# Программирование в "1С:Предприятие 8.2"

В этой главе мы подробно ознакомимся с синтаксисом языка программирования 1С, на примерах рассмотрим применение основных языковых конструкций, после чего читатель вполне сможет самостоятельно писать программные модули или дорабатывать уже имеющиеся. Мы не будем рассматривать все команды, типы данных и языковые конструкции языка программирования 1С в системе "1С:Предприятие 8.2", поскольку их очень много, и для их описания пришлось бы написать как минимум еще одну такую же книгу. Мы разберем основные, наиболее применимые команды, с остальными при желании вы всегда сможете ознакомиться в синтаксис-помощнике системы "1С:Предприятие" (режим Конфигуратора, меню **Справка | Синтаксис-помощник**) или в документации, предоставляемой фирмой "1С" вместе со своими программными продуктами. Также, чтобы не запутывать читателя, мы иногда будем опускать некоторые малозначащие параметры в методах различных объектов конфигурации.

В языке программирования 1С все операторы имеют два написания: русское и английское. К примеру, оператор `Новый("")` аналогичен по смыслу и действию оператору `New("")`. Обычно все же пишут код на одном языке (чаще русском), однако не возбраняется (хотя считается плохим стилем программирования) смешивать оба языка в одном модуле. Мы в описании языковых конструкций будем приводить только русский вариант написания.

Каждая языковая конструкция будет описана в следующем формате:

***ЭлементЯзыка*(*Параметр1*, *Параметр2*, . . . , *ПараметрN*) [*КлючевоеСлово*]**

Здесь: *Параметр1*, *Параметр2*, ..., *ПараметрN* — список параметров; *КлючевоеСлово* — дополнительное ключевое слово, которое может присутствовать или отсутствовать в данной языковой конструкции.

Если у элемента языка нет параметров, скобки опускаются.

Квадратные скобки `[ ]` означают, что параметр или ключевое слово, заключенные в них, необязательны и могут как присутствовать, так и отсутствовать.

Если среди элементов необходимо выбрать только один, они будут разделены следующим образом: `Элемент1|Элемент2|Элемент3`.

Кроме того, будут представлены примеры программного кода с использованием данной языковой конструкции с подробными комментариями.

***Примечание***

В таких текстовых вставках будет приведена дополнительная информация, замечания автора, упоминание о родственных или связанных с рассматриваемой конструкцией элементах языка.

***А как сделать?***

В таких текстовых вставках будут представлены примеры использования данной языковой конструкции в варианте "Вопрос — ответ". Ставится задача и дается способ ее решения. Программный код, приведенный здесь, является типовым способом решения задачи и может быть использован начинающими программистами в собственных наработках.

Приведенные примеры рекомендуется тут же рассматривать на практике. Для этого мы сейчас создадим нашу первую внешнюю обработку, в которой и будем закреплять на практике полученные знания.

# Наша первая обработка

В упрощенном определении *обработка* — это программа, написанная на языке 1С и выполняемая в системе "1С:Предприятие". Обработка может быть как *внутренней*, входящей в конфигурацию и присутствующей в дереве конфигурации в разделе **Обработки**, так и внешней, имеющей расширение epf и запускаемой через пункт меню **Файл | Открыть**.

Мы будем экспериментировать на внешней обработке. Давайте создадим ее. Для этого в режиме Конфигуратора мы должны выбрать меню **Файл | Новый** и в открывшемся списке выбрать вариант **Внешняя обработка**. Откроется окно создания новой обработки (рис. 2.1).

Обработка, как и справочник или документ, может иметь реквизиты, несколько форм и печатных форм-макетов. В данном случае мы пока создадим самую простенькую обработку с одной формой. Нам ее вполне хватит.

Дадим нашей обработке имя (рис. 2.2).

*Имя* (идентификатор) *обработки*, как и имя любого объекта 1С, будь то имя справочника, документа или переменной, не должно содержать пробелов. Регистр не учитывается, т. е. **НашаПервая Обработка** и **нашапервая обработка** — это один и тот же объект, просто первое читается удобнее.

*Синоним* — это представление имени, его записывают для того, чтобы в окнах системы "1С:Предприятие" вместо имени пользователь мог видеть название объекта в привычном и более читаемом виде. При этом обращение в программном коде, конечно, идет по имени, а не по синониму. Обратите внимание, что при вводе имени обработки синоним заполняется автоматически с правильным разделением слова по прописным буквам. Если бы мы назвали обработку "нашапервая обработка", это бы не сработало.

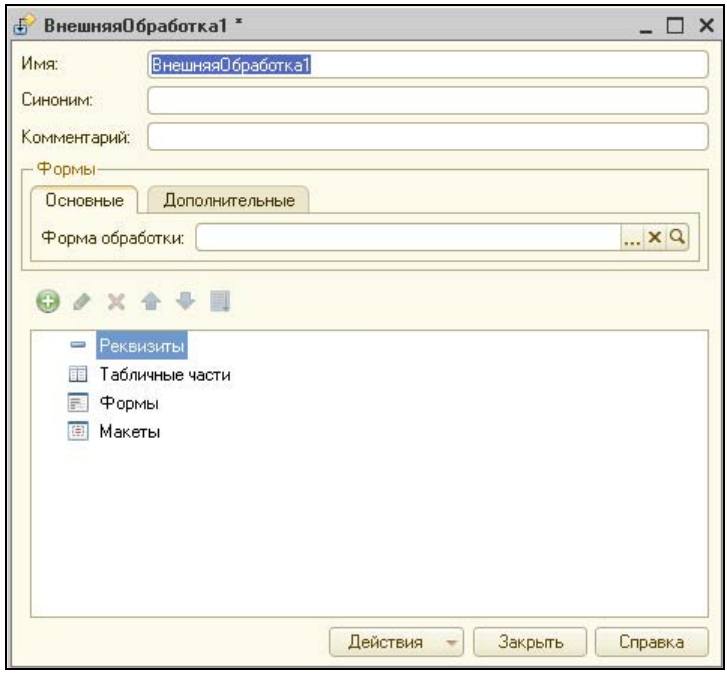



**Рис. 2.1.** Создаем внешнюю обработку

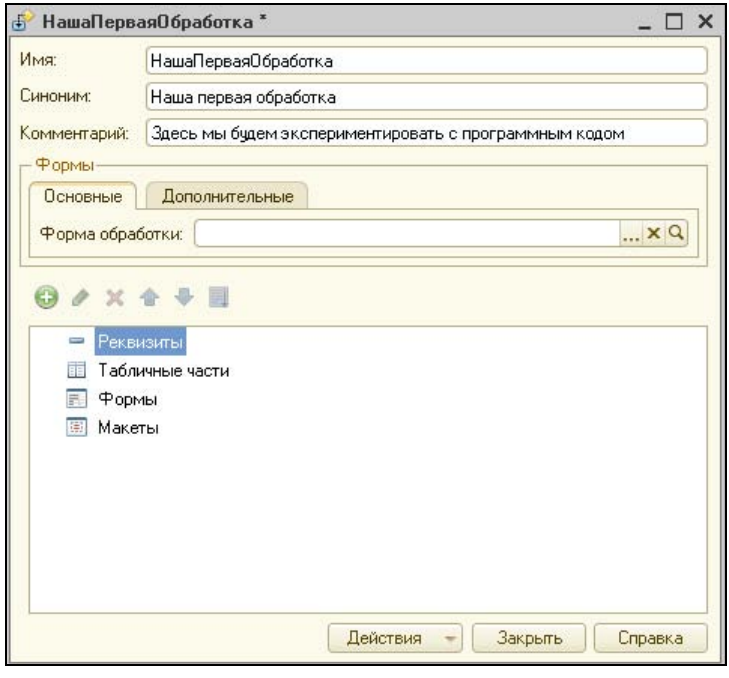



**Рис. 2.2.** Задаем имя и примечание

*Комментарий* предназначен для записи дополнительной информации об объекте.

Теперь создадим форму обработки. Для этого щелкнем правой кнопкой мыши на пункте **Формы** окна создания обработки и выберем пункт **Добавить**. Откроется окно конструктора формы обработки (рис. 2.3).

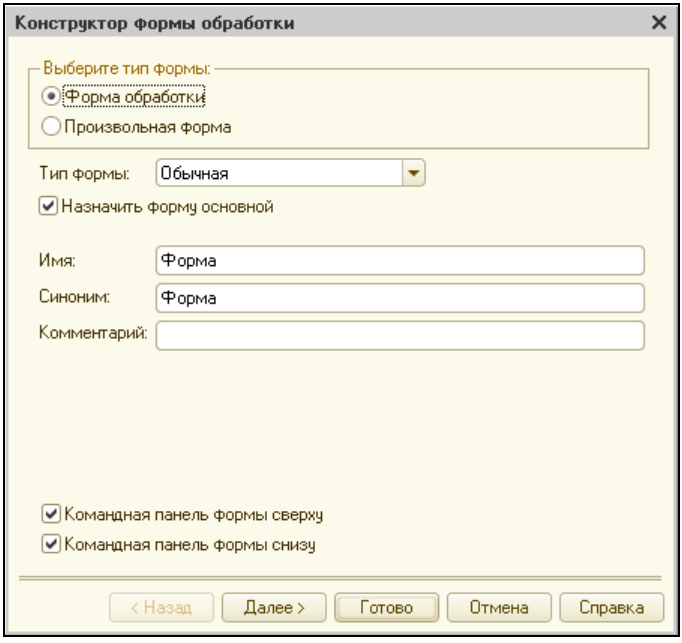



**Рис. 2.3.** Конструктор формы обработки

Здесь мы также можем задать имя, синоним и комментарий, на этот раз для формы, указать тип формы (обычная или управляемая, т. е. для работы через Интернет), определить положение командной панели.

Оставим все по умолчанию и нажмем кнопку **Готово**. Готовая пустая форма выглядит так, как показано на рис. 2.4.

Это пустая форма с командной панелью внизу, на которой расположены три кнопки: **Выполнить** (запускает обработчик выполнения обработки), **Закрыть** (закрывает обработку) и кнопка с точками  , предназначенная для добавления новых кнопок.

Ниже расположены вкладки, относящиеся к данной форме: **Диалог** (собственно форма с элементами, размещенными на ней), **Модуль** (здесь пишется программный модуль формы) и вкладка со списком реквизитов.

Программный модуль формы сразу после создания выглядит так, как показано на рис. 2.5.

На вкладке **Модуль** присутствует одна-единственная процедура-обработчик нажатия кнопки **Выполнить**, расположенной на форме. Обработчик пока пуст, в нем только комментарий, который не является исполняемым программным кодом. По-

этому, если мы откроем обработку в режиме "1С:Предприятие" через пункт меню **Файл | Открыть** и нажмем кнопку **Выполнить**, то никакие действия не произойдут. Процедуру обработчика (равно как и другие процедуры модуля) нам предстоит писать самим.

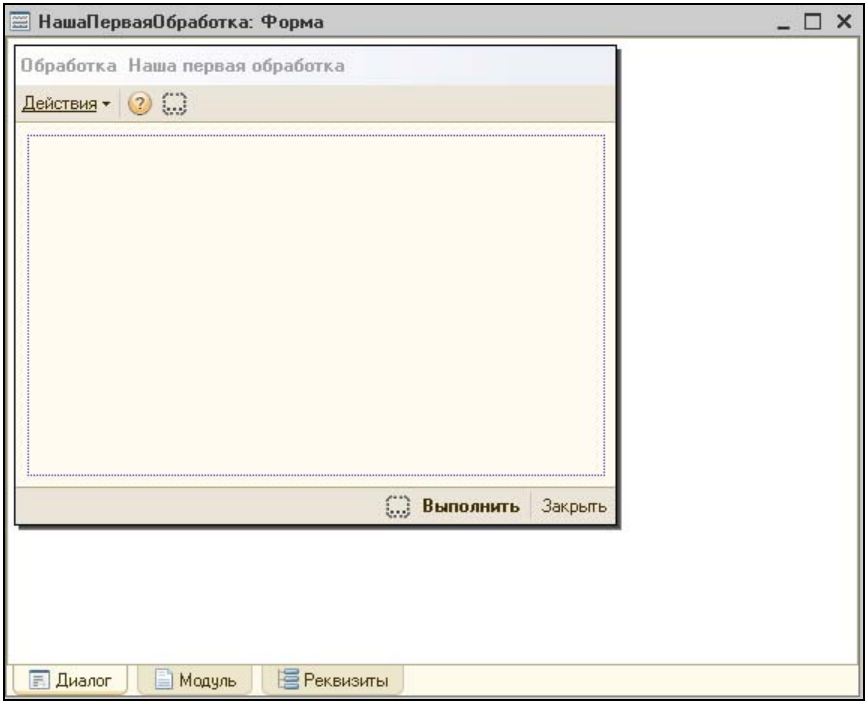



**Рис. 2.4.** Так выглядит пустая форма обработки

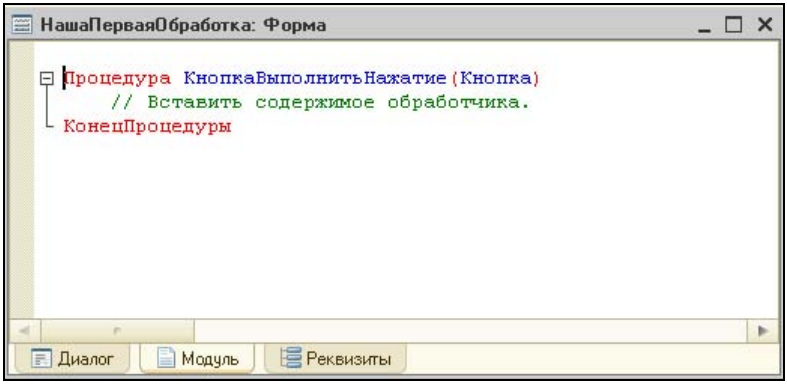



**Рис. 2.5.** Модуль формы новой обработки с одной-единственной пустой процедурой

Теперь давайте рассмотрим, какие бывают программные модули и какова их внутренняя структура.

# Какие бывают модули?

Программные модули в конфигурации системы "1С:Предприятие" не являются самостоятельными программами (за исключением внешних обработок, представляющих собой отдельные файлы). Каждый модуль привязывается к определенному моменту работы системы "1С:Предприятие". Система запущена — запускается содержимое одного модуля. Открыли какой-нибудь справочник — запускается другой модуль. Щелкнули по кнопке на форме — выполняется процедура, "подвешенная" на эту кнопку и находящаяся в модуле формы справочника. Таким образом, программный код в системе "1С:Предприятие" является *контекстно-зависимым*. Вместе с тем программные модули часто связаны между собой и могут быть доступны из других модулей системы.

Существуют области видимости программных элементов, процедур и функций, иначе называемые *контекстом выполнения программного модуля*. Таких контекстов два.

- **Глобальный контекст.** Образуется значениями констант, перечислений, регистров и прочих объектов метаданных, определенных в дереве конфигурации, системными переменными, процедурами и функциями, а также переменными, процедурами и функциями, находящимися в общих модулях конфигурации, объявленными с ключевым словом `Экспорт`. Данные, образующие глобальный контекст, доступны из любых других модулей конфигурации.
- **Локальный контекст конкретного модуля.** Образуется значениями переменных, процедур и функций, находящимися в конкретном программном модуле. Эти значения являются локальными и доступны только внутри модуля, в котором находятся. Исключение — использование в качестве параметров. Например, переменные определены в каком-либо модуле, а потом из этого модуля следует вызов процедуры (или функции), находящейся в одном из общих модулей. В этом случае, значения локальных переменных могут быть использованы в качестве параметров.

В отличие от "1С:Предприятие 7.7" в "1С:Предприятие 8.2" имеется больше различных типов модулей. Рассмотрим, какие именно.

- **Общие модули.** Процедуры и функции, помещенные в такие модули, доступны из любого другого модуля. То есть при проектировании, скажем, документа, мы всегда можем обратиться к любому из общих модулей. Расположены в разделе дерева конфигурации **Общие | Общие модули**.
- **Модуль формы.** Предназначен для обработки действий пользователя с объектом, которому принадлежит форма. Например, если мы поместим на форму кнопку **Выполнить**, то обработчик нажатия этой кнопки помещается в модуль формы. В созданной нами обработке для экспериментов мы уже заходили в модуль формы этой обработки (см. рис. 2.5).
- **Модуль объекта.** Данный модуль предназначен для обработки общих событий объекта. Например, для документа здесь будут располагаться процедуры записи

и проведения документа, а также отмены проведения. Для того чтобы открыть модуль объекта, расположенного в дереве конфигурации, нужно щелкнуть на нем правой кнопкой мыши и выбрать пункт **Открыть модуль объекта**. Для того чтобы открыть модуль нашей внешней обработки для экспериментов (не путаем с модулем формы), нужно в окне обработки (см. рис. 2.2) нажать кнопку **Действия** и выбрать пункт **Открыть модуль объекта**. Для нашей обработки этот модуль пуст.

- **Модуль приложения.** Срабатывает в момент запуска приложения (загрузки конфигурации) и завершения его работы. Сюда помещают программный код, который должен быть выполнен при запуске/закрытии приложения. Модуль доступен в контекстном меню, по щелчку правой кнопкой мыши в самом верхнем пункте дерева конфигурации (там, где название конфигурации). Существуют две разновидности: модуль обычного приложения и модуль управляемого приложения. Модуль обычного приложения предназначен для обычной работы (в режиме "толстого" клиента), режим управляемого приложения — в основном для работы через Интернет (веб-приложение, "тонкий" клиент или "толстый" клиент в режиме управляемого приложения).
- **Модуль сеанса.** Это модуль, в котором записаны параметры начала сеанса работы в системе "1С:Предприятие". Содержит единственную процедуру `УстановкаПараметровСеанса()`. Модуль доступен в контекстном меню по щелчку правой кнопкой мыши в самом верхнем пункте дерева конфигурации (там, где указано название конфигурации).
- **Модуль внешнего соединения.** Назначение модуля аналогично назначению модуля приложения, но только в режиме com-соединения. Модуль доступен в контекстном меню по щелчку правой кнопкой мыши в самом верхнем пункте дерева конфигурации (там, где указано название конфигурации).
- **Модуль менеджера объекта.** Существует для многих объектов конфигурации. Модуль предназначен для переопределения стандартного события выбора, которое возникает в момент ввода по строке. Модуль доступен в контекстном меню по щелчку правой кнопкой мыши по объекту в дереве конфигурации. В стандартных конфигурациях для большинства объектов пуст (не используется).
- **Модуль команды.** Команды — это объекты, подчиненные объектам дерева конфигурации. У каждой команды есть модуль команды, где можно описать предопределенную процедуру `ОбработкаКоманды()`, которая будет срабатывать для данной команды. Если мы пройдемся по дереву конфигурации, то увидим, что для каждого из объектов: справочников, документов, перечислений и т. д. имеется пункт **Команды**. По щелчку на нем правой кнопкой мыши мы можем добавить новую команду и задать для нее обработчик `ОбработкаКоманды()`.

Мы будем рассматривать работу с модулями формы, объекта, а также общими модулями, остальные модули довольно узкоспециализированы, в рабочих конфигурациях модифицируются не так часто и, как правило, не являются постоянно используемыми в работе программиста.

# Структура программного модуля

Программный модуль делится на три части.

- **Раздел описания переменных.** Здесь мы описываем переменные, с помощью оператора `Перем`. Данный раздел размещается от начала текста модуля до первого оператора `Процедура` или `Функция` или любого исполняемого оператора. В общих модулях этот раздел отсутствует.
- **Раздел процедур и функций.** Здесь пишутся все процедуры и функции модуля. Данный раздел размещается от первого оператора `Процедура` или до любого исполняемого оператора вне процедур или функций модуля.
- **Раздел основной программы.** Здесь пишутся команды и языковые конструкции, не относящиеся ни к одной из процедур и функций модуля. Данный раздел размещается от первого исполняемого оператора вне процедур или функций модуля до конца модуля. Здесь могут находиться только исполняемые операторы. Раздел основной программы исполняется в момент запуска модуля на выполнение, поэтому есть смысл помещать сюда, например, инициализацию переменных конкретными значениями. На практике раздел основной программы обычно только называется так, основную часть модуля занимают процедуры и функции, а этот раздел может отсутствовать вовсе. В общих модулях этот раздел отсутствует всегда.

Наличие всех частей модуля не является обязательным. Например, переменные могут объявляться непосредственно в процедурах (отсутствует раздел описания переменных). Или никакие команды не выполняются в момент выполнения модуля (отсутствует раздел основной программы). Наконец, модуль просто может состоять из одной или нескольких процедур или функций, т. е. второго раздела (кстати, на практике чаще всего так и бывает). И больше того, в некоторых объектах модуль вообще может быть пуст.

Пример программного модуля:

```
Перем a;
Перем b;
Перем c;

Процедура Расчет()
  c=a*b;
  Сообщить(c);
КонецПроцедуры

a=2;
b=3;
```

Это простенький программный модуль. Тем не менее он имеет все три части. Сначала объявляются переменные, при открытии модуля они инициализируются числами 2 и 3, а когда выполняется обработка, рассчитывается результат умножения и выдается результат — "6".

Модуль можно было написать и по-другому:

```
Процедура Расчет()
  a=2;
  b=3;
  c=a*b;
  Сообщить(c);
КонецПроцедуры
```

В этом примере отсутствуют разделы объявления переменных и раздел основной программы, имеется только процедура. Переменные `a` и `b` объявляются прямо в начале процедуры, без использования оператора `Перем`, не являющегося обязательным, а переменная `c` вообще объявляется неявно, посредством присваивания. Язык программирования 1С допускает такой стиль программирования, в отличие от более "строгих" языков. Главное — не забывать меру, чтобы написанные вами программные модули оставались понятными.

При написании программного модуля используются ключевые слова языка программирования, а также переменные и их методы.

*Ключевые слова* — это команды языка программирования, зарезервированные и запрещенные к использованию в качестве имен переменных.

*Переменными* являются величины, которые мы сами объявляем и используем, присваивая им имена и определенные значения. Именем переменной, процедуры или функции может быть любая последовательность букв, цифр и знаков подчеркивания "_", начинающаяся с буквы или знака подчеркивания "_". Имена не должны повторять ключевые слова, используемые в языковых конструкциях. Имена являются регистронезависимыми, т. е. `Тов`, `тов` и `ТоВ` — это одна и та же переменная.

*Методами переменных* называются действия, которые могут выполнять данные такого типа. Например, в конструкции `Спр.НайтиПоКоду("12345") Спр` — переменная типа "Справочник" (типы переменных мы сейчас рассмотрим подробней), а `НайтиПоКоду()` — ее метод, в скобках указывается, какой именно код мы пытаемся найти. К какому именно справочнику относится переменная `Спр`, задается отдельно, например, так: `Спр = Справочники.Контрагенты`. В этом случае последующая строка кода `Спр.НайтиПоКоду("12345")` будет искать контрагента с кодом 12345.

# Переменные и константы

*Константы* — это постоянные (условно-постоянные) величины. Они хранят информацию, которая не изменяется или изменяется достаточно редко. Например, константой может быть название фирмы, ее адрес, ФИО директора и т. д.

*Переменная* — это величина, которая в ходе выполнения программного модуля может принимать различные значения.

Значения, которые принимаются константой или переменной, имеют один из используемых в системе "1С:Предприятие" *типов данных*.

## Объявление переменной

Оператор `Перем` объявляет переменную. В процедурах и функциях он не является обязательным, и переменная может быть объявлена путем инициализации, например присваиванием нуля:

```
А=0;
```

Синтаксис:

```
Перем ИмяПеременной [Экспорт];
```

Здесь *Экспорт* — необязательное ключевое слово, означающее, что данная переменная, помещенная в глобальный модуль (в Конфигураторе меню **Действия | Глобальный модуль**), будет доступна всем остальным модулям конфигурации. Ключ `Экспорт` имеет смысл использовать только в глобальном модуле.

Пример:

```
Перем ОбщаяСумма;
```

# Типы данных

Данные в языке программирования 1С делятся на две категории: базовые и агрегатные.

*Базовые типы данных* являются общими и не привязаны непосредственно к объектам конфигурации. Существуют следующие базовые типы данных:

- *числовой* — любое десятичное число;
- *строковый* — любая последовательность символов, в том числе и пустая;
- *дата* — любая дата и время из допустимого диапазона;
- *булево* — принимает одно из двух возможных значений: `Истина` или `Ложь`;
- *не определено* — пустое значение, не принадлежащее ни к какому другому типу.

Базовые типы данных могут быть преобразованы друг в друга функциями преобразования. В разделе, посвященном преобразованию типов данных далее в этой главе, мы рассмотрим, как это делается.

В значениях типа "Число", имеющих дробную часть, в тексте программного целая и дробная части разделяются *точкой*. Вот так: `5.25`.

Значения типа "Строка" в тексте программного модуля заключаются в кавычки. Вот так: `"это – строка"`.

Значения типа "Дата" в тексте программного модуля заключаются в одинарные кавычки. Вот так: `'20120310'` или `'20120310195900'`. Первый пример означает 10 марта 2012 года (2012 год, 03 месяц, 10 день), второй пример — 10 марта 2012 года 19 часов 59 минут 00 секунд (2012 год, 03 месяц, 10 день, 19 час, 59 минута, 00 секунд).

*Агрегатные типы данных* — специальные типы данных, соответствующие типам объектов дерева конфигурации системы "1С:Предприятие". Сюда же относятся

универсальные коллекции значений, такие как СписокЗначений или Массив, а также специфические типы данных, такие как ТекстовыйДокумент или ТабличныйДокумент.

Для наглядности приведем пример работы с агрегатными типами данных:

```
Товары = Справочники.Номенклатура;
РезультатПоиска = Товары.НайтиПоНаименованию("Зеркало круглое");
Если РезультатПоиска <> Товары.ПустаяСсылка() Тогда
    Сообщить("Артикул " + РезультатПоиска.Артикул);
Иначе
    Предупреждение("Нет такого товара!");КонецЕсли;
```

Что делает этот код?

- Объявляет новую переменную Товары типа Справочники.Номенклатура. Это и есть агрегатный тип данных — тип, представленный в дереве конфигурации конкретным объектом.
- Используя метод справочника НайтиПоНаименованию(), ищем конкретное наименование товара в справочнике, присваивая результат переменной РезультатПоиска. У каждого объекта метаданных есть набор собственных методов, которые пишутся в формате *ИмяПеременной.Метод*, т. е. через точку. Аналогично, если у объекта есть поля (например, в справочнике товаров поле "Артикул"), к ним также можно обратиться в формате *ИмяПеременной.ИмяПоля*. Дальше в этой главе мы будем рассматривать методы агрегатных типов данных. Полный список методов можно найти в синтаксис-помощнике (меню **Сервис | Синтаксис-помощник**), где команды и методы сгруппированы по типам. На рис. 2.6 представлено дерево синтаксис-помощника, открытое как раз на методе НайтиПоНаименованию(), принадлежащем объектам типа "Справочник":
- Далее мы сравниваем переменную РезультатПоиска с пустой ссылкой типа Справочники.Номенклатура (переменная Товары именно этого типа, мы это задали в первой строке программного кода). То есть если мы что-то нашли, то ссылка не пуста, если не нашли — пуста. Если результат поиска не равен пустой ссылке на элемент справочника номенклатуры (знак <> означает "не равно"), тогда мы сообщаем артикул найденного товара (предваряя его текстовой строкой "Артикул "; если бы мы это не сделали, то получили бы в сообщении просто значение артикула, без пояснения, что это). В противном случае (Иначе) выводится предупреждение в центре экрана о том, что такого товара нет.

Агрегатные типы данных могут быть как неопределенные, так и конкретизированные. Например, тип "Константа", "Справочник", "Документ", "Перечисление" и т. п. — неопределенные агрегатные типы, которые могут обращаться к любому объекту данного типа. Не существует данных типа "Отчет" или "Обработка", потому что эти элементы дерева метаданных не являются данными, а только средствами манипуляции данными и их обработки.

Тип Справочники.Номенклатура, Документы.ПоступлениеТоваровУслуг, Перечисления.ДаНет и т. п. являются конкретизированными агрегатными типами данных, поскольку переменная данного типа ссылается не просто на какой-либо справочник, документ или перечисление, а на конкретно указанный.

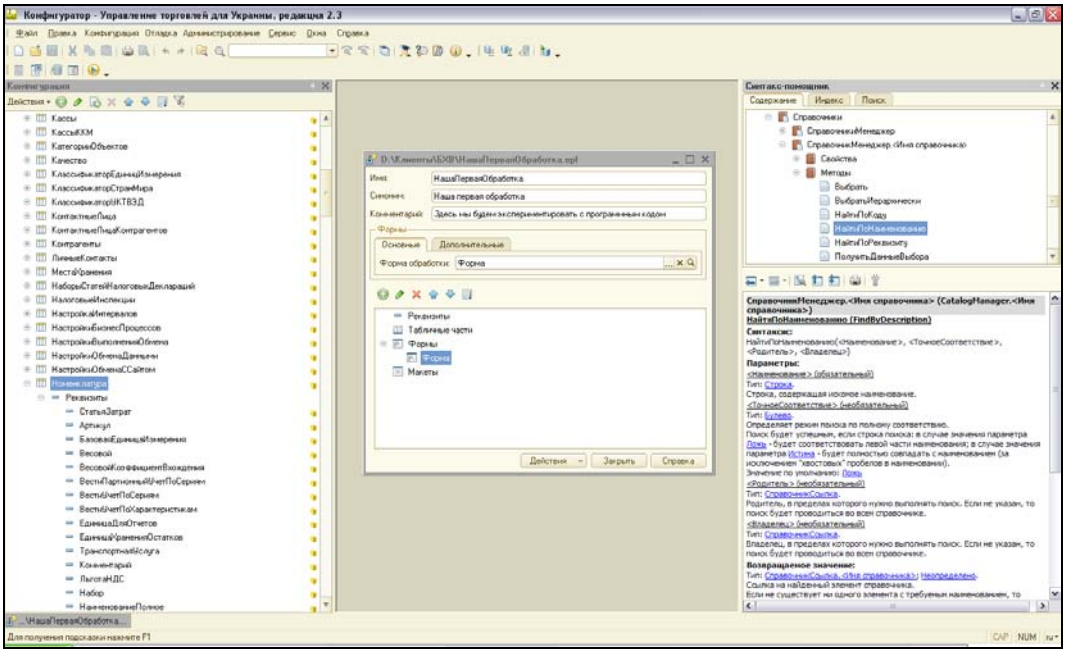


**Рис. 2.6.** Пример использования синтаксис-помощника

# Окончание строки ;

Точка с запятой не является оператором, в полном смысле этого значения. Однако это один из наиболее часто встречающихся элементов в программном модуле. Знаком точки с запятой должна заканчиваться каждая строка программного модуля, за некоторыми исключениями, которые мы рассмотрим далее. Обычно точкой с запятой не закрываются начало и конец процедуры и функции, начало цикла и условия `Если...Тогда`.

Пример:

```
A=2;
B=2;
C=A*B;
Сообщить(C);
```

# Комментарии //

Комментарий используется для размещения пояснений в тексте программного модуля. Для того чтобы разместить комментарий в тексте модуля, необходимо ввести две наклонные черты `//` и после них писать все, что посчитаете нужным: вся строка, находящаяся после `//`, не воспринимается как часть программы и является обычным поясняющим текстом.

Синтаксис:

```
// любое количество любых символов в строке
```

Пример:

```
X=5; // а дальше идет комментарий
// это тоже комментарий, начинающийся с начала строки
// Сообщить(X);
```

Несмотря на то, что введена строка на языке 1С, знак комментария перед ней превращает ее в обычный текст.

> ***А КАК ЗАКОММЕНТИРОВАТЬ СРАЗУ БОЛЬШОЙ БЛОК КОДА?***
>
> Выделить код мышью и нажать кнопку **Добавить комментарий**  . Удаляется блок комментариев аналогично: нужно выделить раскомментируемый блок мышью и нажать кнопку **Удалить комментарий**  .

# Процедуры и функции

## Процедура

*Процедура* — часть программного модуля, которая может быть выполнена обращением к ней по имени. Например, если имеется процедура `Печать()`, а также кнопка на форме в обработчике нажатия которой прописано имя процедуры – `Печать()`, то при нажатии кнопки процедура будет выполнена. Обратите внимание, что скобки ставятся, даже если процедура не имеет параметров. Работа процедуры может быть остановлена, если по ходу ее выполнения в программном коде встретился оператор `Возврат`.

```
Процедура()
   ...
КонецПроцедуры
```

Синтаксис:

```
Процедура ИмяПроцедуры(Параметр1, Параметр2,..., ПараметрN) [Экспорт]
```

Здесь: *Параметр1*, *Параметр2*, ..., *ПараметрN* — список параметров; `Экспорт` — ключевое слово, которое может присутствовать или отсутствовать и означает, что данная процедура, помещенная в глобальном модуле (в Конфигураторе меню **Действия | Глобальный модуль**), будет доступна всем остальным модулям конфигурации. Ключ `Экспорт` имеет смысл использовать только в глобальном модуле.

Пример:

```
Процедура Перемножить()
  A=1;
  B=2;
  C=3;
  D=A*B*C;
КонецПроцедуры
```

***Примечание***

Ключевые слова Процедура и КонецПроцедуры являются не операторами, а так называемыми *операторными скобками*, поэтому их не следует заканчивать точкой с запятой.

## Функция

*Функция* — часть программного модуля, которая может быть выполнена обращением к ней по имени, однако в отличие от процедуры, функция всегда возвращает какое-либо значение. Например, если имеется функция Расчет(), кнопка на форме в обработчике нажатия которой прописано имя процедуры – Расчет(), а также два поля типа "Число" на форме, то по нажатию кнопки можно передать значения числовых полей в функцию и получить результат. Обратите внимание, что скобки ставятся, даже если функция не имеет параметров. В отличие от процедуры, функция обязательно заканчивается оператором Возврат.

```
Функция()
  ...
  Возврат ...
КонецФункции
```

Синтаксис:

```
Функция ИмяФункции (Параметр1, Параметр2,..., ParametrN) [Экспорт]
```

Здесь: *Параметр1*, *Параметр2*, ..., *ПараметрN* — список параметров; Экспорт — ключевое слово, которое может присутствовать или отсутствовать и означает, что данная функция, помещенная в глобальном модуле (в Конфигураторе меню **Действия | Глобальный модуль**), будет доступна всем остальным модулям конфигурации. Ключ Экспорт имеет смысл использовать только в глобальном модуле.

Пример:

```
Перем Первая;
Перем Вторая;

Функция Сложение(А, В)
  С=А+В;
  Возврат С;
КонецФункции

Первая=2;
Вторая=3;
Сообщить(Сложение(Первая, Вторая));
```

В итоге получим результат "5". Мы передаем в функцию значения переменных Первая и Вторая, там они присваиваются параметрам функции А и В, складываются, после чего возвращается результат. Попробуйте ввести подобный код в нашей обработке для экспериментов. При этом модуль формы обработки должен выглядеть так, как показано на рис. 2.7.

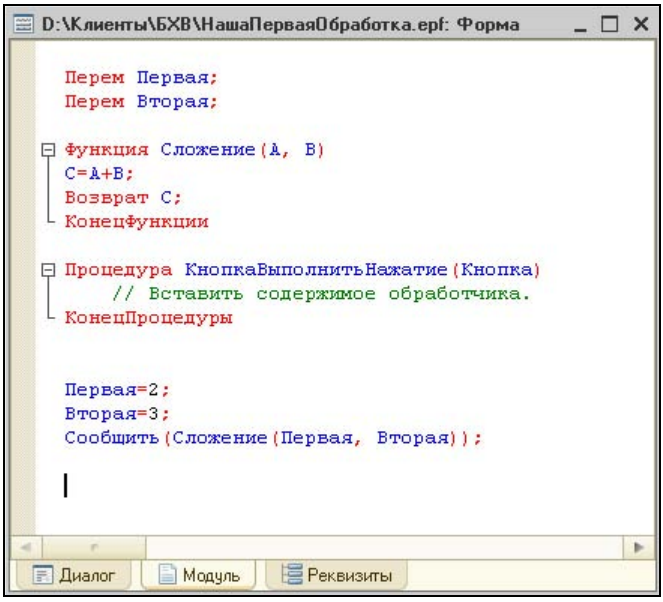



**Рис. 2.7.** Простейшая функция сложения чисел

Сохраните обработку, затем откройте ее в режиме "1С:Предприятие" через меню **Файл | Открыть**. Будет выдан результат: 5.

> ***Примечание***
>
> Ключевые слова `Функция` и `КонецФункции` являются не операторами, а так называемыми *операторными скобками*, поэтому их не следует заканчивать точкой с запятой.

> ***Примечание***
>
> Работу обработки следует проверять в режиме "1С:Предприятие" через меню **Файл | Открыть**. Если пример отработал, и вы стерли его код в обработке и ввели новый, не забудьте перед выполнением сохранить его, а в режиме предприятия — закрыть обработку, после чего открыть на выполнение снова. Открывать обработку заново следует потому, что уже открытая в режиме предприятия обработка — это старая версия обработки на момент предыдущего открытия. В Конфигураторе вы ее уже изменили и сохранили, но в уже открытой ранее обработке эти изменения еще не зафиксированы, и будет выполняться старая версия.

## Оператор *Возврат*

Этот оператор завершает выполнение процедуры или функции, причем в функции возвращает значение в формате `Возврат` *ИмяПеременной*.

Синтаксис:

```
Возврат [Значение];
```

Пример:

```
Функция ОпределениеДняНедели(Номер)
   Если Номер=1 Тогда
      Возврат "понедельник";
```

```
   ИначеЕсли Номер=2 Тогда
      Возврат "вторник";
   ИначеЕсли Номер=3 Тогда
      Возврат "среда";
   ИначеЕсли Номер=4 Тогда
      Возврат "четверг";
   ИначеЕсли Номер=5 Тогда
      Возврат "пятница";
   ИначеЕсли Номер=6 Тогда
      Возврат "суббота";
   ИначеЕсли Номер=7 Тогда
      Возврат "воскресенье";
   Иначе
      Возврат "В неделе семь дней!";
   КонецЕсли;
КонецФункции
```

# Операции

## Арифметические операции

*Арифметические операции* предназначены для выполнения между операндами арифметических действий. В языке программирования 1С существуют следующие виды арифметических операций (табл. 2.1).

***Таблица 2.1.*** *Арифметические операции*

| Операция | Символ операции | Пример |
|---|---|---|
| Сложение | + | Если A=2 и B=3, A+B=5 |
| Вычитание | - | Если A=5 и B=3, A-B=2 |
| Умножение | * | Если A=2 и B=3, A*B=6 |
| Деление | / | Если A=8 и B=2, A/B=4 |
| Остаток от деления | % | Если A=25 и B=5, A%B=0 (нет остатка от деления) |
| Унарный минус | - | Если A=1, -A=-1 |

Арифметические операции могут выполняться между числами, датами, а также обоими типами данных вместе. Например, выполнение следующего кода выдаст результат 10.03.2012 0:00:05:

```
Перем A, B, C;
A= '20120310';
B=5;
C=A+B;
Сообщить (C);
```

К дате 10.03.2012 мы прибавили 5. По умолчанию, если указана только дата без времени, время считается 0 часов 00 минут 00 секунд. Прибавление 5 дает нам 0 часов 00 минут 05 секунд той же даты.

## Операция конкатенации

*Конкатенация* — это присоединение нескольких строк друг к другу, для чего используется знак + (плюс). Если какая-либо из соединяемых строк не имеет тип "Строка", она преобразуется к нему.

Пример:

```
Фамилия= "Иванов";
Имя="Петр";
Отчество="Васильевич";
ФИО=Фамилия+" "+Имя+" "+Отчество;
Сообщить(ФИО);
```

В результате конкатенации трех строковых переменных и двух пробелов (" ") между ними мы получим сообщение "Иванов Петр Васильевич".

## Логические операции

*Логические операции* предназначены для сравнения операндов (табл. 2.2) и возвращают логическое значение `Истина` или `Ложь`. В числовом представлении `Истина` = 1, `Ложь` = 0.

***Таблица 2.2.*** *Логические операции*

| Операция | Символ операции | Пример |
|---|---|---|
| Больше | > | A>B |
| Больше или равно | >= | A>=B |
| Меньше | < | A<B |
| Меньше или равно | <= | A<=B |
| Равно | = | A=B |
| Не равно | <> | A<>B |

## Булевы операции

С логическими операциями тесно связаны так называемые *булевы операции*:

- `И`;
- `ИЛИ`;
- `НЕ`.

Пример:

```
Если (А>В) И (А<>5) Тогда Сообщить("А больше В и не равно 5");
КонецЕсли;
```

Пример:

```
Если (А > 10 = Ложь) ИЛИ (Б > 10 = Ложь) Тогда Сообщить("Хотя бы одна из двух переменных меньше или равно 10");
КонецЕсли;
```

# Присваивание

Оператор присваивания = присваивает переменной определенное значение.

Синтаксис:

***Переменная=Значение;***

Пример:

```
А=5;
Стр="строка текста";
Выч=А+10;
Товары = Справочники.Номенклатура;
```

# Диалог с пользователем

К диалоговым функциям относятся функции, предназначенные для получения данных от пользователя, посредством вопроса или диалогового окна ввода данных. Сюда же можно отнести функции общего назначения, непосредственно не ведущие диалог с пользователем, однако связанные с диалоговыми функциями по смыслу.

- ВвестиЗначение. Пользователю предлагается ввести значение заданного типа.

  Синтаксис:

  **ВвестиЗначение(*Значение*, *Подсказка*, *Тип*);**

  Здесь: *Значение* — имя переменной, которой будет присвоено введенное значение; *Подсказка* — текст подсказки пользователю; *Тип* — тип вводимого значения ("Строка", "Число", элемент справочника или документ и т. п.).

  Пример:

  ```
  Перем КакаяНибудьПеременная;
  ВвестиЗначение(КакаяНибудьПеременная, "Введите строковое значение",
  Тип("Число"));
  Сообщить(КакаяНибудьПеременная);
  ```

  Пример:

  ```
  Перем КакаяНибудьПеременная;
  Если ВвестиЗначение(КакаяНибудьПеременная, "Введите значение переменной",
  Тип("ДокументСсылка.ПоступлениеТоваровУслуг")) Тогда
  ```

```
    Сообщить(КакаяНибудьПеременная);
КонецЕсли;
```

Приведенные примеры практически идентичны, за исключением того, что в первом случае вводилось число, а во втором — документ поступления товаров и услуг (иначе говоря, приходная накладная). При выполнении такой обработки будет открыт журнал документов "Поступление товаров и услуг", и пользователю нужно будет выбрать один из них. Обратите внимание, что при открытии кавычки в параметре "Тип" система предоставляет выбор из списка возможных значений.

- `ВвестиЧисло`. Пользователю предлагается ввести значение числового типа.

  Синтаксис:

  **`ВвестиЧисло(`*`Значение`*`[, `*`Подсказка`*`][, `*`Длина`*`][, `*`Точность`*`]);`**

  Здесь: *`Значение`* — имя переменной, которой будет присвоено введенное значение; *`Подсказка`* — текст подсказки пользователю; *`Длина`* — длина вводимого значения; *`Точность`* — количество знаков после запятой.

  Пример:

```
Перем Количество;
Если ВвестиЧисло(Количество, "Введите количество", 10, 2) Тогда
    КвадратКоличества = Количество*Количество;
КонецЕсли;
Сообщить("Квадрат введенного значения равен "+КвадратКоличества);
```

- `ВвестиСтроку`. Пользователю предлагается ввести значение строкового типа.

  Синтаксис:

  **`ВвестиСтроку(`*`Значение`*`[, `*`Подсказка`*`][, `*`Длина`*`][, `*`Многострочность`*`]);`**

  Здесь: *`Значение`* — имя переменной, которой будет присвоено введенное значение; *`Подсказка`* — текст подсказки пользователю; *`Длина`* — длина строки; *`Многострочность`* — может принимать значение `Истина` или `Ложь` и определяет ввод простой строки или многострочной.

- `ВвестиДату`. Пользователю предлагается ввести значение типа "Дата".

  Синтаксис:

  **`ВвестиДату(`*`Значение`*`[,`*`Подсказка`*`][, `*`ЧастьДаты`*`]);`**

  Здесь: *`Значение`* — имя переменной, которой будет присвоено введенное значение; *`Подсказка`* — текст подсказки пользователю; *`ЧастьДаты`* — задаваемая пользователем часть даты.

  Пример:

```
ДатаНапоминания = РабочаяДата;
Подсказка = "Введите дату и время";
ЧастьДаты = ЧастиДаты.ДатаВремя;
```

```
Если ВвестиДату(ДатаНапоминания, Подсказка, ЧастьДаты) Тогда
  Сообщить(ДатаНапоминания);
КонецЕсли;
```

При выполнении данного примера будут выведены дата и время в окне, в котором мы можем задать нужное нам значение (рис. 2.8).

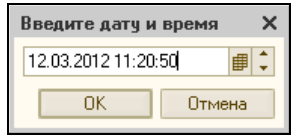



**Рис. 2.8.** Ввод даты и времени

Это и будет `ЧастьДаты` — та часть, которую нам предложено вводить. Если бы строка выглядела так:

```
ЧастьДаты = ЧастиДаты.Время;
```

то нам было бы предложено вводить только время, без даты.

- `Вопрос`. Выдает на экран диалоговое окно, в котором выбирается диапазон дат.

  Синтаксис:

  **`Вопрос(`*`ТекстВопроса`*`, `*`Кнопки`*`[, `*`Таймаут`*`][, `*`КнопкаПоУмолчанию`*`][, `*`Заголовок`*`][, `*`КнопкаТаймаута`*`]);`**

  Здесь: *`ТекстВопроса`* — собственно текст вопроса; *`Кнопки`* — состав и текст кнопок в диалоговм окне (табл. 2.3); *`Таймаут`* — время ожидания системы в секундах (по умолчанию не ограничено, т. е. равно 0); *`КнопкаПоУмолчанию`* — кнопка по умолчанию; *`Заголовок`* — заголовок окна вопроса; *`КнопкаТаймаута`* — кнопка, на которой отображается количество секунд, оставшихся до истечения тайм-аута.

***Таблица 2.3.*** *Режимы вопроса в языке программирования 1С системы "1С:Предприятие 8.2"*

| **Параметр *`Кнопки`*** | **Кнопки в окне диалога** |
|---|---|
| `ДаНет` | Кнопки **Да** и **Нет** |
| `ДаНетОтмена` | Кнопки **Да**, **Нет**, **Отмена** |
| `ОК` | Кнопка **ОК** |
| `ОКОтмена` | Кнопки **ОК** и **Отмена** |
| `ПовторитьОтмена` | Кнопки **Повторить**, **Отмена** |
| `ПрерватьПовторитьПропустить` | Кнопки **Прервать**, **Повторить**, **Пропустить** |

Функция `Вопрос` может возвращать следующие значения:

- `Да`;
- `Нет`;
- `ОК`;
- `Отмена`;
- `Повторить`;
- `Прервать`;
- `Пропустить`;
- `Таймаут`.

Пример:

```
Ответ = Вопрос("Продолжить?", РежимДиалогаВопрос.ДаНет);
Если Ответ = КодВозвратаДиалога.Да Тогда
    Сообщить("Пользователь ответил ДА");
ИначеЕсли Ответ = КодВозвратаДиалога.Нет Тогда
    Сообщить("Пользователь ответил НЕТ");
КонецЕсли;
```

Пример:

```
А=Вопрос("Вы довольны вашей зарплатой?", РежимДиалогаВопрос.ДаНетОтмена);
Если А=КодВозвратаДиалога.Да Тогда
    Сообщить("Ого!");
ИначеЕсли А=КодВозвратаДиалога.Нет Тогда
    Сообщить("Нет!!!");
ИначеЕсли А=КодВозвратаДиалога.Отмена Тогда
    Сообщить("Я еще подумаю...");
КонецЕсли;
```

- `Предупреждение`. Выводит на экран предупреждение.

  Синтаксис:

  **`Предупреждение(`*`ТекстСообщения`*`[, `*`Таймаут`*`][, `*`Заголовок`*`]);`**

  Здесь: *`Таймаут`* — время ожидания ответа от пользователя в секундах, по умолчанию 0 (неограниченное); *`Заголовок`* — заголовок окна предупреждения.

  Пример:

  ```
  Предупреждение("Ошибка!");
  ```

- `Сообщить`. Выводит сообщение в табло, в нижней части экрана.

  Синтаксис:

  **`Сообщить(`*`ТекстСообщения`*`);`**

  Пример:

  ```
  Сообщить("Обработка завершена");
  ```

  Пример:

  ```
  Сообщить("Итого сумма равна "+ВычСумма);
  ```

# Обработка, преобразование и форматирование данных

В процессе работы с программным кодом программисту нередко приходится преобразовывать один тип в другой, обрабатывать и форматировать имеющиеся данные (например, вывести числовое значение только с двумя знаками после запятой или дату без времени в формате "8 марта 2012 года"). В этом разделе мы узнаем о преобразовании и форматировании подробнее.

## Значения типа "Число". Основные функции

- `Цел`. Возвращает целую часть переданного числа без дробной части.

  Синтаксис:

  **Цел(*Число*);**

  Пример:

  ```
  Сообщить(Окр(10.784));
  ```

  Если записать такую строку кода в обработку и запустить, будет выдано 10.

- `Окр`. Возвращает целую часть округленного числа без дробной части.

  Синтаксис:

  **Окр(*Число*);**

  Пример:

  ```
  Сообщить(Окр(10.784));
  ```

  Если записать такую строку кода в обработку и запустить, будет выдано 11.

  ***А КАК ВЫВЕСТИ ПРОПИСЬЮ ВВОДИМОЕ ПОЛЬЗОВАТЕЛЕМ ЧИСЛО БЕЗ ДРОБНОЙ ЧАСТИ?***

  ```
  Перем ВводимоеЧисло;
  ВвестиЧисло(ВводимоеЧисло, "Введите число");
  ФорматнаяСтрока = "Л=ru_RU";
  ПараметрыПредметаИсчисления = " , , , , , , , , 0";
  Сообщить(ЧислоПрописью(ВводимоеЧисло, ФорматнаяСтрока,
  ПараметрыПредметаИсчисления));
  ```

  ***А КАК ВЫВЕСТИ ПРОПИСЬЮ ВВОДИМУЮ ПОЛЬЗОВАТЕЛЕМ СУММУ В РУБЛЯХ И КОПЕЙКАХ?***

  ```
  Перем ВводимоеЧисло;
  ВвестиЧисло(ВводимоеЧисло, "Введите число");
  ПараметрыПредметаИсчисления = "рубль, рубля, рублей, м, копейка, копейки,
  копеек, м, 2";
  ФорматнаяСтрока = "Л=ru_RU; ДП=Истина";
  Сообщить(ЧислоПрописью(ВводимоеЧисло, ФорматнаяСтрока,
  ПараметрыПредметаИсчисления));
  ```

## Значения типа "Строка". Основные функции

- `СтрДлина`. Возвращает количество символов в строке.

  Синтаксис:

  **СтрДлина(*Строка*);**

  Пример:

  ```
  Сообщить(СтрДлина("Определим длину этой строки"));
  ```

  Если записать такую строку кода в обработку и запустить, будет выдано 27.

Пример:

```
Стр = "А это другая строка, и мы присвоим ее переменной";
Сообщить(СтрДлина(Стр));
```

Если записать такую строку кода в обработку и запустить, будет выдано 48.

- СокрЛ. Отсекает незначащие символы (пробелы) в левой части строки.

  Синтаксис:

  **СокрЛ(*Строка*);**

- СокрП. Отсекает незначащие символы (пробелы) в правой части строки.

  Синтаксис:

  **СокрП(Строка);**

- СокрЛП. Отсекает незначащие символы (пробелы) в левой и правой частях строки.

  Синтаксис:

  **СокрЛП(*Строка*);**

  Пример:

```
Сообщить(СокрЛП("     Текстовая строка          "));
```

  Если записать такую строку кода в обработку и запустить, будет выдано "Текстовая строка".

- Лев. Выбирает указанное количество символов в левой части строки.

  Синтаксис:

  **Лев(*Строка, ЧислоСимволов*);**

  Пример:

```
ПроверочнаяСтрока = "Доброе утро, страна!";
ПорезаннаяСтрока = Лев(ПроверочнаяСтрока, 11);
Сообщить(ПорезаннаяСтрока);
```

  Если записать такую строку кода в обработку и запустить, будет выдано "Доброе утро".

- Прав. Выбирает указанное количество символов в правой части строки.

  Синтаксис:

  **Прав(*Строка, ЧислоСимволов*);**

  Пример:

```
ПроверочнаяСтрока = "Доброе утро, страна!";
ПорезаннаяСтрока = Прав(ПроверочнаяСтрока, 7);
Сообщить(ПорезаннаяСтрока);
```

  Если записать такую строку кода в обработку и запустить, будет выдано "страна!".

- `Сред`. Выбирает указанное количество символов, начиная с указанного символа строки.

  Синтаксис:

  ***Сред(Строка, НомерСимвола[, ЧислоСимволов]);***

  Если число символов не указано, то выбирается вся строка.

  Пример:

```
ПроверочнаяСтрока = "Доброе утро, страна!";
ПорезаннаяСтрока = Сред(ПроверочнаяСтрока, 8, 4);
Сообщить(ПорезаннаяСтрока);
```

  Если записать такую строку кода в обработку и запустить, будет выдано "утро".

- **Операция конкатенации** +. Соединяет несколько строк в одну.

  Синтаксис:

  ***Строка1+Строка2+...+СтрокаN;***

  Пример:

```
Строка1 = "Изучение языка программирования 1С";
Строка2 = "не только интересно";
Строка3 = "но и полезно";
Строка4 = Строка1+" "+Строка2+", "+Строка3+"!";
Сообщить(Строка4);
```

  При запуске этот программный код выдаст следующее: "Изучение языка программирования 1С не только интересно, но и полезно!".

  Кроме заданных текстовых строк в конкатенации также участвовали дополнительные строки, содержащие пробелы, запятую и восклицательный знак.

## Значения типа "Дата". Основные функции

- `ТекущаяДата`. Возвращает текущую дату.

  Синтаксис:

  ***ТекущаяДата();***

- `ДобавитьМесяц`. Добавляет к указанной дате указанное количество месяцев. Если это количество отрицательное, то вычитает его из даты.

  Синтаксис:

  ***ДобавитьМесяц(Дата, ЧислоМесяцев);***

  Пример:

```
Сообщить(ДобавитьМесяц(ТекущаяДата(), -12));
```

  Если записать такую строку кода в обработку и запустить, будут выданы дата и время запуска минус год.

- ДеньНедели. Определяет номер дня недели для указанной даты.

  Синтаксис:

  ```
  ДеньНедели(Дата);
  ```

  Пример:

  ```
  Сообщить(ДеньНедели(ТекущаяДата()));
  ```

  Если записать такую строку кода в обработку и запустить, будет выдан номер текущего дня недели. Например, этот фрагмент кода проверялся во вторник, и обработка выдала цифру 2.

- ДеньГода. Определяет номер дня в году для указанной даты.

  Синтаксис:

  ```
  ДеньГода(Дата);
  ```

- НеделяГода. Определяет номер недели в году для указанной даты.

  Синтаксис:

  ```
  НеделяГода(Дата);
  ```

- Год. Определяет год в указанной дате.

  Синтаксис:

  ```
  Год(Дата);
  ```

  Пример:

  ```
  Сообщить(Год(ТекущаяДата()));
  ```

  Этот фрагмент кода проверялся 13 марта 2012 года. Если записать такую строку кода в обработку и запустить, будет выдан текущий год — 2012.

- Месяц. Определяет месяц в указанной дате.

  Синтаксис:

  ```
  Месяц(Дата);
  ```

  Пример:

  ```
  Сообщить(Месяц(ТекущаяДата()));
  ```

  Этот фрагмент кода проверялся 13 марта 2012 года. Если записать такую строку кода в обработку и запустить, будет выдан текущий месяц — 3.

- День. Определяет календарный день в указанной дате.

  Синтаксис:

  ```
  День(Дата);
  ```

  Пример:

  ```
  Сообщить(День(ТекущаяДата()));
  ```

  Этот фрагмент кода проверялся 13 марта 2012 года. Если записать такую строку кода в обработку и запустить, будет выдан текущий день — 13.

- `Час`. Определяет час в указанной дате.

  Синтаксис:

  ```
  Час(Дата);
  ```

  Пример:

  ```
  Сообщить(Час(ТекущаяДата()));
  ```

  Этот фрагмент кода проверялся в 14:44:15. Если записать такую строку кода в обработку и запустить, будет выдан текущий час — 14.

- `Минута`. Определяет минуту в указанной дате.

  Синтаксис:

  ```
  Минута(Дата);
  ```

  Пример:

  ```
  Сообщить(Минута(ТекущаяДата()));
  ```

  Этот фрагмент кода проверялся в 14:44:15. Если записать такую строку кода в обработку и запустить, будет выдана текущая минута — 44.

- `Секунда`. Определяет секунду в указанной дате.

  Синтаксис:

  ```
  Секунда(Дата);
  ```

  Пример:

  ```
  Сообщить(Секунда(ТекущаяДата()));
  ```

  Этот фрагмент кода проверялся в 14:44:15. Если записать такую строку кода в обработку и запустить, будет выдана текущая секунда — 15.

- `НачалоГода`. Определяет дату и время начала года для указанной даты.

  Синтаксис:

  ```
  НачалоГода(Дата);
  ```

  Пример:

  ```
  Сообщить(НачалоГода('20120301'));
  ```

  Если записать такую строку кода в обработку и запустить, будет выдано 01.01.2012 0:00:00.

- `НачалоКвартала`. Определяет дату и время начала квартала для указанной даты.

  Синтаксис:

  ```
  НачалоКвартала(Дата);
  ```

  Пример:

  ```
  Сообщить(НачалоКвартала('20120801'));
  ```

  Если записать такую строку кода в обработку и запустить, будет выдано 01.07.2012 0:00:00.

- НачалоМесяца. Определяет дату и время начала месяца для указанной даты.

  Синтаксис:

  **НачалоМесяца(*Дата*);**

  Пример:

  ```
  Сообщить(НачалоМесяца('20120301'));
  ```

  Если записать такую строку кода в обработку и запустить, будет выдано 01.03.2012 0:00:00.

- НачалоНедели. Определяет дату и время начала недели для указанной даты.

  Синтаксис:

  **НачалоНедели(*Дата*);**

  Пример:

  ```
  Сообщить(НачалоНедели('20120301'));
  ```

  Если записать такую строку кода в обработку и запустить, будет выдано 27.02.2012 0:00:00 (поскольку 27 февраля 2012 года начинается неделя, в которую входит 1 марта 2012 года).

- НачалоДня. Определяет дату и время начала дня для указанной даты.

  Синтаксис:

  **НачалоДня(*Дата*);**

  Пример:

  ```
  Сообщить(НачалоДня('20120301'));
  ```

  Если записать такую строку кода в обработку и запустить, будет выдано 01.03.2012 0:00:00.

- НачалоЧаса. Определяет дату и время начала часа для указанной даты.

  Синтаксис:

  **НачалоЧаса(*Дата*);**

  Пример:

  ```
  Сообщить(НачалоЧаса('20120301'));
  ```

  Если записать такую строку кода в обработку и запустить, будет выдано 01.03.2012 0:00:00.

- НачалоМинуты. Определяет дату и время начала минуты для указанной даты.

  Синтаксис:

  **НачалоМинуты(*Дата*);**

  Пример:

  ```
  Сообщить(НачалоМинуты('20120301'));
  ```

Если записать такую строку кода в обработку и запустить, будет выдано 01.03.2012 0:00:00.

- КонецГода. Определяет дату и время конца года для указанной даты.

  Синтаксис:

  **КонецГода(*Дата*);**

  Пример:

  ```
  Сообщить(КонецГода('20120301'));
  ```

  Если записать такую строку кода в обработку и запустить, будет выдано 31.12.2012 23:59:59.

- КонецКвартала. Определяет дату и время конца квартала для указанной даты.

  Синтаксис:

  **КонецКвартала(*Дата*);**

  Пример:

  ```
  Сообщить(КонецКвартала('20120801'));
  ```

  Если записать такую строку кода в обработку и запустить, будет выдано 30.09.2012 23:59:59.

- КонецМесяца. Определяет дату и время конца месяца для указанной даты.

  Синтаксис:

  **КонецМесяца(*Дата*);**

  Пример:

  ```
  Сообщить(КонецМесяца('20120301'));
  ```

  Если записать такую строку кода в обработку и запустить, будет выдано 31.03.2012 23:59:59.

- КонецНедели. Определяет дату и время конца недели для указанной даты.

  Синтаксис:

  **КонецНедели(*Дата*);**

  Пример:

  ```
  Сообщить(КонецНедели('20120301'));
  ```

  Если записать такую строку кода в обработку и запустить, будет выдано 04.03.2012 23:59:59 (поскольку 4 марта 2012 года заканчивается неделя, в которую входит 1 марта 2012 года).

- КонецДня. Определяет дату и время конца дня для указанной даты.

  Синтаксис:

  **КонецДня(*Дата*);**

Пример:

```
Сообщить(КонецДня('20120301'));
```

Если записать такую строку кода в обработку и запустить, будет выдано 01.03.2012 23:59:59.

- КонецЧаса. Определяет дату и время конца часа для указанной даты.

  Синтаксис:

  **КонецЧаса(*Дата*);**

  Пример:

```
Сообщить(КонецЧаса('20120301'));
```

  Если записать такую строку кода в обработку и запустить, будет выдано 01.03.2012 0:59:59.

- КонецМинуты. Определяет дату и время конца минуты для указанной даты.

  Синтаксис:

  **КонецМинуты(*Дата*);**

  Пример:

```
Сообщить(КонецМинуты('20120301'));
```

  Если записать такую строку кода в обработку и запустить, будет выдано 01.03.2012 0:00:59.

#### *А КАК ПРИБАВИТЬ К ТЕКУЩЕЙ ДАТЕ СУТКИ?*

```
СекундВМинуте = 60;
СекундВЧасе = СекундВМинуте*60;
СекундВСутках = СекундВЧасе*24;
```

#### *А КАК ПРИБАВИТЬ К ЗАДАННОЙ ДАТЕ ПОЛГОДА?*

```
ПолученнаяДата = ДобавитьМесяц('20120315', 6);
```

#### *А КАК ВЫЧЕСТЬ ОТ ВВЕДЕННОЙ ПОЛЬЗОВАТЕЛЕМ ДАТЫ 2 МЕСЯЦА?*

```
Перем ВводимаяДата;
ВведеннаяДата = ВвестиДату(ВводимаяДата,"Введите дату");
ВводимаяДата = ДобавитьМесяц(ВводимаяДата, -2);
```

## Функции преобразования значений

- Число. Преобразует переданное значение в число.

  Синтаксис:

  **Число(*Значение*);**

  Пример:

```
A = "12345";
ПреобразованноеА = Число(А);
Сообщить("ПреобразованноеА");
```

Здесь мы преобразуем строковое представление числа (`А = "12345"`, если бы мы присваивали переменной `А` числовое значение, то писали бы `А = 12345`). Преобразование происходит успешно, после выполнения функции обработка выдаст нам `12345`.

Пример:

```
Б = "какая-то надпись";
ПреобразованноеБ = Число(Б);
Сообщить("ПреобразованноеБ");
```

А здесь мы пытаемся преобразовать в число явно несхожее с числом значение. Такая обработка выдаст 0.

- `Строка`. Преобразует переданное значение в строку.

  Синтаксис:

  ```
  Строка(Значение);
  ```

  Пример:

  ```
  А = 1000;
  Б = Строка(А);
  ```

  Примером использования цифровых литералов в значениях текстового типа может служить, например, справочник номенклатуры конфигурации "Управление Торговлей". Имея код типа "Строка", он, тем не менее, вполне может хранить также визуально "цифровые" коды, например 20245.

- `Дата`. Преобразует переданное значение в дату. Значение может передаваться целиком или по составляющим.

  ```
  Дата(Значение);
  Дата(Год, Месяц, День[, Час][, Минута][, Секунда]);
  ```

  Пример:

  ```
  ПреобразованиеВДату = Дата("20120313181637");
  ```

  Данный код преобразует введенное значение в значение, равное 13 марта 2012 года 18:16:37.

  Пример:

  ```
  ПреобразованиеВДату = Дата(2012,03,13,18,16,37);
  ```

  Данный код по своему действию аналогичен предыдущему, только преобразуется не единое значение, а составляющие его части.

## Форматирование данных

- `Формат`. Форматирует переданное значение в зависимости от заданной форматной строки. В случае отсутствия форматной строки происходит форматирование по умолчанию (как правило, преобразование значения к строковому типу).

Синтаксис:

```
Формат(Значение[, ФорматнаяСтрока]);
```

Пример:

```
Стр = Формат(587950.471, "ЧЦ=10; ЧДЦ=2");
Сообщить(Стр);
```

Здесь `ЧЦ=10` означает длину значения, `ЧДЦ=2` — количество знаков после запятой. Результат выполнения: `587 950,47`.

Пример:

```
Стр = Формат(587950.471, "ЧЦ=10; ЧГ=0; ЧДЦ=2");
Сообщить(Стр);
```

Этот пример практически идентичен предыдущему, за исключением параметра `ЧГ`, означающего способ группировки разрядов числа. `ЧГ=0` означает, что разрядность выводиться не будет (это важно при операциях экспорта-импорта данных в другие программы). Результат выполнения: `587950,47`.

Пример:

```
Стр = Формат(587950.471, "ЧЦ=10; ЧРГ=.; ЧДЦ=2");
Сообщить(Стр);
```

А здесь разделители групп разрядов будут выводиться в виде точки, за что отвечает параметр `ЧРГ`. Результат выполнения: `587.950,47`.

Пример:

```
Стр = Формат('20120313204547', "ДФ=""дд ММММ гггг ЧЧ:мм:сс""");
Сообщить(Стр);
```

Здесь мы преобразуем значение даты в более привычный нам формат:

- параметр `дд` означает номер дня в 12-часовом варианте с лидирующим нулем (т. е. 3 марта будет отражено, как 03 марта);
- параметр `ММММ` — полное название месяца;
- параматр `гггг` — номер года с веком;
- параметр `ЧЧ` — час в 12-часовом формате с лидирующим нулем;
- параметр `мм` — минута с лидирующим нулем;
- параметр `сс` — секунда с лидирующим нулем.

Результат выполнения: 13 марта 2012 20:45:47.

Пример:

```
Стр = Формат('20120313204547', "ДФ=""дд ММММ гггг 'г.' ЧЧ:мм:сс""");
Сообщить(Стр);
```

Этот пример идентичен предыдущему, за исключением того, что в форматную строку добавлено сокращение года `'г.'`. Результат выполнения: 13 марта 2012 г. 20:45:47.

Пример:

```
Стр = Формат('20120313204547', "ДЛФ=Д");
Сообщить(Стр);
```

А это та же самая дата, но в формате "ДЛФ=Д", т. е. просто дата. Результат выполнения: 13.03.2012.

Пример:

```
Стр = Формат('20120313204547', "ДЛФ=В");
Сообщить(Стр);
```

И это та же самая дата, но в формате "ДЛФ=В", т. е. просто время, без даты. Результат выполнения: 20:45:47.

Возможности функции Формат() не исчерпываются приведенными тут примерами, здесь представлены особенно характерные. Полную справку о возможностях функции и используемых в ней параметрах можно получить в документации фирмы "1С" к поставляемым ею программным продуктам или в синтаксис-помощнике (**Справка | Синтаксис-помощник**) в подразделе **Общее описание встроенного языка | Встроенные функции | Функции форматирования**.

- ЧислоПрописью. Представление числового значения прописью в зависимости от заданной форматной строки.

  Синтаксис:

  ***ЧислоПрописью(Число[, ФорматнаяСтрока][, Параметры]);***

  Пример:

  ```
  Стр = "Л = ru_RU; ДП = Истина";
  Вал = "рубль,рубля,рублей,м,копейка,копейки,копеек,м,2";
  Результат = ЧислоПрописью(12751.94, Стр, Вал);
  Сообщить(Результат);
  ```

  Здесь: Л означает локализация (русская); ДП — выводить ли дробную часть (Истина значит выводить); параметры склонения мы присвоили переменной Вал. Результат выполнения: Двенадцать тысяч семьсот пятьдесят один рубль девяносто четыре копейки.

  ***А КАК ВЫВЕСТИ ВВЕДЕННУЮ ПОЛЬЗОВАТЕЛЕМ ДАТУ В ФОРМАТЕ "8 МАРТА 2012 Г."?***

  ```
  Перем ВводимаяДата;
  ВведеннаяДата = ВвестиДату(ВводимаяДата,"Введите дату");
  ВводимаяДата = Формат(ВводимаяДата, "ДЛФ=ДД");
  Сообщить(ВводимаяДата);
  ```

## Прочие функции работы с данными

- Мин. Определяет минимальное значение из заданных параметров.

  Синтаксис:

  ***Мин(Значение1, Значение2, ..., ЗначениеN);***

Пример:

```
Сообщить(Мин(1,14,56,33,2,0,70));
```

Сравнивает числа в списке и выводит минимальное. Результат выполнения: `0`.

Строки, даты и булевы выражения также можно сравнивать с этой функцией.

Пример:

```
Сообщить(Мин("А", "Б", "И", "Я"));
```

Результат выполнения: `"А"`.

- `Макс`. Определяет максимальное значение из заданных параметров.

  Синтаксис:

  **`Макс(`*`Значение1`*`, `*`Значение2`*`, ..., `*`ЗначениеN`*`);`**

  Пример:

  ```
  Сообщить(Макс(1,14,56,33,2,0,70));
  ```

  Сравнивает числа в списке и выводит максимальное. Результат выполнения: `70`.

  Строки, даты и булевы выражения также можно сравнивать с этой функцией.

  Пример:

  ```
  Сообщить(Макс("А", "Б", "И", "Я"));
  ```

  Результат выполнения: `"Я"`.

# Условия

Операторы условия занимают одно из основополагающих мест среди команд и операторов встроенного языка. В зависимости от выполнения или невыполнения условия программа осуществляет то или иное действие.

## Оператор *?*

В зависимости от выполнения условия вычисляется одно или другое выражение.

Синтаксис:

**`?(`*`Условие`*`, `*`Выражение1`*`, `*`Выражение2`*`);`**

Пример:

```
ПроверочнаяПеременная = 7;
БольшеДесятиИлиМеньше = ?(ПроверочнаяПеременная>10, "Да, больше 10","Нет,
меньше или равно 10");
Сообщить(БольшеДесятиИлиМеньше);
```

В приведенном примере мы присваиваем переменной `ПроверочнаяПеременная` значение 7, а затем, сравнивая это значение с 10, присваиваем результат проверки другой

переменной и выводим результат в виде сообщения. Поскольку 7 меньше 10, будет выведен ответ: "Нет, меньше или равно 10".

Пример:

```
Перем ПроверочнаяПеременная;
ВвестиЧисло(ПроверочнаяПеременная, "Введите число");
БольшеДесятиИлиМеньше = ?(ПроверочнаяПеременная>10, Истина, Ложь);
Сообщить(БольшеДесятиИлиМеньше);
```

Тот же самый пример, только немного усложненный. Пользователь не проверяет заданные значения, а предлагает пользователю ввести число самому и сравнивает с 10. Если больше 10 — Истина, если меньше либо равно 10 — Ложь. В зависимости от введенного пользователем числа обработка выдаст "Да" (если Истина) или "Нет" (если Ложь).

## Оператор *Если*

Оператор условия управляет выполнением программы на основании истинности или ложности одного или более логических выражений.

Синтаксис:

```
Если ЛогическоеВыражение Тогда;
  ...
  // Этот блок программного кода сработает, если условие выполняется
  ...
[ИначеЕсли ЛогическоеВыражение Тогда]
  ...
  // Этот блок программного кода сработает, если первое условие
  // не выполняется, а это — выполняется
  ...
[Иначе]
  ...
  // Этот блок программного кода сработает, если ни одно условие
  // из приведенных выше не выполнилось
  ...
КонецЕсли;
```

Задается условие, если оно истинно, то выполняется блок программного кода, идущий следом за Если...Тогда. Если выражение ложно, то блок игнорируется.

Могут быть заданы необязательные блоки ИначеЕсли и Иначе. Блоков ИначеЕсли может быть несколько. Такой блок будет выполнен в том случае, если первое условие Если, а также все условия ИначеЕсли, находящиеся выше, оказались ложными и не выполнились.

Блок программного кода, следующий после слова Иначе, выполняется в том случае, если ни одно из предшествующих условий не было выполнено: ни Если, ни ИначеЕсли.

Пример:

```
Перем ПроверочнаяПеременная;

ВвестиЧисло(ПроверочнаяПеременная, "Введите число");
Если ПроверочнаяПеременная > 10 Тогда
    Сообщить(Истина);
Иначе
    Сообщить(Ложь);
КонецЕсли;
```

Пример, аналогичный разобранному для оператора `?`. Пользователь вводит число, после чего это число сравнивается с 10. Если число больше 10, выдается `Истина`, если меньше либо равно 10 — выдается `Ложь`.

Пример:

```
Перем ПервоеЧисло, ВтороеЧисло;

ВвестиЧисло(ПервоеЧисло, "Введите первое число");
ВвестиЧисло(ВтороеЧисло, "Введите второе число");

Если ПервоеЧисло > ВтороеЧисло  Тогда
    Сообщить("Первое число больше второго");
    Если ПервоеЧисло >= ВтороеЧисло*10 Тогда
        Сообщить("Причем намного больше");
    ИначеЕсли ПервоеЧисло >= ВтороеЧисло*3 Тогда
        Сообщить("Ощутимо больше");
    Иначе
        Сообщить("Менее чем в три раза");
    КонецЕсли;
ИначеЕсли ПервоеЧисло = ВтороеЧисло Тогда
    Сообщить("Первое число равно второму");
ИначеЕсли ПервоеЧисло < ВтороеЧисло Тогда
    Сообщить("Первое число меньше второго");
    Если ПервоеЧисло <= ВтороеЧисло*10 Тогда
        Сообщить("Причем намного меньше");
    ИначеЕсли ПервоеЧисло <= ВтороеЧисло*3 Тогда
        Сообщить("Ощутимо меньше");
    Иначе
        Сообщить("Менее чем в три раза");
    КонецЕсли;
Иначе
    //а без блока Иначе тут вообще можно обойтись
КонецЕсли;
```

А в этом примере реализовано сравнение двух чисел, вводимых пользователем. Сначала сравниваются два числа и выдается, что первое число больше второго, меньше второго или равно ему. В пределах условия, которое явилось истинным,

выполняется вложенное условие Если. Проверяется, насколько первое число больше или меньше второго. В зависимости от этого выводится соответствующее сообщение.

Такие вложенные друг в друга "матрешки" условий используются в практике программирования очень часто, причем их может быть и больше двух. В таких случаях для удобства чтения программного кода очень помогает форматирование кода. Для форматирования нужно выделить текст кода и нажать кнопку  на панели **Текст** или комбинацию клавиш <Alt>+<Shift>+<F>.

Пример:

```
Перем Номер;

ВвестиЧисло(Номер, "Введите номер дня недели");
Если Номер=1 Тогда
   Сообщить ("понедельник");
ИначеЕсли Номер=2 Тогда
   Сообщить ("вторник");
ИначеЕсли Номер=3 Тогда
   Сообщить ("среда");
ИначеЕсли Номер=4 Тогда
   Сообщить ("четверг");
ИначеЕсли Номер=5 Тогда
   Сообщить ("пятница");
ИначеЕсли Номер=6 Тогда
   Сообщить ("суббота");
ИначеЕсли Номер=7 Тогда
   Сообщить ("воскресенье");
Иначе
   Сообщить ("В неделе семь дней!");
КонецЕсли;
Если Номер=6 ИЛИ Номер=7 Тогда
  Сообщить ("это выходной день");
КонецЕсли;
```

В этом примере используется булева операция или. Пользователь вводит номер дня недели, в зависимости от этого ему сообщается название дня недели.

Вторым условием проверяется, является ли введенный номер дня выходным, а также, не выходит ли номер дня за пределы допустимо возможных 7 дней.

Чтобы данное условие выполнилось, нужно чтобы номер дня был или 6 или 7.

Если в условии используется несколько булевых операций, они должны разделяться скобками, например, так, как в следующем примере.

Пример:

```
Перем Номер, СубботникВоскресник, Ответ;

ВвестиЧисло(Номер, "Введите номер дня недели");
```

```
Если Номер=1 Тогда
    Сообщить ("понедельник");
ИначеЕсли Номер=2 Тогда
    Сообщить ("вторник");
ИначеЕсли Номер=3 Тогда
    Сообщить ("среда");
ИначеЕсли Номер=4 Тогда
    Сообщить ("четверг");
ИначеЕсли Номер=5 Тогда
    Сообщить ("пятница");
ИначеЕсли Номер=6 Тогда
    Сообщить ("суббота");
ИначеЕсли Номер=7 Тогда
    Сообщить ("воскресенье");
Иначе
    Сообщить ("В неделе семь дней!");
КонецЕсли;

Если Номер=6 ИЛИ Номер=7  Тогда
    Сообщить ("это выходной день");

    Ответ=Вопрос("Сегодня уборка территории?",РежимДиалогаВопрос.ДаНет);
    Если Ответ = КодВозвратаДиалога.Да Тогда
        СубботникВоскресник = "Да";
    ИначеЕсли Ответ = КодВозвратаДиалога.Нет Тогда
        СубботникВоскресник = "Нет";
    КонецЕсли;
КонецЕсли;

Если (Номер=6 ИЛИ Номер=7) И (СубботникВоскресник = "Да")  Тогда
    Сообщить ("вернее БЫЛ БЫ выходной день, но мы выходим на уборку
территории...");
КонецЕсли;
```

Этот пример отличается от предыдущего дополнительными условиями в случае, если день выходной. `Если (Номер=6 ИЛИ Номер=7) И (СубботникВоскресник = "Да")`, тогда выдается дополнительное сообщение. Такие условия с булевыми операциями `И`, `ИЛИ` и `НЕ` должны отделяться друг от друга скобками.

> ***Примечание***
>
> Если подходить строго, то в последнем примере условие
>
> `"Если (Номер=6 ИЛИ Номер=7) И (СубботникВоскресник = "Да") Тогда"`
>
> можно было не использовать. Достаточно было
>
> `"Если СубботникВоскресник = "Да" Тогда",`
> `ведь переменная СубботникВоскресник определялась там, где условие`
> `"Если Номер=6 ИЛИ Номер=7" выполняется.`
>
> Но поскольку это учебный пример, то дополнительное условие приведено для наглядности разделения булевых операций.

# Циклы

Операторы циклов предназначены для циклического выполнения заданного программного кода, пока выполняются или пока не выполнятся определенные условия.

## Оператор *Для*

Этот оператор циклически выполняет программный код до тех пор, пока значение переменной не превысит заданных граничных условий.

Синтаксис:

```
Для ИмяПеременной = Выражение1 По Выражение2 Цикл
  ... // программный код
  [Продолжить;]
  ... // программный код
  [Прервать;]
  ... // программный код
КонецЦикла;
```

Программный код внутри данной языковой конструкции будет циклически выполняться до тех пор, пока переменная *ИмяПеременной* больше либо равно *Выражение1*, но меньше либо равно *Выражение2*.

Если внутри цикла встречается команда `Продолжить`, то это означает, что в этом месте выполнение цикла будет остановлено и передано на "новый виток" цикла, т. е. операторы, находящиеся ниже команды `Продолжить`, выполнены не будут.

Если внутри цикла встречается команда `Прервать`, это означает, что как только выполнение дойдет до этой команды, работа цикла будет прекращена.

Пример:

```
Перем Счетчик;
Для Счетчик = 1 По 100 Цикл
    Сообщить("Значение счетчика = "+Счетчик);
КонецЦикла;
```

В данном примере перебираются элементы от 1 до 100, текущее значение присваивается переменной `Счетчик` и выводится в виде сообщения. Цикл выполняется "по кругу" до тех пор, пока значение переменной `Счетчик` не дойдет до 100.

Пример:

```
Перем Счетчик;
Для Счетчик = 1 По 100 Цикл
    Если Счетчик > 81 Тогда
        Прервать;
    КонецЕсли;
    Сообщить("Значение счетчика = "+Счетчик);
КонецЦикла;
```

Тот же самый цикл, но выполнение его должно прерваться, когда значение счетчика превысит 81. Когда `Счетчик` будет равняться 82, сработает прерывание цикла. Сообщение о том, что `Счетчик = 82`, выведено уже не будет.

## Оператор *Для каждого*

Данный оператор предназначен для циклического обхода коллекции значений.

Синтаксис:

```
Для каждого Переменная из КоллекцияЗначений Цикл
  ... // программный код
  [Продолжить;]
  ... // программный код
  [Прервать;]
  ... // программный код
КонецЦикла;
```

В этой языковой конструкции осуществляется циклический обход заданной коллекции значений (подборка данных, строки таблицы, какой-то список и т. д.). *Переменная* является счетчиком цикла.

Пример:

```
НДС = 1.2;
Для каждого Стр ИЗ СписокПлатежей Цикл
  Стр.СуммаСНДС = Стр.СуммаБезНДС * НДС;
КонецЦикла;
```

В данном примере перебираются строки таблицы `СписокПлатежей`, в которую внесены суммы платежей без НДС и высчитываются суммы платежей с НДС.

## Оператор *Пока*

Данный оператор предназначен для циклического повторения операторов, находящихся внутри языковой конструкции `Цикл...КонецЦикла`. Цикл выполняется, пока логическое выражение равно `Истина`.

Синтаксис:

```
Пока Выражение Цикл
  ... // программный код
  [Продолжить;]
  ... // программный код
  [Прервать;]
  ... // программный код
КонецЦикла;
```

Программный код внутри данной языковой конструкции будет циклически выполняться до тех пор, пока *Выражение* будет истинно.

Если внутри цикла встречается команда Продолжить, то это означает, что в этом месте выполнение цикла будет остановлено и передано на "новый виток" цикла, т. е. операторы, находящиеся ниже команды Продолжить, выполнены не будут.

Если внутри цикла встречается команда Прервать, это означает, что как только выполнение дойдет до этой команды, работа цикла будет прекращена.

Пример:

```
Счетчик = 1;
Пока Счетчик <= 100 Цикл
     Сообщить("Значение счетчика = "+Счетчик);
     Счетчик = Счетчик + 1;
КонецЦикла;
```

Тот же самый пример с перебором от 1 до 100, который мы рассматривали для оператора Для, но выполненный с помощью оператора Пока. Обратите внимание, что в отличие от оператора Для счетчик при каждом проходе цикла увеличивается не автоматически, а прибавлением единицы.

***А КАК МОЖНО РЕАЛИЗОВАТЬ В ЦИКЛЕ УБЫВАЮЩИЙ СЧЕТЧИК?***

```
НачЗначение = 10;
КонЗначение = 1;
Счетчик = НачЗначение;
Пока Счетчик >= КонЗначение Цикл
    Счетчик = Счетчик — 1;
КонецЦикла;
```

# Массивы

*Массив* — это пронумерованная коллекция значений произвольного типа. К элементу массива можно обращаться по его индексу. Представьте, например, школьную тетрадь "в клеточку". В таком случае тетрадь можно рассматривать как массив, а значение, записанное в каждую из клеточек, — как элемент массива. У каждой такой клеточки есть номер, называемый *индексом*. Массивы бывают одномерные и многомерные (состоящие из нескольких коллекций значений).

Для работы с массивами существуют различные программные методы, которые мы сейчас рассмотрим.

- Добавить. Добавляет элемент в конец массива.

  Синтаксис:

  **Добавить([*Значение*]);**

  Пример:

  ```
  НашМассив = Новый Массив();
  НашМассив.Добавить("Новый элемент");
  НашМассив.Добавить("Еще один элемент");
  Сообщить(НашМассив[0]);
  ```

В этом примере мы создаем новый массив командой Новый, затем командой Добавить последовательно добавляем к массиву два значения строкового типа. После этого мы даем команду сообщить первый элемент массива. Обратите внимание, нумерация массива начинается с нуля. Поэтому, чтобы обратиться к значению "Наш новый элемент", мы должны обращаться к элементу НашМассив[0]. Номер элемента массива указывается в квадратных скобках [].

- Вставить. Вставляет значение в массив по указанному индексу.

  Синтаксис:

  **Вставить(*Индекс*[, *Значение*]);**

  Пример:

```
НашМассив = Новый Массив();
НашМассив.Добавить("Новый элемент");
НашМассив.Добавить("Еще один элемент");
НашМассив.Вставить(0, "Измененный элемент");
Сообщить(НашМассив[0]);
```

  А здесь мы создаем новый массив, добавляем в него элементы, а потом значение одного из элементов меняем на новое.

- Количество. Определяет количество элементов в массиве.

  Синтаксис:

  **Количество();**

  Пример:

```
НашМассив = Новый Массив();
НашМассив.Добавить("Новый элемент");
НашМассив.Добавить("Еще один элемент");
Сообщить(НашМассив.Количество());
```

  При выполнении этот программный код выдаст результат 2.

- Найти. Выполняет поиск элемента в массиве.

  Синтаксис:

  **Найти([*Значение*]);**

  Пример:

```
НашМассив = Новый Массив();
НашМассив.Добавить("Новый элемент");
НашМассив.Добавить("Еще один элемент");
НашМассив.Добавить("Третий элемент");
Если НашМассив.Найти("Еще один элемент") = 1 Тогда
    Сообщить("Элемент найден");
Иначе
    Сообщить("Нет такого элемента!");
КонецЕсли;
```

  При выполнении этот программный код выдаст результат "Элемент найден".

- `Удалить`. Удаляет из массива значение с указанным индексом.

  Синтаксис:

  **`Удалить(Индекс);`**

  Пример:

```
НашМассив = Новый Массив();
НашМассив.Добавить("Новый элемент");
НашМассив.Добавить("Еще один элемент");
НашМассив.Удалить[1];
```

- `Очистить`. Удаляет все значения из массива.

  Синтаксис:

  **`Очистить();`**

  Пример:

```
НашМассив.Очистить();
```

- `ВГраница`. Получает наибольший индекс массива.

  Синтаксис:

  **`ВГраница();`**

  Пример:

```
НашМассив = Новый Массив();
НашМассив.Добавить("Новый элемент");
НашМассив.Добавить("Еще один элемент");
Сообщить(НашМассив.ВГраница);
```

  При выполнении этот программный код выдаст результат `1` (максимальный из имеющихся двух элементов с индексами 0 и 1).

***А КАК СОЗДАТЬ МАССИВ И ПРОГРАММНО ЗАПОЛНИТЬ ЕГО ДАННЫМИ?***

```
СозданныйМассив = Новый Массив (10);
Для Счетчик = 0 По 9 Цикл
    СозданныйМассив.Вставить(Счетчик, Счетчик);
// Вставляем в ячейку массива с номером Счетчик
// числовое значение Счетчик,
// т. е., например, в СозданныйМассив[3] будет записана цифра 3
КонецЦикла;
```

***А КАК ТЕПЕРЬ ПЕРЕБРАТЬ ЭТОТ МАССИВ И ПОКАЗАТЬ, ЧТО В НЕМ ХРАНИТСЯ?***

```
Для Каждого ЭлементМассива Из СозданныйМассив Цикл
    Сообщить(ЭлементМассива);
КонецЦикла;
// результат: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
```

***А КАК СОЗДАТЬ КОПИЮ МАССИВА СО ВСЕМИ ДАННЫМИ?***

```
СтарыйМассив = Новый Массив (10);
Для Счетчик = 0 По 9 Цикл
    СтарыйМассив.Вставить(Счетчик, Счетчик);
КонецЦикла;

Для Каждого ЭлементМассива Из СтарыйМассив Цикл
    Сообщить(ЭлементМассива);
КонецЦикла; // Выводим список элементов старого массива

МассивКопия = Новый Массив;
Для Каждого Элемент Из СтарыйМассив Цикл
    МассивКопия.Добавить(Элемент);
КонецЦикла;

Для Каждого ЭлементМассива Из МассивКопия Цикл
    Сообщить(ЭлементМассива);
КонецЦикла; // Выводим список элементов нового массива
// и убеждаемся, что они равны элементам старого массива
```

# Список значений

*Список значений* — это несохраняемый в базе данных объект, который позволяет строить динамически изменяемые наборы значений и манипулировать ими (добавлять, редактировать, сортировать и удалять элементы). Ограничений на типы хранимых значений нет, т. е. в одном списке значений могут храниться данные разных типов.

Далее мы рассмотрим основные методы работы со списками значений.

- `Добавить`. Добавляет новый элемент в конец списка значений.

  Синтаксис:

  **`Добавить([`*`Значение`*`][, `*`Представление`*`][, `*`Пометка`*`][, `*`Картинка`*`]);`**

  Здесь: *`Значение`* — значение, добавляемое в список значений; *`Представление`* — строковое представление значения; *`Пометка`* — определяет, добавляется ли значение с меткой или нет; *`Картинка`* — визуальное представление значения.

  Все эти параметры не являются обязательным, даже *`Значение`*.

  Пример:

  ```
  СписокПокупок = Новый СписокЗначений;
  СписокПокупок.Добавить("Хлеб");
  СписокПокупок.Добавить("Мясо");
  СписокПокупок.Добавить("Молоко");
  СписокПокупок.Добавить("Масло");
  ```

- `Вставить`. Вставляет новый элемент в список значений на позицию с указанным индексом.

Синтаксис:

```
Вставить(Индекс[, Значение][, Представление][, Пометка][, Картинка]);
```

Здесь: *Индекс* — номер индекса, куда будем вставлять значение; *Значение* — значение, добавляемое в список значений; *Представление* — строковое представление значения; *Пометка* — определяет, добавляется ли значение с меткой или нет; *Картинка* — визуальное представление значения.

Из этих параметров полностью обязательным к использованию является только *Индекс*.

- Количество. Получает количество элементов списка значений.

  Синтаксис:

  ```
  Количество();
  ```

  Пример:

  ```
  СписокПокупок = Новый СписокЗначений;
  СписокПокупок.Добавить("Хлеб");
  СписокПокупок.Добавить("Мясо");
  СписокПокупок.Добавить("Молоко");
  СписокПокупок.Добавить("Масло");
  Сообщить(СписокПокупок.Количество());
  ```

  При выполнении этот программный код выдаст результат 4.

- Удалить. Удаляет элемент из списка значений.

  Синтаксис:

  ```
  Удалить(Индекс);
  ```

  Пример:

  ```
  СписокПокупок = Новый СписокЗначений;
  СписокПокупок.Добавить("Хлеб");   // индекс [0]
  СписокПокупок.Добавить("Мясо");   // индекс [1]
  СписокПокупок.Добавить("Молоко"); // индекс [2]
  СписокПокупок.Добавить("Масло");  // индекс [3]
  СписокПокупок.Удалить("3");       // масло
  ```

- Очистить. Очищает список значений, удаляя из него все элементы.

  Синтаксис:

  ```
  Очистить();
  ```

- НайтиПоЗначению. Осуществляет поиск значения элемента списка значений.

  Синтаксис:

  ```
  НайтиПоЗначению(Значение);
  ```

- СортироватьПоЗначению. Сортирует список значений в порядке возрастания или убывания хранимых элементами значений.

Синтаксис:

**СортироватьПоЗначению(*Направление*);**

Пример:

```
СписокПокупок = Новый СписокЗначений;
СписокПокупок.Добавить("Хлеб");
СписокПокупок.Добавить("Мясо");
СписокПокупок.Добавить("Молоко");
СписокПокупок.Добавить("Масло");
СписокПокупок.СортироватьПоЗначению(НаправлениеСортировки.Возр);
// Сортировка по возрастанию, при сортировке по убыванию было бы
// СписокПокупок.СортироватьПоЗначению(НаправлениеСортировки.Убыв);
```

- `ВыбратьЭлемент`. Вызывает окно для интерактивного выбора одного из элементов, входящих в список значений.

Синтаксис:

**ВыбратьЭлемент([*Заголовок*][, *Элемент*]);**

Здесь: *Заголовок* — заголовок окна выбора; *Элемент* — элемент, на котором изначально позиционируется список значений при выборе.

Пример:

```
Перем Переменная;
СписокПокупок = Новый СписокЗначений;
  СписокПокупок.Добавить("Хлеб");
  СписокПокупок.Добавить("Мясо");
  СписокПокупок.Добавить("Молоко");
  СписокПокупок.Добавить("Масло");
СписокПокупок.СортироватьПоЗначению(НаправлениеСортировки.Возр);
Выбор = СписокПокупок.ВыбратьЭлемент("Выберите элемент списка", Переменная);
Сообщить(Выбор);
```

При выполнении этого программного кода будет выведен список значений, как на рис. 2.9.

Выбранное значение будет присвоено переменной `Выбор` и выведено в сообщении.

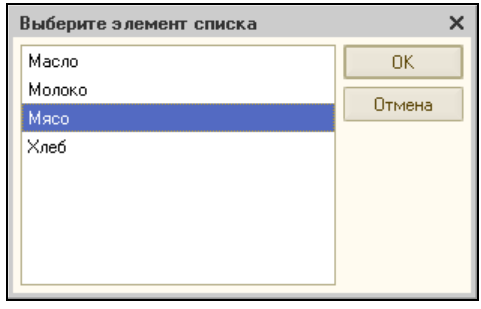



**Рис. 2.9.** Выбор из списка значений

***А КАК ПРОГРАММНО СОЗДАТЬ СПИСОК ЗНАЧЕНИЙ И НАПОЛНИТЬ ЕГО ЗНАЧЕНИЯМИ?***

```
Товары = Новый СписокЗначений;
Товары.Добавить("Лимонад 1 л");
Товары.Добавить("Вода минеральная 1 л");
Товары.Добавить("Квас 2 л");
Товары.Добавить("Вода минеральная 0,5 л");
Товары.Добавить("Вода питьевая очищенная 5 л");
```

***А КАК ТЕПЕРЬ НАЙТИ НУЖНЫЙ ЭЛЕМЕНТ В СПИСКЕ ЗНАЧЕНИЙ И ВЫВЕСТИ ЕГО ИНДЕКС?***

```
ИскомыйЭлемент = Товары.НайтиПоЗначению("Вода минеральная 0,5 л");
Если ИскомыйЭлемент <> Неопределено Тогда
    Сообщить(Товары.Индекс(ИскомыйЭлемент));
КонецЕсли;
```

# Таблица значений

*Таблица значений* — это несохраняемый в базе данных объект, который позволяет строить динамически изменяемые наборы значений и манипулировать ими (добавлять, редактировать, сортировать и удалять элементы). Ограничений на типы хранимых значений нет, т. е. в одном списке значений могут храниться данные разных типов. В отличие от списка значений, в каждой строке которого записано одно значение (по сути, таблица из одного столбца), таблица значений имеет большое количество строк и столбцов.

Далее мы рассмотрим основные методы работы с таблицами значений.

- `Добавить`. Добавляет строку в конец таблицы значений.

  Синтаксис:

  ```
  Добавить();
  ```

  Пример:

```
ТабЗн = Новый ТаблицаЗначений;
ТабЗн.Колонки.Добавить("Товар");
ТабЗн.Колонки.Добавить("Вес");

НоваяСтрока = ТабЗн.Добавить();
НоваяСтрока.Товар = "Макароны весовые";
НоваяСтрока.Вес = 3;

НоваяСтрока = ТабЗн.Добавить();
НоваяСтрока.Товар = "Рис круглый";
НоваяСтрока. Вес = 5;

НоваяСтрока = ТабЗн.Добавить();
НоваяСтрока.Товар = "Крупа гречневая";
НоваяСтрока. Вес = 7;
```

```
Выбор = ТабЗн.ВыбратьСтроку("Выберите строку таблицы значений");
Сообщить(Выбор.Товар);
```

В этом примере мы создали новую таблицу значений, описали для нее два столбца — "Товар" и "Вес", а затем командой `Добавить()` добавили три новых строки и заполнили их значениями.

Далее, аналогично тому, что мы это делали со списком значений, мы выбираем строчку таблицы значений и выводим значение поля "Товар" для выбранной строки.

- `Вставить`. Вставляет новый элемент в таблицу значений на позицию с указанным индексом.

Синтаксис:

```
Вставить(Индекс);
```

Здесь `Индекс` — номер индекса, по которому будем вставлять строку.

Пример:

```
ТабЗн = Новый ТаблицаЗначений;
ТабЗн.Колонки.Добавить("Товар");
ТабЗн.Колонки.Добавить("Вес");

НоваяСтрока = ТабЗн.Добавить();
НоваяСтрока.Товар = "Макароны весовые";
НоваяСтрока.Вес = 3;

НоваяСтрока = ТабЗн.Добавить();
НоваяСтрока.Товар = "Рис круглый";
НоваяСтрока. Вес = 5;

НоваяСтрока = ТабЗн.Добавить();
НоваяСтрока.Товар = "Крупа гречневая";
НоваяСтрока. Вес = 7;

НоваяСтрока = ТабЗн.Вставить(2);
НоваяСтрока.Товар = "Крупа манная";
НоваяСтрока. Вес = 4;

Выбор = ТабЗн.ВыбратьСтроку("Выберите строку таблицы значений");
Сообщить(Выбор.Товар);
```

Если выполнить сначала программный код из предыдущего примера, а потом из текущего, то мы увидим, что в таблицу значений добавилась строка.

- `Количество`. Возвращает количество строк таблицы значений.

Синтаксис:

```
Количество();
```

Пример:

```
ТабЗн = Новый ТаблицаЗначений;
ТабЗн.Колонки.Добавить("Товар");
ТабЗн.Колонки.Добавить("Вес");

НоваяСтрока = ТабЗн.Добавить();
НоваяСтрока.Товар = "Макароны весовые";
НоваяСтрока.Вес = 3;

НоваяСтрока = ТабЗн.Добавить();
НоваяСтрока.Товар = "Рис круглый";
НоваяСтрока. Вес = 5;

НоваяСтрока = ТабЗн.Добавить();
НоваяСтрока.Товар = "Крупа гречневая";
НоваяСтрока. Вес = 7;

НоваяСтрока = ТабЗн.Вставить(2);
НоваяСтрока.Товар = "Крупа манная";
НоваяСтрока. Вес = 4;
Сообщить(ТабЗн.Количество());
```

При выполнении этот программный код выдаст результат `4`.

- `Удалить`. Удаляет строку таблицы значений.

  Синтаксис:

  ```
  Удалить(Строка);
  ```

  Здесь *Строка* — номер строки таблицы значений, которую необходимо удалить.

- `Очистить`. Удаляет все строки таблицы значений.

  Синтаксис:

  ```
  Очистить();
  ```

  Пример:

  ```
  ТабЗн.Очистить();
  ```

- `Найти`. Осуществляет поиск значения в указанных колонках таблицы значений.

  Синтаксис:

  ```
  Найти(Значение[, Колонки]);
  ```

  Здесь: *Значение* — искомое значение; *Колонки* — колонки, в которых осуществляется поиск. Если параметра *Колонки* нет — поиск производится во всей таблице значений.

- `Сортировать`. Сортирует таблицу значений в соответствии с указанными правилами сортировки.

Синтаксис:

```
Сортировать(Колонки);
```

Пример:

```
ТабЗн = Новый ТаблицаЗначений;
ТабЗн.Колонки.Добавить("Товар");
ТабЗн.Колонки.Добавить("Вес");

НоваяСтрока = ТабЗн.Добавить();
НоваяСтрока.Товар = "Макароны весовые";
НоваяСтрока.Вес = 3;

НоваяСтрока = ТабЗн.Добавить();
НоваяСтрока.Товар = "Рис круглый";
НоваяСтрока. Вес = 5;

НоваяСтрока = ТабЗн.Добавить();
НоваяСтрока.Товар = "Крупа гречневая";
НоваяСтрока. Вес = 7;

НоваяСтрока = ТабЗн.Вставить(2);
НоваяСтрока.Товар = "Крупа манная";
НоваяСтрока. Вес = 4;

ТабЗн.Сортировать("Вес Убыв");

Выбор = ТабЗн.ВыбратьСтроку("Выберите строку таблицы значений");
Сообщить(Выбор.Товар);
```

В этом примере мы сортируем таблицу значения по убыванию значений поля "Вес".

- `ВыбратьСтроку`. Вызывает модальное диалоговое окно для интерактивного выбора строки из таблицы значений.

  Синтаксис:

  ```
  ВыбратьСтроку([Заголовок][, НачальнаяСтрока]);
  ```

  Здесь: *Заголовок* — заголовок окна выбора; *НачальнаяСтрока* указывает строку таблицы значений, на которой будет установлен курсор при открытии окна выбора.

  Пример:

  ```
  Выбор = ТабЗн.ВыбратьСтроку("Выберите строку таблицы значений");
  ```

- `Свернуть`. Осуществляет сворачивание таблицы значений по указанным колонкам группировки. Строки, у которых совпадают значения в колонках, указанных в первом параметре, сворачиваются в одну строку. Значения этих строк, храня-

щиеся в колонках, указанных во втором параметре, суммируются. Списки колонок не должны пересекаться. Колонки, не вошедшие ни в один из списков колонок, после выполнения метода удаляются из таблицы значений.

Синтаксис:

**Свернуть(*КолонкиГруппировки*[, *КолонкиСуммирования*]);**

Здесь: *КолонкиГруппировки* — список группируемых колонок; *КолонкиСуммирования* — список суммируемых колонок.

Пример:

```
ТабЗн = Новый ТаблицаЗначений;
ТабЗн.Колонки.Добавить("Товар");
ТабЗн.Колонки.Добавить("ПроизведеноУпаковок");
ТабЗн.Колонки.Добавить("Брак");

НоваяСтрока = ТабЗн.Добавить();
НоваяСтрока.Товар = "Макароны весовые";
НоваяСтрока.ПроизведеноУпаковок = 10;
НоваяСтрока.Брак = 1;

НоваяСтрока = ТабЗн.Добавить();
НоваяСтрока.Товар = "Рис круглый";
НоваяСтрока.ПроизведеноУпаковок = 15;
НоваяСтрока.Брак = 2;

НоваяСтрока = ТабЗн.Добавить();
НоваяСтрока.Товар = "Крупа гречневая";
НоваяСтрока.ПроизведеноУпаковок = 33;
НоваяСтрока.Брак = 0;

НоваяСтрока = ТабЗн.Добавить();
НоваяСтрока.Товар = "Крупа манная";
НоваяСтрока.ПроизведеноУпаковок = 114;
НоваяСтрока.Брак = 7;

НоваяСтрока = ТабЗн.Добавить();
НоваяСтрока.Товар = "Рис круглый";
НоваяСтрока.ПроизведеноУпаковок = 24;
НоваяСтрока.Брак = 3;

НоваяСтрока = ТабЗн.Добавить();
НоваяСтрока.Товар = "Макароны весовые";
НоваяСтрока.ПроизведеноУпаковок = 81;
НоваяСтрока.Брак = 9;

ВыборДоСвертки = ТабЗн.ВыбратьСтроку("Выберите строку таблицы значений");
Сообщить("Товар: " + ВыборДоСвертки.Товар);
```

```
Сообщить("Упаковок: " + ВыборДоСвертки.ПроизведеноУпаковок);
Сообщить("Брак: " + ВыборДоСвертки.Брак);

ТабЗн.Свернуть("Товар","ПроизведеноУпаковок, Брак");
ТабЗн.Сортировать("ПроизведеноУпаковок Возр");

Выбор = ТабЗн.ВыбратьСтроку("Выберите строку таблицы значений");
Сообщить("Товар: " + Выбор.Товар);
Сообщить("Всего упаковок: " + Выбор.ПроизведеноУпаковок);
Сообщить("Всего брака: " + Выбор.Брак);
```

В этом примере мы записали в таблицу значений несколько гипотетических партий товара с указанием общего произведенного количества и количества брака по каждой партии. Некоторые товары указываются разными партиями и, соответственно, разными строками.

На рис. 2.10 показана таблица значений до сворачивания.

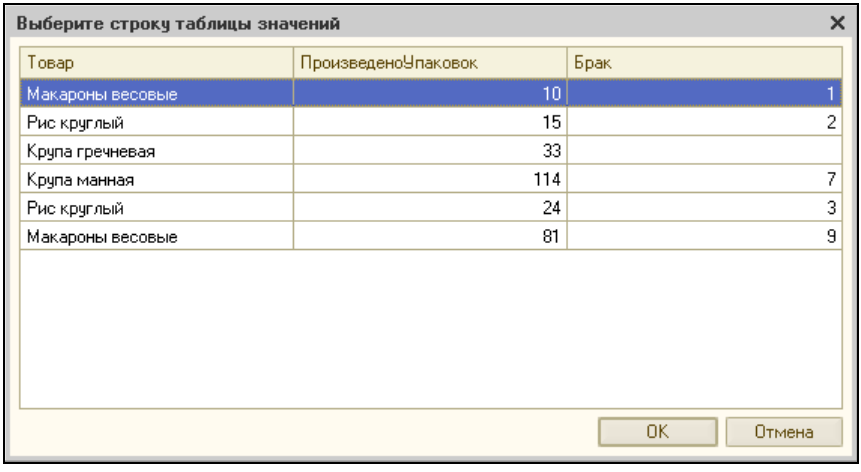



**Рис. 2.10.** Таблица значений до сворачивания

Исходная таблица значений была свернута и отсортирована, результат мы можем видеть на рис. 2.11.

Как видим, свертка — весьма удобный инструмент, который можно использовать для группировки данных и получения итогов.

- `Итог`. Суммирует значения всех строк в указанной колонке. Суммирование производится по числовым типам данных.

  Синтаксис:

  ***Итог(Колонка);***

  Здесь *Колонка* — колонка таблицы значений, по которой производится суммирование.

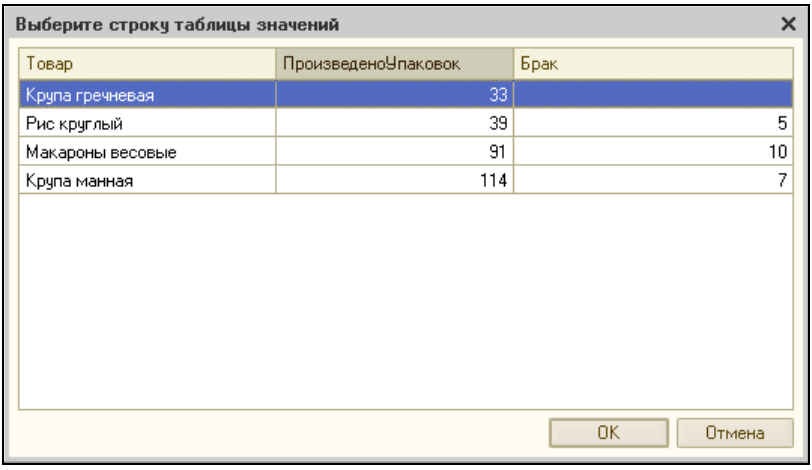



**Рис. 2.11.** Таблица значений после сворачивания и сортировки

Пример:

```
Сообщить(ТабЗн.Итог("Брак"));
```

Если добавить эту строку в предшествующий пример сворачивания данных в таблице значений, то получим общий итог по всей колонке "Брак".

### *А КАК ПРОГРАММНО СОЗДАТЬ ТАБЛИЦУ ЗНАЧЕНИЙ И НАПОЛНИТЬ ЕЕ ДАННЫМИ?*

```
ИнвентаризационнаяВедомость = Новый ТаблицаЗначений;

// Создаем колонки таблицы.
ИнвентаризационнаяВедомость.Колонки.Добавить("Наименование");
ИнвентаризационнаяВедомость.Колонки.Добавить("Остаток");

// Добавляем новую строку со значениями.
НоваяСтрока = ИнвентаризационнаяВедомость.Добавить();
НоваяСтрока.Наименование = "Лимонад 1 л";
НоваяСтрока.Остаток = 200;

// Добавляем новую строку и задаем значения в колонках таблицы.
НоваяСтрока = ИнвентаризационнаяВедомость.Добавить();
НоваяСтрока.Наименование = "Вода минеральная 1 л";
НоваяСтрока.Остаток = 250;

// Добавляем новую строку со значениями.
НоваяСтрока = ИнвентаризационнаяВедомость.Добавить();
НоваяСтрока.Наименование = "Квас 2 л";
НоваяСтрока.Остаток = 87;

// Добавляем новую строку со значениями.
НоваяСтрока = ИнвентаризационнаяВедомость.Добавить();
```

```
НоваяСтрока.Наименование = "Вода минеральная 0,5 л";
НоваяСтрока.Остаток = 740;

// Добавляем новую строку со значениями.
НоваяСтрока = ИнвентаризационнаяВедомость.Добавить();
НоваяСтрока.Наименование = "Вода питьевая очищенная 5 л";
НоваяСтрока.Остаток = 124;
```

#### *А КАК ПРОСМОТРЕТЬ ЗАПИСИ ТАБЛИЦЫ ЗНАЧЕНИЙ?*

```
Для Каждого ТекущаяСтрока Из ИнвентаризационнаяВедомость Цикл
    Сообщить(ТекущаяСтрока.Наименование);
    Сообщить(ТекущаяСтрока.Остаток);
    Сообщить("----------------------------------");
КонецЦикла;
```

#### *А КАК УДАЛИТЬ СТРОКУ ТАБЛИЦЫ ЗНАЧЕНИЙ ПО НОМЕРУ?*

```
ИнвентаризационнаяВедомость.Удалить(3);
// Строка № 3. Так как нумерация начинается с нуля,
// то для приведенного выше примера это "Вода минеральная 0,5 л"
```

#### *А КАК УДАЛИТЬ СТРОКУ КОЛОНКУ ЗНАЧЕНИЙ ПО НОМЕРУ?*

```
ИнвентаризационнаяВедомость.Колонки.Удалить(1);
```

#### *А КАК ЗАПОЛНИТЬ ВСЮ КОЛОНКУ ТАБЛИЦЫ ЗНАЧЕНИЙ НУЖНЫМ ЗНАЧЕНИЕМ?*

```
ИнвентаризационнаяВедомость.ЗаполнитьЗначения(0, "Остаток");
// Обнулили остатки в инвентаризационной ведомости
```

# Исключительные ситуации. Оператор *Попытка*

Оператор `Попытка` управляет выполнением программы, основываясь на возникающих при выполнении модуля ошибочных (исключительных) ситуациях, и задает действия, которые должны быть выполнены при возникновении подобной ситуации. При ошибочной ситуации выполнение программы может быть прервано с ошибкой. Во избежание таких ситуаций и используется языковая конструкция `Попытка...Исключение`.

Синтаксис:

```
Попытка
   ... // операторы попытки
Исключение
   ... // операторы исключения
КонецПопытки;
```

Пример:

```
Перем ПервоеЧисло, ВтороеЧисло, ТретьеЧисло;
ВвестиЧисло(ПервоеЧисло, "Введите первое число");
ВвестиЧисло(ВтороеЧисло, "Введите второе число");
Попытка
    ТретьеЧисло = ПервоеЧисло/ВтороеЧисло;
    Сообщить("ПервоеЧисло/ВтороеЧисло = "+ТретьеЧисло);
Исключение
    Сообщить("На 0 делить нельзя!");
КонецПопытки;
```

В этом примере реализован расчет деления одного из введенных пользователем чисел на другое. Оператор исключительной ситуации нужен здесь, чтобы исключить деление на ноль, в противном случае выполнение программы было бы прервано с ошибкой.

# Работа с файлами

Язык программирования системы "1С:Предприятие" позволяет также работать с файловой системой Windows. Далее мы рассмотрим основные из этих команд.

- `СоздатьКаталог`. Создает новый каталог (папку).

  Синтаксис:

  **СоздатьКаталог(*ИмяКаталога*);**

  Пример:

  ```
  СоздатьКаталог("D:\Работа");
  ```

  Создает каталог Работа на диске D:.

- `НайтиФайлы`. По заданной маске осуществляет поиск файлов и каталогов, расположенных в заданном каталоге.

  Синтаксис:

  **НайтиФайлы(*Путь*[, *Маска*][, *ИскатьВПодкаталогах*]);**

  Параметр *ИскатьВПодкаталогах* имеет тип "Булево". Если он принимает значение `Истина`, то поиск производится также во вложенных подкаталогах текущего каталога.

  Пример:

  ```
  Найдено = НайтиФайлы("D:\Работа", "*.txt");
  ```

- `КопироватьФайл`. Копирует файл-источник в файл-приемник.

  Синтаксис:

  **КопироватьФайл(*ИмяФайлаИсточника*, *ИмяФайлаПриемника*);**

  Пример:

  ```
  КопироватьФайл("D:\Работа\test.txt", "C:\test.txt");
  ```

Пример:

```
КопироватьФайл("D:\Работа\test.txt", "D:\Работа\тестовый.txt");
```

В этом примере в каталоге исходного файла создается его копия с другим именем.

- `ПереместитьФайл`. Выполняет перемещение указанного файла из адреса-источника по адресу-приемнику.

  Синтаксис:

  ```
  ПереместитьФайл(ИмяФайлаИсточника, ИмяФайлаПриемника);
  ```

  Пример:

  ```
  ПереместитьФайл("D:\Работа\test.txt", "D:\test.txt");
  ```

- `ПолучитьФайл`. Получает файл и сохраняет его в файловой системе пользователя.

  Синтаксис:

  ```
  ПолучитьФайл(Адрес[, ИмяФайла][, Интерактивно]);
  ```

- `РазделитьФайл`. Разделяет указанный файл на несколько частей (файлов) заданного размера. Имя каждой части образуется из имени исходного файла с прибавлением порядкового номера.

  Синтаксис:

  ```
  РазделитьФайл(ИмяФайла, РазмерЧасти[, Путь]);
  ```

  Здесь: *РазмерЧасти* исчисляется в байтах; *Путь* — путь к каталогу, в котором должны быть размещены получившиеся файлы. Если путь не указан, то они будут размещены в одном каталоге с исходным файлом.

  Пример:

  ```
  РазделитьФайл("D:\Работа\test.txt", 1024*2);
  ```

  В каталоге D:\Работа имеется файл test.txt объемом 12 Кбайт. Программный код данного примера делит его на части по 2 Кбайт ((1024 байт = 1 Кбайт) × 2). В итоге создаются шесть файлов с именами test.txt.1, test.txt.2, test.txt.3, test.txt.4, test.txt.5 и test.txt.6.

- `ОбъединитьФайлы`. Объединяет несколько файлов (частей) в один файл.

  Синтаксис:

  ```
  ОбъединитьФайлы(Шаблон, ИмяРезультирующегоФайла);
  ```

  Пример:

  ```
  ОбъединитьФайлы("D:\Работа\test.txt.*", "D:\Работа\собранный.txt");
  ```

  А этот пример противоположен предыдущему. Из полученных в этом примере шести файлов мы опять собираем один, используя в шаблоне символ `*` (любое количество любых символов). В результате получим файл собранный.txt размером 6 × 2 Кбайт = 12 Кбайт.

Обратите внимание на объем нового файла. Если у вас он получился в два раза больше, чем суммарный размер шести исходных файлов, это значит, что вы не удаляли из каталога файл test.txt из предыдущего примера. А ведь он тоже подпадает под шаблон `test.txt.*` и объединяется вместе с шестью остальными файлами.

- `УдалитьФайлы`. Удаляет заданные файлы.

  Синтаксис:

  ```
  УдалитьФайлы(Путь[, Маска]);
  ```

  Пример:

  ```
  УдалитьФайлы ("D:\Работа");
  ```

  Этот программный код удаляет наш каталог D:\Работа вместе со всем содержимым.

***А КАК ЗАПРОГРАММИРОВАТЬ ДИАЛОГ ВЫБОРА ФАЙЛА ПОЛЬЗОВАТЕЛЕМ?***

```
Режим = РежимДиалогаВыбораФайла.Открытие;
ДиалогОткрытияФайла = Новый ДиалогВыбораФайла(Режим);
ДиалогОткрытияФайла.ПолноеИмяФайла = "";
Фильтр = "Текст(*,txt)|*.txt";
ДиалогОткрытияФайла.Фильтр = Фильтр;
ДиалогОткрытияФайла.МножественныйВыбор = Ложь;
ДиалогОткрытияФайла.Заголовок = "Выберите текстовый файл";

Если ДиалогОткрытияФайла.Выбрать() Тогда
    // т. е. если пользователь выбрал файл
    ПутьКФайлу = ДиалогОткрытияФайла.ПолноеИмяФайла;
    Сообщить("Выбран файл "+ПутьКФайлу);
КонецЕсли;
```

# Справочная система и синтаксис-помощник

Кроме встроенного языка программирования система "1С:Предприятие" также располагает справочной системой, посвященной языку программирования и среде разработки.

Вызывается справочная система через пункт меню **Справка** в режиме Конфигуратора.

Окно справки выглядит так, как показано на рис. 2.12.

В верхней части окна расположена панель навигации, в самом тексте справки находятся ссылки на связанную информацию. По ссылкам можно переходить аналогично тому, как это делается в браузере при просмотре интернет-страниц, кнопками **Вперед** и **Назад**.

Также режим Конфигуратора располагает бесценным помощником программиста — синтаксис-помощником. В данной главе мы уже неоднократно упоминали этот полезный инструмент и видели, как он выглядит на рис. 2.6.

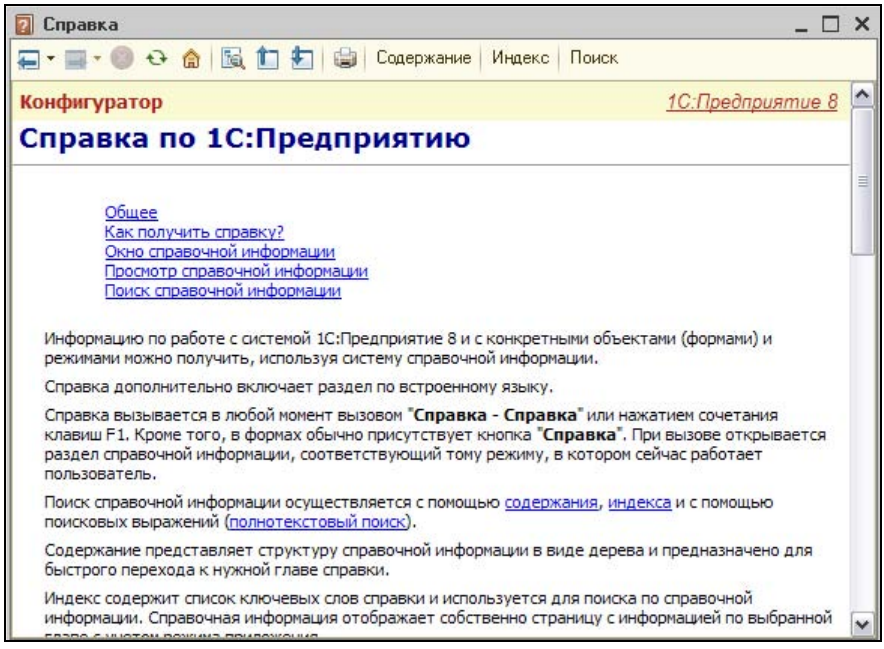



**Рис. 2.12.** Окно справки

Запускается синтаксис-помощник через пункт меню **Справка | Синтаксис-помощник**.

Также, если вы забыли синтаксис какой-либо команды и хотите себя проверить — всегда можно написать команду в программном коде модуля, выделить ее мышью, щелкнуть правой кнопкой мыши и выбрать пункт контекстного меню **Поиск в Синтаксис-Помощнике**. В этом случае синтаксис-помощник будет открыт именно на интересующей вас команде или, если таких команд несколько, будет выведен список, из которого нужно сделать выбор. Например, при поиске оператора Цикл будет выведен список из трех различных вариантов (рис. 2.13).

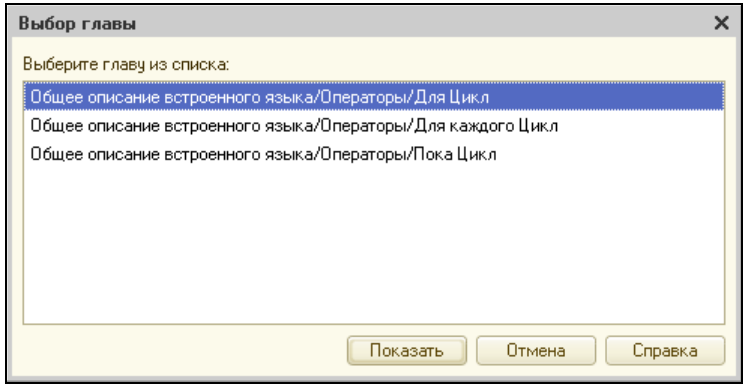



**Рис. 2.13.** Выбор нужного раздела в синтаксис-помощнике

Для многих команд список гораздо обширнее, например для команды `Выбрать`.

Выбираем нужный раздел, нажимаем кнопку **Показать**, и синтаксис-помощник сразу открывается на интересующем нас разделе. Иногда при этом надо также найти местонахождение раздела в общем дереве команд. Автоматическое позиционирование не происходит, поэтому находим раздел в дереве команд кнопкой  синтаксис-помощника. На рис. 2.14 показан синтаксис-помощник после нажатия этой кнопки.

В верхней части окна расположено дерево команд встроенного языка, а в нижней части — описание выбранного оператора.

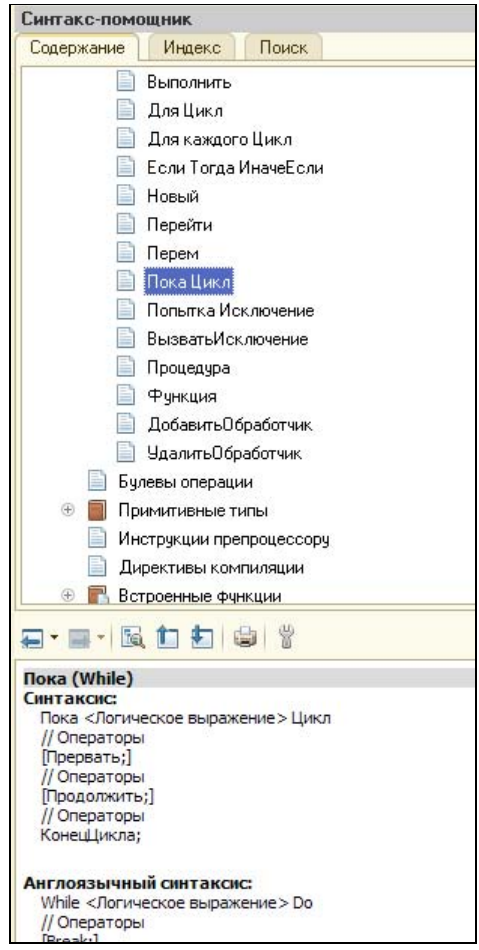



**Рис. 2.14.** Синтаксис-помощник

ГЛАВА 3

# Работа с прикладными объектами. Справочники, документы, регистры и проводки

В предыдущей главе мы рассмотрели наиболее распространенные операторы и языковые конструкции языка программирования системы "1С:Предприятие 8.2". Теперь осталось применить полученные знания на практике, каковой является разработка и доработка конфигураций "1С:Предприятие". Ведь одно дело, когда мы осуществляем выборку каких-либо абстрактных чисел из списка в учебной обработке, другое дело, когда, используя те же самые методы, мы работаем с реальными записями в работающей базе данных предприятия. В этой главе мы закрепим полученные знания, применив их на практике, а также углубим их, изучив свойства и типовые команды работы с прикладными объектами дерева конфигурации.

## Создание нового справочника

*Справочники* — это средство для работы со списками однородных элементов данных. При помощи справочников организуется ввод стандартной информации в документы, ее просмотр и изменение. Обычно справочниками являются списки товаров, организаций, валют, сотрудников и др. Для наглядности справочники можно сравнить с классической бумажной картотекой, где каждая карточка — это отдельный товар, покупатель, сотрудник и т. п.

Типовая конфигурация, такая как "Управление торговлей" или "Бухгалтерия", уже содержит в себе большое количество различных справочников. Наша задача — изучить свойства этих справочников и программные методы работы с ними, а также научиться добавлять в конфигурацию новые справочники и модифицировать имеющиеся.

Сейчас давайте добавим в нашу учебную конфигурацию новый справочник. Предполагаем, что конфигурация не пуста, а представляет собой типовую конфигурацию "1С:Предприятие 8.2", такую как "Управление торговлей". Если у вас нет конфигурации "Управление торговлей", то для экспериментов подойдет любая типовая конфигурация. Тогда при изучении примеров, приведенных в книге, делайте по-

правку на это, если вам встретятся некоторые различия в объектах вашей конфигурации и конфигурации, которая используется в качестве учебной.

На предприятии, которое ведет учет в "1С:Управление торговлей 8.2", развита система грузоперевозок заказчикам. Необходимо вести справочник автомобилей, рассчитывать, какая машина какую поставку осуществляет, как далеко и с каким расходом топлива, а также кто из водителей будет управлять автомобилем (а значит, нам нужно также вести учет сотрудников, которые работают водителями).

Начнем со справочника "Автомобили". Для его создания щелкнем правой кнопкой мыши по пункту **Справочники** дерева конфигураций и в контекстном меню выберем пункт **Добавить**.

Откроется окно создания нового справочника (рис. 3.1).

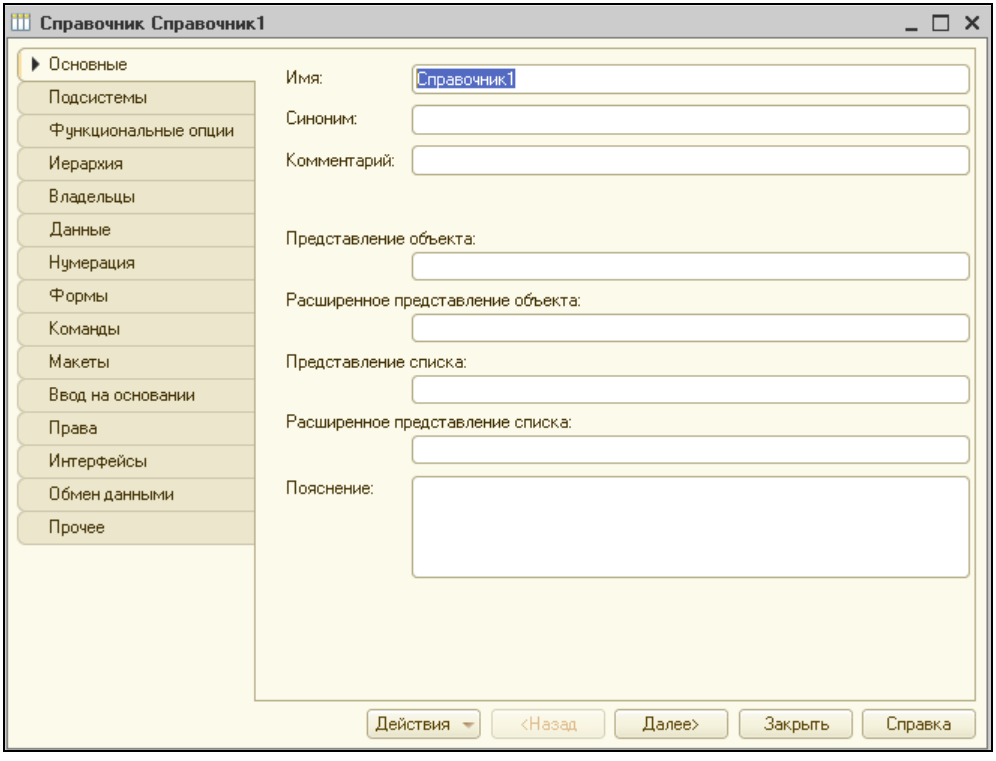



**Рис. 3.1.** Создаем новый справочник

Все свойства справочника, которые мы будем задавать, сгруппированы в левой части окна в виде вкладок. При нажатии на какую-либо вкладку в окне конструирования справочника открывается раздел, соответствующий этой вкладке. При создании справочника мы попадаем в самый первый раздел — **Основные**. Здесь мы задаем **Имя** справочника (как мы будем обращаться к нему в дальнейшем), **Синоним** (отображаемый вид имени, делающий его более читаемым для пользователя, все-таки привычнее видеть в формах названия, подобные "Статьи движения денежных

средств", чем "СтатьиДенежныхСредств") и **Комментарий** с кратким описанием справочника.

Заполните форму так, как показано на рис. 3.2.

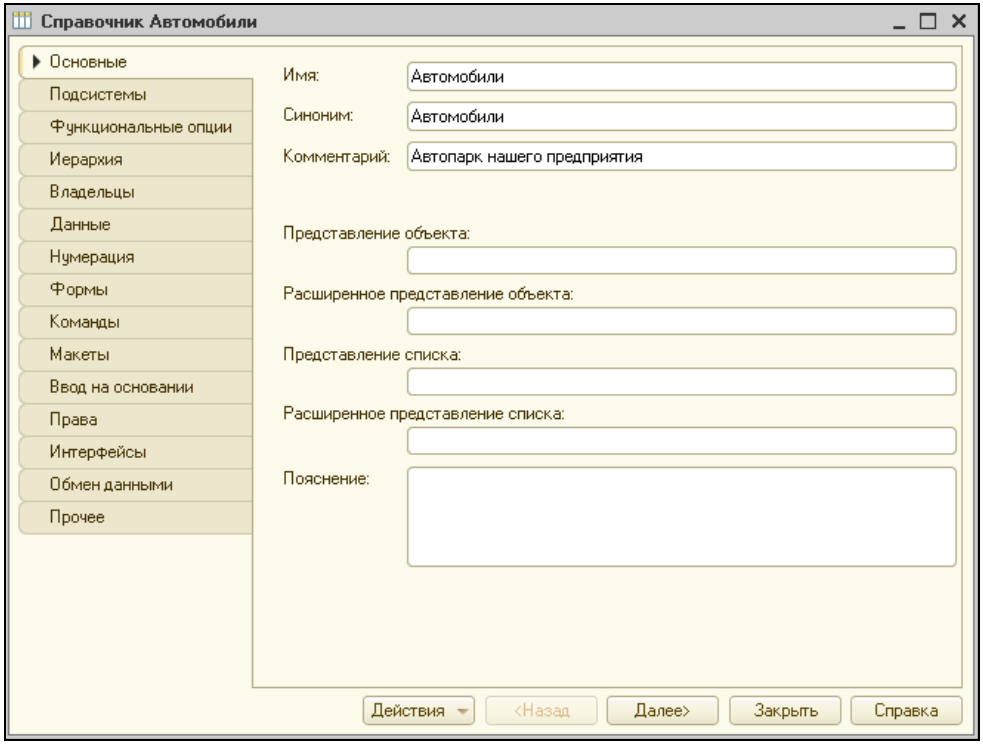



**Рис. 3.2.** Новый справочник "Автомобили". Вкладка **Основные**

Для перехода между разделами справочника используйте кнопку **Далее** или просто выберите мышью интересующую вас вкладку-раздел.

Сейчас нас интересует раздел **Подсистемы**. Для чего используются подсистемы в "1С:Предприятие"?

Представьте себе ведение учета в крупной промышленно-торговой фирме. Как много различных специалистов будет работать с базой данных этой фирмы? Производственники, управленцы, персонал, занимающийся продажами, бухгалтеры, кладовщики... Нужно ли им в работе использовать весь функционал конфигурации? Очевидно, что нет. Менеджер по продажам будет использовать отчеты по продажам и движению товаров, но оборотно-сальдовую ведомость — вряд ли. С другой стороны, для бухгалтера оборотно-сальдовая ведомость — один из главнейших инструментов. Документы, связанные с производством, вряд ли понадобятся зав. складом, который работает со складскими документами, ведомость об инвентаризации на розничном складе вряд ли заинтересуют начальника производства.

В общем, если каждому работнику выводить весь функционал конфигурации — это будет, как минимум, неудобно. Как максимум, не любую информацию необходимо

видеть каждому сотруднику. Поэтому и относят элементы метаданных к различным подсистемам. Иначе говоря, подсистему можно трактовать как набор элементов интерфейса. Например, в конфигурации "Управление торговлей" среди прочих существуют подсистемы УправлениеЗакупками, УправлениеЗапасами и УправлениеЗатратами, к которым относятся только те объекты, которые необходимы для соответствующего им рода деятельности. Чтобы посмотреть, какие элементы входят в ту или иную подсистему, нужно сделать на этой подсистеме двойной щелчок левой кнопкой мыши и перейти на вкладку **Состав**. Открывшееся окно будет выглядеть так, как показано на рис. 3.3.

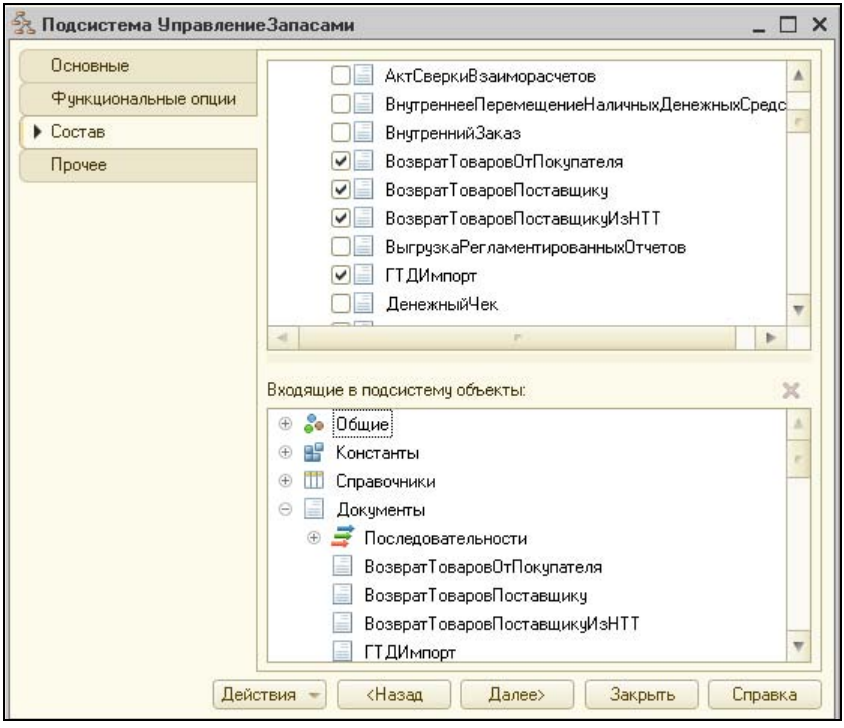



**Рис. 3.3.** Состав подсистемы УправлениеЗапасами конфигурации "Управление торговлей"

В правой части окна находится дерево конфигурации, в которой отмечаются входящие в подсистему объекты, а ниже — то же дерево, но только из одних входящих в подсистему объектов.

Вернемся к нашему справочнику "Автомобили". Определим, что использоваться он будет в подсистемах УправлениеЗакупками, УправлениеЗапасами и УправлениеПродажами, т. е. как раз пользователями, которые отвечают за товарный учет. Пользователям, которые занимаются, скажем, налоговым учетом (и имеют доступ к элементам соответствующей подсистемы), работать с нашим справочником вряд ли понадобится.

Окно подсистем для справочника "Автомобили" будет выглядеть так, как показано на рис. 3.4.

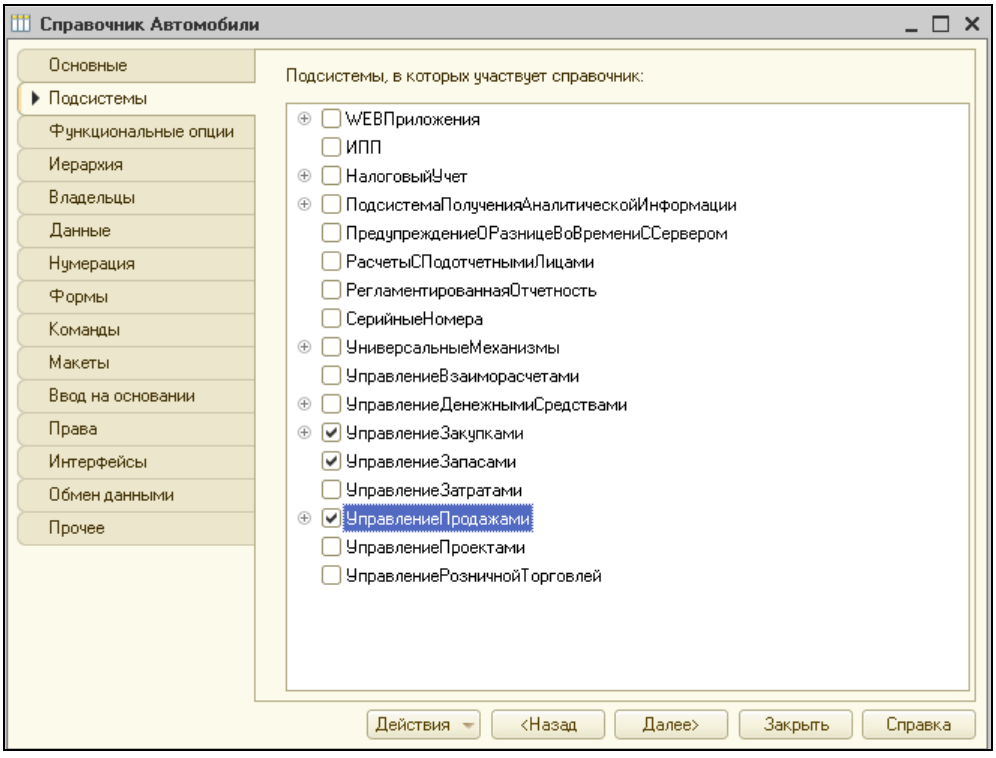



**Рис. 3.4.** Новый справочник "Автомобили". Определяем, в каких подсистемах доступен

Теперь перейдем на вкладку **Данные**, чтобы определить список реквизитов справочника. В сложных справочниках с большим объемом хранящейся информации мы бы еще заполнили вкладку **Иерархия**. Например, контрагенты могут быть разбиты на группы "Продавцы" и "Покупатели", товары могут быть разбиты на группы "Товар", "Услуга". Наш справочник "Автомобили" не настолько объемен, и особого смысла делать его иерархическим нет. При желании читатель может проделать это самостоятельно — выбрать опцию **Иерархический справочник** и указать количество вложенных подуровней справочника (уровней иерархии), а впоследствии разбить автомобили предприятия на легковые и грузовые или по маркам автомобилей.

На вкладке **Данные** мы определяем собственно состав хранящейся в справочнике информации (рис. 3.5).

Каждый справочник, даже при отсутствии реквизитов, уже имеет два реквизита по умолчанию. Это код и наименование. В верхней части окна мы определяем максимально возможную длину кода и наименования, какого типа будет код справочника (тип "Строка" или "Число") и основное представление справочника. Основное представление задает, по какому реквизиту элемент справочника будет идентифицирован и как будет отображаться. Возможно, пользователю нужно, чтобы элемент в списках и отчетах имел вид "Автомобиль Hyundai Accent серебристый". А возможно, специфика работы предприятия делает более удобным отображения в виде

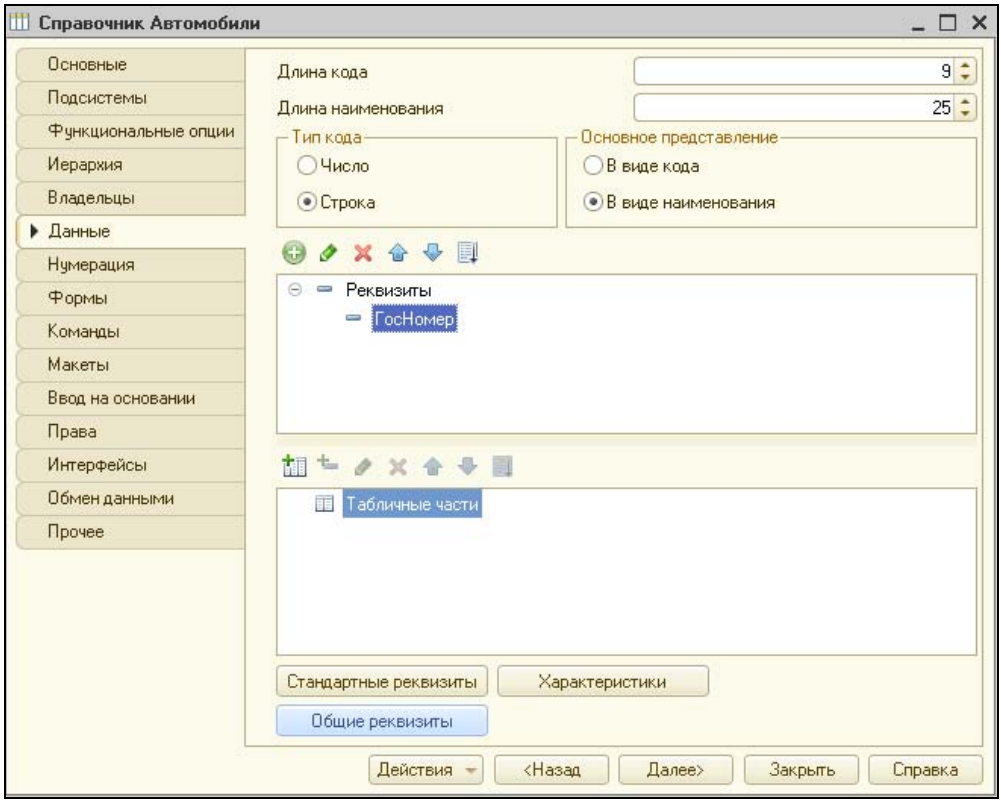



**Рис. 3.5.** Новый справочник "Автомобили". Добавляем новый реквизит

кода (где кодом может выступить номер). Обычно, все-таки отображают в виде наименования.

Ниже отображается список прочих реквизитов. Он пока пуст, на рис. 3.5 я создал только один реквизит **ГосНомер**. Для добавления нового реквизита щелкните правой кнопкой мыши по заголовку **Реквизиты** и в контекстном меню выберите пункт **Добавить** (он там единственный). Добавится новый реквизит. В правой части экрана — окно свойств реквизита (рис. 3.6).

Имя реквизита, по которому к нему будем обращаться впоследствии — **ГосНомер**, синоним подставляется автоматически, можем так и оставить. Поскольку в номере автомобиля могут присутствовать и буквы, и цифры — в качестве типа выбираем вариант **Строка**, указываем длину 12 символов. Свойство **Проверка заполнения** по умолчанию имеет значение **Не проверять**. Выбираем вариант **Выдавать ошибку**. Это делается для того, чтобы не создавали карточки автомобилей предприятия без указания гос. номера.

Аналогичным образом создайте остальные реквизиты справочника так, как показано на рис. 3.7.

В табл. 3.1 приведены имена реквизитов и типы их значений.

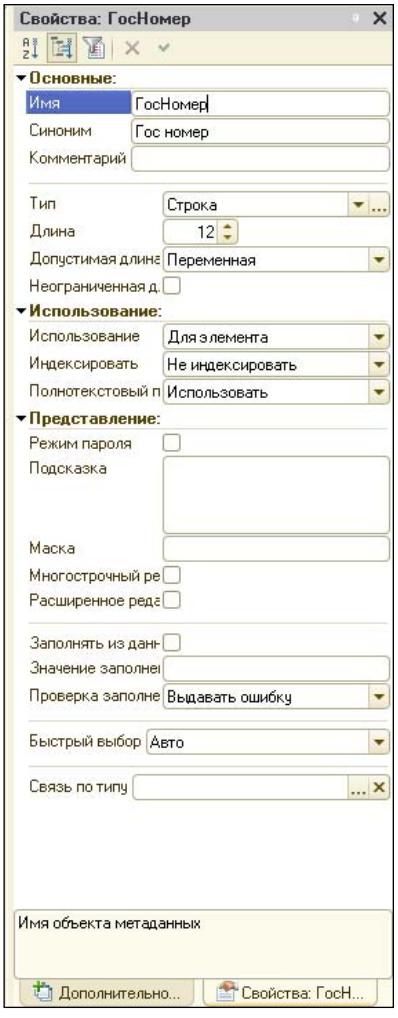



**Рис. 3.6.** Окно свойств реквизита

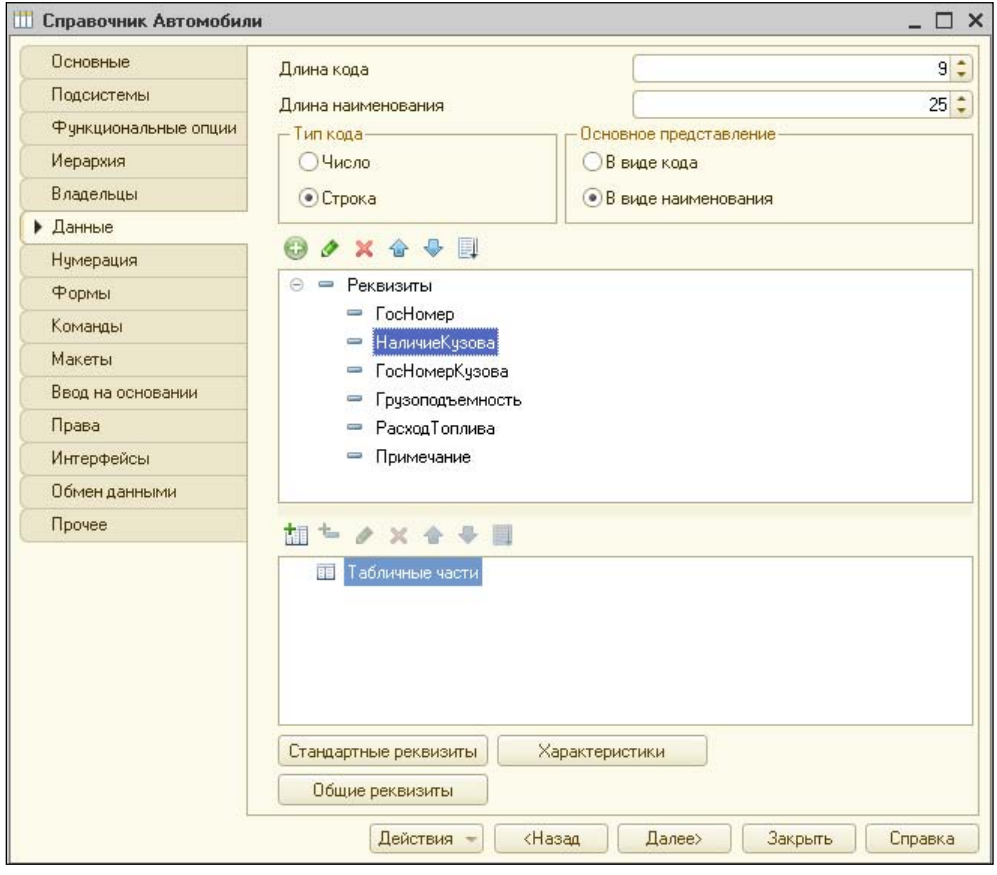



**Рис. 3.7.** Новый справочник "Автомобили". Список реквизитов

***Таблица. 3.1.*** *Список реквизитов справочника "Автомобили", их тип и длина*

| Имя реквизита | Тип реквизита | Длина/принимаемые значения |
|---|---|---|
| **ГосНомер** | Строка | 12 |
| **НаличиеКузова** | Булево | Истина/Ложь |
| **ГосНомерКузова** | Строка | 12 |
| **Грузоподъемность** | Число | 10.2 (10 длина целой части, 2 — длина дробной части) |
| **РасходТоплива** | Число | 8.2 (8 длина целой части, 2 — длина дробной части) |
| **Примечание** | Строка | 50 |

Не стоит задавать реквизитам излишне большую длину "на всякий случай, пусть будет", на больших объемах информации это даст существенный прирост объема базы данных, потери в скорости работы, "торможение" отчетов. Места под хранение данных следует отводить достаточно, но не избыточно. Да и какой нам смысл отводить под наименование 200—300 символов?

Ниже списка реквизитов в окне **Данные** расположен список табличных частей справочника. Да, в отличие от "1С:Предприятие 7.7" в "1С:Предприятие 8.2" у справочников могут быть табличные части. В нашем справочнике табличные части нам пока что не нужны, и мы их не создаем.

После создания нужных реквизитов переходим на вкладку **Нумерация** (рис. 3.8).

Здесь мы задаем правила нумерации новых элементов справочника. Будет ли нумерация автоматической, не будет ли повторяющихся номеров (контроль уникальности), каким образом будет проводиться нумерация: так называемая "сквозная" (т. е. по всему справочнику) или в пределах подчинения (т. е. для каждой группы элементов нумерация будет отдельная). При нумерации в пределах подчинения, скажем, для справочника товаров, нумерация в пределах каждой товарной группы будет вестись отдельно, т. е. в группе "Товары" может быть товар с кодом "000001" и в группе "Услуги" также может быть товар с кодом "000001", при этом контроль уникальности не нарушается, группы-то разные.

Обычно все же нумерация берется по всему справочнику, а для нашего случая этот вариант единственный, все равно у нас справочник не иерархический и групп не имеет.

Дальше нам нужно создать формы для справочника, задать, как будет выглядеть форма элемента (карточки) справочника, а как — список этих элементов. Для этого перейдем на вкладку **Формы** и добавим новую форму точно так же, как мы это делали с реквизитами (щелкнем правой кнопкой мыши по заголовку **Формы** в окне и выберем пункт **Добавить**). Откроется конструктор формы справочника (рис. 3.9).

Для начала мы создадим форму элемента справочника, поэтому можно оставить все по умолчанию так, как на рис. 3.9. По нажатию кнопки **Готово** откроется готовая

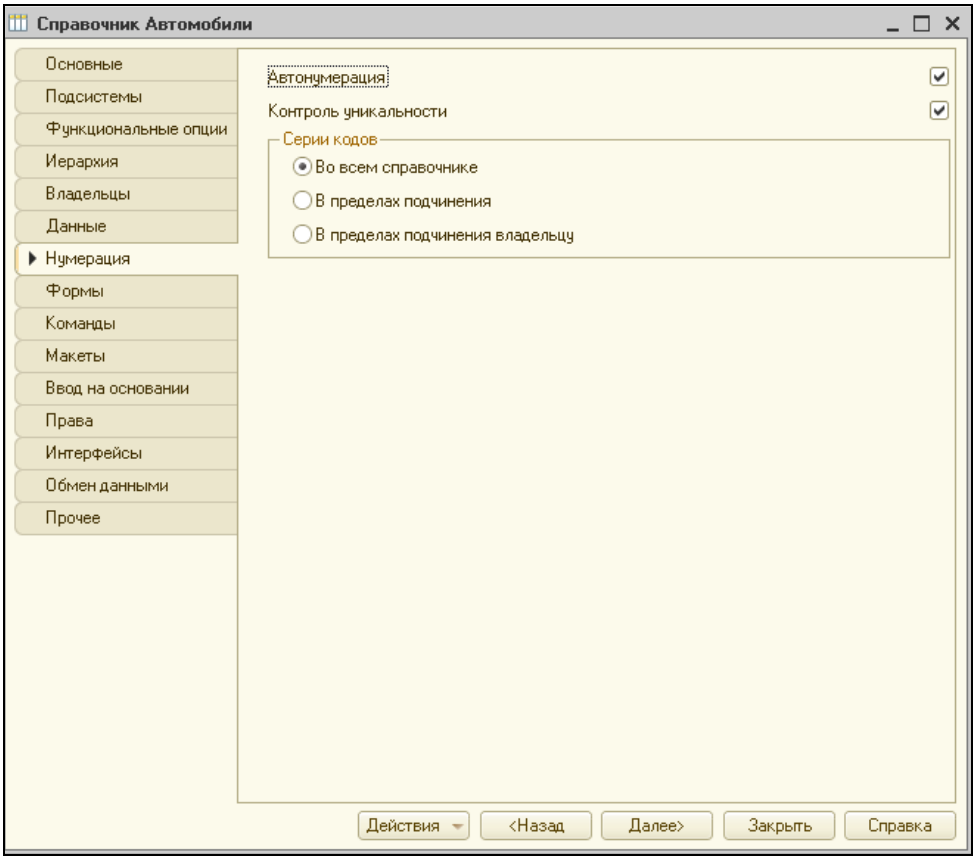



**Рис. 3.8.** Новый справочник "Автомобили". Нумерация и контроль уникальности

форма элемента, все поля и элементы управления в ней располагаются автоматически (рис. 3.10).

Если такая форма вас устроит — ее можно оставить, как есть. В противном случае элементы формы можно расположить по своему усмотрению, перетаскивая их мышью по форме, например, так, как показано на рис. 3.11.

При переносе элементов управления обратите внимание, что система помогает вам выравнивать элементы, показывая линии выравнивания по горизонтали и вертикали. Вообще, в "1С:Предприятие 8.2" не всегда получается расположить элементы быстро, правильно и красиво. Какие-то из элементов упорно не хотят выравниваться вместе с другими и при масштабировании формы (изменении ее размера мышью) не всегда все элементы вытягиваются пропорционально. Например, изображенная на рис. 3.11 форма вполне удобна в работе, но если в режиме Предприятия начать менять размер формы, растягивая или сужая ее, увидим, что по левому краю масштабируются все поля, а по правому — только поля **Наименование** и **Примечание**, выровненные по одной линии. Так что иногда приходится экспериментировать с расположением элементов.

Далее аналогичным способом мы создаем форму списка справочника (рис. 3.12).

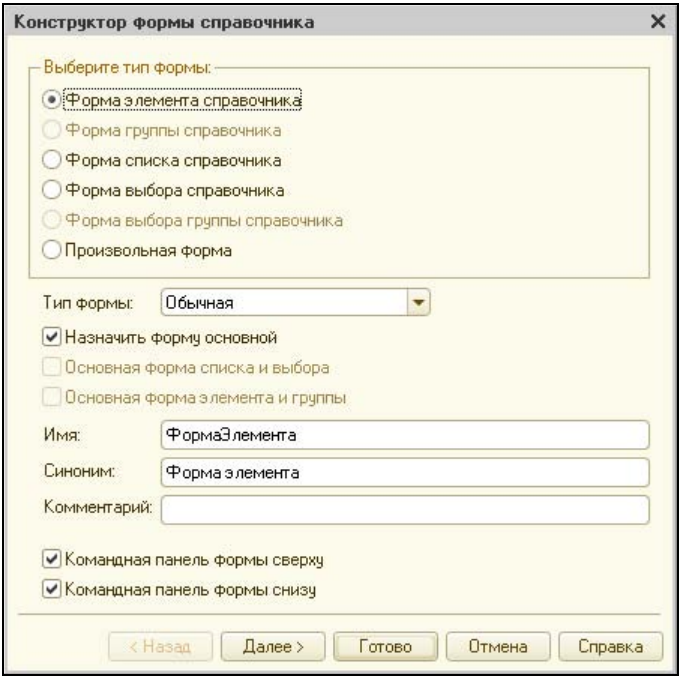



**Рис. 3.9.** Новый справочник "Автомобили". Добавляем форму элемента

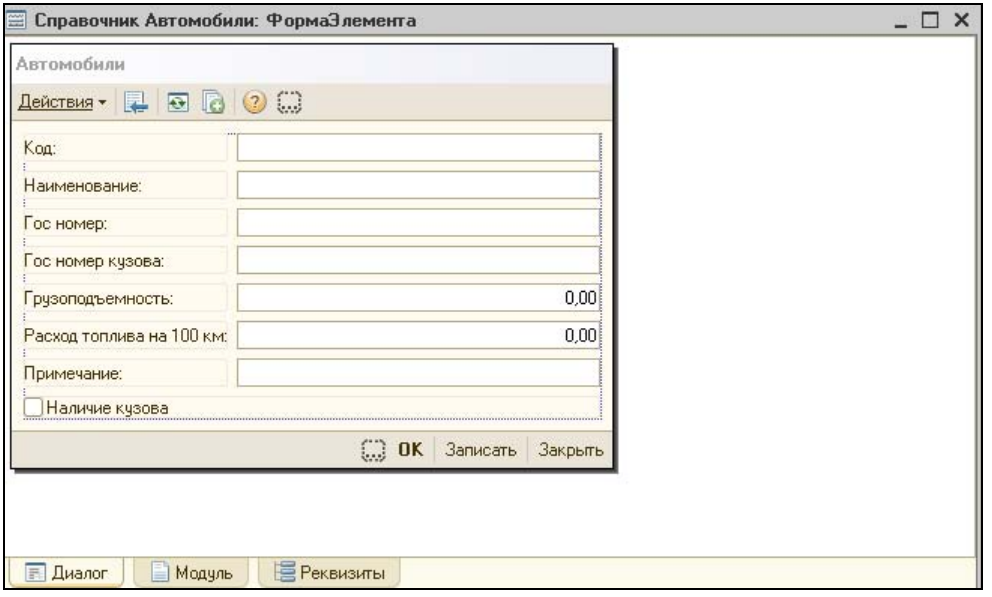



**Рис. 3.10.** Новый справочник "Автомобили". Автоматически созданная форма элемента

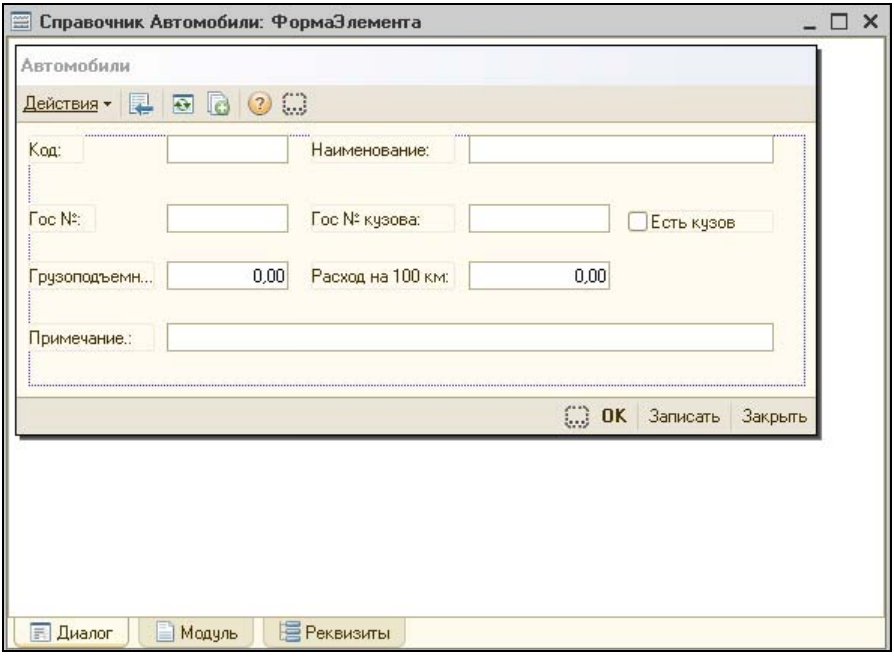



**Рис. 3.11.** Новый справочник "Автомобили". Та же форма с размещенными вручную элементами

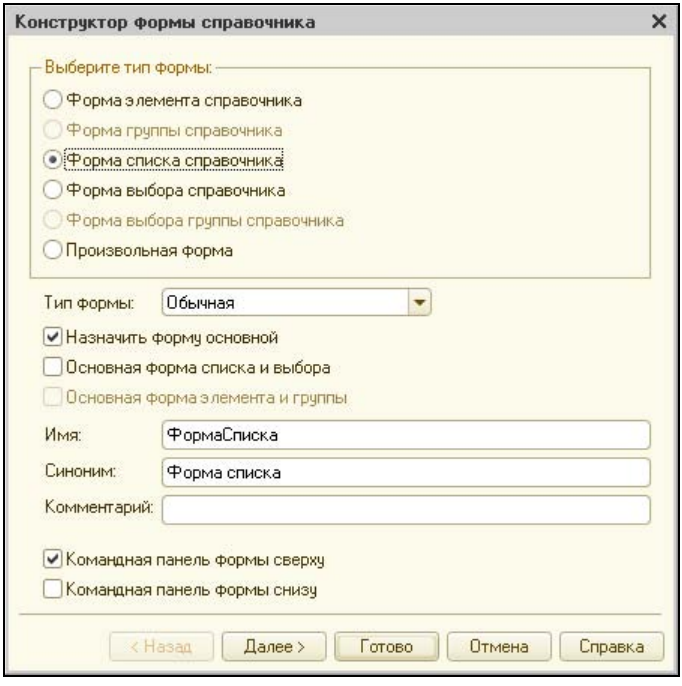



**Рис. 3.12.** Новый справочник "Автомобили". Добавляем форму списка

Все почти так же, как и при создании формы элемента, просто выбран другой тип. Оставим все, как показано на рисунке, и нажмем кнопку **Готово**. Форма списка формируется автоматически (рис. 3.13).

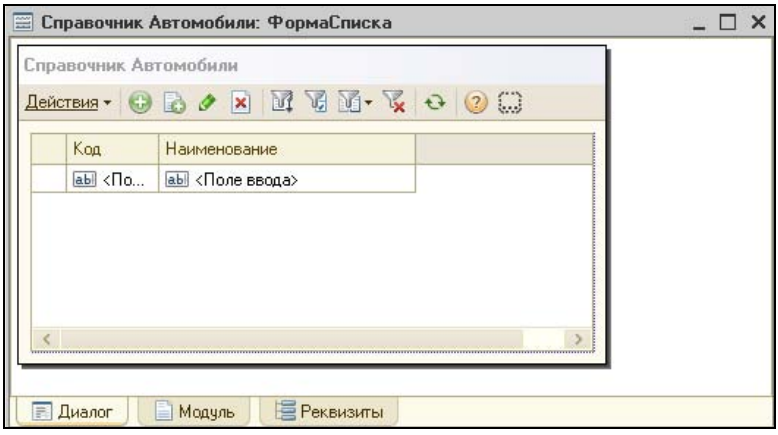



**Рис. 3.13.** Новый справочник "Автомобили". Автоматически созданная форма списка

Все бы хорошо, но автоматически показаны только поля **Код** и **Наименование**. Для того чтобы показать остальные реквизиты, нам нужно их добавить. Для этого щелкнем по табличной части правой кнопкой мыши и выберем в контекстном меню пункт **Добавить колонку**.

Откроется окно свойств новой колонки (рис. 3.14).

Мы должны присвоить колонке имя, обязательно выбрать, что будет отображаться в колонке, используя выпадающий список **Данные** (там будут показаны имена полей нашего справочника: **Код**, **Наименование**, **ГосНомер**, **РасходТоплива** и т. д.), задать текст шапки в одноименном свойстве колонки. Обратите внимание на флажки-свойства **Доступность** и **Видимость**. Они используются для всех визуальных элементов: полей ввода, кнопок, флажков, колонок таблиц и т. п. Если флажок **Доступность** снят, то элемент будет виден, но пользователь не сможет с ним выполнять никаких действий. Если же снят флажок **Видимость**, то элемент виден не будет. Разумеется, эти свойства можно устанавливать программно, что широко используется, показывая пользователю только тот объем данных, который ему нужно видеть. Программные методы работы с доступностью и видимостью элементов, как и другие, относящиеся к формам, мы рассмотрим далее в этой главе. Добавьте в список колонки для нашего справочника "Автомобили", за исключением поля **НаличиеКузова** (если кузов есть, мы увидим его номер и так). Результат должен выглядеть так, как показано на рис. 3.15.

Теперь у нашего справочника уже две формы: форма элемента и форма списка (рис. 3.16).

В форме элемента пользователь будет вносить данные по каждой новой записи справочника, а в форме списка сможет просматривать сразу все эти записи в том виде, который мы определили при конструировании формы.

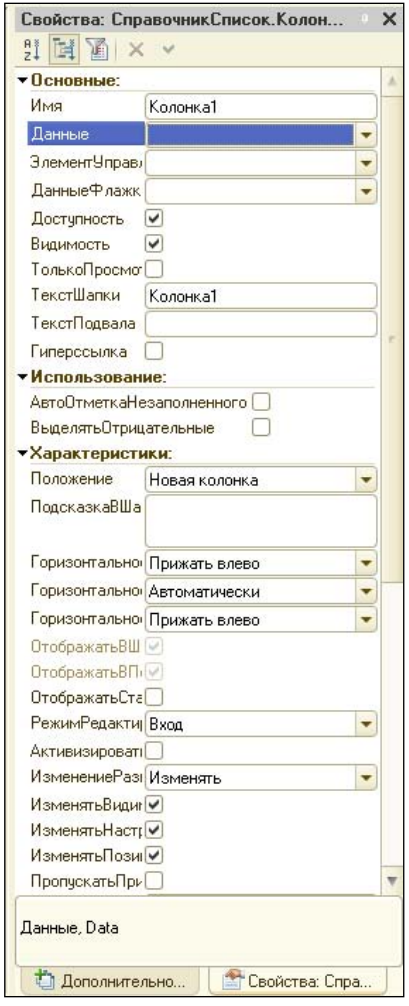



**Рис. 3.14.** Свойства новой колонки в форме списка

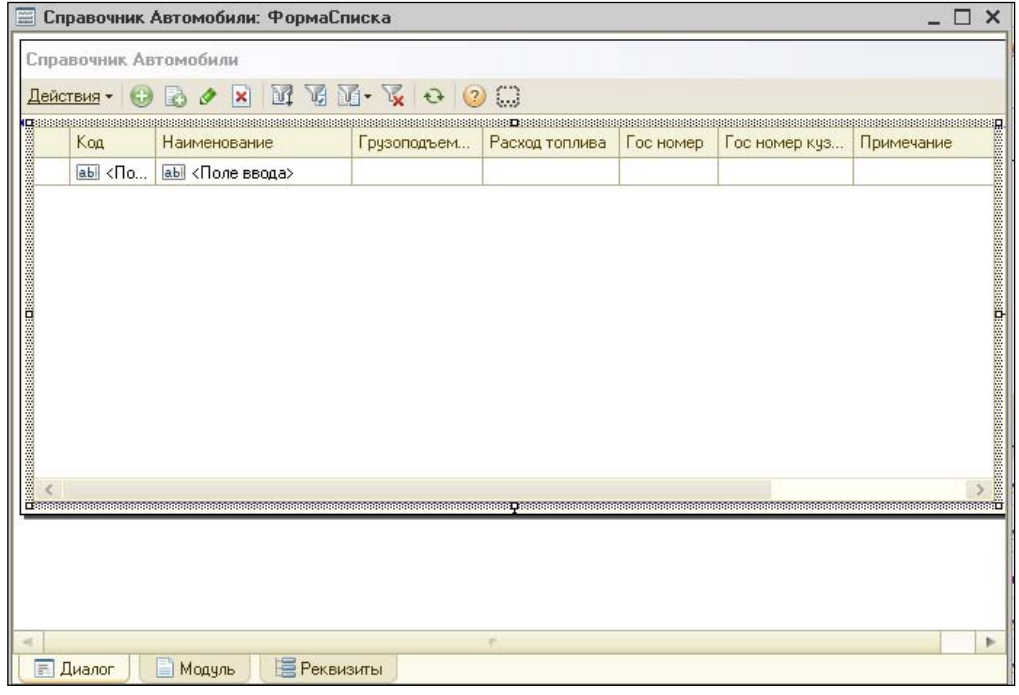



**Рис. 3.15.** Новый справочник "Автомобили". Форма списка, модифицированная вручную

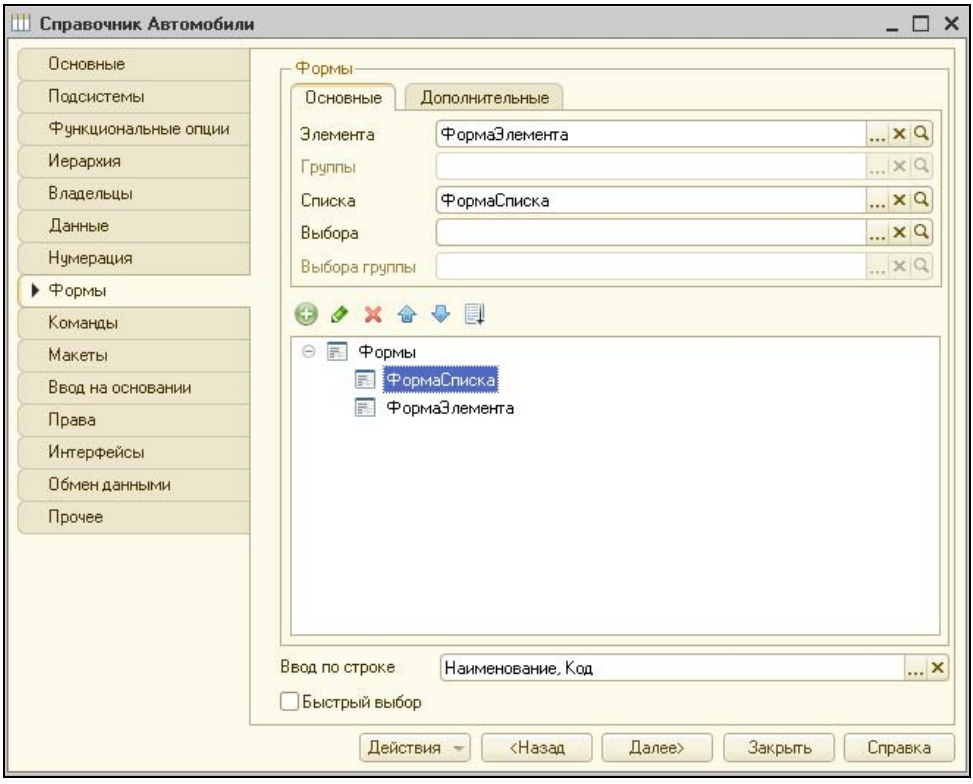



**Рис. 3.16.** Список форм справочника "Автомобили"

Следующая вкладка, которую мы будем настраивать, — **Права**. По названию уже понятно, что здесь пользователям задаются права на работу со справочником. Это выглядит так, как показано на рис. 3.17.

Если сказать точнее, то права задаются даже не пользователям, а *ролям пользователей*. Могут ведь быть пользователи Иванов, Петров, Сидоров, но роль **Администратор** или **МенеджерПоПродажам** может быть не обязательно только у одного из них.

В верхней части окна отображен список ролей. Выбирая каждую из них мышью, мы открываем в нижней части окна список действий, которые со справочником могут производить пользователи *выбранной в верхней части окна роли*. На рис. 3.17 видно, что пользователи роли **Кладовщик** могут видеть и просматривать записи справочника, добавлять новые и изменять имеющиеся записи, а вот удалять не могут.

Далее мы должны указать, в каком интерфейсе (интерфейс следует понимать, как совокупность кнопок, пунктов меню и прочих управляющих элементов, доступных пользователю) будет отображаться наш справочник. То есть, будет ли справочник "Автомобили" доступен в пользовательском меню того же **Кладовщика** или **Бухгалтера**.

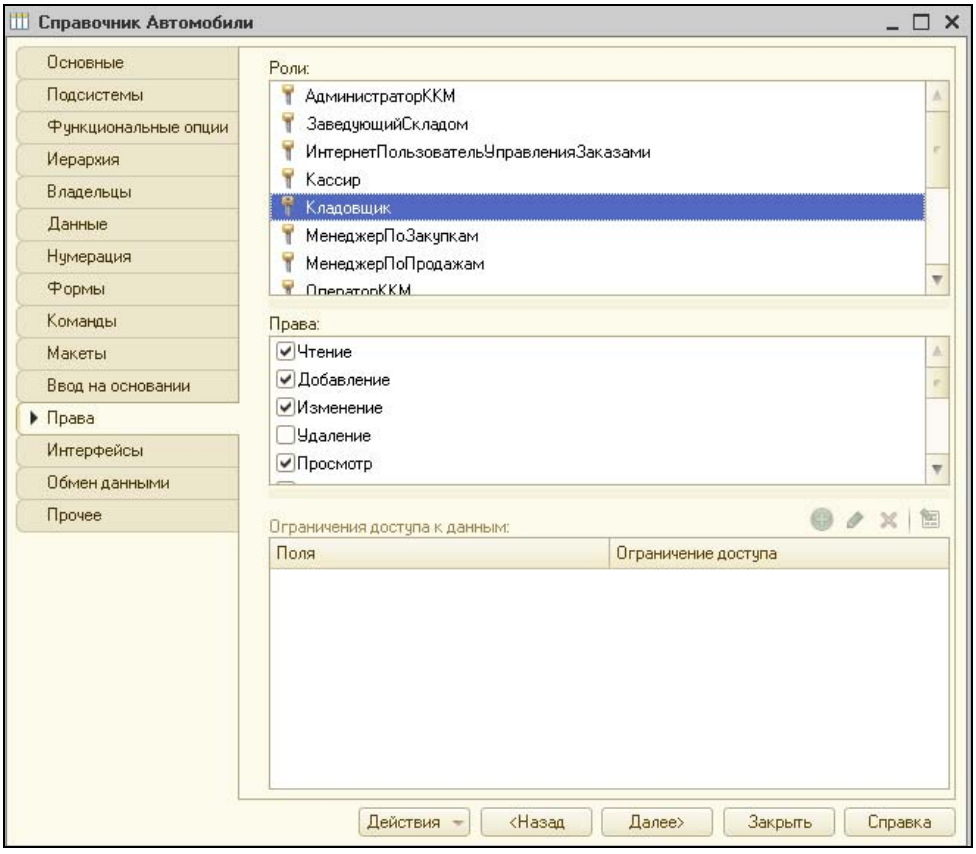



**Рис. 3.17.** Новый справочник "Автомобили". Назначаем права доступа

Включение нового справочника в интерфейсы довольно-таки похоже на назначение прав, которое мы рассмотрели только что, да и окно выглядит практически так же (рис. 3.18).

Как и при назначении прав, в верхней части окна находится список интерфейсов, а в нижней — список действий со справочником для этого интерфейса. Собственно, команд, которые можно включить в интерфейс, всего две — открыть список записей справочника или создать новый элемент. Включаем сразу оба, хотя создание нового элемента включать особой необходимости нет. Все равно в открываемом списке есть кнопка **Новый**.

Создание справочника завершено. Сохраняем конфигурацию, также не забываем обновить конфигурацию базы данных (напомню, это делается из пункта меню **Конфигурация** или при помощи кнопок на панелях инструментов).

Теперь давайте перейдем в режим Предприятия и внесем в справочник несколько новых записей, например, как показано на рис. 3.19.

На рисунке мы видим форму списка справочника "Автомобили" со всей необходимой информацией. Внесенные данные об автомобилях мы можем программно

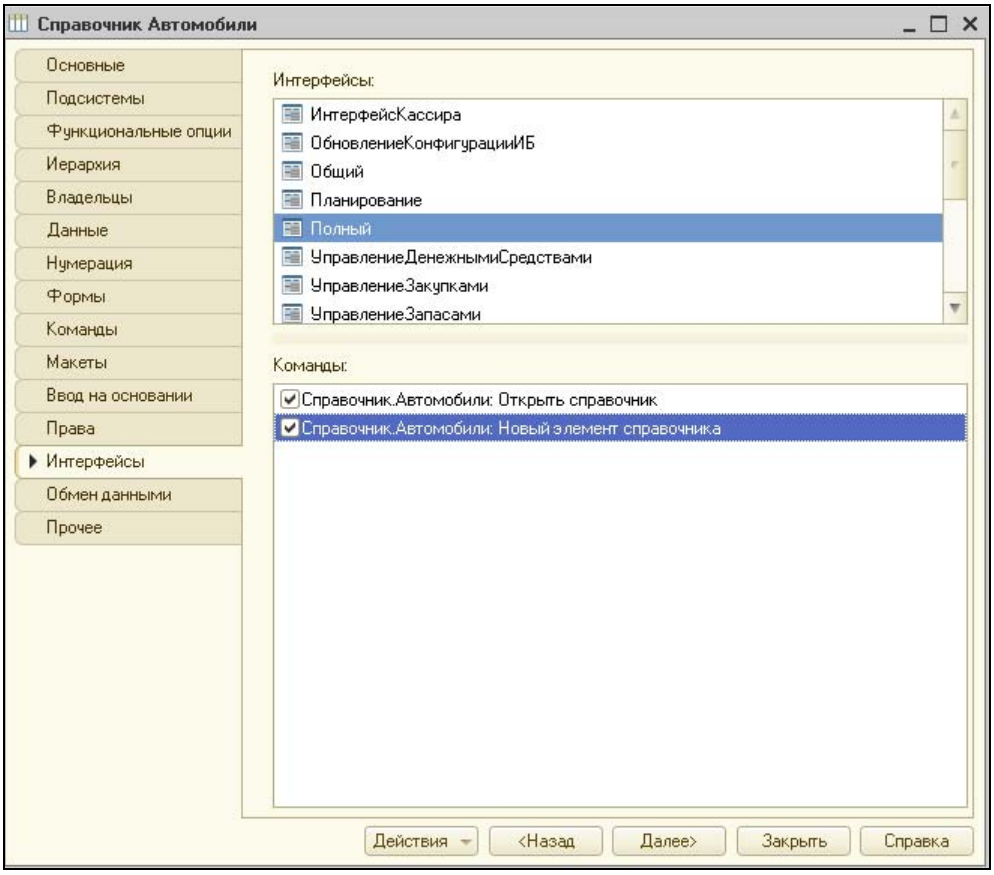
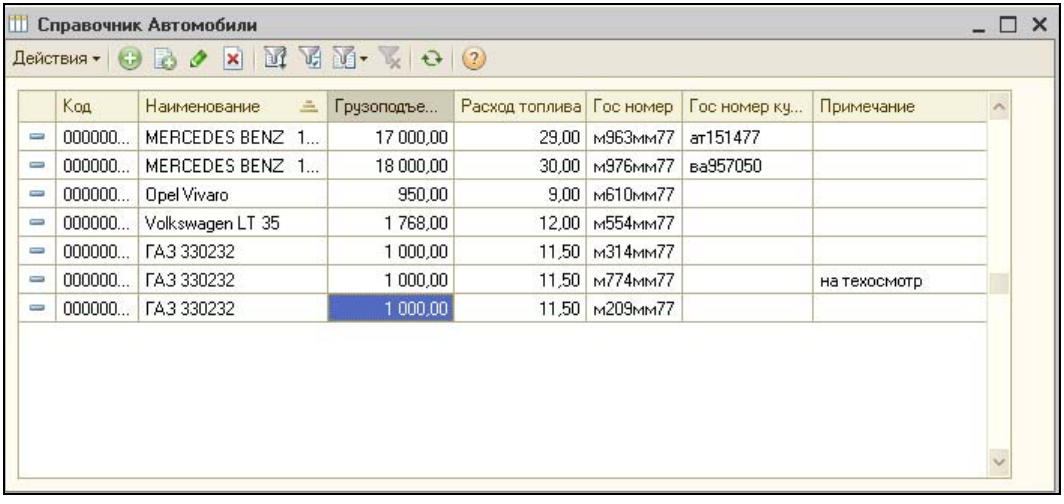



**Рис. 3.18.** Новый справочник "Автомобили". В каких интерфейсах используется




| Код | Наименование | Грузоподъе... | Расход топлива | Гос номер | Гос номер ку... | Примечание |
|---|---|---|---|---|---|---|
| 000000... | MERCEDES BENZ 1... | 17 000,00 | 29,00 | м963мм77 | ат151477 | |
| 000000... | MERCEDES BENZ 1... | 18 000,00 | 30,00 | м976мм77 | ва957050 | |
| 000000... | Opel Vivaro | 950,00 | 9,00 | м610мм77 | | |
| 000000... | Volkswagen LT 35 | 1 768,00 | 12,00 | м554мм77 | | |
| 000000... | ГАЗ 330232 | 1 000,00 | 11,50 | м314мм77 | | |
| 000000... | ГАЗ 330232 | 1 000,00 | 11,50 | м774мм77 | | на техосмотр |
| 000000... | ГАЗ 330232 | 1 000,00 | 11,50 | м209мм77 | | |

**Рис. 3.19.** Заполнение справочника автомобилей

искать, выбирать, использовать и модифицировать, как нам необходимо. При работе со справочниками мы можем использовать как методы работы с данными, рассмотренные нами в предыдущей главе, так и собственно методы работы со справочниками. Для просмотра работы примеров на практике используйте ту же самую обработку "для экспериментов", которую использовали при прочтении предыдущей главы.

Пример:

```
Выборка = Справочники.Автомобили.Выбрать();
Пока Выборка.Следующий() Цикл
  Сообщить(Выборка.ГосНомер);
КонецЦикла;
```

Это самый обыкновенный цикл Пока, синтаксис и примеры работы которого мы разбирали в предыдущей главе. Однако здесь мы перебираем не какой-то абстрактный набор значений, а вполне конкретные элементы справочника "Автомобили", выводя по каждому элементу гос. номер.

## Методы работы со справочниками

Рассмотрим несколько часто используемых функций для работы со справочниками.

- НайтиПоКоду.

  Синтаксис:

  **НайтиПоКоду(*Код*[, *ПоискПоПолномуКоду*][, *Родитель*][, *Владелец*])**

  Здесь: *Код* — искомый код; *ПоискПоПолномуКоду* — режим поиска по полному коду (может принимать значение ИСТИНА или ЛОЖЬ, по умолчанию — ЛОЖЬ); *Родитель* — родитель, в пределах которого нужно выполнить поиск; *Владелец* — владелец, в пределах которого нужно выполнить поиск.

  Здесь стоит также рассмотреть, что такое родитель и что такое владелец.

  Под *родителем* понимается старшая ступень в иерархической структуре. Например, если справочник "Товары" разбит на группы "Товары" и "Услуги", а группа "Товары", в свою очередь, разбита на подгруппы "Молочные продукты", "Соки, воды" и "Хлебо-булочные изделия", то для товара "Плюшка Московская с маком" родителем будет группа "Хлебо-булочные изделия", а для этой группы родителем будет группа "Товары".

  Под *владельцем* понимается старший справочник из двух взаимосвязанных справочников. Например, у справочника "Товары" есть различные единицы измерения — штука, килограмм, упаковка. Но "Единицы измерения" — это тоже справочник, связанный со справочником "Товары" таким образом, что у одной товарной позиции может быть несколько разных единиц измерения. В этом случае справочник "Товары" является владельцем.

  В случае команды НайтиПоКоду(), если не указывать Родителя или Владельца, поиск будет проводиться по всему справочнику.

Пример:

```
Поиск = Справочники.Автомобили.НайтиПоКоду("000000004");
Если Поиск.Пустая() = ЛОЖЬ Тогда    // Пустая() означает, что
                                    // элемент не найден
  Сообщить(Поиск.Наименование);
КонецЕсли;
```

Ищем код элемента в пределах справочника "Автомобили", если поиск прошел успешно (ссылка результата не пуста), то выдаем наименование найденного элемента. Для примера из рис. 3.19 это "ГАЗ 330232".

- `НайтиПоНаименованию`.

Синтаксис:

**`НайтиПоНаименованию(`*`Наименование`*`[, `*`ТочноеСоответствие`*`][, `*`Родитель`*`] [, `*`Владелец`*`])`**

Здесь: *`Наименование`* — искомое наименование (или его часть, в зависимости от следующего параметра); *`ТочноеСоответствие`* означает, полностью ли соответствует искомое значение значению параметра `Код`. Может принимать значение `ИСТИНА` или `ЛОЖЬ`. Если `ТочноеСоответствие` равно `ИСТИНА`, то параметр `Код` должен полностью совпадать с искомым кодом (за исключением пробелов в конце), если же `ТочноеСоответствие` равно `ЛОЖЬ`, то поиск ведется по части кода, начиная слева. По умолчанию `ТочноеСоответствие = ЛОЖЬ`.

*`Родитель`* — родитель, в пределах которого нужно выполнить поиск; *`Владелец`* — владелец, в пределах которого нужно выполнить поиск, аналогично с методом `НайтиПоКоду()`.

Пример:

```
Поиск = Справочники.Автомобили.НайтиПоНаименованию("Opel");
Если Поиск.Пустая() = ЛОЖЬ Тогда
  Сообщить(Поиск.Наименование);
КонецЕсли;
```

Ищем наименование элемента в пределах справочника "Автомобили" по части наименования. Если поиск прошел успешно (ссылка результата не пуста), то выдаем наименование найденного элемента. Для примера из рис. 3.19 это "Opel Vivaro".

```
Поиск = Справочники.Автомобили.НайтиПоНаименованию("Opel", Истина);
Если Поиск.Пустая() = Ложь Тогда
  Сообщить(Поиск.Наименование);
КонецЕсли;
```

Этот пример отличается от предыдущего всего одним параметром. Мы включили поиск по точному соответствию (`ТочноеСоответствие = ИСТИНА`), поэтому по примеру из рис. 3.19 мы ничего не найдем.

Пример:

```
Поиск = Справочники.Автомобили.НайтиПоНаименованию("Opel Vivaro", Истина);
Если Поиск.Пустая() = ЛОЖЬ Тогда
  Сообщить(Поиск.Наименование);
КонецЕсли;
```

А здесь при поиске по точному соответствию мы изменили искомое наименование. Наименование одной из записей справочника "Автомобили" соответствует искомому значению, поэтому для примера с рис. 3.19 будет найден "Opel Vivaro".

- `НайтиПоРеквизиту`.

Синтаксис:

**НайтиПоРеквизиту(*ИмяРеквизита*, *ЗначениеРеквизита*[, *Родитель*][, *Владелец*])**

Здесь: *ИмяРеквизита* — имя реквизита, по которому будем искать; *ЗначениеРеквизита* — искомое значение реквизита; *Родитель* — родитель, в пределах которого нужно выполнить поиск; *Владелец* — владелец, в пределах которого нужно выполнить поиск, аналогично с методами `НайтиПоКоду()` и `НайтиПоНаименованию()`.

Пример:

```
СтрокаПоиска = "на техосмотр";
Поиск = Справочники.Автомобили.НайтиПоРеквизиту("Примечание", СтрокаПоиска);
Если Поиск.Пустая() = Ложь Тогда
  Сообщить(Поиск.Наименование+" номер: "+Поиск.ГосНомер);
КонецЕсли;
```

В этом примере мы ищем автомобиль, у которого в поле **Примечание** ввели пометку "на техосмотр", и выводим название автомобиля и его номерной знак. Для примера с рис. 3.19 результат будет "ГАЗ 330232 номер: м774мм77".

- `ВыбратьИерархически`. Формирует иерархическую выборку элементов справочника по заданным условиям. При иерархической выборке для каждого элемента сначала выбираются элементы, для которых он является родителем, а потом — элементы следующего уровня.

Синтаксис:

**ВыбратьИерархически([*Родитель*][, *Владелец*][, *Отбор*][, *Порядок*])**

Здесь: *Родитель* — родитель, по которому осуществляется выборка; *Владелец* — владелец, по которому осуществляется выборка; *Отбор* задает поле и значение отбора открываемой выборки; *Порядок* — реквизит, по которому сортируется выборка, и направление выборки (может принимать значение `"Возр"` и `"Убыв"`, по умолчанию отбор идет по возрастанию). Например, параметр *Порядок* может иметь значение `"Код Убыв"`, где `Код` — реквизит документа, по которому будем сортировать, `Убыв` — сортировка по убыванию.

Пример:

```
Выборка = Справочники.Контрагенты.ВыбратьИерархически();
Пока Выборка.Следующий() Цикл
```

```
   Сообщить(Выборка.Наименование);
КонецЦикла;
```

Этот программный код перебирает все элементы справочника "Контрагенты" и по каждому элементу сообщает его наименование.

Результат выполнения может выглядеть следующим образом:

```
Покупатели
Астра
Гаврилов И.С.
Иванов С.П.
Макаричева С.Т.
Петрова И.А.
ЧП Сергеев
Поставщики
Евромебель
Завод мягких игрушек
Мебельный комбинат №7
ООО Баглей
Ввод начальных остатков
```

Жирным шрифтом выделены группы — **Поставщики** и **Покупатели.**

- Выбрать. Формирует выборку элементов справочника по заданным условиям.

Синтаксис:

**Выбрать([*Родитель*][, *Владелец*][, *Отбор*][, *Порядок*])**

Здесь: *Родитель* — родитель, по которому осуществляется выборка; *Владелец* — владелец, по которому осуществляется выборка; *Отбор* задает поле и значение отбора открываемой выборки; *Порядок* — реквизит, по которому сортируется выборка, и направление выборки (может принимать значение "Возр" и "Убыв", по умолчанию отбор идет по возрастанию). Например, параметр *Порядок* может иметь значение "Код Убыв", где Код — реквизит документа, по которому будем сортировать, Убыв — сортировка по убыванию.

Пример:

```
Выборка = Справочники.Контрагенты.Выбрать();
Пока Выборка.Следующий() Цикл
   Сообщить(Выборка.Наименование);
КонецЦикла;
```

Этот программный код перебирает все элементы справочника "Контрагенты" и по каждому элементу сообщает его наименование.

Результат выполнения может выглядеть следующим образом:

```
Астра
Ввод начальных остатков
Гаврилов И.С.
Евромебель
Завод мягких игрушек
```

```
Иванов С.П.
Макаричева С.Т.
Мебельный комбинат №7
ООО Баглей
Петрова И.А.
Покупатели
Поставщики
ЧП Сергеев
```

Жирным шрифтом выделены группы — **Поставщики** и **Покупатели.**

В чем разница между последними двумя примерами? При обычной выборке происходит перебор элементов справочника подряд, вне зависимости от того, группа это или элемент справочника. При иерархической выборке сначала отбирается группа (т. е. родитель для всех элементов в нее входящих), затем входящие в нее элементы, потом следующая группа и входящие в нее элементы и т. д.

- `СоздатьГруппу`. Создает новую группу справочника.

  Синтаксис:

  **СоздатьГруппу()**

  Пример:

```
НовыйОбъект = Справочники["Контрагенты"].СоздатьГруппу();
НовыйОбъект.Наименование = "Комиссионеры";
НовыйОбъект.Записать();
```

- `СоздатьЭлемент`. Создает новый элемент справочника.

  Синтаксис:

  **СоздатьЭлемент()**

  Пример:

```
НовыйОбъект = Справочники["Склады"].СоздатьЭлемент();
НовыйОбъект.Наименование = "Склад-холодильник";
НовыйОбъект.ВидСклада = Перечисления.ВидыСкладов.Оптовый;
НовыйОбъект.ТипЦенРозничнойТорговли =
Справочники.ТипыЦенНоменклатуры.НайтиПоНаименованию(СокрЛП("Оптовая"));
НовыйОбъект.Записать();
```

  Данный программный код создает новый элемент справочника складов и заполняет реквизиты `Наименование`, `ВидСклада` и `ТипЦенРозничнойТорговли`, после чего записывает новый элемент.

- `Записать`. Записывает элемент в базу данных.

  Синтаксис:

  **Записать()**

  Пример:

```
НовыйОбъект = Справочники["Склады"].СоздатьЭлемент();
НовыйОбъект.Наименование = "Выносная торговля";
```

```
НовыйОбъект.ВидСклада = Перечисления.ВидыСкладов.Розничный;
НовыйОбъект.ТипЦенРозничнойТорговли =
Справочники.ТипыЦенНоменклатуры.НайтиПоНаименованию(
СокрЛП("Розничная"));
НовыйОбъект.Записать();
```

- `УстановитьПометкуУдаления`. Помечает на удаление элемент справочника.

  Синтаксис:

  **`УстановитьПометкуУдаления(`*`ПометкаУдаления`*`[, `*`ВключаяПодчиненные`*`])`**

  Здесь *`ПометкаУдаления`* может принимать значения `ИСТИНА` или `Ложь`. При значении `ИСТИНА` элемент будет помечен на удаление, при значении `Ложь` — пометка на удаление будет снята с элемента. *`ВключаяПодчиненные`* — может принимать значения `Истина` или `Ложь`. При значении `Истина` элемент будет помечен на удаление вместе со всеми подчиненными ему элементами, при значении `Ложь` — помечен на удаление будет только сам элемент. Значение по умолчанию — `Истина`.

  Пример:

```
Спр = Справочники.Номенклатура.НайтиПоКоду("00000000108").ПолучитьОбъект();
Спр.УстановитьПометкуУдаления(Истина);
```

  В этом примере мы ищем элемент справочника с заданным кодом, получаем его командой `ПолучитьОбъект()` и помечаем на удаление. С данным примером связана еще одна важная команда работы со справочниками — `ПолучитьОбъект()`.

- `ПолучитьОбъект`. Получает объект по ссылке, после чего полученный объект можно читать, модифицировать и удалять.

  Синтаксис:

  **`ПолучитьОбъект()`**

  Пример:

```
Спр = Справочники.Номенклатура.НайтиПоКоду("00000000108").ПолучитьОбъект();
Спр.УстановитьПометкуУдаления(Ложь);
```

  Этот пример отменяет результат предыдущего примера — ищет элемент справочника номенклатуры с заданным кодом и снимает с него пометку на удаление.

- `ПустаяСсылка`. Получает пустое значение ссылки на справочник заданного типа.

  Синтаксис:

  **`ПустаяСсылка()`**

  Пример:

```
Спр = Справочники.Контрагенты.ПустаяСсылка();
```

Теперь давайте внесем последние коррективы в наш справочник.

Обратите внимание на то, что при работе с формой справочника "Автомобили", вне зависимости от того, есть кузов у автомобиля или нет, значение номерного знака кузова внести все равно можно. Так проектировать форму можно, но неправильно.

Гораздо лучше сделать так, чтобы если уж кузова нет, то и никакой информации по нему пользователь внести не смог.

Для того чтобы сделать это, откройте форму элемента справочника "Автомобили" (для этого открываем дерево конфигурации, ищем в разделе **Справочники** справочник "Автомобили", открываем его содержимое знаком "+", выбираем в подразделе **Формы** ранее созданную нами форму элемента и открываем ее двойным щелчком). Если до этого вы все делали, как описывалось в этой главе ранее, то форма элемента должна выглядеть подобно рис. 3.11. Теперь выполняем следующие действия:

1. Щелкнем правой кнопкой мыши по полю **ГосНомерКузова** и в контекстном меню выберем пункт **Свойства**. При этом, как и в рассмотренных ранее примерах, в правой части окна откроется окно свойств объекта.
2. В окне свойств снимем флажок **Доступность**.
3. Щелкнем правой кнопкой мыши по флажку **НаличиеКузова** ("Есть кузов") и в контекстном меню выбираем пункт **Свойства**. При этом в правой части окна откроется окно свойств выбранного флажка.
4. В нижней части окна свойств найдите раздел **События**, в котором есть опция **При изменении** (рис. 3.20).

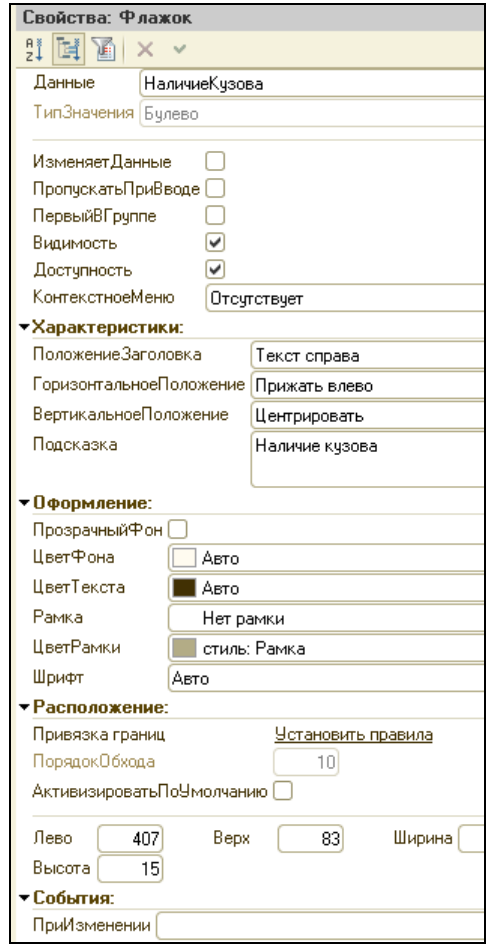



**Рис. 3.20.** Свойства флажка на форме элемента справочника

Подобных опций-действий у каждого элемента управления на форме может быть много, все зависит от типа элемента формы: **ПриИзменении**, **ПриВыборе**, **ПриОткрытии**, **ПриНажатии** и т. д. Пройдитесь мышью по элементам управления на форме, выбирая их и просматривая в свойствах, какие бывают действия. Для каждого из таких действий может быть написана программная процедура, которая выполняется, если такое действие над элементом формы будет

произведено. Нажали кнопку — выполнилась процедура, щелкнули по флажку — выполнилась процедура. Если, конечно, мы эту процедуру создадим. Сейчас именно это мы сделаем с процедурой **ПриИзменении** для флажка. Других событий у этого типа элемента управления нет. Поставили флажок, сбросили флажок — это все подпадает под категорию "При изменении".

5. Щелкните мышью на значке с изображением увеличительного стекла в правой части строки **ПриИзменении**.
6. Мы сразу попадем в модуль формы и увидим автоматически созданную процедуру (рис. 3.21).

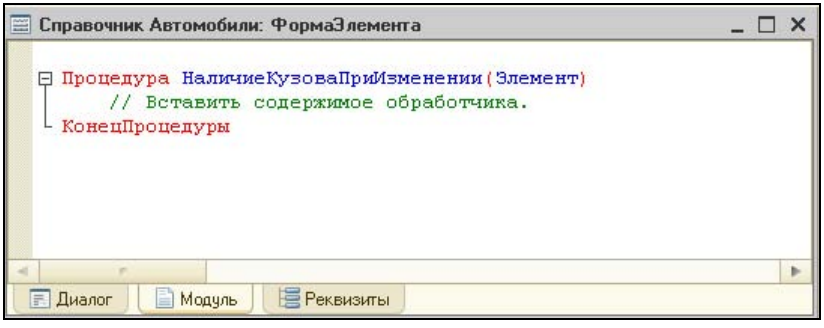



**Рис. 3.21.** Автоматически созданная процедура "При изменении" для флажка на форме

7. Модуль формы справочника пока еще пуст, других процедур там нет. Да и вновь созданная процедура также пуста, только лишь автоматически созданные начало и конец процедуры с автоматически же заданным именем и комментарий, который показывает, где мы должны создавать тело процедуры.
8. Изменим код процедуры так, как показано на рис. 3.22.

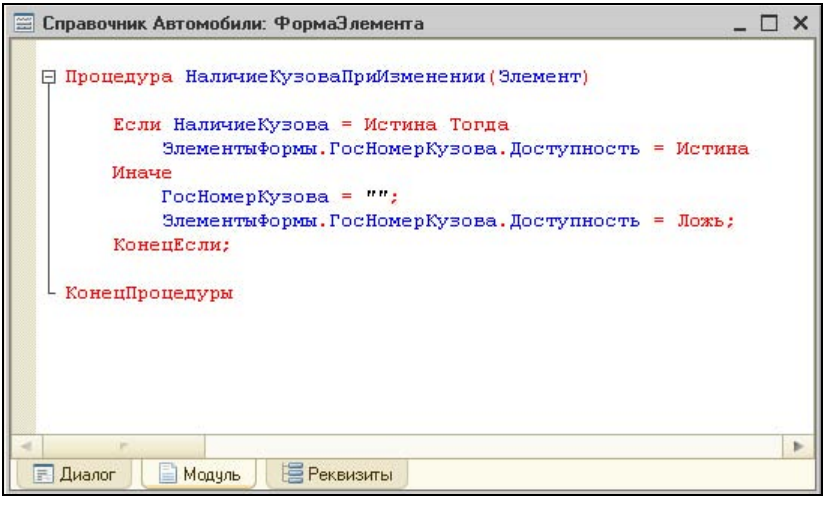



**Рис. 3.22.** А так надо переписать процедуру, приведенную на рис. 3.21

9. Строка `Если НаличиеКузова = Истина Тогда` могла быть написана по-другому, вот так: `Если НаличиеКузова Тогда`, без явного сравнения со значением `Истина`. Разбирая чужой программный код, вы неоднократно встретите такой стиль написания. По своему действию он абсолютно идентичен сравнению со значением `Истина`, показанному в примере на рис. 3.22. Если же значение сравнивается со значением `Ложь`, то программный код `Если НаличиеКузова = Ложь Тогда` будет идентичен такому программному коду: `Если Не НаличиеКузова Тогда`. Попробуйте переписать пример, показанный на рис. 3.22, разными способами.
10. А что делает строка `ГосНомерКузова = ""`? Она предназначена для того, чтобы очистить номер кузова, если пользователь ввел номер кузова, потом передумал, но стирать не стал, а просто убрал флажок **НаличиеКузова**. Если бы мы не прописали эту строку, то в поле **ГосНомерКузова** осталось бы значение, пусть и недоступное к редактированию.
11. Сохраните конфигурацию и конфигурацию базы данных, перейдите в режим Предприятия и проверьте, как работает форма справочника "Автомобили" теперь. Поле номера кузова теперь доступно для внесения данных, только при наличии кузова.

#### *Примечание*

Подобным же образом работает метод элемента формы **Видимость**, только в отличие от метода **Доступность**, при `Видимость = Ложь` объект становится не недоступным для редактирования, а попросту не виден пользователю.

#### *Примечание*

Обратите внимание на то, что для того чтобы обратиться к элементу на форме, мы используем не просто имя реквизита, а обращаемся к имени через **ЭлементыФормы**. Элемент формы может содержать в себе тот или иной реквизит справочника, но сам по себе реквизит справочника не может быть недоступен или невидим. А вот элемент формы, в котором реквизит отображается, — может.

#### *А как перебрать элементы справочника?*

```
Выборка = Справочники.Номенклатура.ВыбратьИерархически();
Пока Выборка.Следующий() Цикл
    Сообщить(Выборка.Наименование);
КонецЦикла;
```

#### *А как перебрать строки табличной части справочника?*

```
ЭлементСправочника = Справочники.Контрагенты.НайтиПоКоду("000000009");
Для Каждого ТекущаяСтрока Из ЭлементСправочника.ВидыДеятельности Цикл
  Сообщить(ТекущаяСтрока.ВидДеятельности);
КонецЦикла;
```

#### *А как программно создать элемент справочника?*

```
НовыйЭлемент = Справочники.Валюты.СоздатьЭлемент();
НовыйЭлемент.Код = 96;
```

```
НовыйЭлемент.Наименование = "MNT";
НовыйЭлемент.НаименованиеПолное = "Монгольский тугрик";
НовыйЭлемент.Записать();
```

#### А КАК ПРОГРАММНО ПОМЕТИТЬ НА УДАЛЕНИЕ ЭЛЕМЕНТ СПРАВОЧНИКА?

```
ИскомыйЭлемент = Справочники.Валюты.НайтиПоКоду("96");
Если ИскомыйЭлемент.Пустая() = Ложь Тогда
  УдаляемыйЭлемент = ИскомыйЭлемент.ПолучитьОбъект();
  УдаляемыйЭлемент.УстановитьПометкуУдаления(Истина);
КонецЕсли;
```

#### А КАК ПРОГРАММНО ОТКРЫТЬ ФОРМУ СПИСКА СПРАВОЧНИКА?

```
ФормаСписка = Справочники.Автомобили.ПолучитьФормуСписка();
ФормаСписка.Открыть();
```

# Создание нового документа

*Документы* — основное средство совершения хозяйственных операций в системе "1С:Предприятие". С их помощью осуществляются все движения товарно-денежных потоков на предприятии. Чаще всего документы дублируют собой реально существующую в бумажном варианте документацию, используемую в деятельности предприятия. В зависимости от настроек конфигурации, каждый объект имеет экранную форму ввода и изменения данных, включающую средства формирования документа, и одну или несколько печатных форм, используемых для вывода некоторого представления документа на печать. По аналогии с бумажной документацией документы хранятся в журналах, предоставляющих возможность удобного просмотра, фильтрации списка документов и доступа к ним.

Типовая конфигурация, такая как "Управление торговлей" или "Бухгалтерия", уже содержит в себе большое количество различных документов. Наша задача — изучить свойства этих документов и программные методы работы с ними, а также научиться добавлять в конфигурацию новые документы и модифицировать имеющиеся.

Ранее для изучения работы со свойствами справочников и конфигурирования форм мы создали новый справочник. Теперь точно так же создадим новый документ. Раз уж мы начали создавать на нашем предприятии систему перевозок, то продолжим данное начинание и создадим документ "Развозка", в котором будут фиксироваться все поездки автотранспорта предприятия, связанные с развозкой проданного товара клиентам.

Новый документ создаем так же, как ранее создавали справочник. В дереве конфигурации найдем группу **Документы**, щелкнем на ней правой кнопкой мыши и в контекстном меню выберем единственный пункт **Добавить**. Откроется окно создания нового документа, очень похожее на то, с которым мы работали при создании справочника (рис. 3.23).

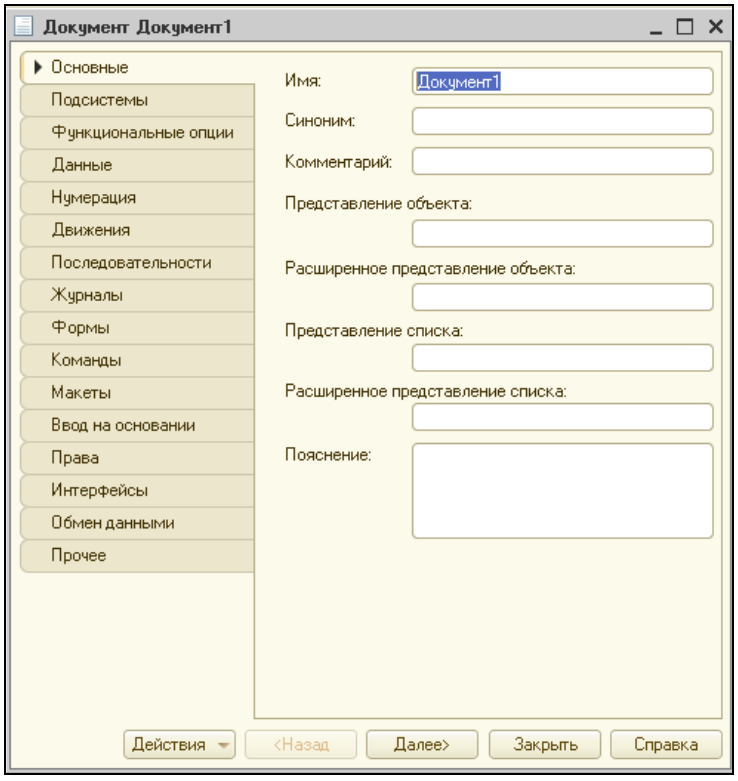



**Рис. 3.23.** Создаем новый документ

Как и у любого другого объекта метаданных, у нового документа должно быть свое уникальное имя. Также можно прописать синоним для лучшей читаемости и пояснение-комментарий. Заполните форму так, как показано на рис. 3.24.

Поле **Синоним** можно было и не заполнять, все равно имя документа состоит из одного слова. Но для единообразия с другими объектами пусть будет. Далее, как и раньше со справочниками, мы должны отнести наш будущий документ к одной или нескольким подсистемам. Переходим на вкладку **Подсистемы** (рис. 3.25).

Документ "Развозка" будут использовать те же самые пользователи, которые будут работать со справочником "Автомобили", функционально они взаимосвязаны. Так что подсистемы выбираем те же, что и для справочника "Автомобили" — УправлениеЗакупками, УправлениеПродажами и УправлениеЗапасами.

Далее мы должны определить состав и структуру данных, которые будут использоваться в нашем новом документе. Как мы уже знаем, справочники имеют два постоянных реквизита по умолчанию: **Код** и **Наименование**. У документов также есть реквизиты по умолчанию. Это **Номер** и **Дата**. Остальные реквизиты мы создаем сами.

Список реквизитов и типы их значений приведены в табл. 3.2.

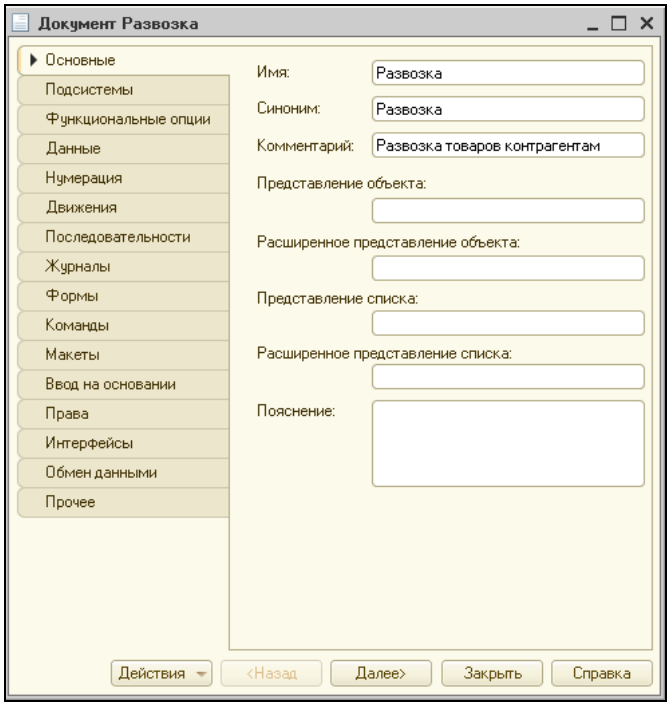



**Рис. 3.24.** Новый документ "Развозка". Вкладка **Основные**

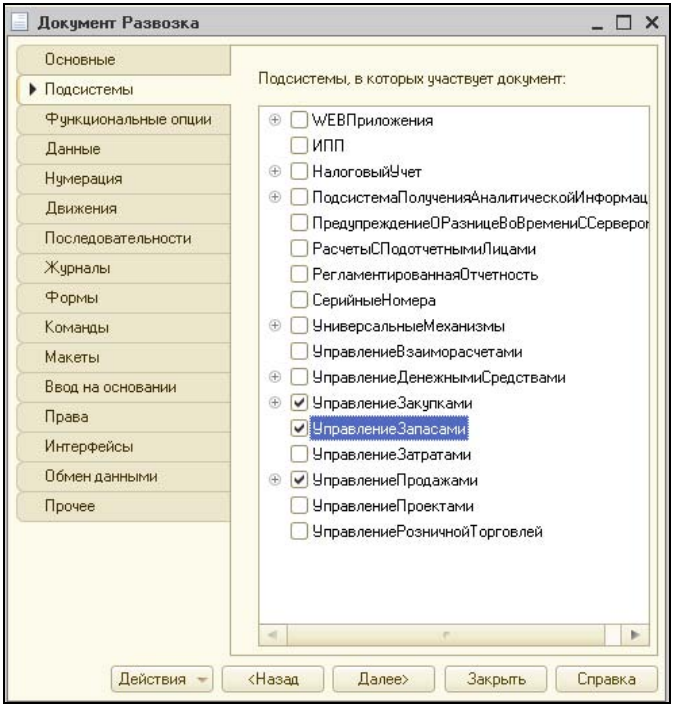



**Рис. 3.25.** Новый документ "Развозка". В каких подсистемах доступен

***Таблица 3.2.*** *Список реквизитов документа "Развозка", их тип и длина*

| Имя реквизита | Тип реквизита | Длина/принимаемые значения |
|---|---|---|
| **Документ** | Документы.РеализацияТоваровУслуг | |
| **Контрагент** | Справочники.Контрагенты | |
| **Адрес** | Строка | 50 |
| **АдресВручную** | Булево | Истина/Ложь |
| **Автомобиль** | Справочники.Автомобили | |
| **Водитель** | Справочники.ФизическиеЛица | |
| **Примечание** | Строка | 100 |

Внесите реквизиты из таблицы в список реквизитов вкладки **Данные**, как показано на рис. 3.26. Новый реквизит добавляется, как обычно, щелчком правой кнопкой мыши на заголовке реквизитов и выбором единственного пункта **Добавить**, после чего в окне свойств реквизита вносятся необходимые изменения.

В результате список реквизитов будет выглядеть так, как на рис. 3.26.

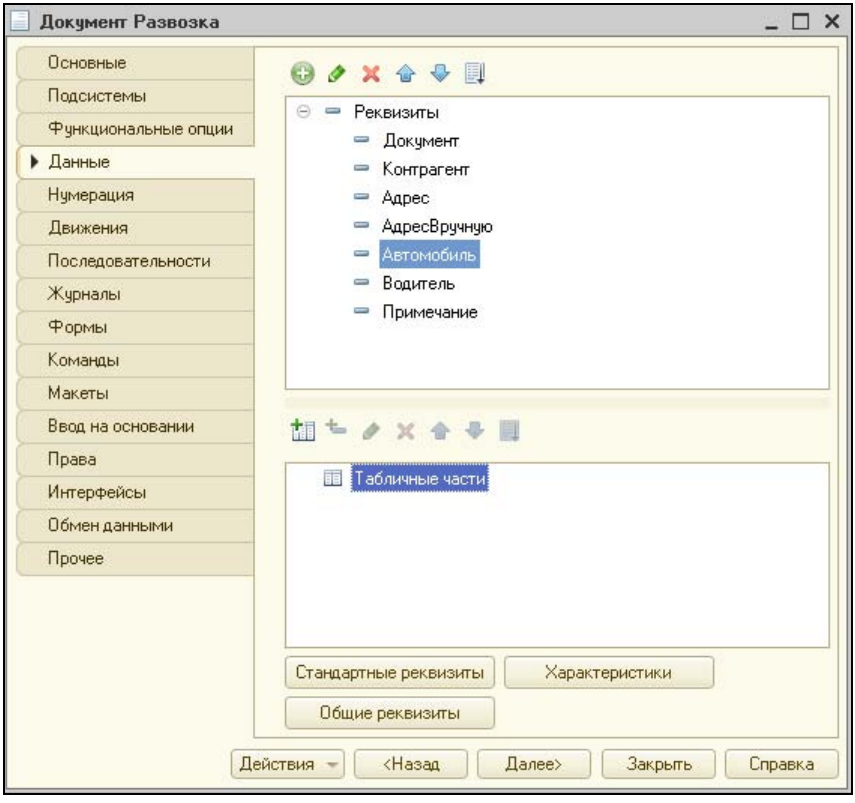



**Рис. 3.26.** Новый документ "Развозка". Список реквизитов

Как правило, документ содержит в себе не только шапку с общими реквизитами, но и табличную часть. Наш документ "Развозка" не будет исключением. Создадим для него табличную часть, в которую автоматически будет переноситься список номенклатуры развозимой расходной накладной (РеализацияТоваровУслуг) и количество. Эта табличная часть также должна будет распечатываться в виде товаротранспортной накладной (ТТН) для водителя. Табличные части добавляем точно так же, как и реквизиты: в нижней части окна, там, где должен располагаться список табличных частей, щелкнем правой кнопкой мыши по заголовку **Табличные части** и выберем единственный пункт **Добавить**. Табличных частей у документа может быть несколько, но нам пока достаточно одной. Сделайте так, как показано на рис. 3.27.

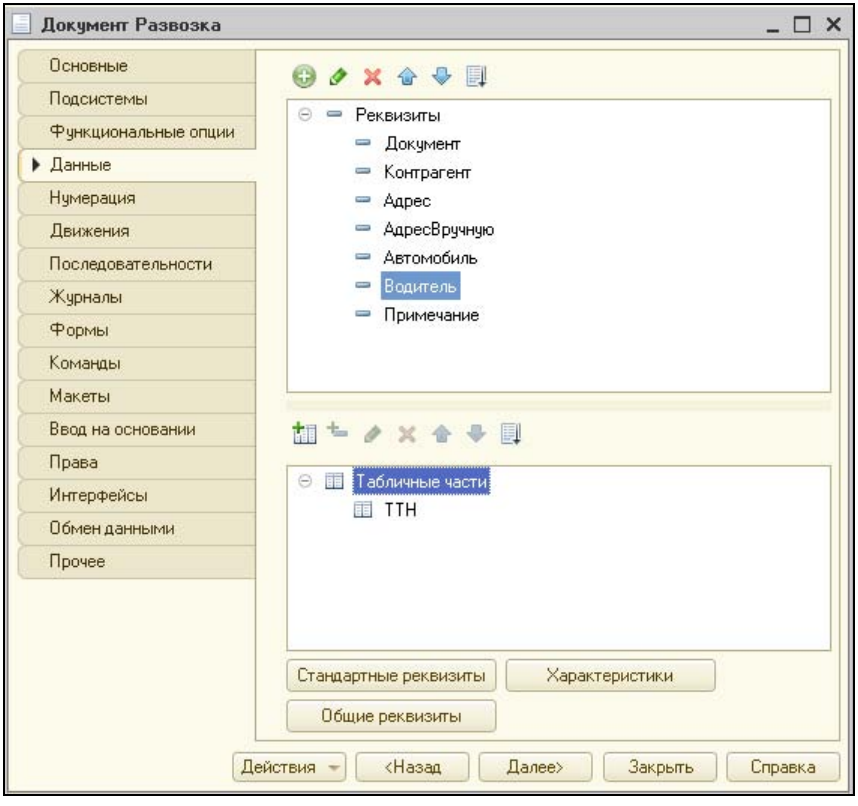



**Рис. 3.27.** Новый документ "Развозка". Добавляем табличную часть

Теперь, аналогично тому, как создавали реквизиты документа, создаем реквизиты табличной части. Щелкнем правой кнопкой мыши по заголовку **ТТН**, выберем пункт **Добавить**, а в выпадающем списке — пункт **Реквизит табличной части**. Список реквизитов и типы их значений приведены в табл. 3.3.

После добавления всех реквизитов вкладка **Данные** должна выглядеть так, как показано на рис. 3.28.

***Таблица 3.3.*** *Список реквизитов табличной части документа "Развозка", их тип и длина*

| Имя реквизита | Тип реквизита | Длина/принимаемые значения |
|---|---|---|
| **Товар** | Справочники.Номенклатура | |
| **Количество** | Число | 12.2 |
| **ЕдиницаИзмерения** | Справочники.ЕдиницыИзмерения | |

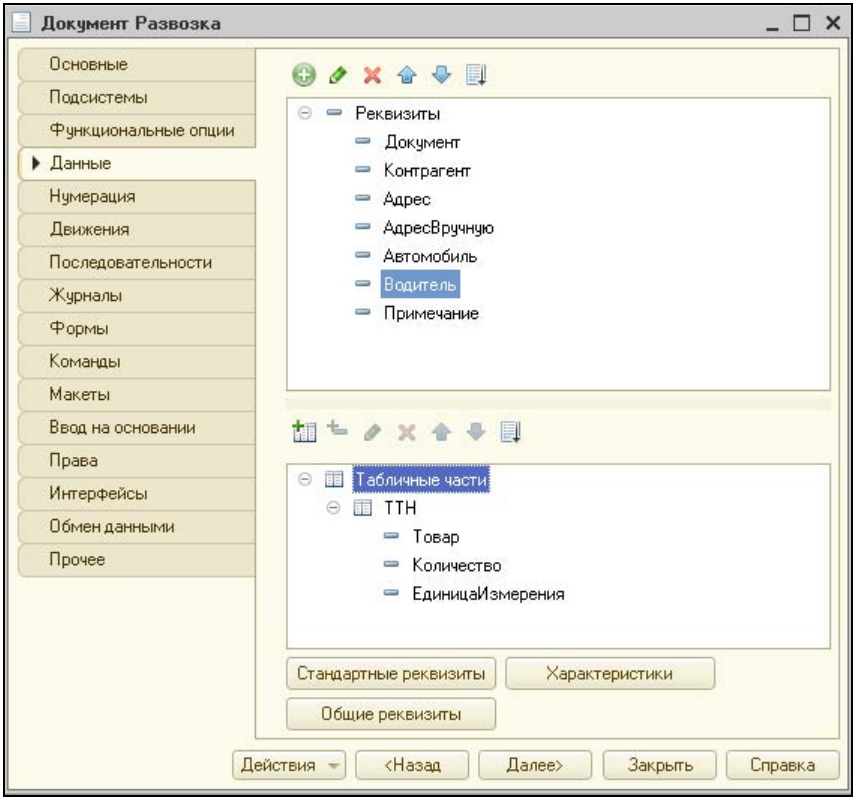



**Рис. 3.28.** Новый документ "Развозка". Добавляем реквизиты табличной части

Далее мы переходим на вкладку **Нумерация**. Делаем все, как показано на рис. 3.29. Нам нужно, чтобы документы автоматически нумеровались системой при создании, причем номер был уникальным. Длина номера — 9 символов, этого нам вполне хватит. Тип номера — **Строка**, что означает, что допускаются номера с префиксом, например, так: Ф-0000089. Префиксы чаще всего назначаются для обозначения принадлежности к определенной фирме (при многофирменном учете). При желании мы можем назначить документу периодичность (раз в день, месяц, квартал или год нумерация будет начинаться сначала). Подобная периодичность нумерации регламентирована законодательно для некоторых документов бухгалтерской от-

четности, например для налоговых накладных. Для нашего нового документа, нестандартного и к тому же "для служебного использования", подобная периодичность нумерации смысла не имеет, так что оставляем все как есть.

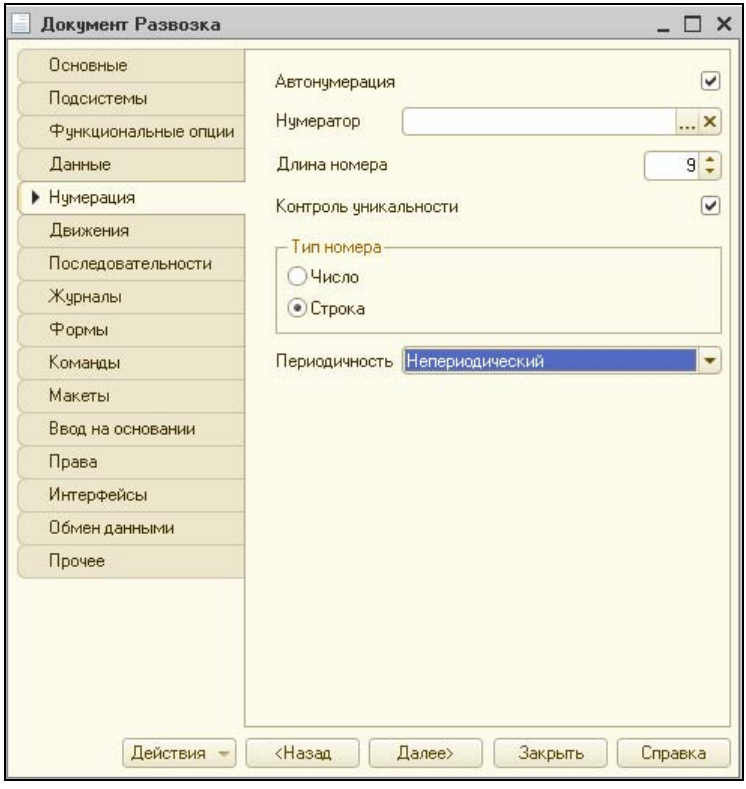



**Рис. 3.29.** Новый документ "Развозка". Задаем нумерацию

Далее на вкладке **Журналы** мы можем задать принадлежность нашего нового документа одному или нескольким журналам. Что такое *журнал*? Это "контейнер" для хранения списка документов. Точно по аналогии с картонной папкой или пластиковым файлом, в которых хранятся документы. Журнал сам по себе не является источником информации, однако предназначен для отображения хранимой информации в удобном для пользователя виде. Для нового документа надо бы завести отдельный журнал. Но пока мы его не завели, привяжем документ "Развозка" к журналу складских документов, как к наиболее близкому по типу и смыслу хранимой информации (рис. 3.30).

Далее нам нужно задать формы документа, как обычную, так и форму списка документов. Для этого переходим на вкладку **Формы** (рис. 3.31).

Для создания новой формы щелкнем по заголовку **Формы** правой кнопкой мыши и выберем единственный пункт **Добавить**. Нам предложат указать тип формы (рис. 3.32).

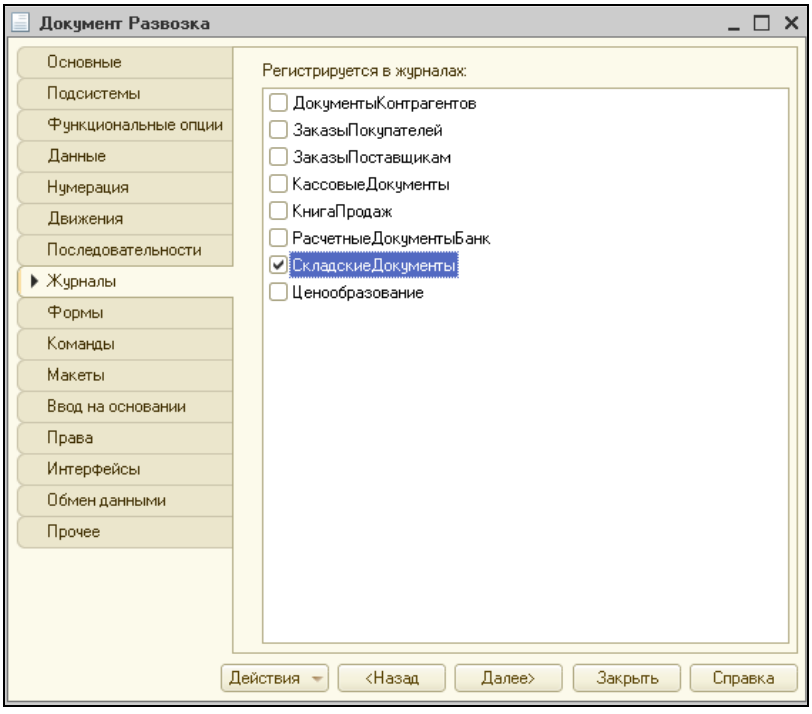



**Рис. 3.30.** Новый документ "Развозка". В каких журналах будет доступен

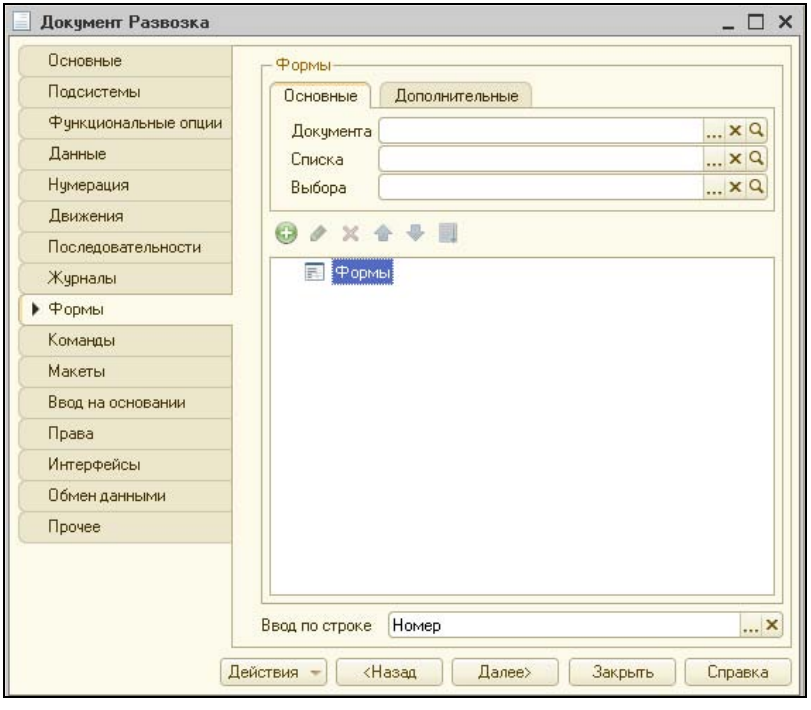



**Рис. 3.31.** Новый документ "Развозка". Вкладка **Формы**

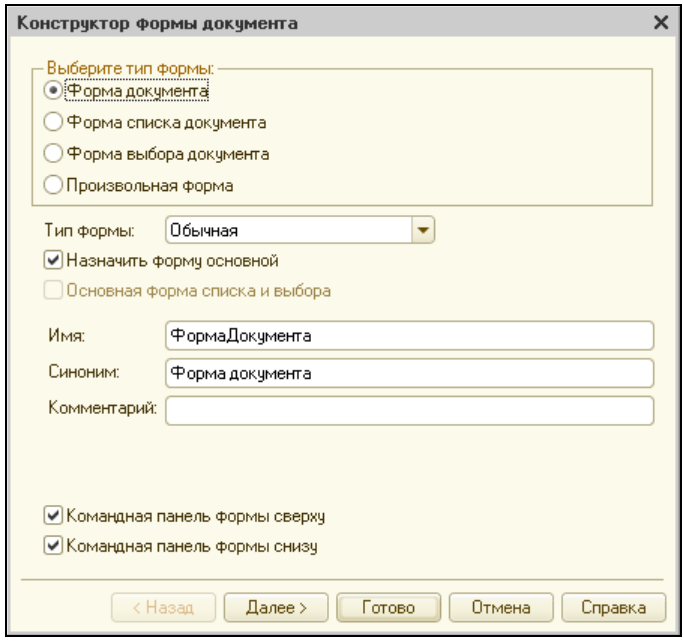



**Рис. 3.32.** Создаем форму документа

Пока что нам нужно создать обычную форму документа, так что оставим все так, как показано на рис. 3.32, и нажмем кнопку **Готово**. Как и в случае со справочником "Автомобили", для нашего нового документа система автоматически построит форму с реквизитами (рис. 3.33).

Если такая форма устраивает, можно оставить все как есть, в противном случае элементы формы можно разместить так, как удобно разработчику. Я оставил почти все, как было, переместив только поля **Номер** и **Дата** (рис. 3.34).

После создания формы нам нужно задать права на новый документ. На вкладке **Права** в верхней части окна перечислены доступные в конфигурации роли, а в нижней части — возможные действия с документом для выбранной роли (рис. 3.35).

Дадим права на работу с документом менеджеру по закупкам, менеджеру по продажам и кладовщику.

Следующим шагом на вкладке **Интерфейсы** зададим, в каких пользовательских интерфейсах будет использоваться документ "Развозка" (рис. 3.36).

Ну и, наконец, на вкладке **Прочие** мы можем перейти в модуль документа (модуль объекта) и в модуль менеджера одноименными кнопками (рис. 3.37). Также доступ к модулям документа и менеджера можно получить из раскрывающегося меню кнопки **Действия**.

Первая часть формирования документа завершена. В дереве конфигурации, наравне со стандартными документами, появился вновь созданный документ "Развозка" со всеми своими реквизитами и формами (рис. 3.38).

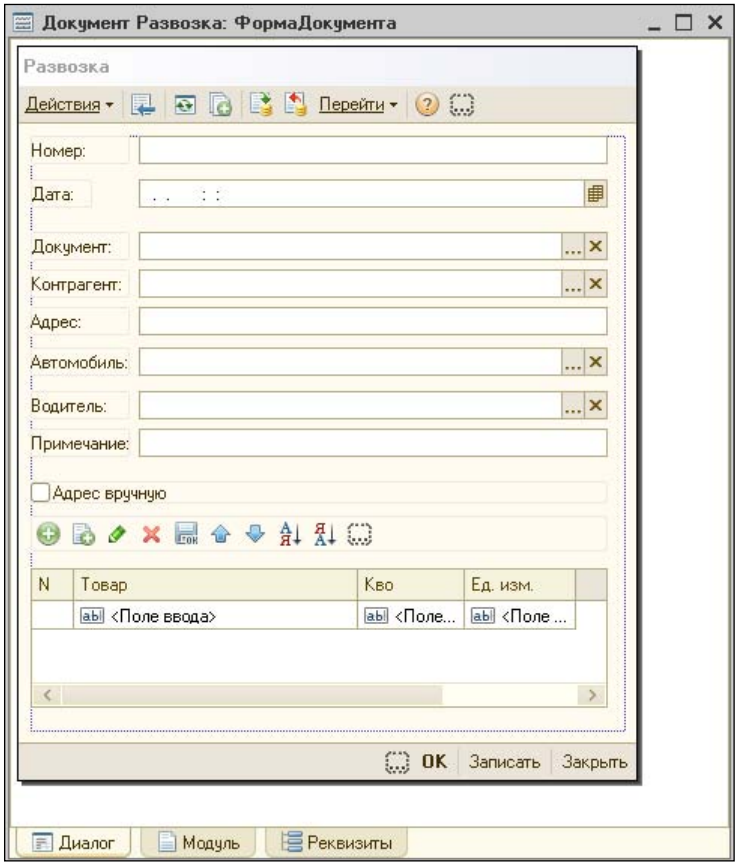



**Рис. 3.33.** Новый документ "Развозка". Автоматически скомпонованная форма документа

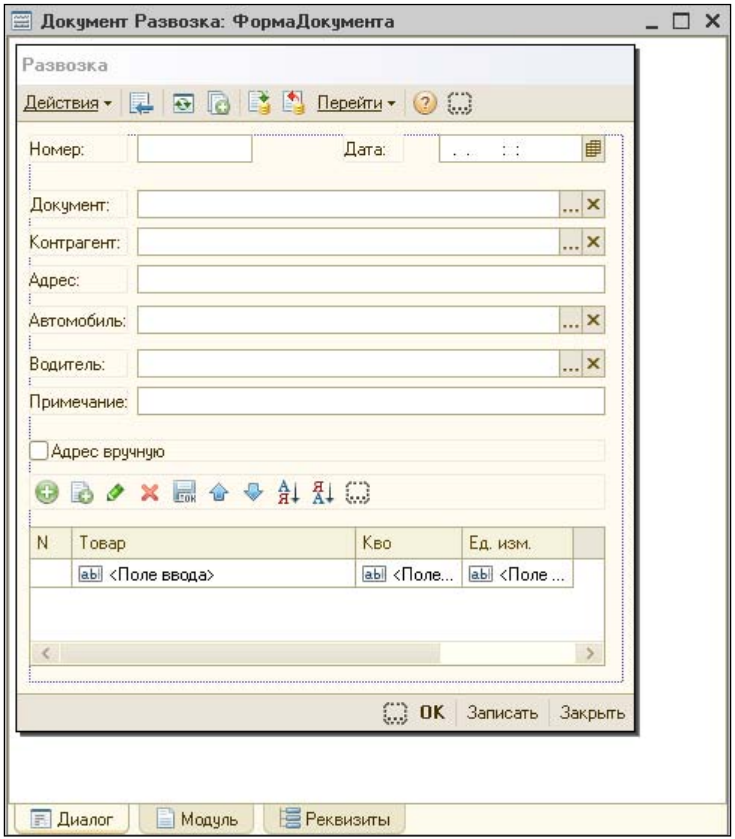



**Рис. 3.34.** Новый документ "Развозка". Форма документа изменена вручную

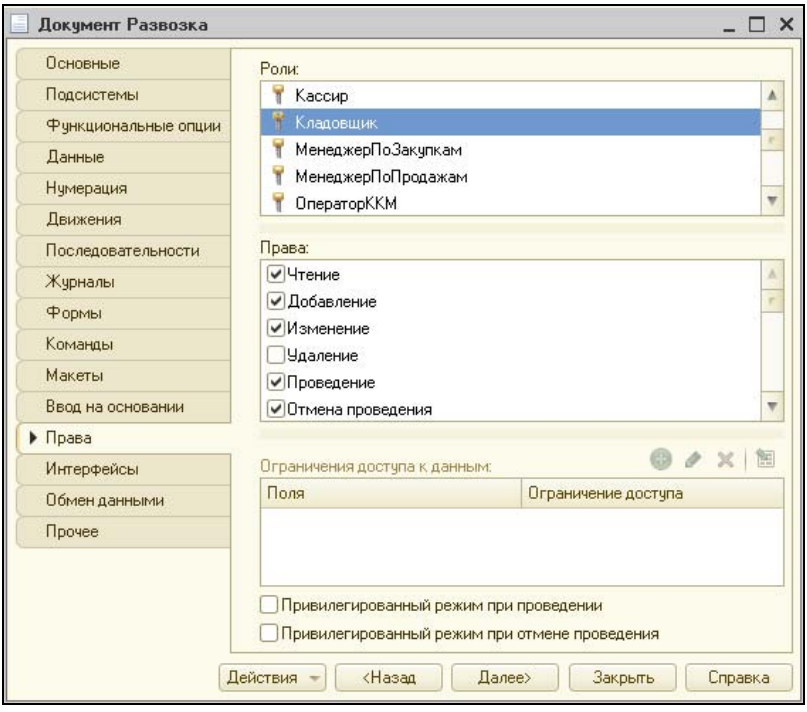



**Рис. 3.35.** Новый документ "Развозка". Назначение прав доступа

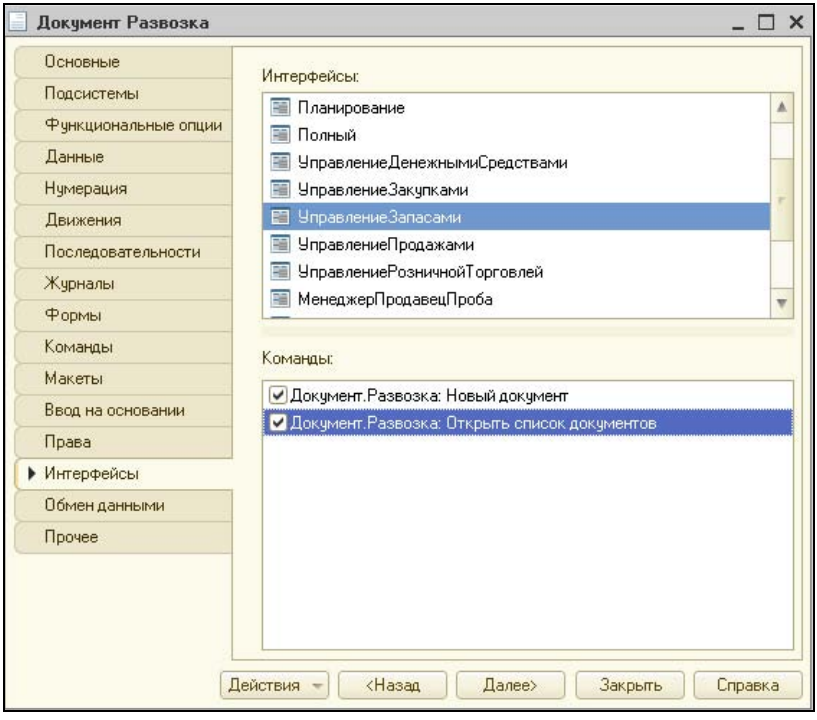



**Рис. 3.36.** Новый документ "Развозка". Назначение доступа в пользовательских интерфейсах

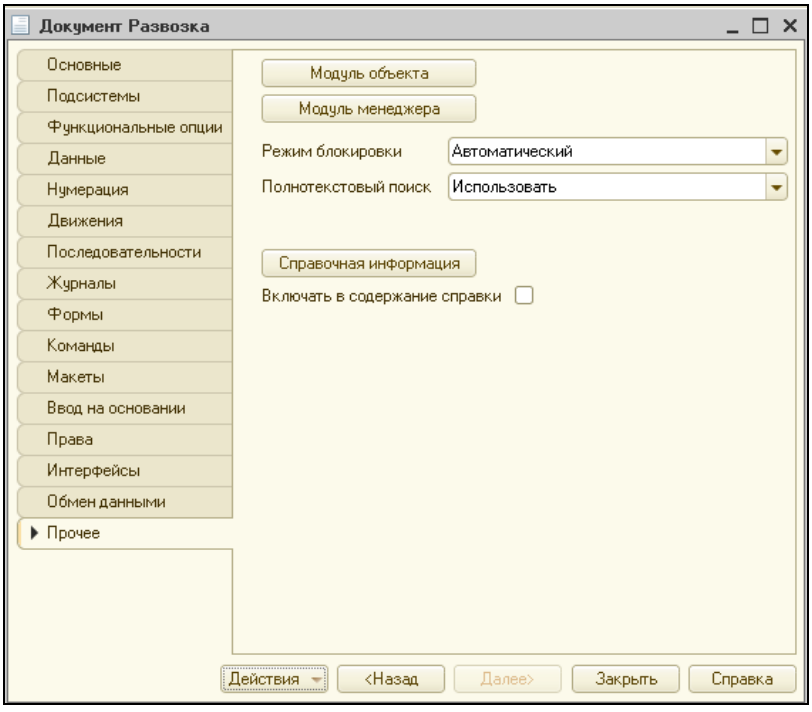



**Рис. 3.37.** Новый документ "Развозка". Вкладка **Прочее**, доступ к программным модулям

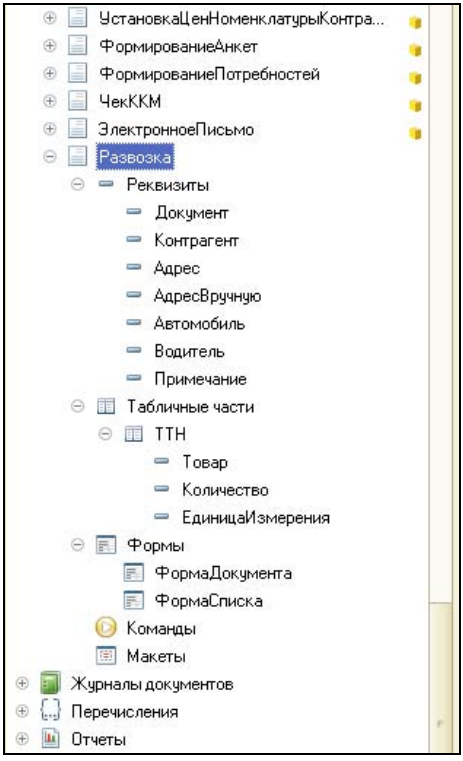



**Рис. 3.38.** Новый документ "Развозка" в дереве конфигурации

Теперь нам нужно посмотреть, как выглядит наш документ в списке. Форму списка мы ему специально не создавали, журнала тоже пока нет, привязали к стандартному журналу, хотя большинства полей нового документа в журнале просто нет.

Список документов, доступный через меню **Операции | Документы | Развозка**, в режиме Предприятия выглядит не очень хорошо. Много лишних полей, автоматически вынесенных в список одно за другим (рис. 3.39).

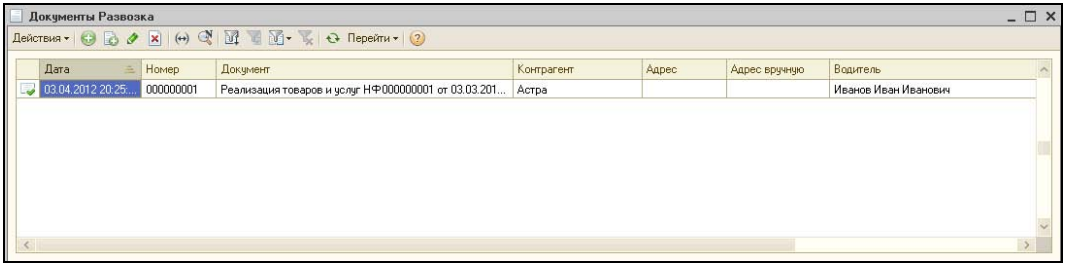



**Рис. 3.39.** Форма списка документа "Развозка" по умолчанию

Примерно аналогичным образом выглядит пробный документ "Развозка" в журнале складских документов: дата, номер, ни суммы, ни контрагента. Неудобно (рис. 3.40).

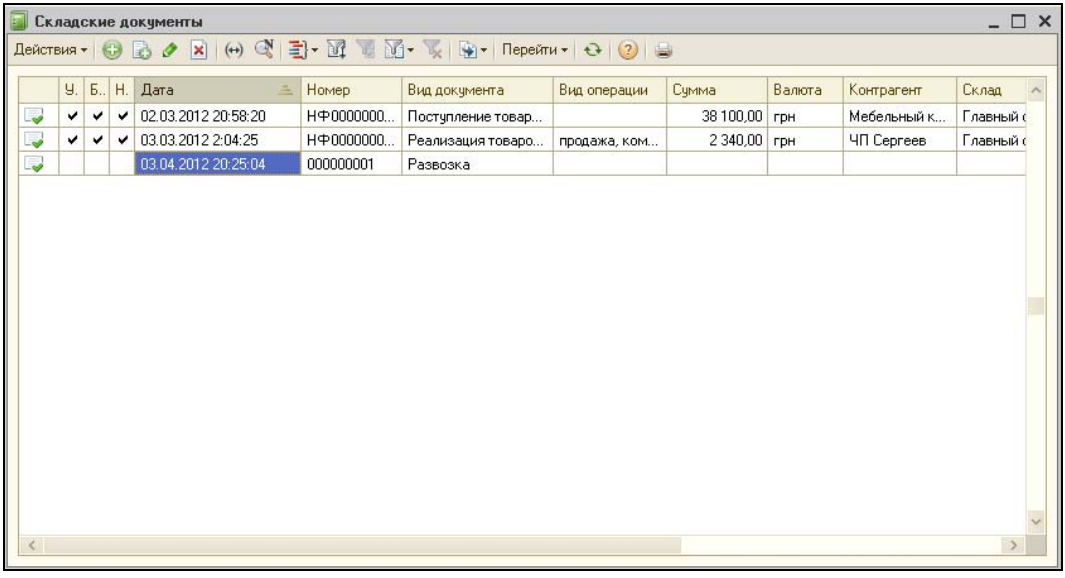



**Рис. 3.40.** Документ "Развозка" в журнале складских документов

Чтобы вывести в журнале максимально информации для пользователей, лучше создадим новый журнал и графы в него вынесем такие, которые будут соответствовать реквизитам, нужным пользователю при просмотре списка документов.

Итак, найдем в дереве конфигурации раздел **Журналы документов**, щелкнем на нем правой кнопкой мыши и выберем единственный пункт контекстного меню — **Добавить**. Откроется окно, как на рис. 3.41.

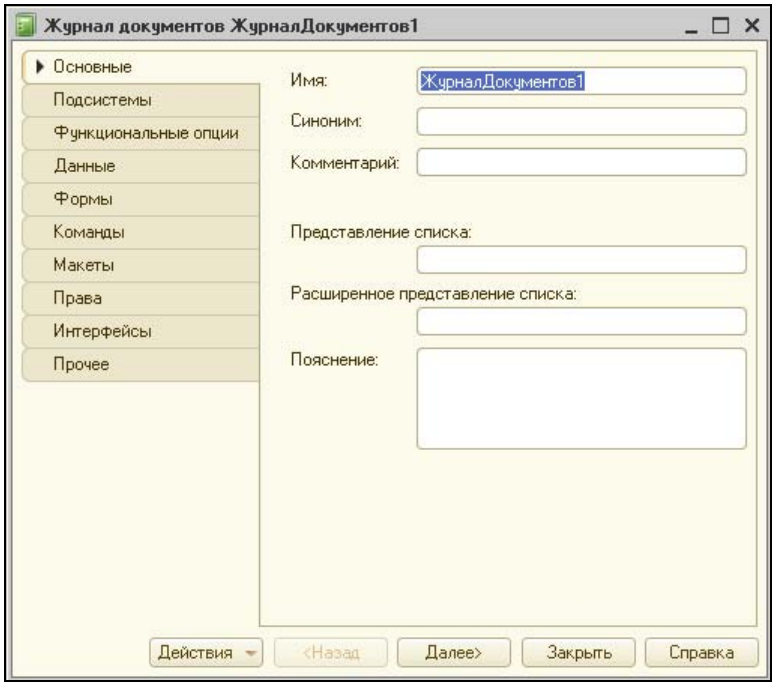



**Рис. 3.41.** Создание нового журнала документов

Аналогично тому, как мы это делали для нового справочника и документа, заполните поля **Имя**, **Синоним** и **Комментарий** (рис. 3.42).

Как и для любого объекта, новый журнал документов должен принадлежать каким-то подсистемам. В данном случае — тем же самым, которым принадлежит документ "Развозка", который будет отображаться в нашем новом журнале (рис. 3.43).

Далее перейдите на вкладку **Данные** (рис. 3.44).

В верхней части окна расположен список документов, которые будут отображаться в нашем новом журнале. Список пока пуст. В нижней части окна находится список граф будущего журналы, графы также еще не заданы.

Сначала внесем в список регистрируемых документов документ "Развозка". Для этого нажимаем кнопку с изображением карандаша, расположенную в правом верхнем углу на рис. 3.44. Откроется окно выбора документов, в котором необходимо выбрать вносимые документы (рис. 3.45).

Выберем документ "Развозка" и нажмем кнопку **OK**. Теперь документ "Развозка" привязан к нашему новому журналу (рис. 3.46).

Теперь нужно добавить графы журнала. Щелкнем правой кнопкой мыши по одноименному заголовку в нижней части окна и выберем пункт **Добавить** или просто нажмем зеленую кнопку с плюсом там же. Будет создана новая графа, которой нужно приписать имя и привязать к конкретным данным. Делается это в окне свойств графы, в правой части экрана (появляется сразу при создании новой графы). Введем имя и синоним (рис. 3.47).

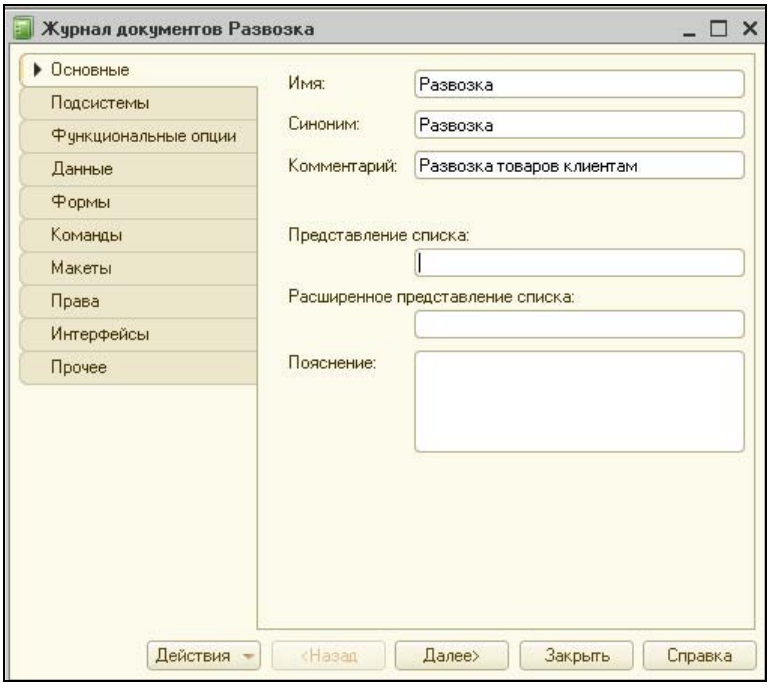



**Рис. 3.42.** Журнал документов "Развозка". Вкладка **Основные**

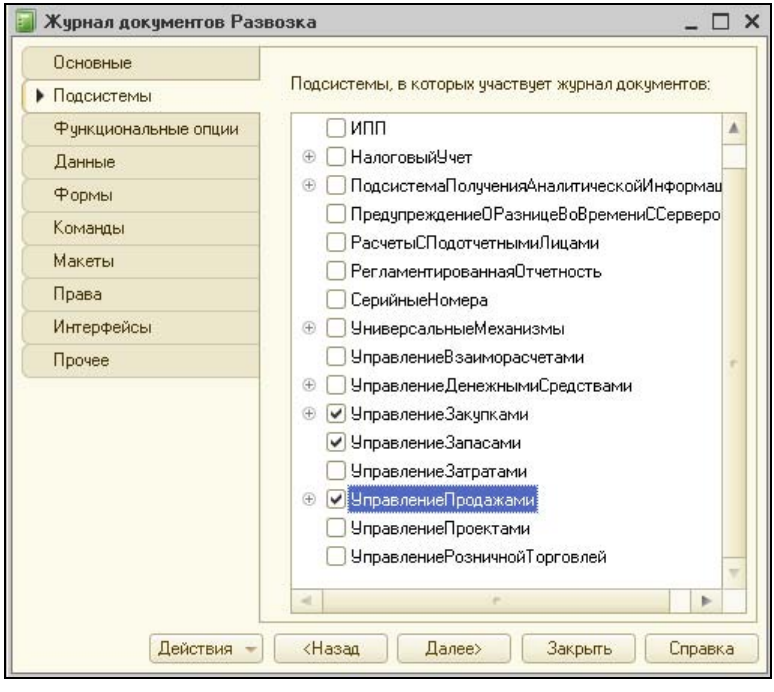



**Рис. 3.43.** Журнал документов "Развозка". В каких подсистемах доступен

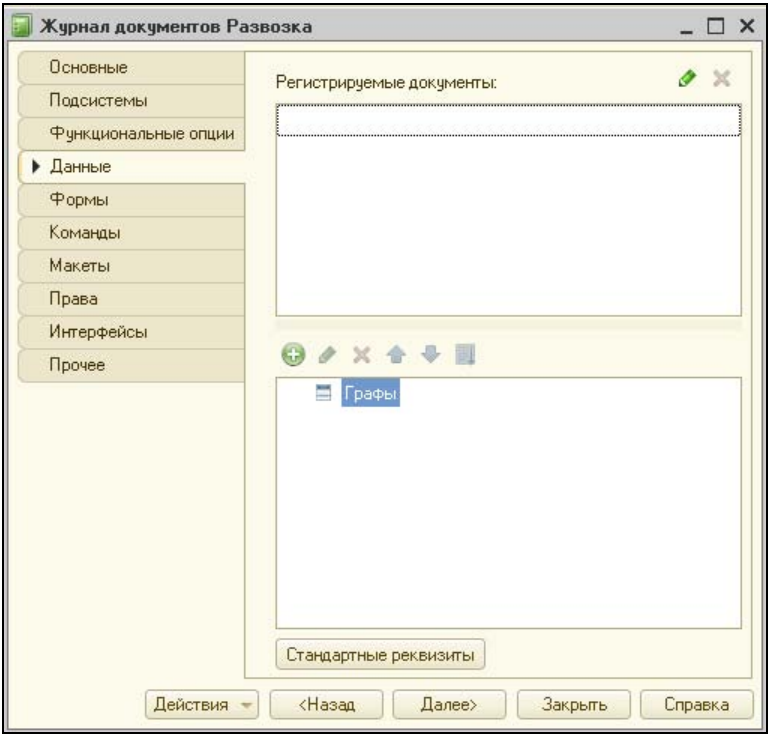



**Рис. 3.44.** Журнал документов "Развозка". Определяем, какие данные будет показывать журнал

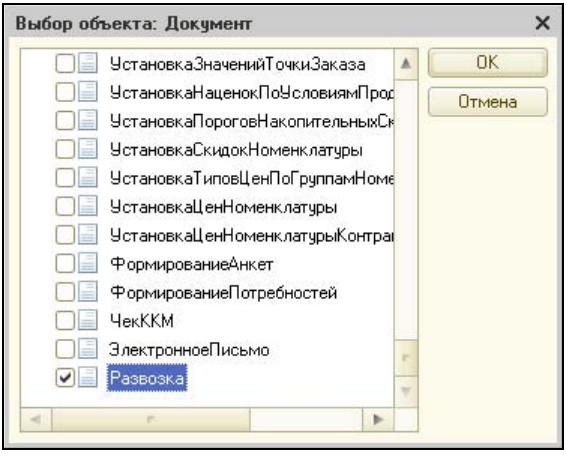



**Рис. 3.45.** Журнал документов "Развозка", регистрация хранимых документов

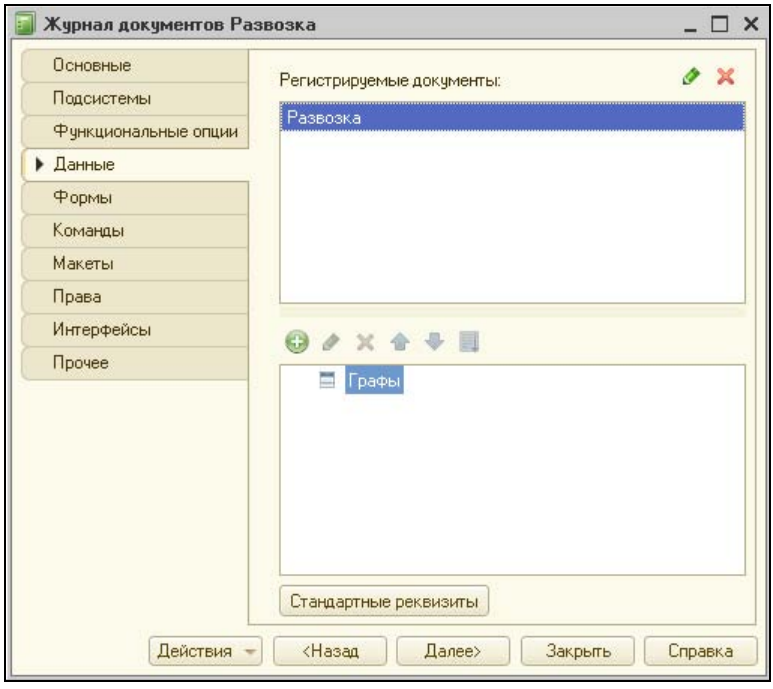



**Рис. 3.46.** Журнал документов "Развозка" с зарегистрированным документом

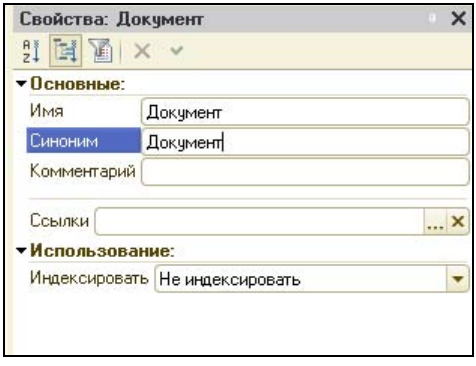



**Рис. 3.47.** Свойства новой графы журнала

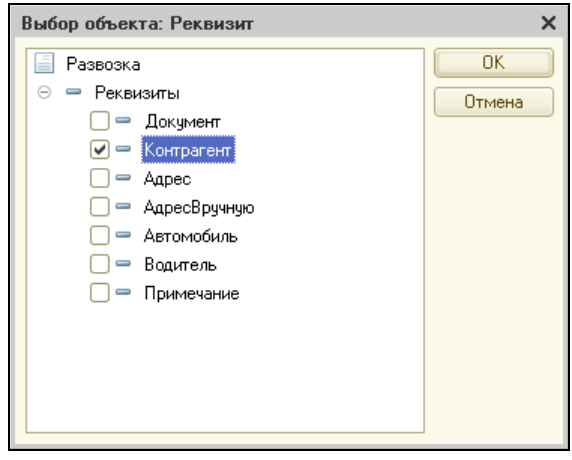



**Рис. 3.48.** Привязка данных к графе журнала

После этого здесь же, в окне свойств, привяжем графу к нужным данным, для чего нажмем кнопку с тремя точками в правой части строки **Ссылки**. Откроется окно выбора реквизита, причем в списке будут только реквизиты документа, который привязан к журналу (в нашем случае "Развозка"). Выберем графу **Контрагент** и нажмем кнопку **OK** (рис. 3.48).

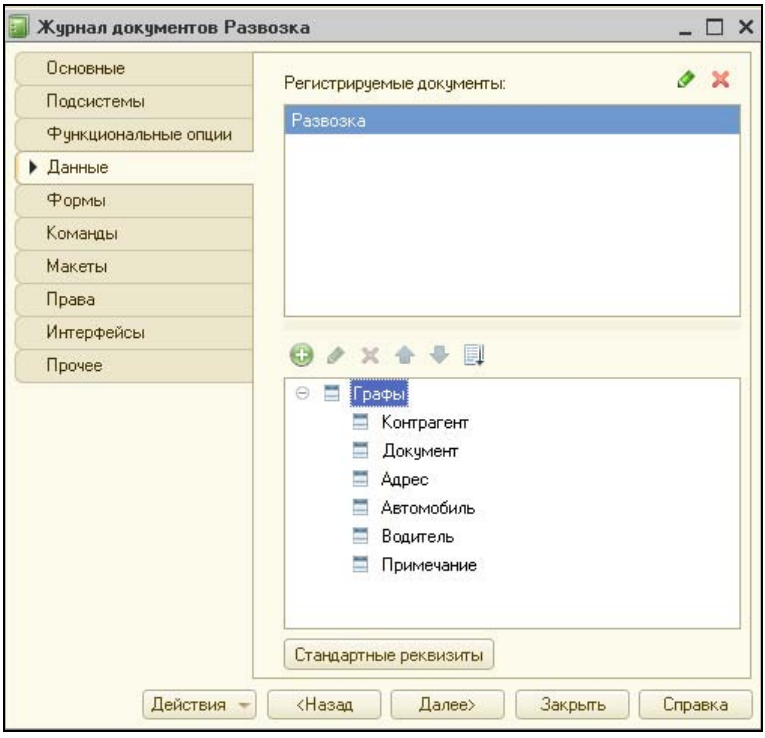



**Рис. 3.49.** Журнал документов "Развозка" с зарегистрированным документом и списком граф

Затем точно так же добавим остальные графы, привязывая к ним соответствующие реквизиты документа. Результат должен получиться таким, как показано на рис. 3.49.

После того как определились с графами, создадим форму нового журнала, по аналогии тому, как создавали форму справочника "Автомобили" и документа "Развозка".

Перейдем на вкладку **Формы** (рис. 3.50).

Щелкнем правой кнопкой мыши по заголовку **Формы** и выберем пункт **Добавить**. Откроется конструктор формы журнала (рис. 3.51).

Оставим все, как показано на рис. 3.51, и нажмем кнопку **Готово**. Форма списка журнала "Развозка" будет выглядеть так, как показано на рис. 3.52.

Далее мы должны дать пользователям права на использование журнала. Выберем пользовательские роли в верхней части окна, а в нижней проставим допустимые для выбранной роли действия (рис. 3.53).

Затем пропишем использование журнала в пользовательских интерфейсах, как мы уже делали это ранее (рис. 3.54).

Создание журнала закончено, осталось доработать сам документ "Развозка". Откроем его, перейдем на вкладку **Формы** и в дополнение к уже имеющейся форме документа создадим форму списка. Форму списка по умолчанию мы уже видели на рис. 3.39 и сейчас ее немного подправим. В открывшемся окне конструктора

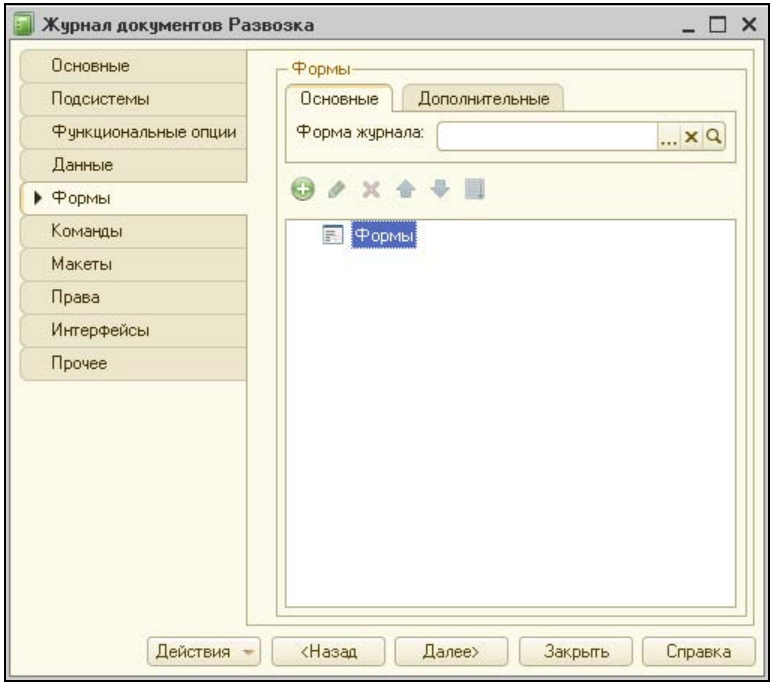



**Рис. 3.50.** Журнал документов "Развозка". Вкладка **Формы**

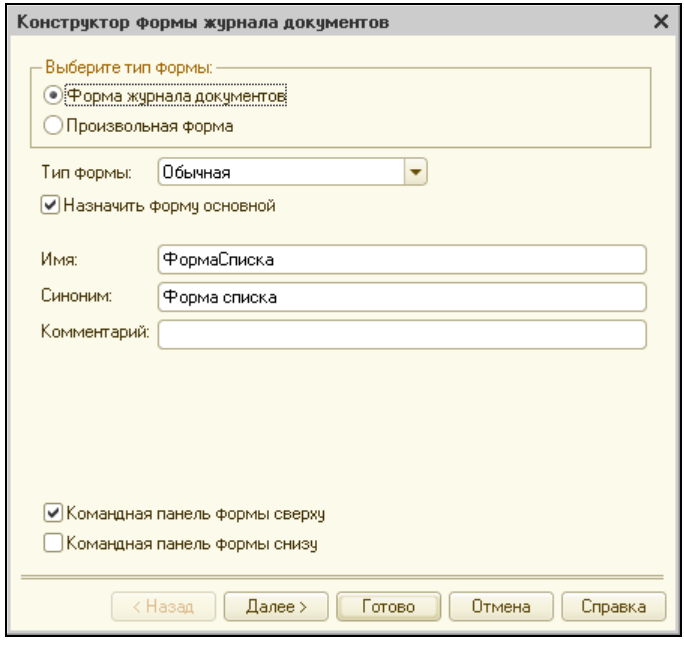



**Рис. 3.51.** Конструктор формы журнала

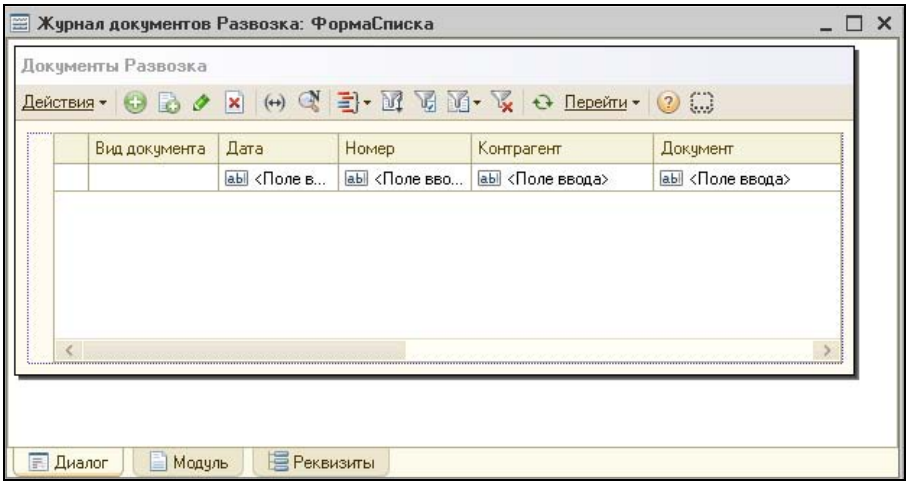



**Рис. 3.52.** Форма списка журнала "Развозка"

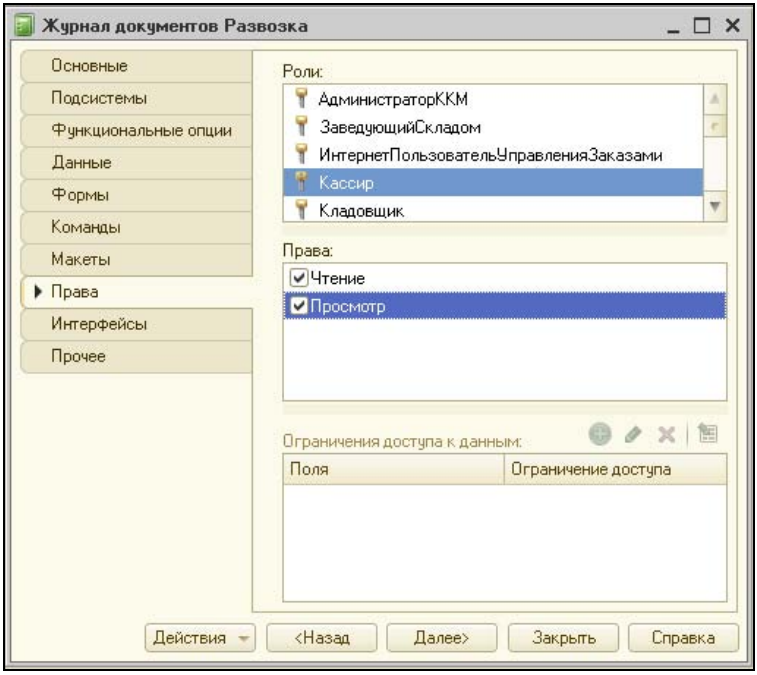



**Рис. 3.53.** Журнал документов "Развозка", назначение прав

формы выберем тип формы — **Форма списка документа**, после чего нажмем кнопку **Готово** (рис. 3.55).

После нажатия кнопки **Готово** будет создана форма списка документа, которую мы можем поправить вручную, как нам будет угодно, добавляя, удаляя или меняя местами колонки. Для того чтобы поменять местами колонки, их достаточно просто перетащить мышью, удерживая за заголовок колонки. Результат должен получиться примерно такой, как на рис. 3.56.

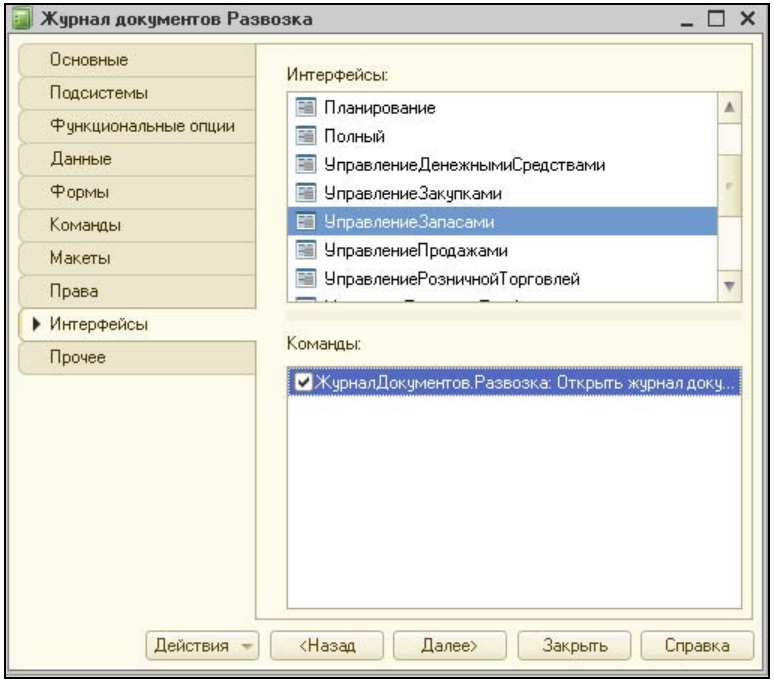



**Рис. 3.54.** Журнал документов "Развозка", задание доступа в пользовательских интерфейсах

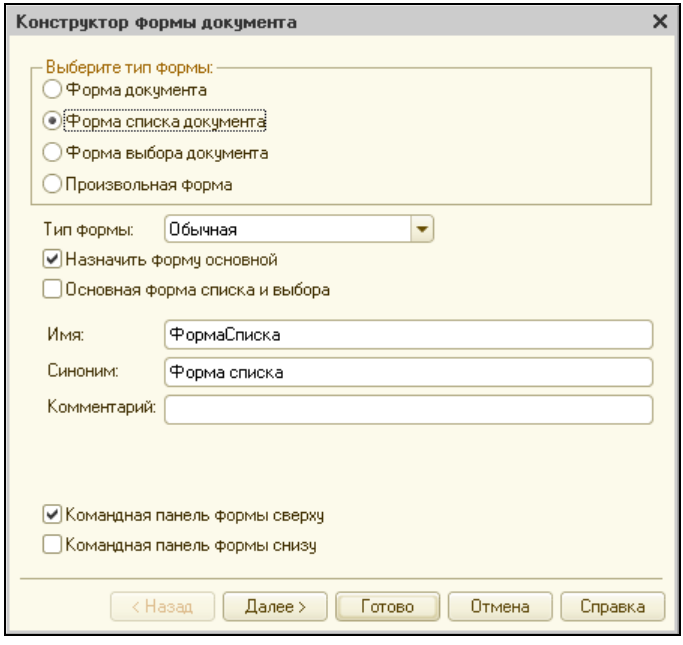



**Рис. 3.55.** Конструктор формы списка документа "Развозка"

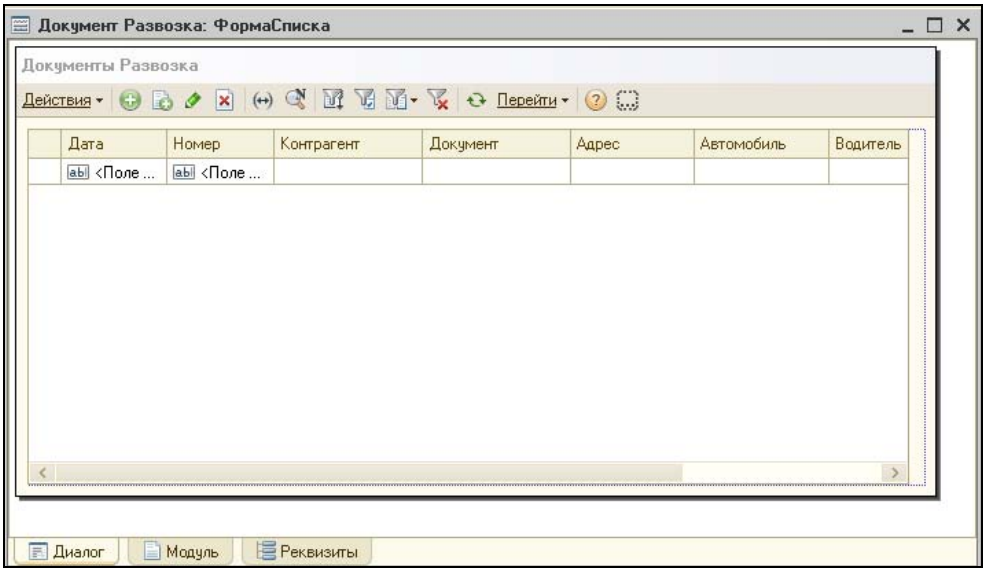



**Рис. 3.56.** Форма списка документа "Развозка"

Теперь перейдите на вкладку **Журналы** нашего документа "Развозка". На рис. 3.30 мы отнесли документ "Развозка" к журналу складских документов, поскольку отдельный журнал еще не создали. На рис. 3.57 видно, что документ теперь привязан как к журналу складских документов, так и к журналу "Развозка".

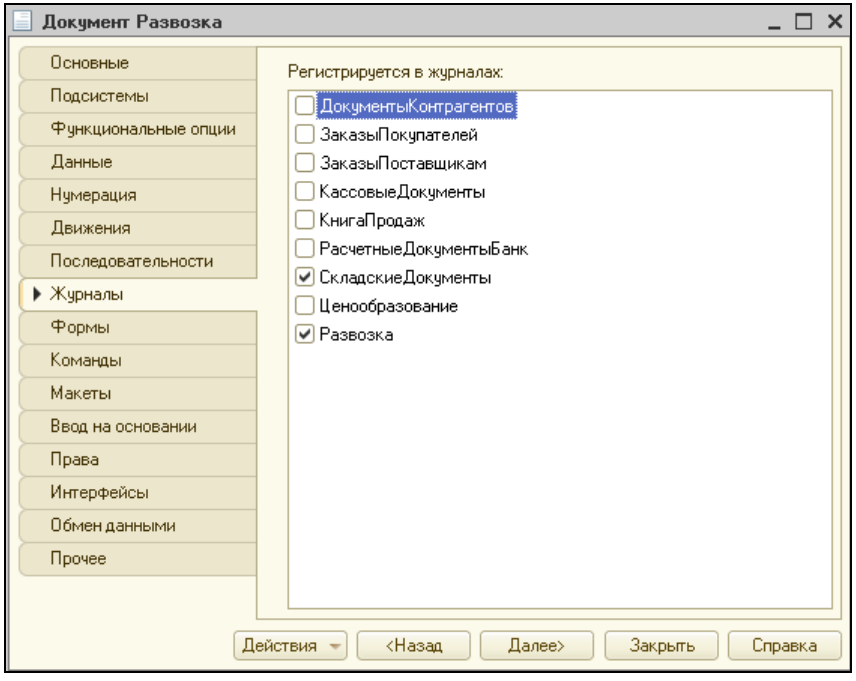



**Рис. 3.57.** Документ "Развозка", принадлежность к новому журналу добавлена автоматически

Теперь, при желании, мы можем отключить документ от журнала складских документов, поскольку создали специализированный журнал. А можем и оставить, как есть.

Документ сконструирован, но до завершения работы над ним еще далеко. Сохраните конфигурацию и перейдите в режим Предприятия. Попробуйте создать пробный документ "Развозка", заполните его данными.

Я на тестовой базе сформировал нечто подобное (рис. 3.58).

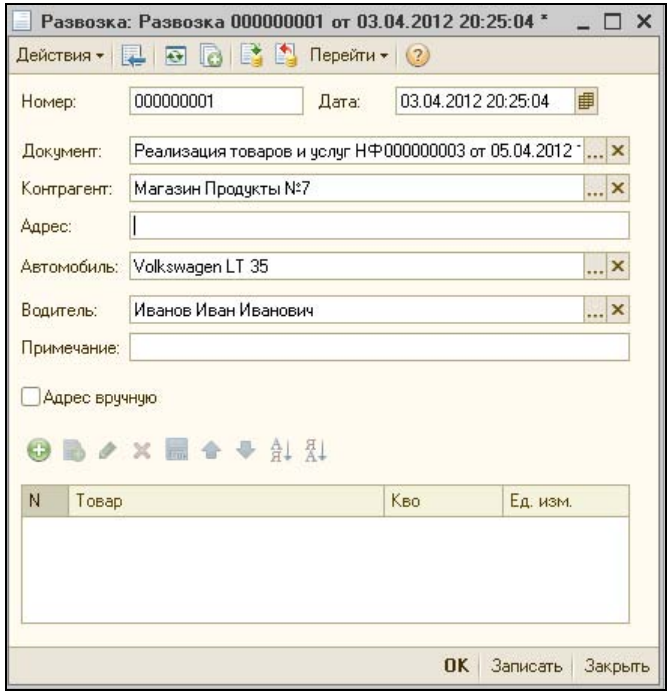



**Рис. 3.58.** Первый документ "Развозка"

При заполнении формы документа сразу бросаются в глаза недоработки.

1. Поля **Документ** и **Контрагент** не взаимосвязаны. Если выбрать другой документ, даже с другим контрагентом, то значение не меняется, а должно бы.
2. Поле **Контрагент** лучше сделать недоступным для пользователя, данные в него должны заполняться на основании поля **Документ.**
3. В поле **Адрес** хорошо бы подтягивать значение фактического адреса контрагента.
4. Флажок **Адрес вручную** предназначен для того, чтобы показывать, когда фактический адрес контрагента подтягивать не нужно (у контрагента ведь может быть несколько складов). Но функционал флажка нужно прописывать программно, это еще не сделано.
5. В табличную часть не загружается список номенклатуры и количества из накладной.

6. Документ нельзя распечатать, а ведь планировалось печатать товарно-транспортную накладную.

Наша задача сейчас — реализовать первые пять пунктов из списка. Печатную форму мы сформируем чуть позже, в разделе, посвященном работе с печатными формами (макетами).

Перейдем снова в режим Конфигуратора и откроем форму документа "Развозка" (ранее на рис. 3.34 мы ее рассматривали). Делаем двойной щелчок мышью по полю **Документ**. Справа откроется окно свойств поля. Опустим ползунок прокрутки в самый низ этого окна и увидим раздел **События**. Он нам и нужен (рис. 3.59).


**Рис. 3.59.** Нижняя часть окна свойств реквизита, раздел "События"

Как уже говорилось ранее, любой элемент формы имеет какие-то события: кнопку можно нажать, флажок можно установить или снять, в поле выбора можно выбрать значение... Все это события, к каждому из которых можно привязать программную процедуру. Именно этим мы сейчас и займемся.

Щелкните мышью по значку с изображением увеличительного стекла в правой части события `ПриИзменении`. Откроется модуль документа, в котором сразу автоматически будет создана пустая процедура:

```
Процедура ДокументПриИзменении(Элемент)
  // Вставить содержимое обработчика.
КонецПроцедуры
```

После этого снова перейдите на форму, откройте окно свойств поля **Контрагент** (двойным щелчком или правой кнопкой, а затем выберите пункт **Свойства**), найдите в окне свойств опцию **Доступность** и отключите ее.

Теперь сделайте двойной щелчок по флажку `АдресВручную`. Точно так же, как для поля ввода **Документ**, перейдите в нижнюю часть окна свойств и щелкните по значку с увеличительным стеклом в правой части события `ПриИзменении` (для флажка оно будет единственным). Будет автоматически создана еще одна пустая процедура:

```
Процедура АдресВручнуюПриИзменении(Элемент)
  // Вставить содержимое обработчика.
КонецПроцедуры
```

А теперь переписываем модуль документа так, чтобы у нас получились две заполненные процедуры и функция:

```
Функция ПолучитьАдресКонтрагента(Объект)
  ФункцияВернет = "";
  Запрос = Новый Запрос;
  Запрос.Текст = "
    |ВЫБРАТЬ РАЗРЕШЕННЫЕ
    | КонтактнаяИнформация.Тип КАК Тип,
    | КонтактнаяИнформация.Вид КАК Вид,
    | КонтактнаяИнформация.Представление КАК Представление
    |ИЗ
    | РегистрСведений.КонтактнаяИнформация КАК КонтактнаяИнформация
    |ГДЕ
    | КонтактнаяИнформация.Объект = &Объект";

  Запрос.УстановитьПараметр("Объект", Объект);
  РезультатПоиска =
Справочники.ВидыКонтактнойИнформации.НайтиПоКоду("000000002");
  Результат = Запрос.Выполнить().Выбрать();

  Пока Результат.Следующий() Цикл
    Если Результат.Тип<>Перечисления.ТипыКонтактнойИнформации.Адрес
    Тогда
      Продолжить;
    КонецЕсли;

    Если Результат.Вид = РезультатПоиска Тогда
      ФункцияВернет = Результат.Представление;
      Прервать;
    Иначе
      ФункцияВернет = "";
    КонецЕсли;
  КонецЦикла;
  Возврат ФункцияВернет;
КонецФункции

Процедура ДокументПриИзменении(Элемент)
  ЭлементыФормы.Контрагент.Значение = Документ.Контрагент.Ссылка;
  Если ЭлементыФормы.АдресВручную.Значение = Ложь Тогда
    ФАдрес = ПолучитьАдресКонтрагента(ЭлементыФормы.Контрагент.Значение);
    ЭлементыФормы.Адрес.Значение = ФАдрес;
  КонецЕсли;
КонецПроцедуры

Процедура АдресВручнуюПриИзменении(Элемент)
  Если ЭлементыФормы.АдресВручную.Значение = Истина Тогда
    ЭлементыФормы.Адрес.Значение = "";
```

```
  ИначеЕсли ЭлементыФормы.АдресВручную.Значение = Ложь Тогда
    ФАдрес = ПолучитьАдресКонтрагента(ЭлементыФормы.Контрагент.Значение);
    ЭлементыФормы.Адрес.Значение = ФАдрес;
  КонецЕсли;
КонецПроцедуры
```

Рассмотрим, что же мы такое написали.

В процедуре `ДокументПриИзменении()` мы заполняем поле **Контрагент** на форме, "вытягивая" его из реквизита **Контрагент**, выбранного документе. За это отвечает строка

```
ЭлементыФормы.Контрагент.Значение = Документ.Контрагент.Ссылка;
```

При этом, поскольку мы работаем с элементами формы, мы используем методы `Значение` и `Ссылка`.

Затем мы проверяем, установлен ли флажок `АдресВручную` (`Истина`) или нет (`Ложь`).

Если флажок не установлен (`Ложь`), то адрес мы заносим не вручную, и нам нужно записать его в поле **Адрес** из фактического адреса контрагента. Здесь есть нюанс — в типовых конфигурациях "1С:Предприятие 8.2" адрес в виде реквизитов документа не хранится, как это было в "1С:Предприятие 7.7". Вся контактная информация контрагентов находится в специальном регистре (многомерной таблице) КонтактнаяИнформация. О регистрах мы поговорим еще чуть позже. Напрямую к адресу мы обратиться не можем и пишем программный код, позволяющий получить адрес из регистра. Поскольку адрес можно получать в разных частях кода (в разных процедурах, например), то, чтобы не усложнять программный код, не дублировать его и не делать излишне громоздким, оформим получение адреса в виде обычной функции, к которой мы можем обратиться из любой части модуля. Это функция `ПолучитьАдресКонтрагента()`. В типовых конфигурациях есть подобная же типовая, в общем модуле `УправлениеКонтактнойИнформацией()`, но мы не стали усложнять пример и написали свою, простенькую функцию `ПолучитьАдресКонтрагента()`.

В функции `ПолучитьАдресКонтрагента()` используется запрос, синтаксис которого мы рассмотрим в *главе 4*. Пока же поясним его принцип работы (если читатель знаком с языком запросов SQL, то логика запроса ему будет вполне понятна). В запросе мы выбираем тип контактной информации (адрес, телефон и т. п.), вид (юридический адрес или фактический, рабочий телефон или мобильный и т. п.) и представление (собственно контактная информация) из регистра сведений, при условии, что объект выборки равен переданному объекту (контрагенту). Запрос отбирает значение только типа "Адрес" и вида "000000002" (в справочнике контактной информации — код фактического адреса контрагента). В итоге функция возвращает найденный результат.

В процедуре `АдресВручнуюПриИзменении()` осуществляется проверка, установлен флажок или снят. Если флажок установлен, то значит, адрес будет вноситься вручную и поле адреса очищается. Если же флажок снят, то поле **Адрес** заполняется адресом контрагента с использованием функции `ПолучитьАдресКонтрагента()`.

С шапкой документа мы закончили. Теперь нужно запрограммировать заполнение табличной части из документа, который мы выбираем в шапке.

Возвратимся в модуль документа и поменяем процедуру `ДокументПриИзменении()` вот так:

```
Процедура ДокументПриИзменении(Элемент)
  ЭлементыФормы.Контрагент.Значение = Документ.Контрагент.Ссылка;

  Если ЭлементыФормы.АдресВручную.Значение = Ложь Тогда
    ФАдрес = ПолучитьАдресКонтрагента(ЭлементыФормы.Контрагент.Значение);
    ЭлементыФормы.Адрес.Значение = ФАдрес;
  КонецЕсли;

  ТТН.Очистить();
  Для Каждого ТекущаяСтрока Из Документ.Товары Цикл
    НоваяСтрокаТТН = ТТН.Добавить();
    НоваяСтрокаТТН.Товар = ТекущаяСтрока.Номенклатура;
    НоваяСтрокаТТН.Количество = ТекущаяСтрока.Количество;
    НоваяСтрокаТТН.ЕдиницаИзмерения = ТекущаяСтрока.ЕдиницаИзмерения;
  КонецЦикла
КонецПроцедуры
```

Что мы добавили? Цикл, перебирающий табличную часть **Товары документа**, выбранного в поле формы **Документ**. При каждом проходе цикла создается новая строка в таблице ТТН, в поле **Товар** записывается значение `ТекущаяСтрока.Номенклатура` (а `ТекущаяСтрока` — это строка документа, который перебирается циклом), так же из текущей строки документа в текущую строку ТТН записываются количества и единицы измерения. Перед циклом мы поставили команду `ТТН.Очистить()`, это сделано на случай, если пользователь выберет другой расходный документ на форме документа "Развозка". ТТН при этом нужно заполнить заново, а старые значения — удалить. Именно это мы и делаем.

Пока работа над документом завершена. Сохраняем конфигурацию, также не забываем обновить конфигурацию базы данных (напомню, это делается из пункта меню **Конфигурация** или при помощи кнопок на панелях инструментов).

Теперь давайте перейдем в режим Предприятия и внесем несколько документов развозки. Поэкспериментируйте с документами, поменяйте документ и посмотрите, как будет меняться адрес и товарный состав в табличной части. У меня готовый документ выглядит примерно так, как показано на рис. 3.60.

Наш новый документ выглядит уже вполне презентабельно, но доработан он еще не полностью. Во-первых, его еще нельзя печатать. Во-вторых, хоть в нем и можно сохранять данные, но основной механизм хранения данных — регистры (а для бухгалтерских документов — проводки) — в нем никак не задействован. Далее в этой главе мы доработаем документ и устраним эти недостатки. Сейчас же давайте рассмотрим некоторые основные методы работы с документами. Для наглядности и отработки использования методов на практике перейдите в режим Предприятия и создайте несколько документов "Развозка". Вот так у меня выглядят в списке созданные документы (рис. 3.61).

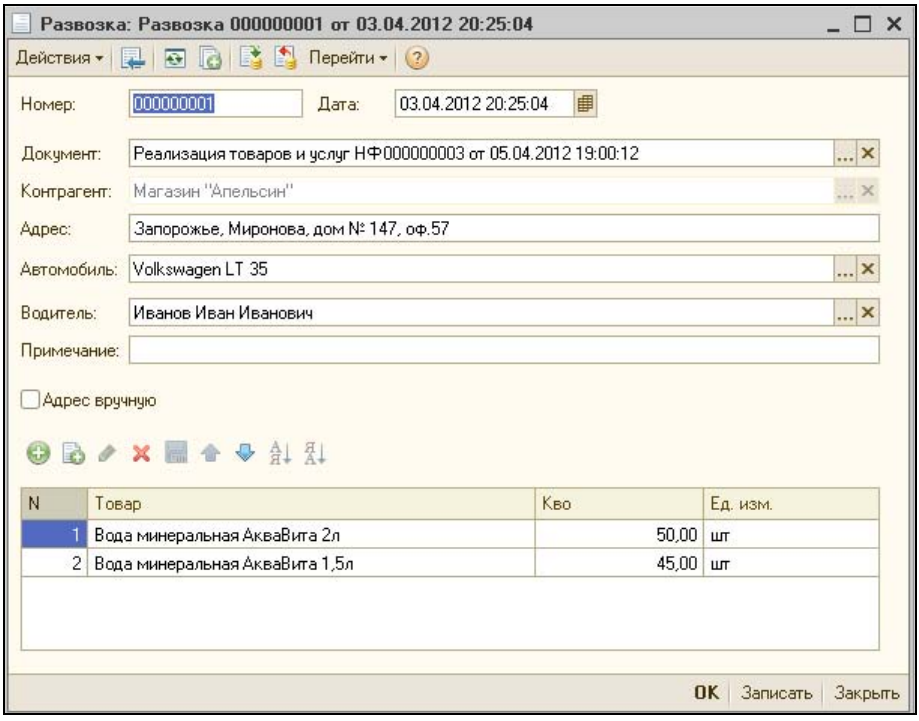



**Рис. 3.60.** Почти готовый документ "Развозка". Осталось совсем немного

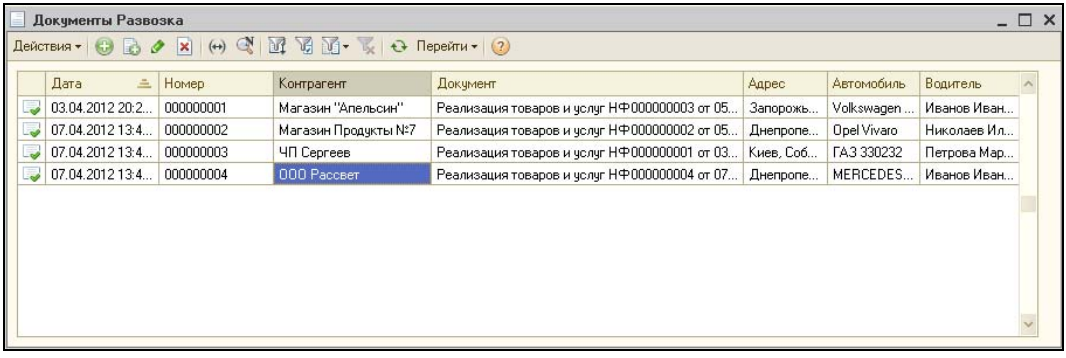



**Рис. 3.61.** Образец документов "Развозка" в списке

## Методы работы с документами

Рассмотрим некоторые важные и применимые методы работы с документами в системе "1С:Предприятие".

- `Выбрать`. Формирует выборку документов за заданный отрезок времени.

  Синтаксис:

  **`Выбрать([ДатаНачала][, ДатаОкончания][, Отбор][, Порядок])`**

Здесь: *ДатаНачала* — начало периода; *ДатаОкончания* — конец периода; *Отбор* задает поле и значение отбора открываемой выборки; *Порядок* — реквизит, по которому сортируется выборка, и направление выборки (может принимать значение "Возр" и "Убыв", по умолчанию отбор идет по возрастанию). Например, параметр *Порядок* может принимать значение "Дата Убыв", где Дата — реквизит документа, по которому будем сортировать, Убыв — сортировка по убыванию.

Пример:

```
Выборка = Документы.Развозка.Выбрать();
Пока Выборка.Следующий() Цикл
  Сообщить(Выборка);
  Сообщить(Выборка.Документ);
  Сообщить("------------------------------");
КонецЦикла;
```

Этот пример выбирает документы "Развозка" без параметров, за весь возможный период. Пока выборка не закончится, мы сообщаем документ выборки, документ реализации, по которому производится развозка и который внесен в документ развозки, а также разделитель "---------------" для лучшей читаемости выводимого результата.

Если читатель выполнил указанные в книге примеры, то, запустив этот код в обработке "для экспериментов", в результате получит:

```
Развозка 000000001 от 03.04.2012 20:25:04
Реализация товаров и услуг НФ000000003 от 05.04.2012 19:00:12
------------------------------
Развозка 000000002 от 07.04.2012 13:43:27
Реализация товаров и услуг НФ000000002 от 05.04.2012 18:57:48
------------------------------
Развозка 000000003 от 07.04.2012 13:44:24
Реализация товаров и услуг НФ000000001 от 03.03.2012 2:04:25
------------------------------
Развозка 000000004 от 07.04.2012 13:44:50
Реализация товаров и услуг НФ000000004 от 07.04.2012 13:41:46
------------------------------
```

Это те самые четыре документа развозки, которые мы видели на рис. 3.61

Пример:

```
Выборка = Документы.Развозка.Выбрать(НачДата, КонДата,,"Дата Убыв");
Пока Выборка.Следующий() Цикл
  Сообщить(Выборка);
  Сообщить(Выборка.Документ);
  Сообщить("------------------------------");
КонецЦикла;
```

Пример, аналогичный предыдущему, но выборка производится за период времени, определяемый двумя переменными — НачДата и КонДата (они могут задаваться прямо в программном коде или быть вынесены на форму, чтобы пользо-

ватель сам проставил нужные даты). Сортировка выборки ведется по дате документов в порядке убывания.

- `ПустаяСсылка`. Получает пустое значение ссылки на документ заданного типа.

  Синтаксис:

  **`ПустаяСсылка()`**

  Пример:

```
Док = Документы.ЗаказПокупателя.ПустаяСсылка();
```

- `НайтиПоНомеру`. Выполняет поиск документа по номеру.

  Синтаксис:

  **`НайтиПоНомеру(`*`НомерДокумента`*`[, `*`ДатаИнтаервала`*`])`**

  Здесь: *`НомерДокумента`* — искомый номер; *`ДатаИнтервала`* — необязательный параметр, дата из интервала, в котором проводится поиск по номеру. Сам интервал определяется как период уникальности номеров документа, в который входит указанная дата. Параметр используется для документов с периодической нумерацией. Предположим, если нумерация документов уникальна в течение месяца, то поиск будет проводиться в рамках того месяца, к которому относится указанная дата. Например, если указана дата 15 июня, то поиск будет проводиться с 1 по 30 июня.

  Пример:

```
Поиск = Документы.Развозка.НайтиПоНомеру("000000001");
Если Поиск.Пустая() = Ложь Тогда   // Пустая() означает, что
                                   // элемент не найден
  Сообщить(Поиск.Дата);  // Если нашли, сообщим дату документа
КонецЕсли;
```

  Ищем документ "Развозка" с номером 000000001. Если поиск прошел успешно (ссылка результата не пуста), то выдаем дату найденного документа.

- `НайтиПоРеквизиту`. Поиск документа по заданному реквизиту.

  Синтаксис:

  **`НайтиПоРеквизиту(`*`ИмяРеквизита`*`, `*`ЗначениеРеквизита`*`)`**

  Здесь: *`ИмяРеквизита`* — имя реквизита, по которому будем искать; *`ЗначениеРеквизита`* — искомое значение реквизита.

  Пример:

```
Спр = Справочники.Автомобили.НайтиПоНаименованию(СокрЛП("Opel Vivaro"));
Док = Документы.Развозка.НайтиПоРеквизиту("Автомобиль", Спр);
Сообщить(Док.Номер + " от " + Док.Дата);
```

  В этом примере мы позиционируем переменную `Спр` на элемент "Opel Vivaro" справочника "Автомобили", а затем ищем документ развозки, в котором этот автомобиль был задействован, и сообщаем номер документа и его дату.

  Результат: `000000002 от 07.04.2012 13:43:27`.

***Примечание***

Если бы в предыдущем примере нашей целью было не найти один документ, а, предположим, отобрать все документы развозки, где был использован автомобиль "Opel Vivaro", то программный код следовало бы построить по-другому, с использованием цикла. Например, так:

```
Спр = Справочники.Автомобили.НайтиПоНаименованию(СокрЛП("Opel Vivaro"));
Док = Документы.Развозка.Выбрать();
Пока Док.Следующий() Цикл
  Если Док.Автомобиль = Спр Тогда
    Сообщить(Док.Номер + " от " + Док.Дата);
  КонецЕсли;
КонецЦикла;
```

- `СоздатьДокумент`. Создает новый документ.

  Синтаксис:

  **`СоздатьДокумент()`**

  Пример:

```
Док = Документы.ЗаказПокупателя.ПустаяСсылка();
```

- `ПустаяСсылка`. Получает пустое значение ссылки на документ заданного типа.

  Синтаксис:

  **`ПустаяСсылка()`**

  Пример:

```
Спр = Справочники.Автомобили.НайтиПоНаименованию(СокрЛП("MERCEDES BENZ
1838 LS"));
Водитель = Справочники.ФизическиеЛица.НайтиПоНаименованию("Николаев Илья
Петрович");
РезультатПоискаДокумента =
Документы.РеализацияТоваровУслуг.НайтиПоНомеру("НФ000000004",
ТекущаяДата());

ДокументРеализации = РезультатПоискаДокумента.Ссылка.ПолучитьОбъект();
Док = Документы.Развозка.СоздатьДокумент();
Док.Дата = ТекущаяДата();
Док.Документ = ДокументРеализации.Ссылка;
Док.Контрагент = Док.Документ.Контрагент;
Док.Автомобиль = Спр;
Док.Водитель = Водитель;
Док.АдресВручную = Истина;

Для Каждого ТекущаяСтрока Из ДокументРеализации.Товары Цикл
  НоваяСтрокаТТН = Док.ТТН.Добавить();
  НоваяСтрокаТТН.Товар = ТекущаяСтрока.Номенклатура;
  НоваяСтрокаТТН.Количество = ТекущаяСтрока.Количество;
```

```
  НоваяСтрокаТТН.ЕдиницаИзмерения = ТекущаяСтрока.ЕдиницаИзмерения;
КонецЦикла;

Док.Записать(РежимЗаписиДокумента.Проведение);
```

В этом примере мы программно создаем и заполняем документ "Развозка", который затем записываем в режиме проведения. Параметр `РежимЗаписиДокумент` может принимать три значения: `Проведение`, `ОтменаПроведения` и `Запись`.

***Примечание***

Если бы в приведенном примере в строке

```
РезультатПоискаДокумента =
Документы.РеализацияТоваровУслуг.НайтиПоНомеру("НФ000000004",
ТекущаяДата());
```

не был использован второй параметр (`ТекущаяДата()`), то документ не был бы найден. Это связано с тем, что в этом документе установлена периодичность в пределах года (нумерация начинается сначала раз в год). Если бы периодичности не было, правильно было бы писать так:

```
РезультатПоискаДокумента =
Документы.РеализацияТоваровУслуг.НайтиПоНомеру("НФ000000004");
```

- `УстановитьПометкуУдаления`. Помечает документ на удаление. Здесь *`ПометкаУдаления`* — параметр, который может принимать значение `Истина` или `Ложь`. Если `Истина` — документ будет помечен на удаление. Если `Ложь` — пометка на удаление будет снята.

Синтаксис:

**`УстановитьПометкуУдаления(`*`ПометкаУдаления`*`)`**

Пример:

```
Док = Документы.Развозка.НайтиПоНомеру("000000014");
Если Док.Пустая() = Ложь Тогда
  ДокОбъект = Док.ПолучитьОбъект();
  ДокОбъект.УстановитьПометкуУдаления(Истина);
КонецЕсли;
```

В этом примере мы ищем документ развозки по номеру. В случае, если нашли — получаем объект и помечаем его на удаление.

***А как перебрать документы и по каждому из них перебрать строки?***

```
ВыбираемыйДокумент = Документы.Развозка.Выбрать();
Пока ВыбираемыйДокумент.Следующий() Цикл
  Если ВыбираемыйДокумент.Проведен = Ложь Тогда
    // непроведенные и помеченные на удаление пропускаем
    Продолжить;
  КонецЕсли;
  Сообщить("В документе "+ВыбираемыйДокумент+" перевезен товары: ");
  Для Каждого ТекущаяСтрока Из ВыбираемыйДокумент.ТТН Цикл
```

```
    // организовали вложенный цикл
    Сообщить(ТекущаяСтрока.Товар);
  КонецЦикла
КонецЦикла
```

### А КАК ВЫГРУЗИТЬ ТАБЛИЧНУЮ ЧАСТЬ ДОКУМЕНТА?

```
ВыбранныйДокумент = Документы.Развозка.НайтиПоНомеру("000000002");
ТаблицаПеревозки = ВыбранныйДокумент.ТТН.Выгрузить();
```

### А КАК ПРОГРАММНО СКОПИРОВАТЬ ДОКУМЕНТ?

```
ВыбранныйДокумент = Документы.Развозка.НайтиПоНомеру("000000002");
КопияДокумента = ВыбранныйДокумент.Скопировать();
КопияДокумента.Дата = ТекущаяДата();
КопияДокумента.Записать();
```

### А КАК ПОСЧИТАТЬ ИТОГ ПО КОЛОНКЕ СУММА В ТАБЛИЧНОЙ ЧАСТИ ДОКУМЕНТА?

```
ВыбранныйДокумент =
Документы.РеализацияТоваровУслуг.НайтиПоНомеру("НФ000000004",
ТекущаяДата());
ВсегоПоКолонкеСумма = ВыбранныйДокумент.Товары.Итог("Сумма");
Сообщить(ВсегоПоКолонкеСумма);
```

### А КАК РАСПРОВЕСТИ ДОКУМЕНТ?

```
Док = Документы.Развозка.НайтиПоНомеру("000000015");
Если Док.Пустая() = Ложь Тогда
  ДокОбъект = Док.ПолучитьОбъект();
  Если Док.Проведен = Истина Тогда
    ДокОбъект.Записать(РежимЗаписиДокумента.ОтменаПроведения);
  КонецЕсли;
КонецЕсли;
```

В следующем разделе мы продолжим конструирование нашего нового документа "Развозка", но теперь мы не будем довольствоваться простым сохранением данных в полях документа. Мы научимся задействовать в своем документе специальные средства хранения данных, внесенных в базу данных документами. А именно — регистры и проводки.

# Хранение данных, или регистры

В оперативном учете первичные данные, внесенные в систему посредством документов, хранятся в регистрах. *Регистр* представляет собой многомерную таблицу, в которой хранятся данные и откуда их можно извлечь в удобном для пользователя виде посредством отчетов. Конечно, информацию в отчетах можно получить и путем обработки документов. Но очень часто, особенно при большом документообороте, такой метод чересчур громоздок, занимает слишком много времени и машинных ресурсов.

Регистр по своей структуре не должен быть чересчур сложен. Проектировать его следует так, чтобы получить информацию из него было несложно.

С регистром непосредственно связаны такие понятия, как *измерения* и *ресурсы*.

*Измерения* определяют то, в каких разрезах мы храним информацию. Например, мы можем хранить ее в разрезе складов (сколько товара находится на том или ином складе) или фирм (сколько каждая из наших фирм должна поставщикам), или товаров. Измерение — это "что учитываем".

*Ресурсы* определяют то, что хранится в регистре, конкретные количественные или суммовые данные, например, количество товаров или денежные суммы. Ресурс — это "сколько того, что мы учитываем".

Можно сказать так, что к каждому *измерению* регистра относится некоторое количество *ресурсов*.

Например, к каждому складу (склад — это измерение) относится некоторое количество (количество — это ресурс) товара (товар — это тоже измерение).

Регистры бывают разных видов.

- Регистры сведений 1С — таблицы, для хранения различной информации, наподобие таблиц MS Excel. В регистрах сведений можно, например, хранить информацию о ценах и скидках номенклатуры по разным прайс-листам или информацию о курсах валют.
- Регистры накопления 1С — таблицы, в которых хранятся остатки, обороты и накапливаемые итоги. Например, если у нас было некоторого товара 20 штук и 3 штуки были проданы, то итоговый остаток, 17 штук, будет храниться в регистре накопления.
- Регистры бухгалтерии 1С — таблицы, основанные на бухгалтерских планах счетов. Такие таблицы используются для ведения бухгалтерского учета, именно в регистры бухгалтерии записываются бухгалтерские проводки.
- Регистры расчетов 1С — таблицы, основанные на планах видов расчетов. Такие таблицы используются для ведения учета по начислению заработной платы.

В системе "1С:Предприятие 7.7" регистры и проводки являли собой различные объекты дерева метаданных. В системе "1С:Предприятие 8.2" бухгалтерские проводки записываются в один из видов регистров: регистры бухгалтерии. Упомянем вкратце о принципах построения бухгалтерского учета и структуре бухгалтерских проводок.

*Бухгалтерский учет* — это упорядоченная система сбора, регистрации и обобщения информации в денежном выражении о состоянии имущества, обязательств организации и их изменениях (движении денежных средств) путем сплошного, непрерывного и документального учета всех хозяйственных операций.

Объектами бухгалтерского учета являются: имущество организаций, их обязательства и хозяйственные операции, осуществляемые организациями в процессе их деятельности.

Основными задачами бухгалтерского учета являются:

- формирование полной и достоверной информации о деятельности организации и ее имущественном положении;
- обеспечение информацией, необходимой внутренним и внешним пользователям бухгалтерской отчетности для контроля над соблюдением законодательства при осуществлении организацией хозяйственных операций и их целесообразностью, наличием и движением имущества и обязательств, использованием материальных, трудовых и финансовых ресурсов в соответствии с утвержденными нормами, нормативами и сметами;
- предотвращение отрицательных результатов хозяйственной деятельности организации и выявление внутрихозяйственных резервов обеспечения ее финансовой устойчивости.

Бухгалтерский учет ведется в соответствии с утвержденным Законодательством планом счетов на основе принципа двойной записи.

*Бухгалтерский счет* — это способ группировки и отражения в учете отдельных видов средств, их источников и хозяйственных процессов. То есть объект бухгалтерской аналитики. Совокупность таких объектов, охватывающая все сферы деятельности предприятия и установленная законодательно, представляет собой *план счетов*.

Разумеется, в деятельности отдельно взятого предприятия весь план счетов никогда не используется. Кто-то занимается производством и торговлей, кто-то — оказанием услуг, а кто-то — выращиванием нутрий. В каждом отдельно взятом случае будет задействован не весь план счетов, а лишь часть его.

Каждому бухгалтерскому счету присвоен свой номер и название, установленные Законодательством.

В плане счетов выделяются и группируются экономически однородные счета, например счета учета основных средств (10), подразделяющиеся на 10.1 (земельные участки), 10.2 (капитальные затраты по улучшению земель), 10.3 (дома и сооружения), 10.4 (машины и оборудование) и т. д., или например 36 (расчеты с покупателями и заказчиками), подразделяющиеся на 36.1 (расчеты с отечественными покупателями), 36.2 (расчеты с иностранными покупателями) и т. д.

В конфигурациях, использующих планы счетов, они доступны в меню **Операции | Планы счетов**. В конфигурации "Управление торговлей", которую я взял в качестве примера, планы счетов не используются, поэтому в списке планов счетов пусто.

Движение денежных средств между бухгалтерскими счетами называется *проводкой* и может быть записано, например, следующим образом: Д31.1/К36.1. Данная проводка представляет собой поступление в банк оплаты от покупателя.

Д31.1 означает "дебет 31.1" — деньги пришли на счет 31.1 (текущие счета в национальной валюте), т. е. *дебет* проводки — это счет, *на который* приходят деньги.

К36.1 означает "кредит 36.1" — деньги пришли со счета 36.1 (расчеты с отечественными покупателями), т. е. *кредит* проводки — это счет, *с которого* приходят деньги.

*Сальдо* — это остаток денег на счете, он может быть как положительным, так и отрицательным. Остаток получается сложением всех сумм, которые приходили на счет и которые уходили с него.

Некоторые счета всегда имеют положительный остаток (например, деньги на расчетном счете или в кассе, товары на складе). Такие счета являются *активными*, и сальдо у них *дебетовое*.

Некоторые счета всегда имеют отрицательный остаток (например, реализация товаров, на склад товар пришел по одной цене, продался уже с другой, более высокой, т. е. со счета "Реализация товаров" уйдет больше денег, чем придет на него). Такие счета являются *пассивными*, и сальдо у них *кредитовое*.

Наконец, часть счетов может иметь положительное сальдо, а может и отрицательное (например, какой-либо контрагент может задолжать нам, а можем и мы ему). Такие счета называются *активно-пассивными*.

Если сложить сальдо по всем счетам, сумма обязательно должна быть равна нулю. Если это так, то бухгалтерский баланс сходится.

Остаток по счетам позволяет нам видеть итоговую картину: сумму товаров на складах, сумму денег на расчетных счетах и в кассе, задолженность перед поставщиками и т. д. Однако этого недостаточно. Нам также важно видеть сумму не в общем, а по конкретному складу или даже товару, задолженность не перед всеми поставщиками, а перед конкретным. Для этого используются единицы аналитического учета, которые называются *субконто*. Например, для счета 28.1 (товары на складе) может иметься два субконто — Номенклатура (т. е. товары) и Склады.

Итак, самая различная информация, как для оперативного учета, так и для бухгалтерского, внесенная документами, может храниться в регистрах. Давайте рассмотрим основные методы работы с регистрами, а затем модернизируем наш учебный документ "Развозка" таким образом, чтобы он формировал записи в регистры.

## Методы работы с регистрами

В этом разделе мы рассмотрим некоторые распространенные и употребимые методы работы с регистрами.

### Регистры сведений

- `Выбрать`. Формирует выборку записей из регистра.

  Синтаксис:

  **`Выбрать([НачалоИнтервала][, КонецИнтервала][, Отбор][, Порядок])`**

  Здесь: *`НачалоИнтервала`* — начало интервала выборки; *`КонецИнтервала`* — конец интервала выборки; *`Отбор`* задает поле и значение отбора открываемой выборки; *`Порядок`* — реквизит или измерение, по которому сортируется выборка, и направление выборки (может принимать значение `"Возр"` и `"Убыв"`, по умолчанию отбор идет по возрастанию).

Синтаксис:

**Выбрать([*Отбор*][, *Порядок*])**

Вариант синтаксиса для непериодического регистра значений.

Пример:

```
Организация = Справочники.Организации.НайтиПоКоду("000000001");
Выборка = РегистрыСведений.КонтактнаяИнформация.Выбрать(Новый
Структура("Объект", Организация));
Пока Выборка.Следующий() Цикл
  Если Строка(Выборка.Вид) = "Юридический адрес организации" Тогда
    ЮрАдрес = Выборка.Представление;
    Сообщить(ЮрАдрес);
  ИначеЕсли Строка(Выборка.Вид) = "Почтовый адрес организации" Тогда
    ПАдрес = Выборка.Представление;
    Сообщить(ПАдрес);
  ИначеЕсли Строка(Выборка.Вид) = "Телефон организации" Тогда
    Телефон = Выборка.Представление;
    Сообщить(Телефон);
  КонецЕсли;
КонецЦикла;
```

В этом примере мы перебираем регистр контактной информации для организации, которую ищем по заданному коду. Найденную контактную информацию выводим в виде сообщений.

- `ВыбратьПоРегистратору`. Формирует выборку записей из регистра по указанному регистратору.

  Синтаксис:

  **ВыбратьПоРегистратору(*Регистратор*)**

  Здесь *Регистратор* — документ, создавший запись в регистре.

- `Получить`. Получает ресурсы регистра сведений по указанным ключевым полям.

  Синтаксис:

  **Получить(*Период* [, *Отбор*])**

  Здесь: *Период* — временной период, на который получаем данные; *Отбор* — структура, содержащая отбор по измерениям регистра.

## Регистры накопления

- `Выбрать`. Формирует выборку записей из регистра.

  Синтаксис:

  **Выбрать([*НачалоИнтервала*][, *КонецИнтервала*][, *Отбор*][, *Порядок*])**

  Здесь: *НачалоИнтервала* — начало интервала выборки; *КонецИнтервала* — конец интервала выборки; *Отбор* задает поле и значение отбора открываемой выборки; *Порядок* — реквизит или измерение, по которому сортируется выборка, и на-

правление выборки (может принимать значение "Возр" и "Убыв", по умолчанию отбор идет по возрастанию).

Пример:

```
УчетНоменклатуры = РегистрыНакопления.УчетНоменклатуры;
ОтборПоТовару = Новый Структура("Номенклатура");
ОтборПоТовару.Номенклатура = ВыбТовар;
НачДата = НачалоГода(ТекущаяДата());
КонДата = ТекущаяДата();
Выборка = УчетНоменклатуры.Выбрать(НачДата,КонДата,ОтборПоТовару);
Расход =0;
Пока Выборка.Следующий() Цикл
  Если Выборка.ВидДвижения = ВидДвиженияНакопления.Расход Тогда
    Расход = Расход + Выборка.Количество;
  КонецЕсли;
КонецЦикла;
Сообщить("Отгружено с начала года """ + СокрЛП(ВыбТовар) + """ =
"+ Расход + " шт.");
```

В этом примере мы рассчитываем количество отгруженного с начала товара. Товар можно задать как поиском по коду/наименованию, так и выбором через элемент управления ВыбТовар на форме.

- ВыбратьПоРегистратору. Формирует выборку записей из регистра по указанному регистратору.

  Синтаксис:

  **ВыбратьПоРегистратору(*Регистратор*)**

  Здесь *Регистратор* — документ, создавший запись в регистре.

- Обороты. Получает обороты регистра накопления за заданный период времени.

  Синтаксис:

  **Обороты([*НачалоПериода*][, *КонецПериода*][, *Отбор*][, *Измерения*][, *Ресурсы*])**

  Здесь: *НачалоПериода* — начало временного интервала, в котором надо получить обороты; *КонецПериода* — конец временного интервала, в котором надо получить обороты; *Отбор* — структура, содержащая отбор по измерениям регистра; *Измерения* — список измерений регистра, для которых нужно разворачивать обороты; *Ресурсы* — список ресурсов, для которых нужно получить обороты.

- Остатки. Получает остатки регистра накопления на заданный момент времени.

  Синтаксис:

  **Остатки([*МоментВремени*][, *Отбор*][, *Измерения*][, *Ресурсы*])**

  Здесь: *МоментВремени* — дата, на которую получаем остатки; *Отбор* — структура, содержащая отбор по измерениям регистра; *Измерения* — список измерений регистра, для которых нужно получить остатки; *Ресурсы* — список ресурсов, для которых нужно получить остатки.

Пример:

```
Фильтр = Новый Структура;
Фильтр.Вставить("Контрагент",ТекущаяСтрока.Контрагент);
Сум=РегистрыНакопления.ВзаиморасчетыСКонтрагентами.Остатки
  (ПериодРегистрации,Фильтр);.
```

Получаем остатки регистра накопления по взаиморасчетам с контрагентами.

- `ПересчитатьИтоги`. Выполняет полный пересчет итогов регистра накопления.

  Синтаксис:

  **`ПересчитатьИтоги()`**

  Пример:

  ```
  Регистры.Остатки.ПересчитатьИтоги();
  ```

- `Добавить`. Добавляет новую запись в набор записей регистра.

  Синтаксис:

  **`Добавить()`**

- `ДобавитьПриход`. Добавляет новую запись регистра накопления в наборе с установленным видом движения "Приход".

  Синтаксис:

  **`ДобавитьПриход()`**

  Пример:

  ```
  Движение = Движения.УчетНоменклатуры.ДобавитьПриход();
  Движение.Период = Дата;
  Движение.Номенклатура = СтрокаСостава.Номенклатура;
  Движение.Склад = Склад;
  Движение.Количество = СтрокаСостава.Количество;
  ```

- `ДобавитьРасход`. Добавляет новую запись регистра накопления в наборе с установленным видом движения "Расход".

  Синтаксис:

  **`ДобавитьРасход()`**

- `Выгрузить`. Создает таблицу значений, в которую выгружает из регистра массив данных по заданным строкам и колонкам.

  Синтаксис:

  **`Выгрузить([`*`Строки`*`][, `*`Колонки`*`])`**

  Здесь: *`Строки`* — строки выгрузки; *`Колонки`* — колонки выгрузки.

  Если строки и колонки не указаны, выгружается весь набор записей.

  Пример:

  ```
  Рег = РегистрыНакопления.Затраты;
  НаборЗаписей = Рег.СоздатьНаборЗаписей();
  НаборЗаписей.Отбор.Документ.Значение = Ссылка;
  ОсновныеДанные = НаборЗаписей.Выгрузить();
  ```

- Загрузить. Загружает набор записей значениями из переданной таблицы значений.

  Синтаксис:

  **Загрузить(*ТаблицаЗначений*)**

  Пример:

```
Рег = РегистрыНакопления.Продажи;
НаборЗаписей = Рег.СоздатьНаборЗаписей();
Таб = Новый ТаблицаЗначений;
// ...
// здесь формируется таблица значений
// ...
НаборЗаписей.Загрузить(Таб);
```

- Итог. Суммирует значения всех строк набора записей регистра по указанной колонке.

  Синтаксис:

  **Итог(*ИмяПоля*)**

- Количество. Получает количество записей в наборе.

  Синтаксис:

  **Количество()**

  Пример:

```
Рег = РегистрыНакопления.ВзаиморасчетыСКонтрагентами;
Сообщить(Рег.Обороты().Количество());
```

- Прочитать. Считывает записи по установленному отбору.

  Синтаксис:

  **Прочитать()**

- Удалить. Удаляет запись из регистра накопления по указанному индексу или объекту записи регистра накопления.

  Синтаксис:

  **Удалить(*Запись*)**

  Здесь *Запись* — индекс записи, которая будет удаляться, либо сама запись.

## Регистры бухгалтерии

- Выбрать. Формирует выборку записей из регистра бухгалтерского учета.

  Синтаксис:

  **Выбрать([*НачалоИнтервала*][, *КонецИнтервала*][, *Отбор*][, *Порядок*])**

  Здесь: *НачалоИнтервала* — начало интервала выборки; *КонецИнтервала* — конец интервала выборки; *Отбор* задает поле и значение отбора открываемой выборки;

*Порядок* — реквизит или измерение, по которому сортируется выборка, и направление выборки (может принимать значение "Возр" и "Убыв", по умолчанию отбор идет по возрастанию).

- ВыбратьПоРегистратору. Формирует выборку записей из регистра бухгалтерского учета по указанному регистратору.

  Синтаксис:

  **ВыбратьПоРегистратору(*Регистратор*)**

  Здесь *Регистратор* — документ, создавший запись в регистре.

- Обороты. Получает обороты по регистру бухгалтерии на заданный момент. Обороты получаются в разрезе заданных измерений и по заданным ресурсам. В измерения входят счет, субконто, измерения, корр. счет, кор. субконто, кор. измерения.

  Синтаксис:

  **Обороты([*НачалоПериода*][, *КонецПериода*][, *ВидыСубконто*]**
  **[, *ВидыКорСубконто*][, *Отбор*][, *Измерения*][, *Ресурсы*])**

  Здесь: *НачалоПериода* — начало временного периода выборки; *КонецПериода* — конец временного периода выборки; *ВидыСубконто* — используемые виды субконто; *ВидыКорСубконто* — используемые виды корреспондирующих субконто; *Отбор* — структура, содержащая отбор по измерениям регистра; *Измерения* — измерения, для которых получаем обороты; *Ресурсы* — ресурсы, для которых получаем обороты.

- ОборотыДтКт. Получает обороты по регистру бухгалтерии на заданный момент. Обороты получаются в разрезе заданных измерений и по заданным ресурсам. В измерения входят счет дебета, субконто дебета, счет кредита, субконто кредита, измерения кредита, измерения (для балансовых) и измерения дебета, измерения кредита (для небалансовых).

  Синтаксис:

  **Обороты([*НачалоПериода*][, *КонецПериода*][, *ВидыСубконтоДт*]**
  **[, *ВидыСубконтоКт*][, *Отбор*][, *Измерения*][, *Ресурсы*])**

  Здесь: *НачалоПериода* — начало временного периода выборки; *КонецПериода* — конец временного периода выборки; *ВидыСубконтоДт* — дебетовые субконто; *ВидыСубконтоКт* — кредитовые субконто; *Отбор* — структура, содержащая отбор по измерениям регистра; *Измерения* — измерения, для которых получаем обороты; *Ресурсы* — ресурсы, для которых получаем обороты.

- Остатки. Получает остатки регистра накопления на заданный момент времени.

  Синтаксис:

  **Остатки([*МоментВремени*][*ВидыСубконто*][, *Отбор*][, *Измерения*][, *Ресурсы*])**

  Здесь: *МоментВремени* — дата, на которую получаем остатки; *ВидыСубконто* — используемые виды субконто; *Отбор* — структура, содержащая отбор по измерени-

ям регистра; *Измерения* — список измерений регистра, для которых нужно получить остатки; *Ресурсы* — список ресурсов, для которых нужно получить остатки.

- Добавить. Добавляет новую запись регистра в набор данных.

  Синтаксис:

  **Добавить()**

- ДобавитьДебет. Добавляет новую запись с видом движения "Дебет" в набор данных.

  Синтаксис:

  **ДобавитьДебет()**

- ДобавитьКредит. Добавляет новую запись с видом движения "Кредит" в набор данных.

  Синтаксис:

  **ДобавитьКредит()**

- Итог. Суммирует значения всех строк в указанном поле набора записей.

  Синтаксис:

  **Итог(*ИмяПоля*)**

- Количество. Получает количество строк в наборе записей.

  Синтаксис:

  **Количество()**

- Очистить. Удаляет все записи из регистра бухгалтерского учета.

  Синтаксис:

  **Очистить()**

- Удалить. Удаляет запись с указанным индексом из набора данных.

  Синтаксис:

  **Удалить(*Индекс*)**

## Регистры расчета

- Выбрать. Формирует выборку записей из регистра расчета.

  Синтаксис:

  **Выбрать([*НачалоИнтервала*][, *КонецИнтервала*][, *Отбор*][, *Порядок*])**

  Здесь: *НачалоИнтервала* — начало интервала выборки; *КонецИнтервала* — конец интервала выборки; *Отбор* задает поле и значение отбора открываемой выборки; *Порядок* — реквизит или измерение, по которому сортируется выборка, и направление выборки (может принимать значение "Возр" и "Убыв", по умолчанию отбор идет по возрастанию).

- `ВыбратьПоРегистратору`. Формирует выборку записей из регистра расчета по указанному регистратору.

  Синтаксис:

  **ВыбратьПоРегистратору(*Регистратор*)**

  Здесь *Регистратор* — документ, создавший запись в регистре.

- `СоздатьНаборЗаписей`. Создает набор записей регистра расчета.

  Синтаксис:

  **СоздатьНаборЗаписей()**

- `Количество`. Получает количество записей в наборе.

  Синтаксис:

  **Количество()**

- `Очистить`. Удаляет все записи из регистра расчета.

  Синтаксис:

  **Очистить()**

- `Удалить`. Удаляет запись с указанным индексом из набора данных.

  Синтаксис:

  **Удалить(*Индекс*)**

# Разработка модуля проведения документа

Теперь, когда читатель получил некоторое количество теоретических знаний о регистрах, пора применить их на практике в нашем документе "Развозка", который мы создавали и дорабатывали на протяжении всей главы. В самом начале главы мы планировали учитывать в документе "Развозка" пройденное машиной расстояние и расход топлива, но так и не заложили необходимый функционал. Сейчас мы это сделаем.

Сначала давайте добавим в документ "Развозка" два реквизита: **Километраж** (Число 12.2) и **ЗатратыТоплива** (Число 12.2). После добавления реквизитов вынесите их на форму документа. На всякий случай напомню, что реквизиты нужно добавлять на вкладке **Данные** окна документа "Развозка", а формы модифицируются на вкладке **Формы**. Добавили?

Теперь давайте создадим новый регистр, в котором будем хранить данные по пройденному пути и затраченному топливу. Регистр создается, как и любой другой объект дерева конфигурации, щелчком правой кнопкой мыши по заголовку **Регистры накопления** в дереве конфигурации и выбором пункта **Добавить.**

Открывшееся окно нам уже вполне знакомо. Заполните его, как показано на рис. 3.62.

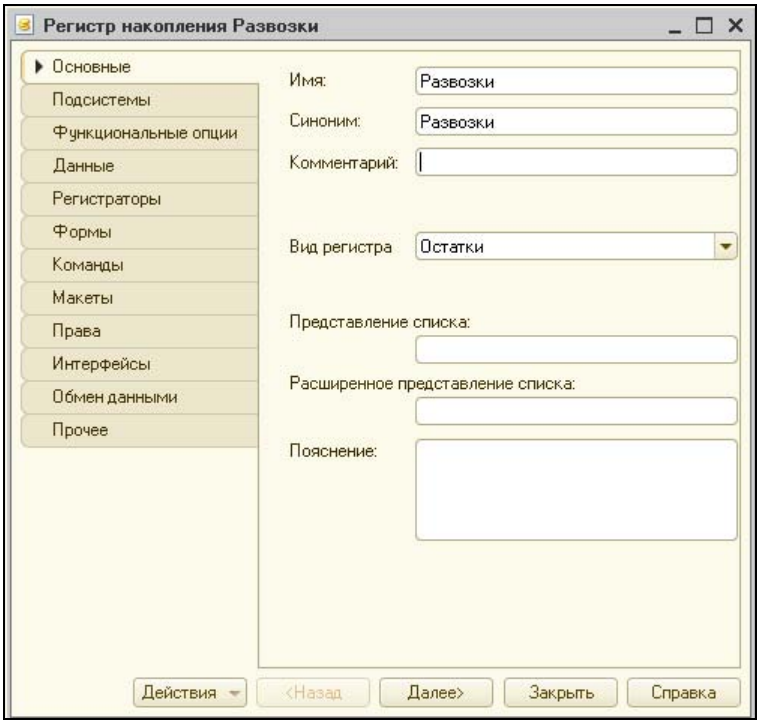



**Рис. 3.62.** Создание нового регистра накопления "Развозки"

Теперь мы должны сформировать структуру регистра, определить, каким образом в регистре будут храниться данные и какие именно. Перейдем на вкладку **Данные** и создадим измерения и ресурсы, как показано на рис. 3.63.

Добавление измерений и ресурсов стандартно, через контекстное меню или кнопку с плюсом. Хранить мы будем документ реализации товаров и услуг, связанный с документом отгрузки и являющийся его первопричиной (Документы.РеализацияТоваровУслуг), информацию об автомобиле (Справочники.Автомобили), а также водителе, осуществляющем развозку (Справочники.ФизическиеЛица). Числовые значения — **Километраж** и **Запас** топлива (Число 12.2).

Теперь нужно связать вновь созданный регистр с документом, который будет осуществлять запись в регистр. Переходим к документу "Развозка" на вкладку **Движения** (рис. 3.64).

В нижней части списка регистров появился еще один, наш регистр "Развозки". Его и выбираем, после чего становится активной кнопка **Конструктор движений**. С ее помощью мы сейчас создадим модуль проведения документа. Конечно, мы могли бы его создать и вручную (нажать кнопку **Действия** в окне документа и выбрать команду **Открыть модуль объекта**, а затем написать обработку проведения вручную), но зачем? Сейчас мы все сконструируем в автоматическом режиме. После нажатия кнопки откроется окно конструктора движений (рис. 3.65).

В левой части указан наш регистр накопления, в правой — реквизиты документа "Развозка". Центральная часть окна предназначена для связывания полей регистра

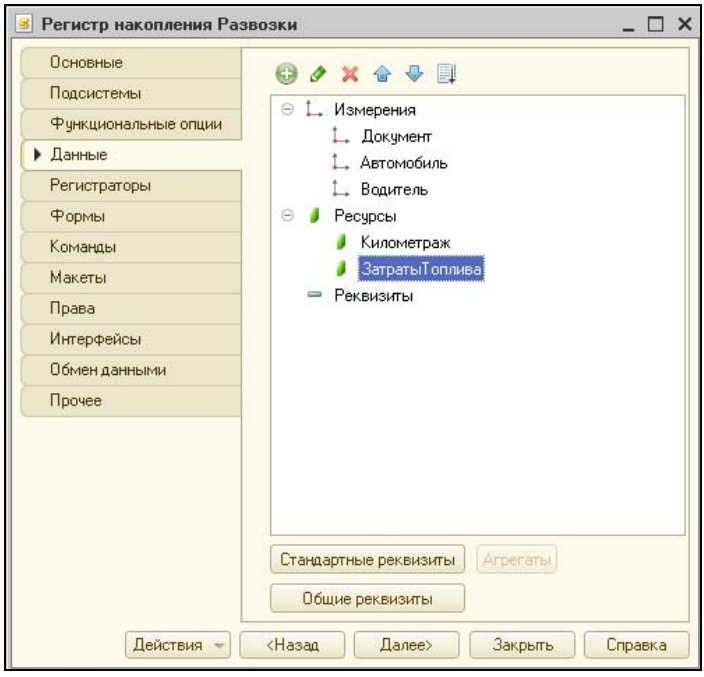



**Рис. 3.63.** Определяем, какие данные будут храниться в регистре

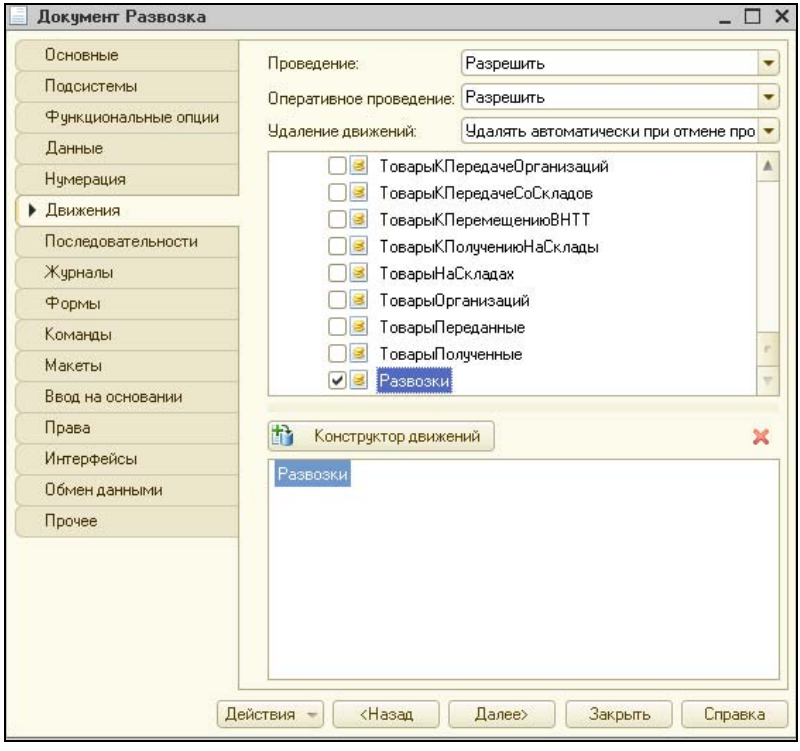



**Рис. 3.64.** Привязываем документ "Развозка" к новому регистру

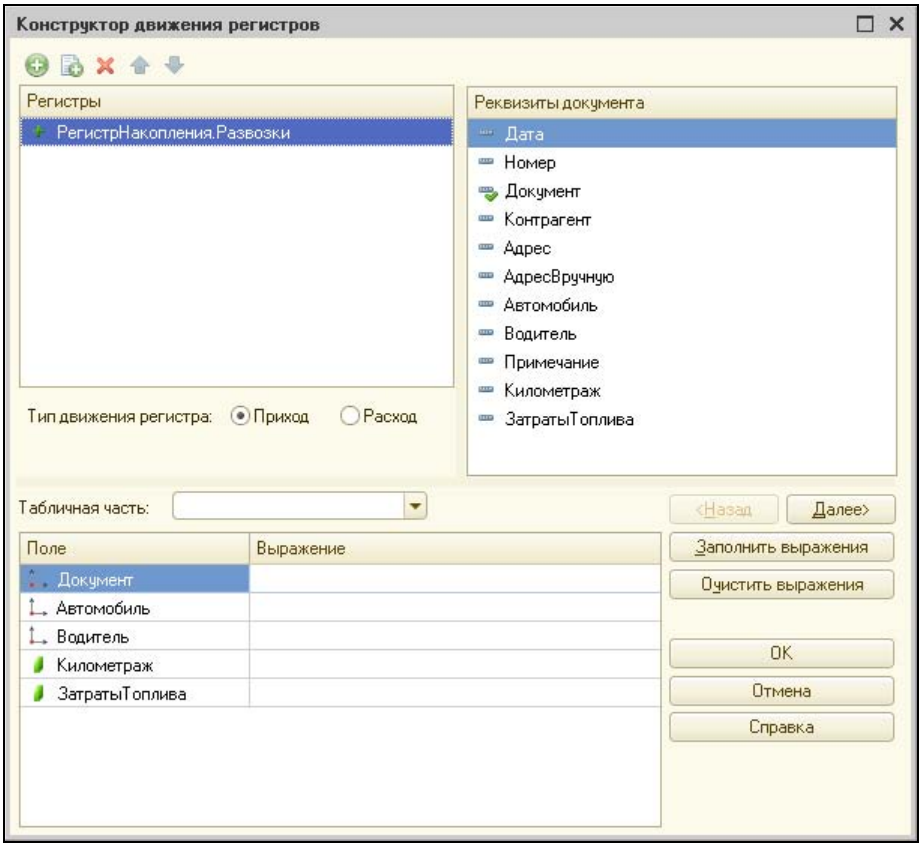



**Рис. 3.65.** Конструктор движения регистров

и документа. Из реквизитов документа мы видим только реквизиты шапки. Для того чтобы подключить также реквизиты табличной части, выберите табличную часть документа в одноименном выпадающем списке на форме конструктора, а затем нажмите кнопку **Заполнить выражения** для связывания полей документа и регистра. Результат представлен на рис. 3.66.

В списке реквизитов документа появились строки табличной части, а поля документа и регистра оказались связанными между собой.

> ***Примечание***
>
> Обратите внимание на выбор типа движения регистра в форме конструктора. Поскольку мы учитываем километраж, который увеличивается с каждой развозкой, а также затраты топлива, которые тоже увеличиваются, мы оставили тип **Приход**, как увеличивающееся значение. Если бы мы учитывали остатки топлива на складе, приходовали его специальным документом и списывали для развозок — регистр, учитывающий расход топлива, должен был бы иметь тип **Расход**.

Конструирование завершено, нажмем кнопку **OK** в окне конструктора, после чего попадаем в модуль документа с автоматически созданной обработкой проведения документа (рис. 3.67).

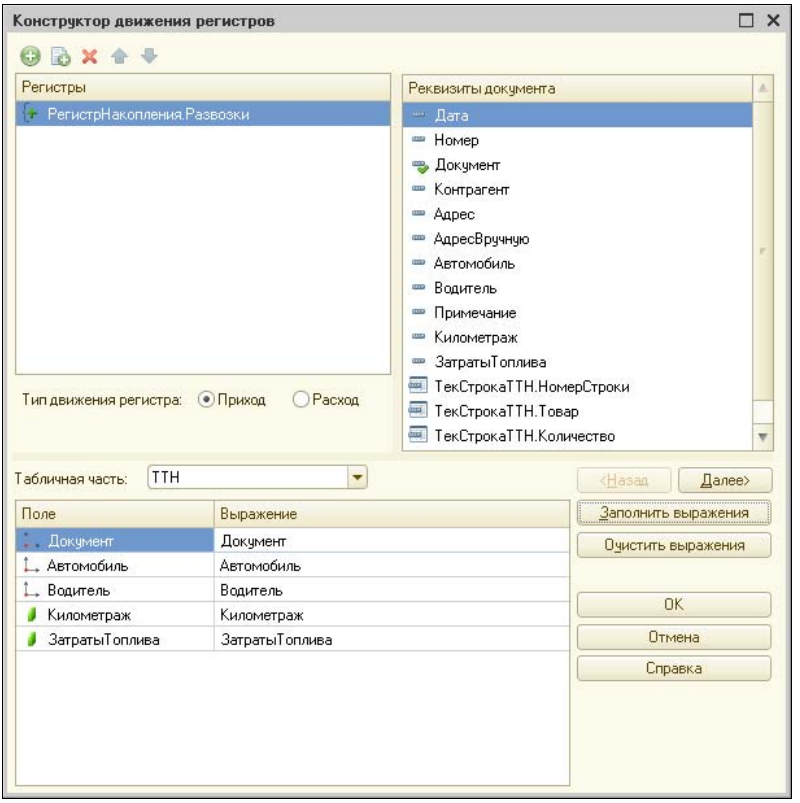



**Рис. 3.66.** Связали поля документа с полями регистра

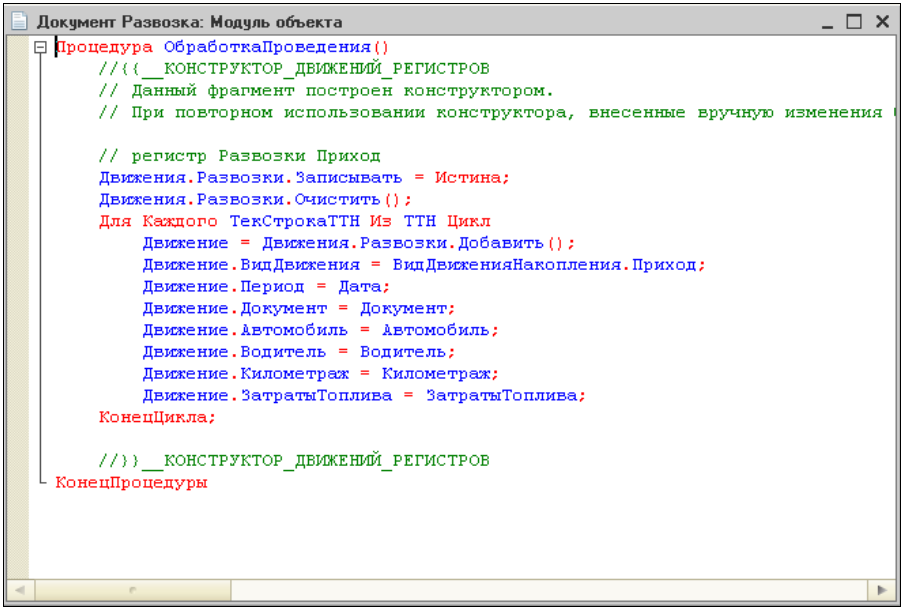



**Рис. 3.67.** Так выглядит автоматически созданная обработка проведения

Теперь давайте создадим пробный документ развозки и проведем его, а затем посмотрим, как же отразилось в регистрах то, что было внесено в документ. Для этого в режиме Предприятия выберем пункт меню **Операции | Отчет** и в списке отчетов — **Движения документов по регистрам** (рис. 3.68).

В окне формы отчета выберем нашу развозку (рис. 3.69).

После того как выбрали документ, нажмем кнопку **Сформировать**. В результате получим окно отчета с информацией по выбранному документу (рис. 3.70).

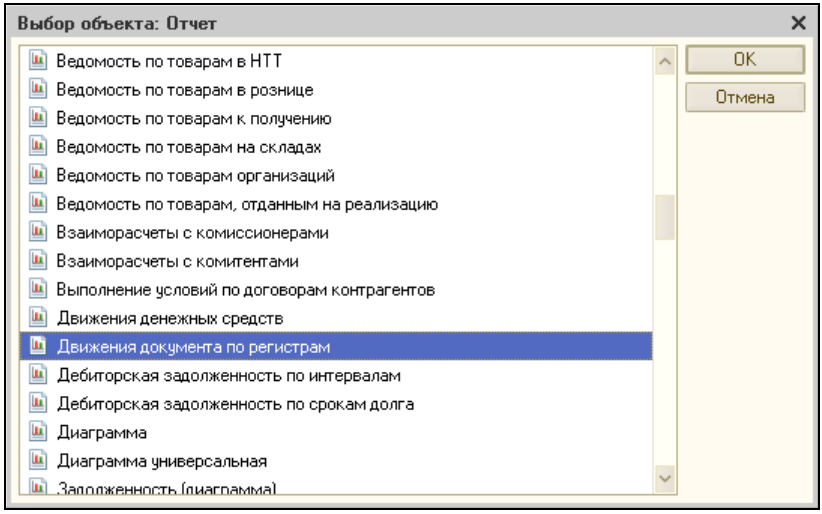



**Рис. 3.68.** Движения документа по регистрам

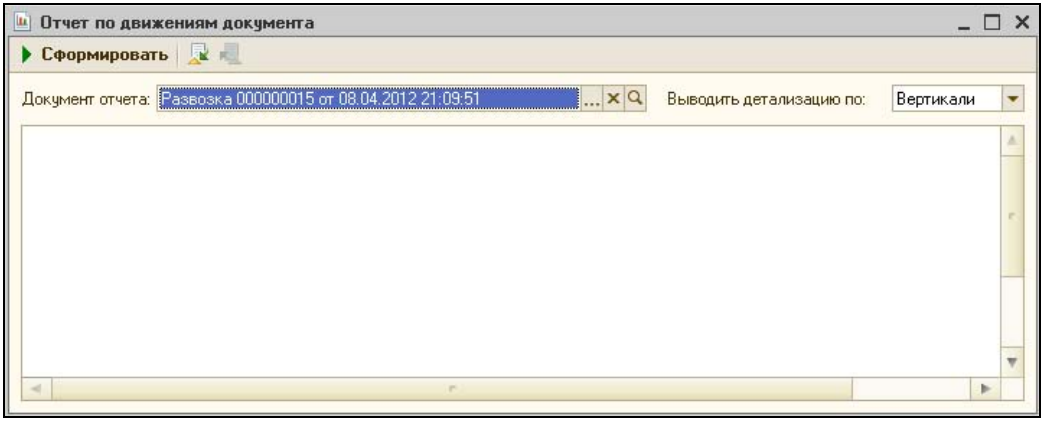



**Рис. 3.69.** Форма отчета по движению выбранного документа

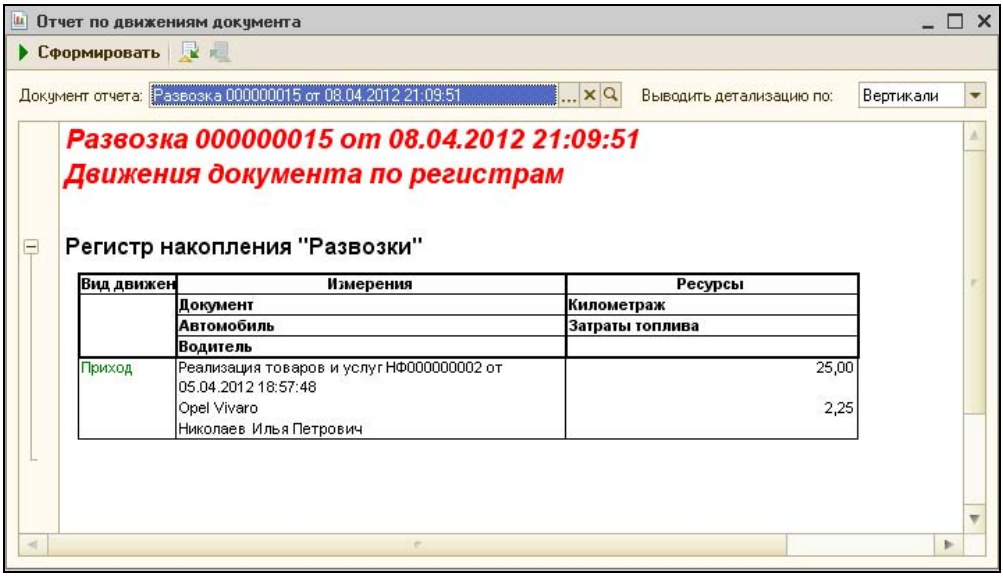



**Рис. 3.70.** Так выглядит информация, записанная в регистр при проведении документа "Развозка"

# Конструирование печатных форм (макетов)

Глава подходит к своему завершению, новый документ, который мы сконструировали, также вполне готов к использованию в учете. Единственное, что нам еще осталось сделать в нем, — это организовать возможность распечатать товарно-транспортную накладную.

Для конструирования печатной формы документа перейдем на вкладку **Макет** в окне документа. Список макетов пока что пуст (рис. 3.71).

Нажмем кнопку **Конструкторы** и в выпадающем списке выберем пункт **Конструктор печати**. Сначала нам предлагают выбрать, будет ли у нас обычная печатная форма или для управляемого режима (через Интернет). Выберем обычный режим и нажмем кнопку **Далее** (рис. 3.72).

На следующем шаге нам предложено выбрать модуль, в котором будет создана процедура печати документа (рис. 3.73).

Зададим создание процедуры в модуле формы и нажмем кнопку **Далее**, после чего нам потребуется указать, какие реквизиты должны печататься в шапке документа (рис. 3.74).

В левой части окна расположен список реквизитов документа, вправо кнопками со стрелками переносятся те из них, которые должны печататься в шапке. Далее мы выбираем реквизиты, которые будут печататься в табличной части (рис. 3.75).

Затем определяем, какие реквизиты будут печататься в подвале, т. е. области печатной формы, расположенной в нижней части печатного листа (рис. 3.76).

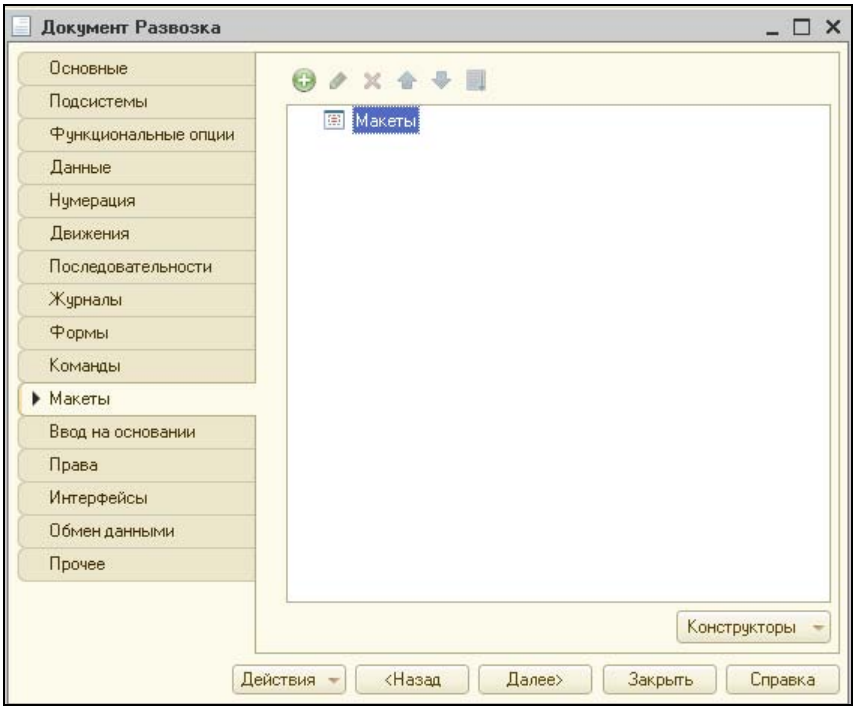



**Рис. 3.71.** Вкладка **Макет** — печатные формы

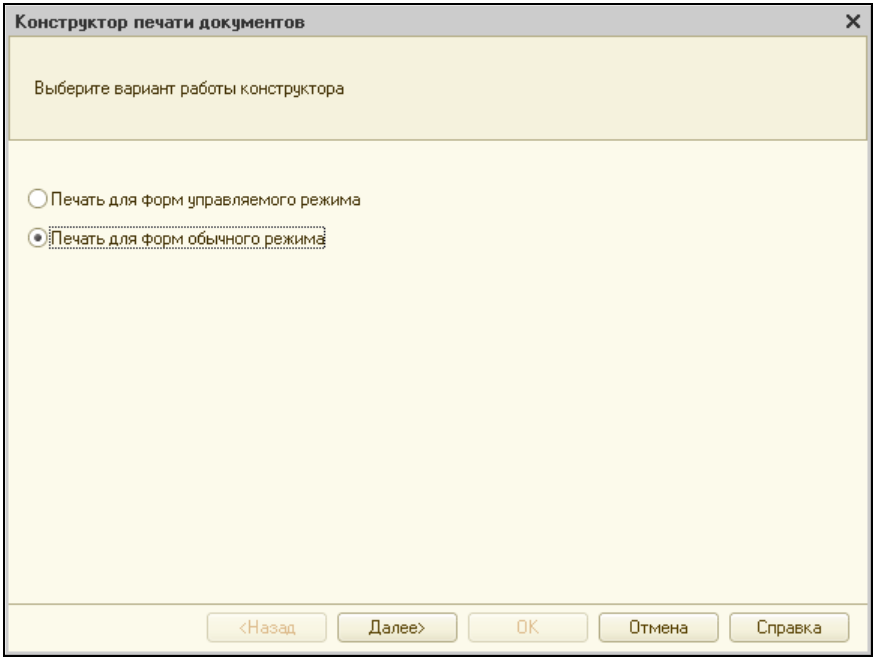



**Рис. 3.72.** Конструктор печати

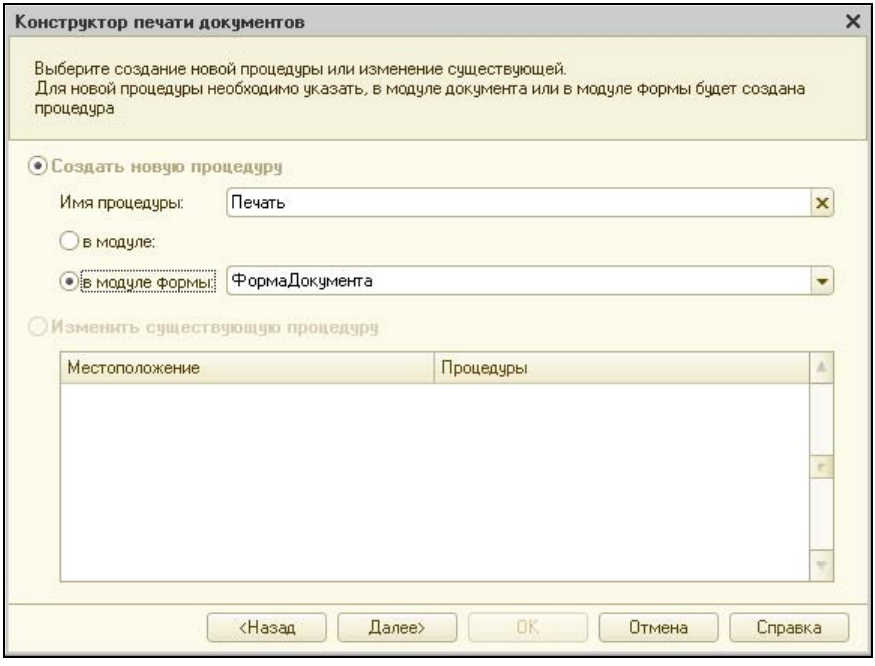



**Рис. 3.73.** Выбор местонахождения процедуры печати

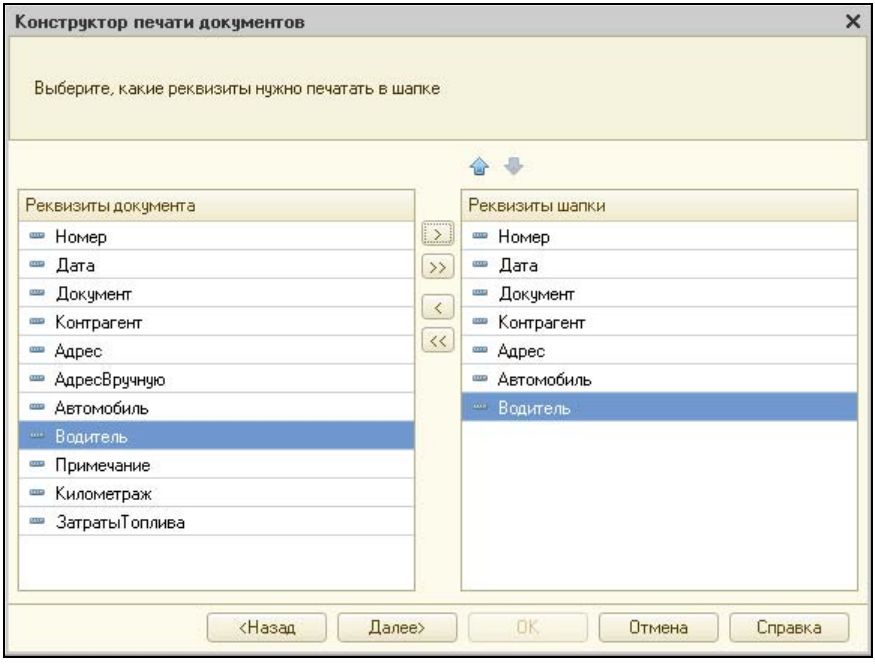



**Рис. 3.74.** Список реквизитов шапки

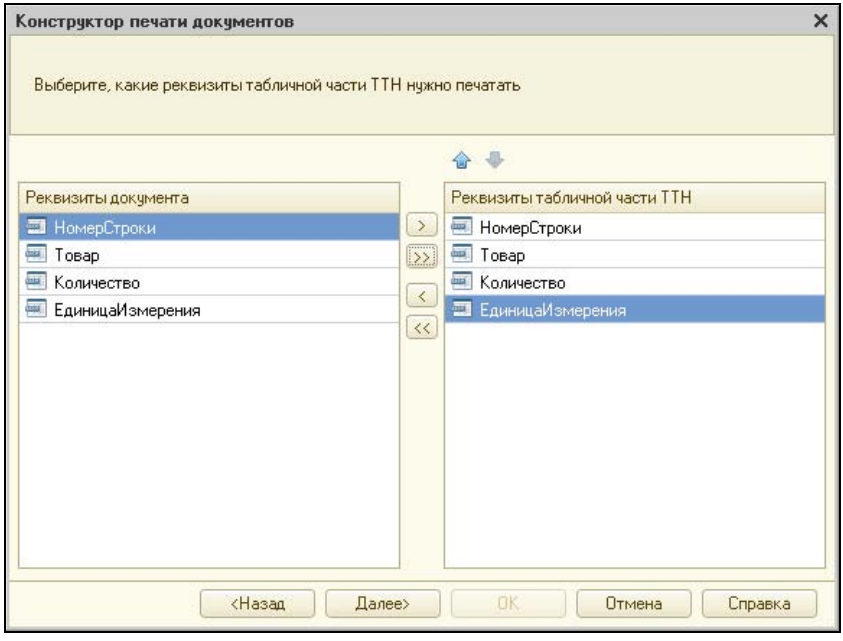



**Рис. 3.75.** Список реквизитов табличной части

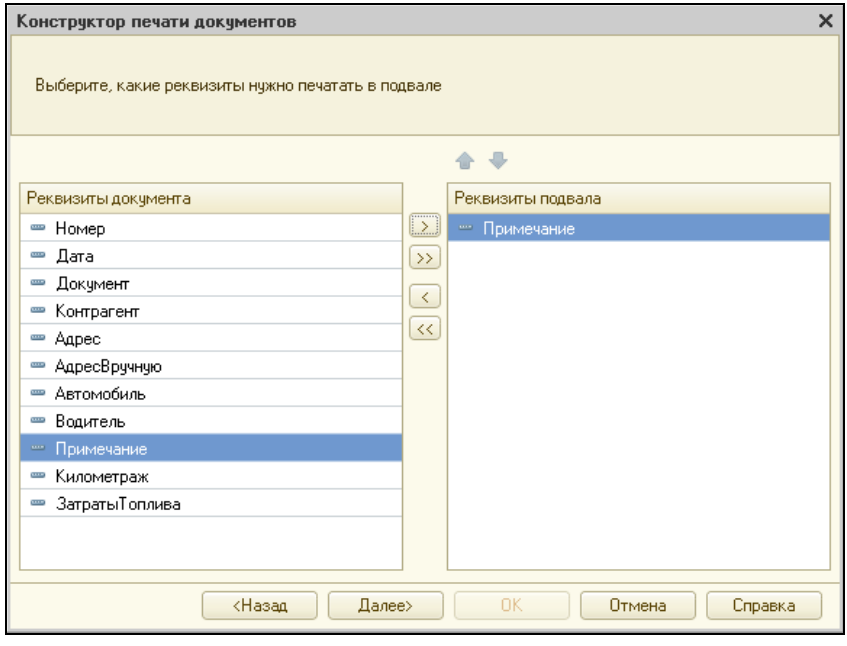



**Рис. 3.76.** Список реквизитов подвала (нижняя часть макета)

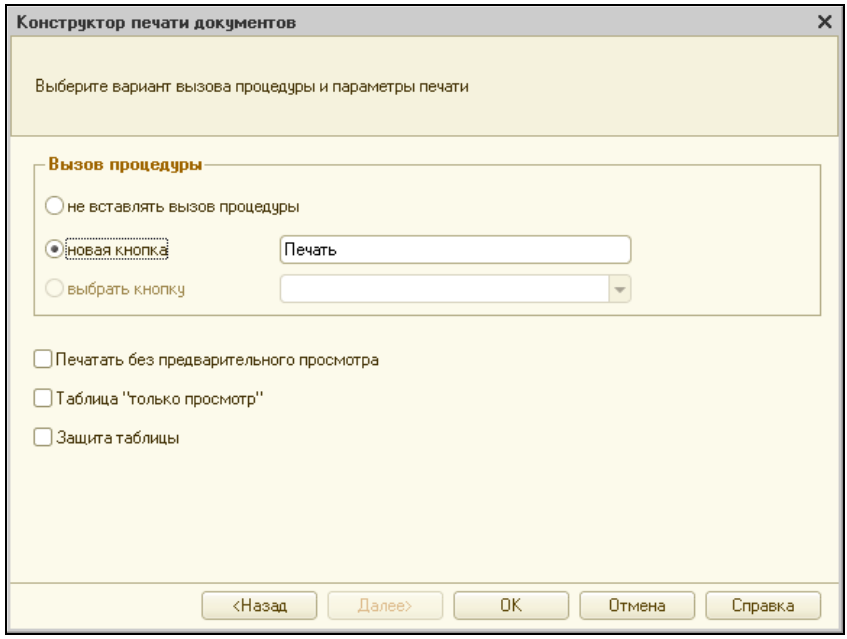



**Рис. 3.77.** Добавляем кнопку **Печать** на форму документа

После нажатия кнопки **Далее** определим, будет ли выведен вызов процедуры печати в виде кнопки на форме, а также некоторые второстепенные параметры, вроде защиты печатной формы от ручных изменений (рис. 3.77).

Конструирование печатной формы завершено. Нажмите кнопку **OK** для просмотра полученного результата (рис. 3.78).

Изображенный на рис. 3.78 макет сконструирован в автоматическом режиме. При необходимости его можно подправить вручную.

Сохраните конфигурацию и перейдите в режим Предприятия. Откройте ранее проведенный документ "Развозка". Теперь он завершен, кнопка **Печать** добавлена в автоматическом режиме при конструировании макета (рис. 3.79).

Если нажать кнопку **Печать**, то на печатную форму будет выведена печатная форма, которая изображена на рис. 3.80.

А что с процедурой печати? Процедура была создана автоматически при конструировании макета. Найти ее можно в модуле формы документа "Развозка":

```
Процедура Печать(Элемент)
  //{{_КОНСТРУКТОР_ПЕЧАТИ_ЭЛЕМЕНТ(Печать)
  // Данный фрагмент построен конструктором.
  // При повторном использовании конструктора
  // внесенные вручную изменения будут утеряны!!!
  ТабДок = Новый ТабличныйДокумент;
  Макет = Документы.Развозка.ПолучитьМакет("Печать");
  // Заголовок
  Область = Макет.ПолучитьОбласть("Заголовок");
  ТабДок.Вывести(Область);
```

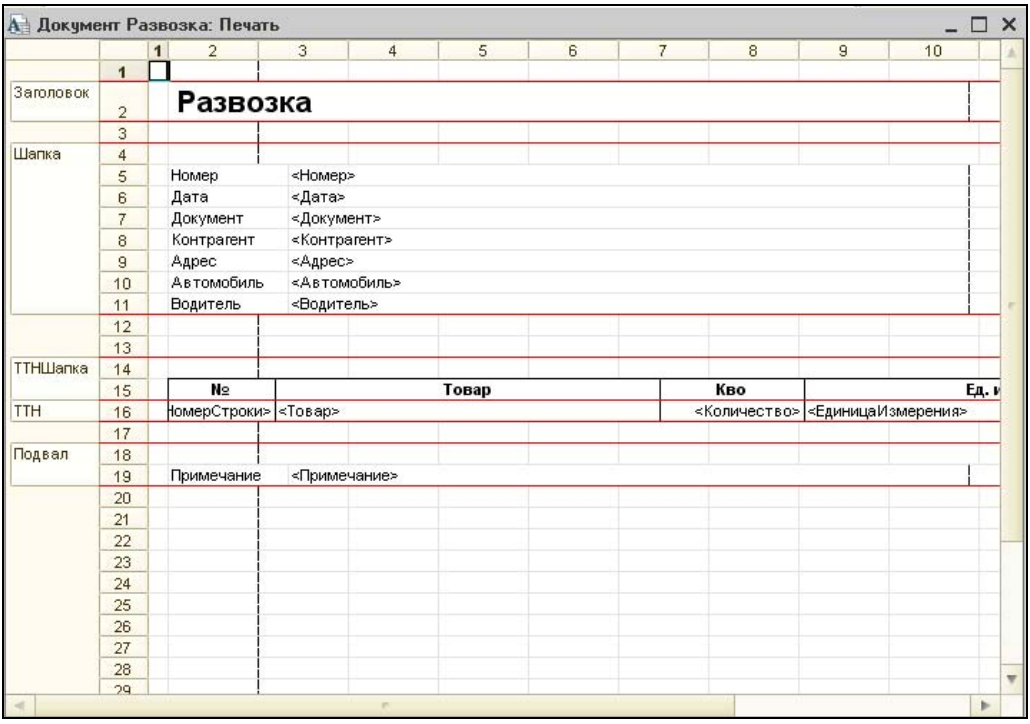



**Рис. 3.78.** Автоматически сконструированный макет

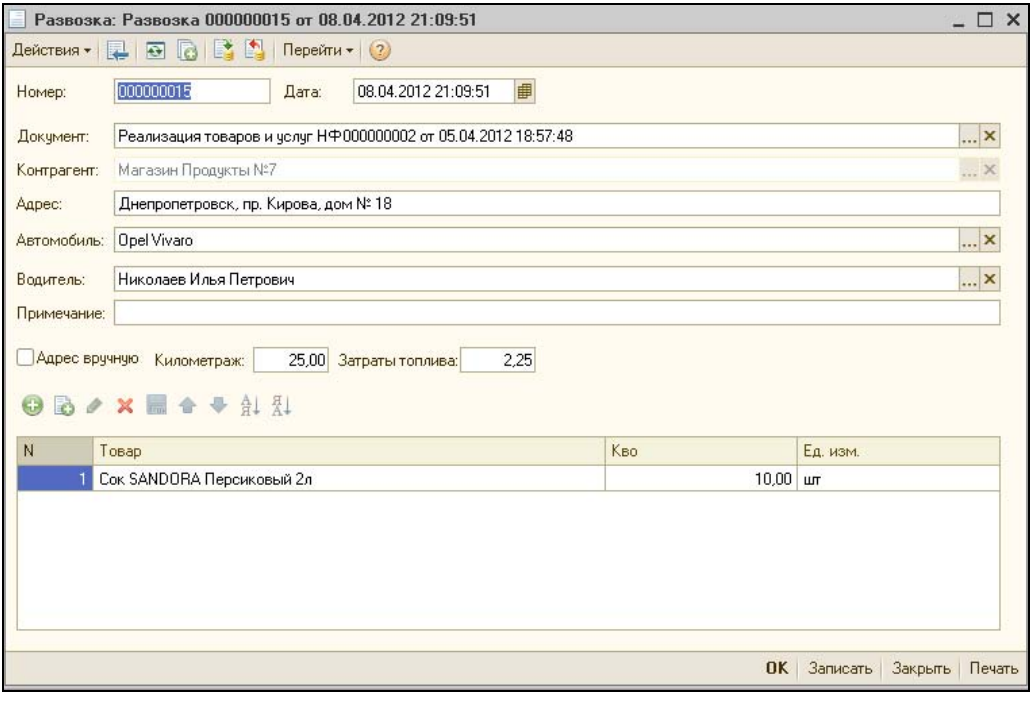



**Рис. 3.79.** Завершенный документ "Развозка"

```
  // Шапка
  Шапка = Макет.ПолучитьОбласть("Шапка");
  Шапка.Параметры.Заполнить(ЭтотОбъект);
  ТабДок.Вывести(Шапка);
  // ТТН
  Область = Макет.ПолучитьОбласть("ТТНШапка");
  ТабДок.Вывести(Область);
  ОбластьТТН = Макет.ПолучитьОбласть("ТТН");
  Для Каждого ТекСтрокаТТН Из ТТН Цикл
    ОбластьТТН.Параметры.Заполнить(ТекСтрокаТТН);
    ТабДок.Вывести(ОбластьТТН);
  КонецЦикла;
  // Подвал
  Подвал = Макет.ПолучитьОбласть("Подвал");
  Подвал.Параметры.Заполнить(ЭтотОбъект);
  ТабДок.Вывести(Подвал);

  ТабДок.ОтображатьСетку = Ложь;
  ТабДок.Защита = Ложь;
  ТабДок.ТолькоПросмотр = Ложь;
  ТабДок.ОтображатьЗаголовки = Ложь;
  ТабДок.Показать();
  //}}_КОНСТРУКТОР_ПЕЧАТИ_ЭЛЕМЕНТ
КонецПроцедуры
```

На этом мы завершаем работу над документом "Развозка", так же как и эту главу. В следующей же главе мы научимся конструировать отчеты различной сложности, как с использованием встроенного языка запросов, так и с помощью языка программирования системы "1С:Предприятие 8.2".


**Рис. 3.80.** Так выглядит автоматически сконструированный макет на печати

ГЛАВА 4

# Запросы и отчеты

В предыдущей главе вы познакомились с построением документов, изучили механизм их проведения и знаете, что документы являются первичным носителем информации об остатках и оборотах товарно-денежных средств. А как получать сводную, более-менее упорядоченную информацию? Нам мало знать, что такому-то контрагенту в течение месяца отгружалось некоторое количество накладных. Сколько продано товара? Какого? На какую сумму? Какую мы получили прибыль? Сколько нам должны и кто, сколько мы должны и кому? Эти и многие другие вопросы возникают постоянно, и на них не ответишь, просто просматривая документы в журнале. Необходим механизм, выбирающий данные из документов и из регистров, структурирующий и выводящий их на экран в удобном пользователю виде. Это и есть отчеты.

Отчеты в системе "1С:Предприятие" бывают двух видов: внутренние и внешние. Внутренние отчеты встроены в конфигурацию и находятся в дереве метаданных в группе **Отчеты**. Внешние отчеты — это обработки с расширением ert, отдельные файлы, которые запускаются через меню **Файл | Открыть.**

Обработка — понятие в некотором роде собирательное. Ее можно понимать, как форму или совокупность форм с блоком программного кода, она может храниться как в пределах конфигурации (внутренняя обработка), так и в виде отдельного epf-файла (внешняя обработка). Обработки могут являться отчетами, т. е. выбирать некоторые данные. А могут играть и совсем другую роль, например пометить на удаление все счета, выписанные покупателям, за указанный период времени. Или перенести некоторые данные из одной базы данных в другую. Или вывести список товаров, по которым не было продаж последние полгода. Или... В общем, обработки — мощный механизм и могут уметь многое, был бы написан соответствующий программный код.

## Простые выборки данных с использованием языка программирования 1С

Отчеты могут создаваться как вручную, так и с использованием мощных средств автоматизации, конструкторов запросов и печатных форм. При создании запросов

применяется специальный язык запросов. Однако отчеты, равно как и печатные формы для них, можно создавать и полностью в ручном режиме. В этом разделе мы рассмотрим создание простого отчета с использованием только языка программирования 1С и ручное конструирование печатной формы (макета), а в дальнейших разделах обсудим более "продвинутые" способы.

Итак, задача: нам нужно рассчитать затраты топлива на развозку товара каждой из машин предприятия за заданный период времени. Также нужно высчитать общее расстояние, которое прошла каждая из машин. Отчет должен выводиться на печать.

1. Создадим отчет в виде внешней обработки. Делается это в режиме Конфигуратора через пункт меню **Файл | Новый**. В открывшемся списке выбираем вариант **Внешняя обработка.**
2. Откроется окно создания обработки. Это стандартное окно нового объекта, такие мы уже видели неоднократно при работе с данной книгой. Сейчас нам нужно назначить обработке имя, синоним, а также создать форму (щелчок правой кнопкой мыши по слову "Формы" — и выбираем пункт **Добавить**, тип формы — "Форма отчета"). В результате окно новой обработки должно выглядеть так, как показано на рис. 4.1.

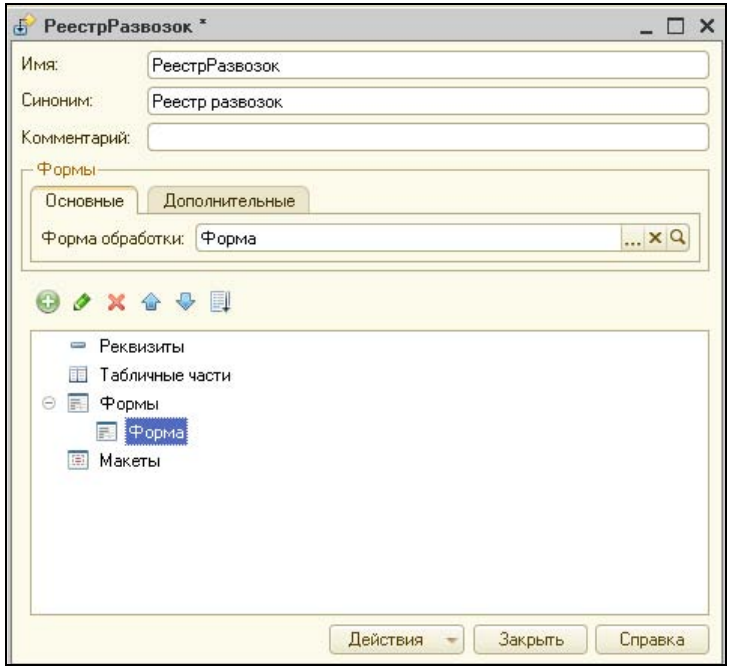



**Рис. 4.1.** Создаем новую обработку

3. Теперь двойным щелчком переходим на форму и начинаем заполнять ее элементами. Для вставки различных элементов формы существует панель кнопок **Элементы управления** (обычно расположена в нижней части окна, но может быть и сверху, как поместите). На этой панели расположены заготовки для самых

разных элементов — полей ввода, кнопок, флажков, надписей, таблиц и т. д. Можно просто выбирать их мышью, затем щелкать по форме — и элемент создан, остается только задать его свойства. Мы воспользуемся другой кнопкой — **Вставить элемент управления**  . По сути, это то же самое, только вместо выбора из большого количества кнопок мы нажимаем одну кнопку, а потом в диалоговом окне (рис. 4.2) выбираем элемент управления.

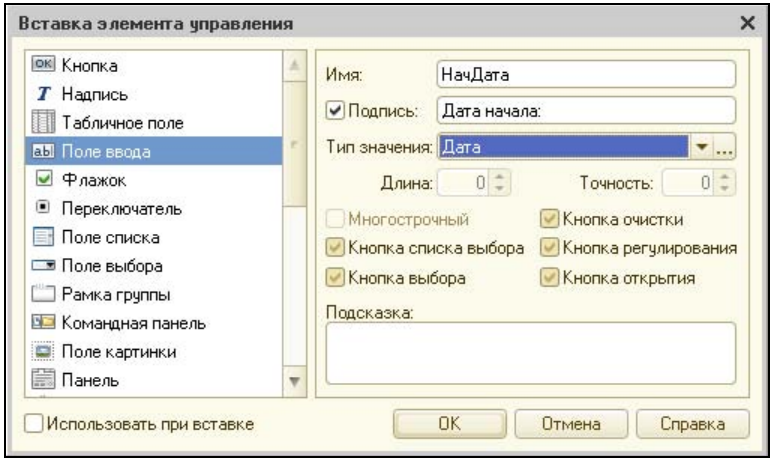



**Рис. 4.2.** Вставка элемента управления

4. В левой части окна мы можем выбрать тип элемента управления, в правой части — задать имя, надпись, тип и прочие параметры будущего элемента. Нам сейчас нужно создать два поля выбора даты для того, чтобы задавать период отбора. Одно поле мы назовем `НачДата`, другое — `КонДата`, тип значения — `Дата`. Результат должен выглядеть подобно тому, что показано на рис. 4.3.
5. Теперь нам нужно разместить на форме таблицу значений. Мы будем отбирать данные в нее, а уже потом выводить содержимое на печать. Снова нажмем кнопку  , но теперь выберем тип **Табличное поле**, имя — `Реестр`, тип значения — `ТаблицаЗначений` (рис. 4.4).
6. Разместим таблицу значений на форме так, чтобы она заняла все пространство. Кроме полей ввода даты, это будет основное информационное поле (рис. 4.5).
7. Теперь в таблицу значений надо добавить колонки. Делается это щелчком правой кнопкой мыши на таблице значений, после чего необходимо выбрать пункт **Добавить колонку**, а затем в свойствах новой колонки задать имя — `Автомобиль` и тип значения — `Справочник.Автомобили` (рис. 4.6).
8. После того как заданы свойства, форма с новой колонкой будет выглядеть так, как показано на рис. 4.7.
9. Теперь, аналогично тому, как мы создали колонку "Автомобиль", создайте остальные колонки таблицы значений. Список всех колонок и типы хранимых в них значений приведены в табл. 4.1.

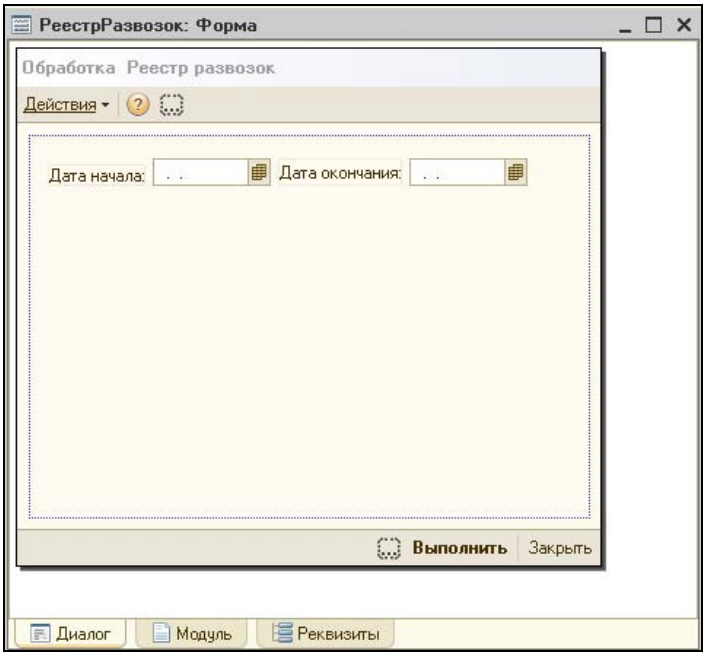



**Рис. 4.3.** Так выглядят элементы выбора периода

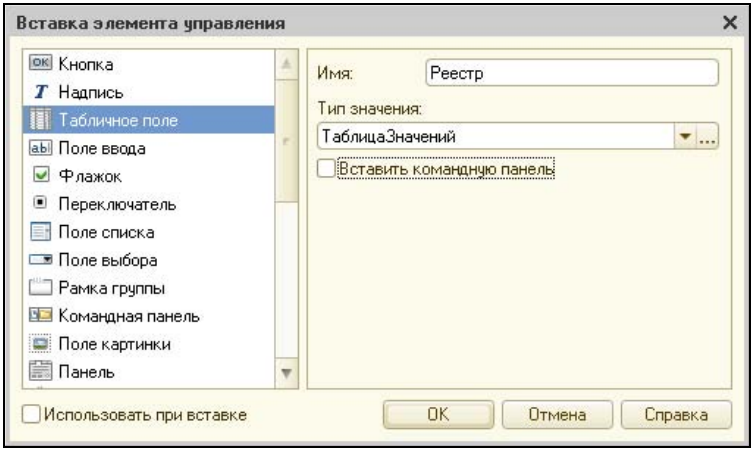



**Рис. 4.4.** Создаем таблицу значений для формы

***Таблица. 4.1.*** *Список колонок таблицы значений в отчете "Реестр развозок"*

| Имя колонки | Тип значения | Длина |
|---|---|---|
| Автомобиль | Справочник.Автомобили | |
| Номер | Строка | 12 |
| Километраж | Число | 12.2, неотрицательное |
| ЗатратыТоплива | Число | 12.2, неотрицательное |

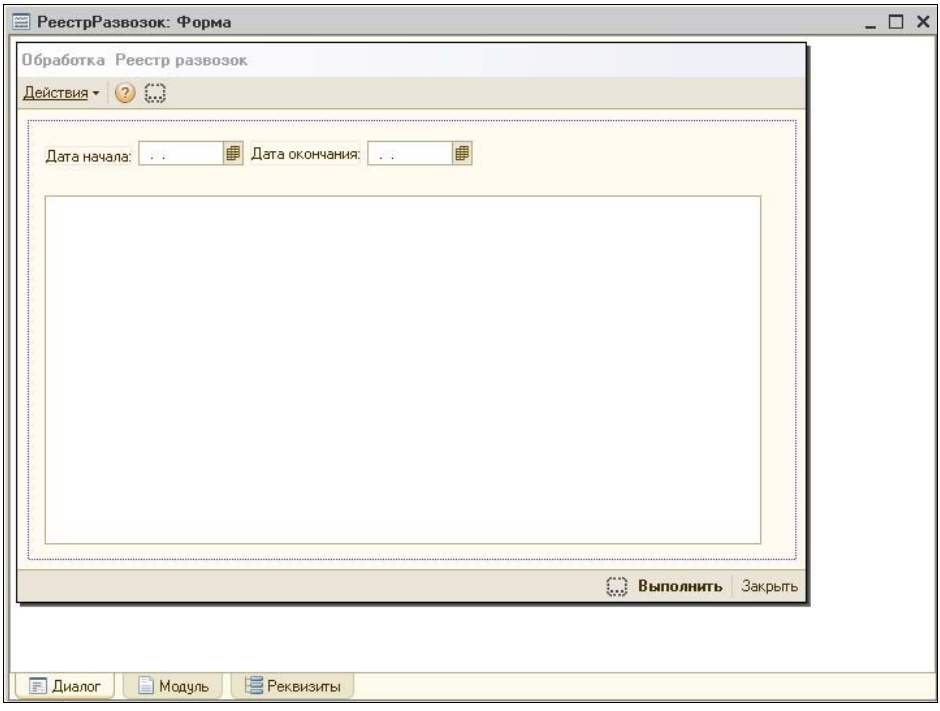



**Рис. 4.5.** "Каркас" отчета

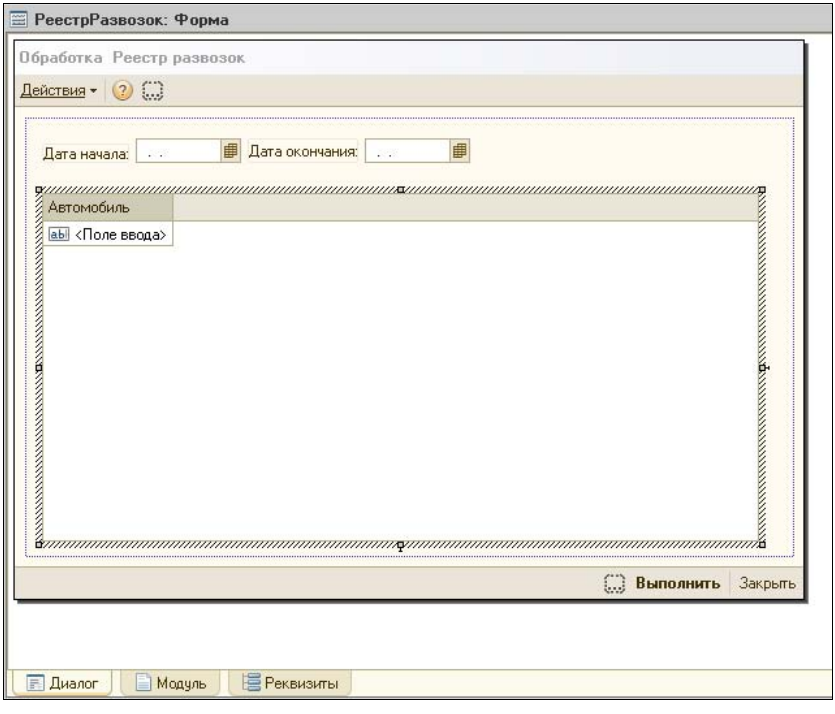



**Рис. 4.6.** Здесь задается тип значения колонки таблицы значений

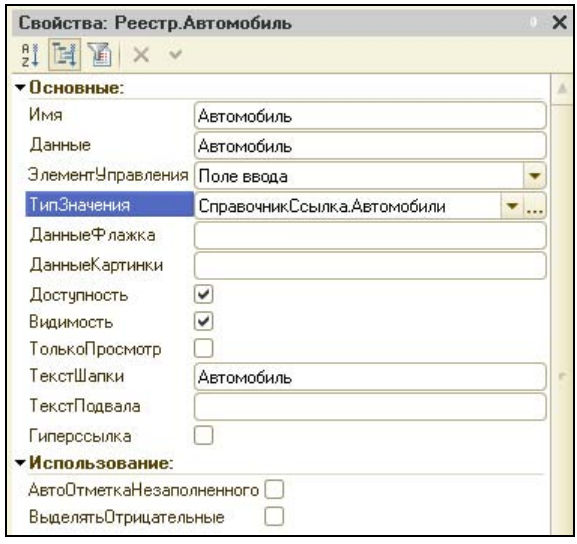



**Рис. 4.7.** Добавляем колонку в таблицу значений

10. В соответствии с табл. 4.1 добавим колонки в таблицу значений, результат должен выглядеть так, как показано на рис. 4.8.

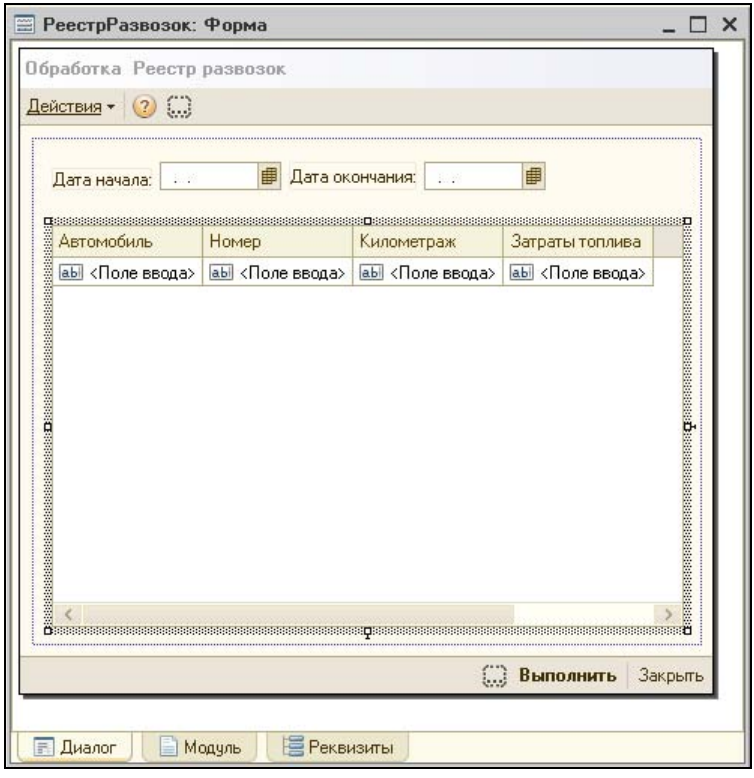



**Рис. 4.8.** Готовая форма реестра развозок, осталось написать модуль и сделать печатную форму

11. Форма отчета вполне готова. Осталось главное — заполнить модуль программным кодом и создать печатную форму-макет. Пока начнем с модуля. Сперва сделаем так, чтобы в поля даты начала и даты окончания интервала отбора при открытии формы сразу подставлялись какие-то даты. Конечно, пользователь может выбирать их и вручную, но гораздо удобнее, если уже будет стоять какой-то стандартный интервал. А если он пользователя не устроит, тогда он и выберет другой. Читатель, который уже имел опыт программирования в системе "1С:Предприятие 7.7", сейчас скажет: "Нужно написать предопределенную процедуру `ПриОткрытии()`". И напишет ее. И процедура эта работать не будет. Почему? Потому, что в "1С:Предприятие 8.2" вручную написать процедуру `ПриОткрытии()` недостаточно. Надо сначала открыть окно свойств формы (можно просто двойным щелчком на заголовке формы) и в нижней части этого окна найти раздел **События** (рис. 4.9).


**Рис. 4.9.** Этот раздел окна свойств содержит список предопределенных событий

Именно здесь описываются все стандартные процедуры, которые могут произойти с формой. Щелкнем мышью по значку в виде увеличительного стекла события `ПриОткрытии` — и попадаем в модуль с уже автоматически созданной процедурой `ПриОткрытии()`. К слову сказать, название этой процедуры несущественно, можно назвать хоть `ПриОткрытииНашегоПрекрасногоОтчета()`, лишь бы эта процедура была связана с событием формы `ПриОткрытии`. Да, если вы попробуете поэкспериментировать и поменяете в модуле название процедуры, не забудьте выбрать его в выпадающем списке события формы `ПриОткрытии`, т. к. при автоматическом создании процедуры туда уже прописалось имя `ПриОткрытии`. Процедура создана, код, который в нее нужно вписать, представлен в п. 13.

12. Теперь пропишем процедуру, которая будет связана с нажатием кнопки **Выполнить** в окне формы. Для того чтобы ее открыть, можно просто щелкнуть по кнопке, а затем в окне свойств выбрать значок с увеличительным стеклом в свойстве `Действие`. А можно и просто перейти на вкладку **Модуль**, поскольку при создании формы кнопка **Выполнить** создается автоматически и пустая процедура для нее тоже создается автоматически.

13. Текст программного модуля должен выглядеть так:

```
Процедура КнопкаВыполнитьНажатие(Кнопка)
  Реестр.Очистить();
  Выборка = Документы.Развозка.Выбрать(НачДата, КонДата);
  Пока Выборка.Следующий() Цикл
    Если Выборка.Проведен = Ложь Тогда
      Продолжить;
    КонецЕсли;
```

```
      НоваяСтрока = Реестр.Добавить();
      НоваяСтрока.Автомобиль = Выборка.Автомобиль;
      НоваяСтрока.Номер = Выборка.Автомобиль.ГосНомер;
      НоваяСтрока.Километраж = Выборка.Километраж;
      НоваяСтрока.ЗатратыТоплива = Выборка.ЗатратыТоплива;
    КонецЦикла;
    Реестр.Свернуть("Автомобиль, Номер","Километраж, ЗатратыТоплива");
КонецПроцедуры

Процедура ПриОткрытии()
    НачДата = НачалоДня(НачалоМесяца(ТекущаяДата()));
    КонДата = КонецДня(ТекущаяДата()+ 86400);
КонецПроцедуры
```

14. Теперь поясним, что мы тут написали. В процедуре `ПриОткрытии()` мы присвоили переменной `НачДата` начало дня начала месяца текущей даты. То есть, если говорить проще, 0 часов 0 минут 0 секунд первого числа текущего месяца. Переменной `КонДата` мы присвоили значение `ТекущаяДата() + 86400`. А почему не просто `ТекущаяДата()`? Потому, что тогда в интервал не попадут документы, выписанные сегодня, потому что интервал закончится сегодня ночью, в 0 часов 0 минут 0 секунд. Прибавив 86 400 секунд, т. е. 1 сутки, мы заканчиваем наш интервал в 0 часов 0 минут 0 секунд дня, следующего за текущим.

15. В процедуре `КнопкаВыполнитьНажатие()` мы сначала очищаем таблицу (вдруг пользователь сформирует отчет несколько раз подряд, тогда данные в таблицу будут добавляться к уже сформированным ранее), затем циклом выбираем документы "Развозка" в пределах дат `НачДата` и `КонДата`, при этом пропуская непроведенные (и помеченные на удаление тоже, поскольку они также являются непроведенными). В цикле мы при каждом проходе добавляем новую строку и присваиваем значению колонки таблицы значений соответствующий реквизит документа. Если бы мы этим ограничились, то получили бы список всех документов развозки за указанный период, но нам нужно не это. Поэтому после цикла мы командой `Свернуть()` сворачиваем выборку по колонкам "Автомобиль" и "Номер", при этом суммируя данные в колонках "Километраж" и "ЗатратыТоплива". Результаты представлены на рис. 4.10.

16. Отчет формируется, теперь бы его научиться выводить на печать, заодно показывая общую сумму по затраченному топливу. Перейдем в окно нашего отчета (см. рис. 4.1) и создадим новый макет (правой кнопкой мыши по слову "Макеты" и выбираем пункт **Добавить**). Откроется окно создания нового макета (рис. 4.11).

17. Оставим все как на рисунке и нажмем кнопку **Готово**. Теперь окно обработки выглядит так, как показано на рис. 4.12.

18. Откроем макет двойным щелчком по нему и наметим его структуру наподобие того, как создаются таблицы в MS Excel. Введем заголовок и разметим таблицу, которая впоследствии будет заполняться данными. Результат получится таким,

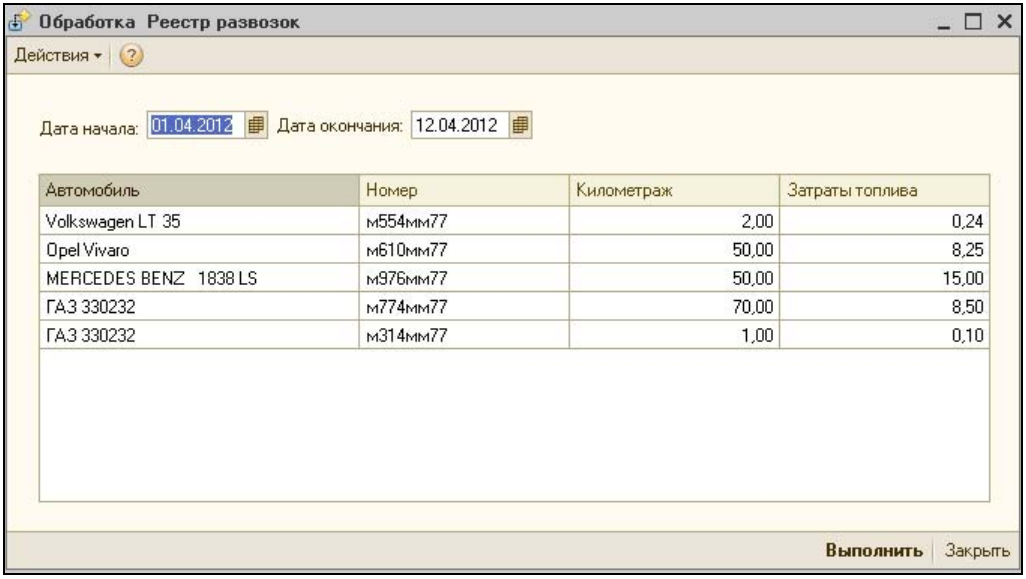



**Рис. 4.10.** Формирование реестра развозок, таблица значений заполнена данными

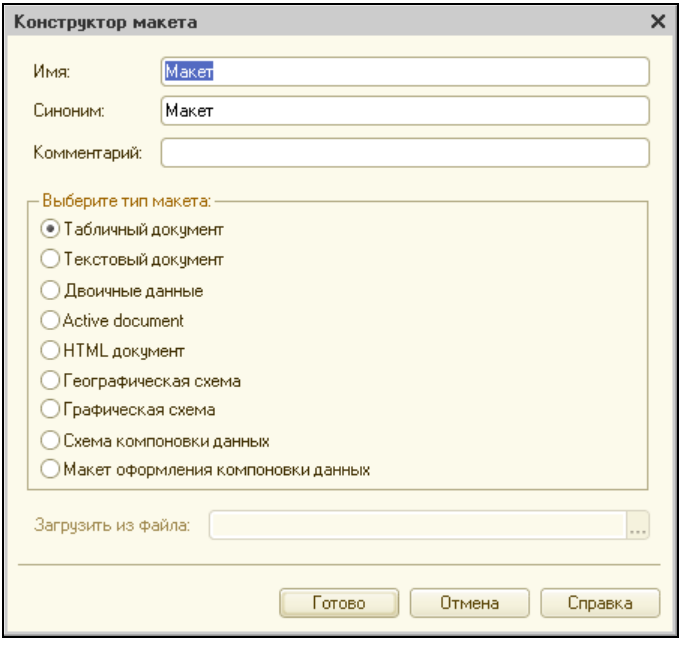



**Рис. 4.11.** Создаем новый макет (печатную форму)

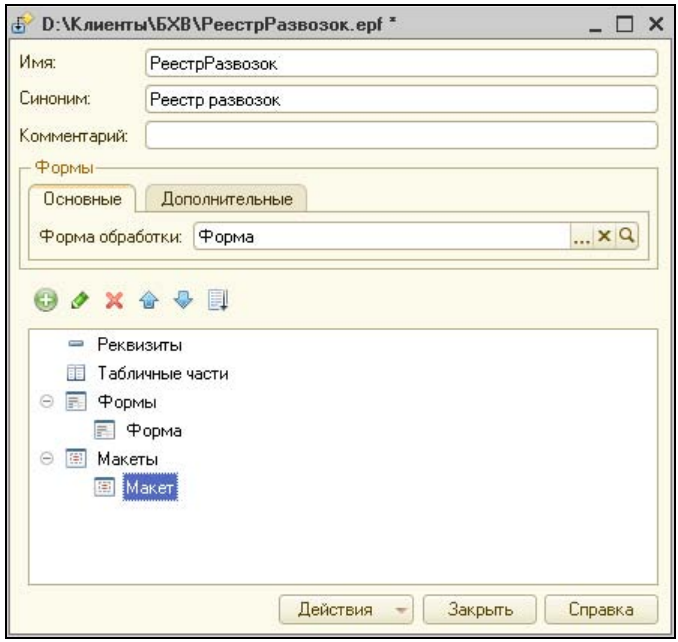



**Рис. 4.12.** Так должно выглядеть окно отчета после добавления макета

| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | | | | | |
| 2 | | Список развозок за период НачДата - КонДата | | | |
| 3 | | | | | |
| 4 | № | Автомобиль | Номер | Километраж | Затраты топлива |
| 5 | | | | | |
| 6 | | | | | |
| 7 | | | | | |
| 8 | | | | | |
| 9 | | | | | |
| 10 | | | | | |
| 11 | | | | | |
| 12 | | | | | |
| 13 | | | | | |
| 14 | | | | | |

**Рис. 4.13.** Создаем макет вручную. Пока все как в MS Excel

как на рис. 4.13. То, что в табличной части всего одна строка, пусть вас не смущает.

19. Оформление ячеек (шрифт, цвет текста, начертание, границы и т. д.) расположено в разделе **Оформление** свойств ячейки (рис. 4.14).

20. После того как мы сделали "картинку" будущей печатной формы, ее нужно заполнить данными. Сначала выберем заголовок "Список развозок за период" (как и в MS Excel, достаточно выбрать одну ячейку, в которой и был написан заголовок), откроем свойства ячейки и внесем изменения. В разделе **Макет** свойство **Заполнение** изменяем на **Шаблон**, а в свойстве **Текст** мы обрамляем квадратными скобками переменные `НачДата` и `КонДата` (рис. 4.15).

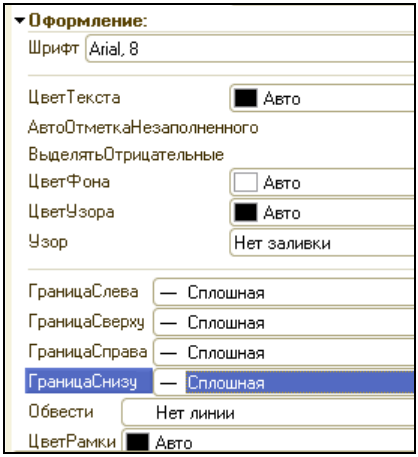



**Рис. 4.14.** Раздел окна свойств ячейки, в котором можно задать оформление

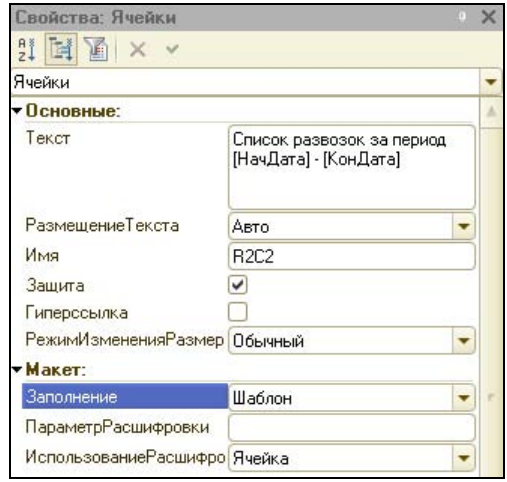



**Рис. 4.15.** Здесь задаем тип содержимого ячейки и правим текст

21. Что мы только что сделали? Сейчас поясню. Дело в том, что содержимое ячейки может быть трех типов: **Текст** — просто обычный текст, **Выражение** — программный код и **Шаблон** — совокупность первых двух типов, когда в текст вставляются фрагменты программного кода. Вот именно этот тип мы и присвоили нашей ячейке. "Список развозок за период" — обычный текст, но вместо слов [НачДата] и [КонДата] будут подставлены соответствующие значения переменных. То, что `НачДата` и `КонДата` — не просто текст, а части кода, было указано нами с помощью квадратных скобок. Результат будет выглядеть так, как показано на рис. 4.16.

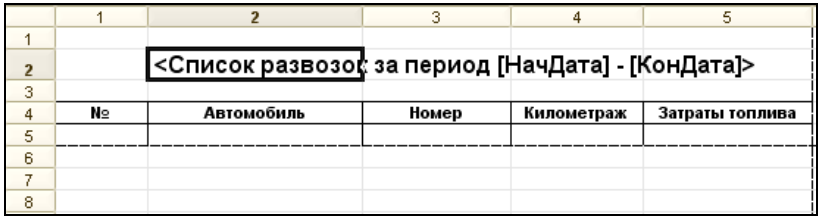



**Рис. 4.16.** Заголовок макета — уже не текст

22. Теперь заполняем ячейки параметрами, как показано на рис. 4.17: свойство **Заполнение** со значением **Параметр**. Переменные, которые проставлены в ячейки (`Ном`, `ПечАвтомобиль` и т. д.), мы еще не определяли, но до этого мы сейчас дойдем.
23. Теперь нам нужно объединить строки макета в секции, каждая из которых будет выводиться в заданный нами момент. Выделите мышью строки с 1 по 4 включительно, как бы вы это делали в MS Excel (т. е. выделяя не все строки, а по сути, только их номера). После этого в меню **Таблица | Имена** выберите пункт **Назначить имя**. Нам предложат задать имя для выделенной области (рис. 4.18).


**Рис. 4.17.** Заполняем содержимое ячеек макета

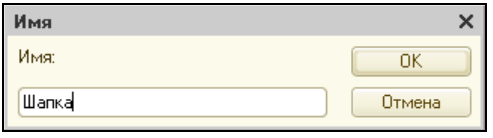



**Рис. 4.18.** Задаем имя группы ячеек

24. Задаем имя `Шапка` диапазону с 1 по 4 строки, `Строка` — 5-й строке и `Итог` — 6-й строке. Также в секции `Итог` для одной из ячеек задайте имя `ИтогоТоплива` и тип **Параметр**, выделите эту ячейку жирным шрифтом. Готовый к работе макет должен выглядеть так, как показано на рис. 4.19.

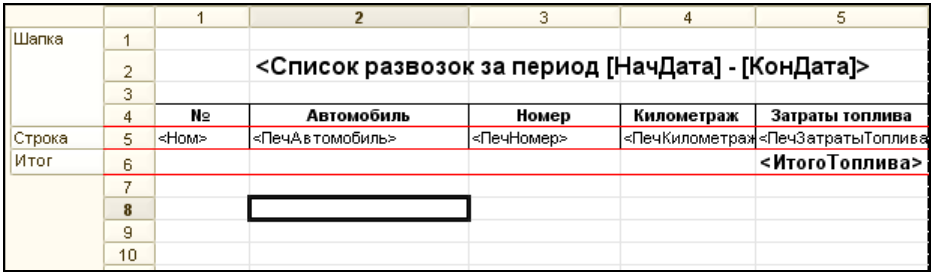



**Рис. 4.19.** Готовый макет реестра развозок

25. Макет у нас есть, осталось запрограммировать саму процедуру печати, а также кнопку, запускающую эту процедуру. Сначала создадим кнопку. Для этого перейдем на форму обработки и сделаем щелчок правой кнопкой мыши на кнопке с тремя точками, расположенной слева от кнопки **Выполнить**. В открывшемся контекстном меню выберем пункт **Добавить**. На панели, левее кнопки **Выполнить,** появится новая кнопка, свойства которой мы должны задать, как это уже делали для самых разных объектов — двойным щелчком и установкой свойств в окне свойств кнопки. Итоговый результат должен выглядеть так, как на рис. 4.20, а окно свойств кнопки — как на рис. 4.21.

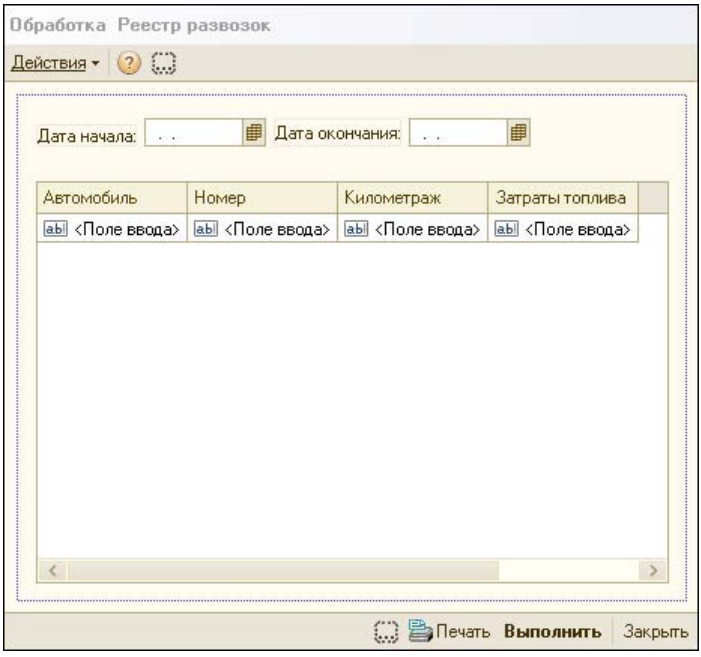



**Рис. 4.20.** Добавляем кнопку **Печать** на форму

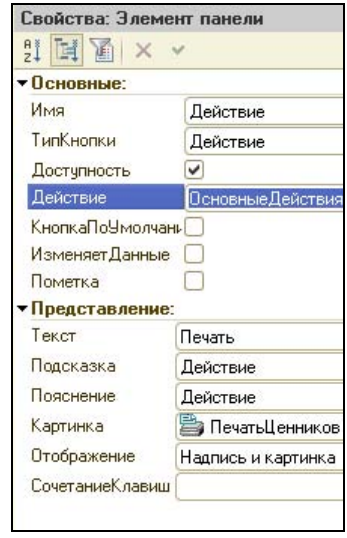



**Рис. 4.21.** Задаем отображение кнопки и привязываем кнопку к процедуре печати

26. Для красоты мы задали для кнопки картинку в одноименном поле, а в свойстве **Отображение** показали, что хотим, чтобы отображалась и картинка, и поясняющая надпись. В свойстве **Действие** мы, как обычно, делаем щелчок мышью, создавая процедуру, связанную с кнопкой **Печать**. Сама процедура должна выглядеть следующим образом.

```
Процедура ОсновныеДействияФормыДействие(Кнопка)
  ТабДокумент = Новый ТабличныйДокумент;
  Макет = ЭтотОбъект.ПолучитьМакет("Макет");
  ОбластьШапка = Макет.ПолучитьОбласть("Шапка");
  ОбластьСтрока = Макет.ПолучитьОбласть("Строка");
  ОбластьИтог = Макет.ПолучитьОбласть("Итог");
  ОбластьШапка.Параметры.НачДата = Формат(НачДата, "ДФ=dd.MM.yyyy");
  ОбластьШапка.Параметры.КонДата = Формат(КонДата, "ДФ=dd.MM.yyyy");
  ТабДокумент.Вывести(ОбластьШапка);
  Ном = 1;
  ИтогоТоплива = 0;
  Для Каждого ТекущаяСтрока Из Реестр Цикл
    ОбластьСтрока.Параметры.Ном = Ном;
    ОбластьСтрока.Параметры.ПечАвтомобиль = ТекущаяСтрока.Автомобиль;
    ОбластьСтрока.Параметры.ПечНомер = ТекущаяСтрока.Номер;
    ОбластьСтрока.Параметры.ПечКилометраж = ТекущаяСтрока.Километраж;
    ОбластьСтрока.Параметры.ПечЗатратыТоплива = ТекущаяСтрока.ЗатратыТоплива;
```

```
        ИтогоТоплива = ИтогоТоплива + ТекущаяСтрока.ЗатратыТоплива;
        ТабДокумент.Вывести(ОбластьСтрока);
        Ном = Ном + 1;
    КонецЦикла;
    ОбластьИтог.Параметры.ИтогоТоплива = ИтогоТоплива;
    ТабДокумент.Вывести(ОбластьИтог);
    ТабДокумент.ТолькоПросмотр = Истина;
    ТабДокумент.АвтоМасштаб = Истина;
    ТабДокумент.ОтображатьСетку = Ложь;
    ТабДокумент.Показать();
КонецПроцедуры
```

Что написано в процедуре? Сначала мы объявляем, с чем будем работать. Табличный документ (тот, что мы формировали чуть раньше), макет обработки (у нас он один, но могло бы быть и несколько), области, которые мы именовали выше. Затем начинаем привязывать данные. `НачДата` на форме и `НачДата` в макете — это разные переменные, поэтому мы должны присвоить одну другой. Обращение к переменной в макете идет как `ИмяОбласти.Параметры.ИмяПеременной`. `НачДата` и `КонДата` мы отформатировали затем, чтобы в печатную форму не выводилось время, иначе это будет неинформативно и выглядеть некрасиво.

Далее мы формируем цикл, который перебирает строки таблицы значений и присваивает переменным печатной формы соответствующие переменные таблицы значений. Переменные `Ном` и `ИтогоТоплива` являются вычисляемыми: к первой при каждом проходе цикла прибавляется единица, ко второй прибавляется расход топлива по текущей строке. Таким образом, когда цикл заканчивает свою работу, мы получаем итоговый расход топлива, который и записываем в ячейку `ОбластьИтог.Параметры.ИтогоТоплива`.

Далее в процедуре задаются параметры макета (запрет редактирования, автомасштабирование, запрет вывода сетки) и дается команда вывода получившегося макета на экран `ТабДокумент.Показать()`.

27. Работа над отчетом завершена. Сохраните его и запустите в режиме Предприятия. Если все было сделано правильно, то после отработки отчет заполнит таблицу значений (рис. 4.22).

28. При нажатии кнопки **Печать** должна быть выведена печатная форма, представленная на рис. 4.23.

Мы завершили работу над отчетом. Однако, мягко говоря, это не самый сложный отчет, который может понадобиться пользователю системы "1С:Предприятие". Сложные отчеты тоже могут быть созданы в ручном режиме, но также для них можно (и нужно) использовать конструкторы, позволяющие упростить и автоматизировать труд программиста. Об этом — в следующем разделе.

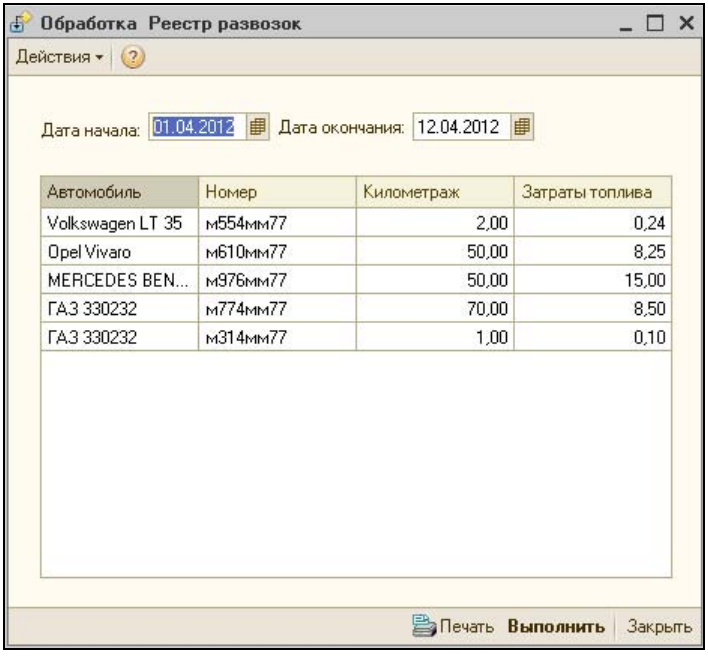



**Рис. 4.22.** Работающий отчет "Реестр развозок"…

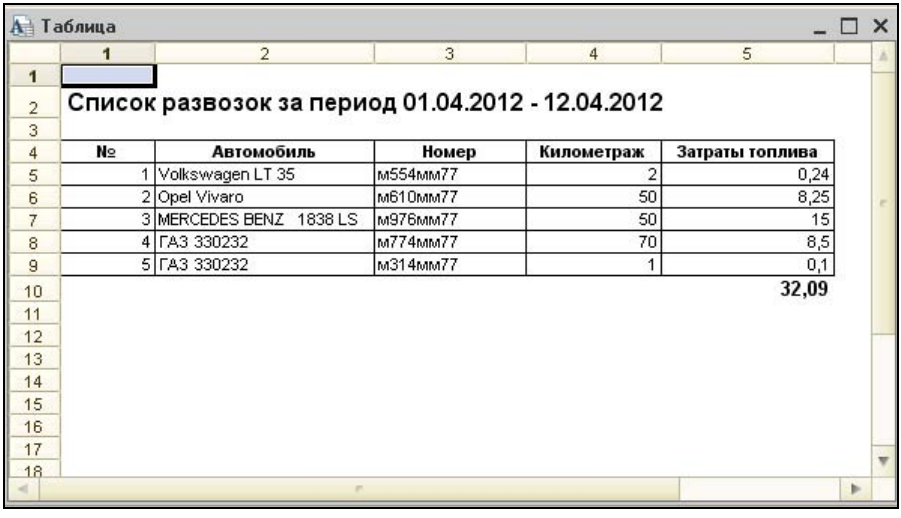



**Рис. 4.23.** ... и его печатная форма

# Использование системы компоновки данных. Конструктор запросов

Теперь рассмотрим, как можно автоматизировать получение данных и создание отчетов с использованием мощного инструмента — системы компоновки данных.

Система компоновки данных предназначена для визуального проектирования отчетов и позволяет при этом не прибегать собственно к программированию. Программный код на внутреннем языке запросов формируется автоматически на основе настроек, выбранных пользователем. Итак, сначала мы реализуем тот же самый отчет "Реестр развозок", который создавали в предыдущем разделе, но на этот раз используя систему компоновки данных.

Итак, для разнообразия создадим внутренний отчет, т. е. отчет в дереве конфигурации (хотя точно так же с компоновкой данных можно работать и во внешних отчетах, создаваемых через пункт меню **Файл | Новый | Внешний отчет**). Отчет создаем щелчком правой кнопкой мыши по заголовку раздела **Отчеты** дерева конфигурации и выбором пункта **Добавить**. Откроется стандартное окно создания нового объекта конфигурации (рис. 4.24).

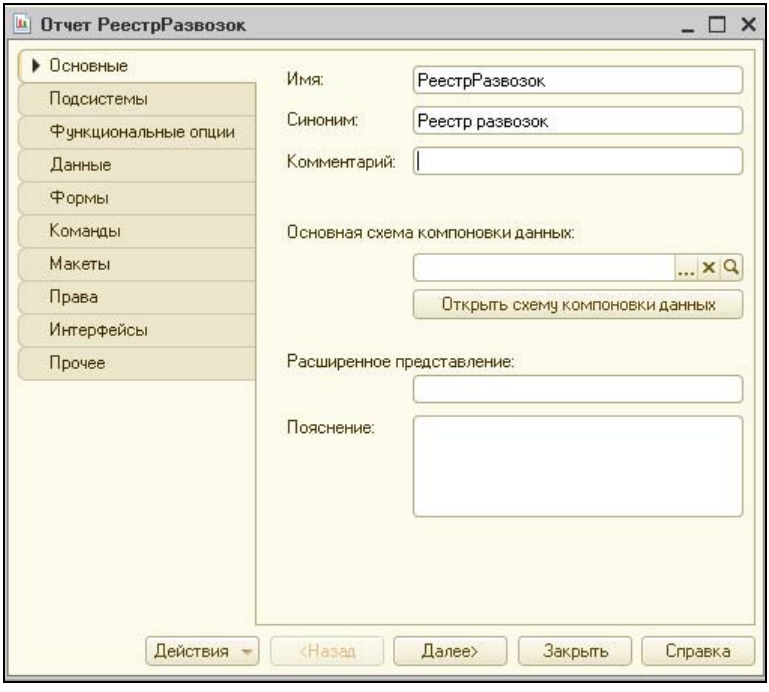



**Рис. 4.24.** Создаем новый отчет в дереве конфигурации

Зададим имя и синоним, а затем нажмем кнопку **Открыть схему компоновки данных**. Откроется уже знакомое нам окно конструктора макета, однако на этот раз для выбора доступен только один переключатель, мы создаем новую схему компоновки данных (рис. 4.25).

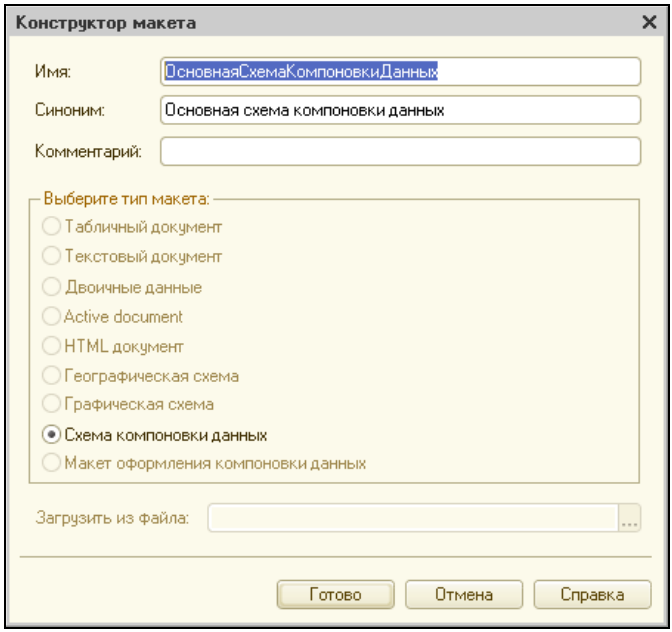



**Рис. 4.25.** Создаем схему компоновки данных

Нажмем кнопку **Готово** и попадем в окно схемы компоновки данных, пока еще пустое (рис. 4.26).

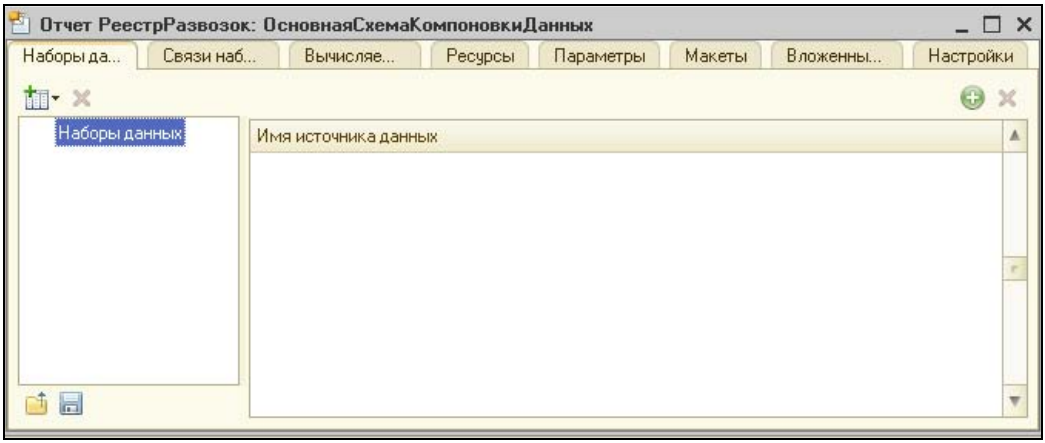



**Рис. 4.26.** Новая схема компоновки данных пока пуста

Щелкнем правой кнопкой мыши по заголовку **Наборы данных** и в контекстном меню выберем пункт **Добавить набор данных — запрос**. Формируется окно набора данных и будущего запроса, пока еще пустые (рис. 4.27).

Переименуйте **НаборДанных1** во что-то более осмысленное, например в **РеестрРазвозок** (сделать это можно двойным щелчком мыши). Затем нажмите кнопку **Конструктор запроса**. Откроется окно, показанное на рис. 4.28.

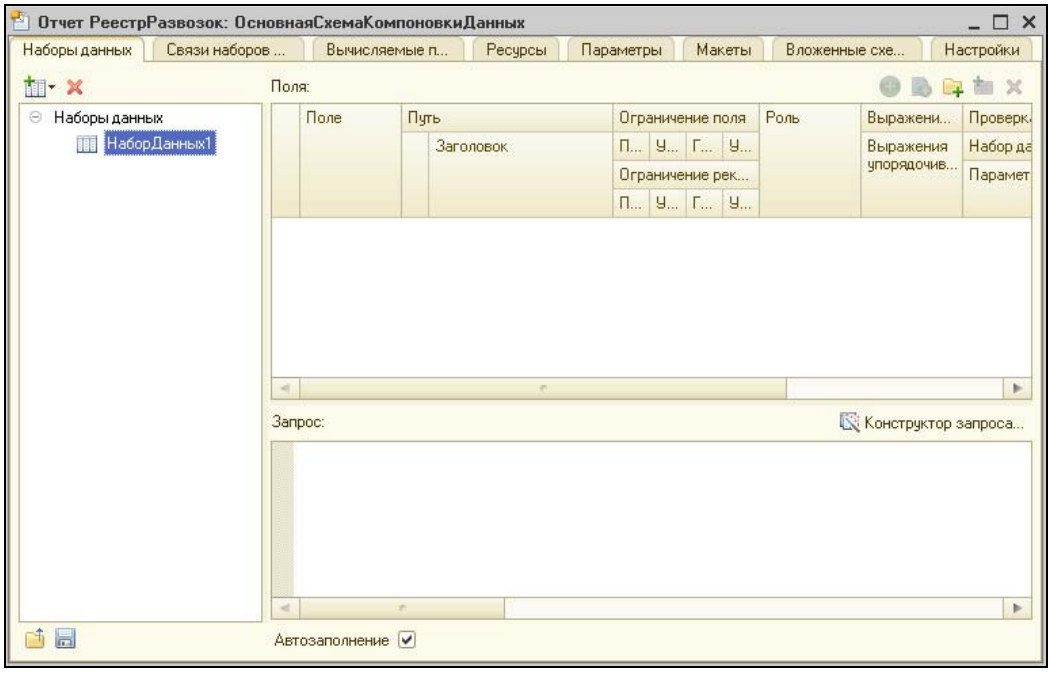



**Рис. 4.27.** Новый набор данных в схеме компоновки

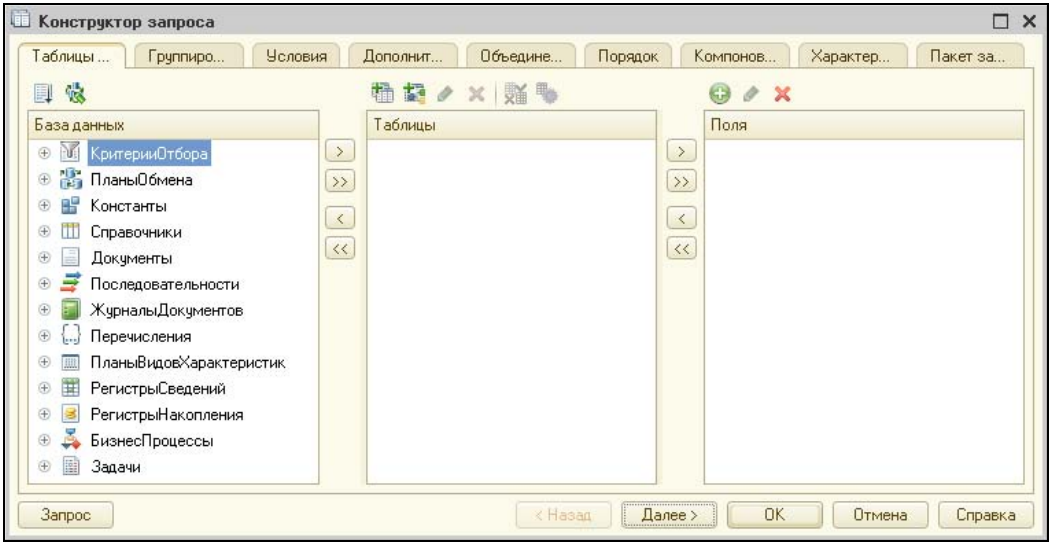



**Рис. 4.28.** Конструктор запроса

В левой части окна конструктора показано полное дерево метаданных конфигурации. С помощью кнопок со стрелками необходимо перенести вправо те таблицы, которые будут использоваться запросом для отбора данных. В свою очередь из таблиц переносим в правую треть окна используемые запросом поля. В нашем отчете по развозкам используется одна-единственная таблица "Развозки", поля же

нам понадобятся: автомобиль (ссылка), номер автомобиля, километраж, затраты топлива и дата. Все эти поля есть в таблице "Развозка". Переносим их в правую треть окна (рис. 4.29).

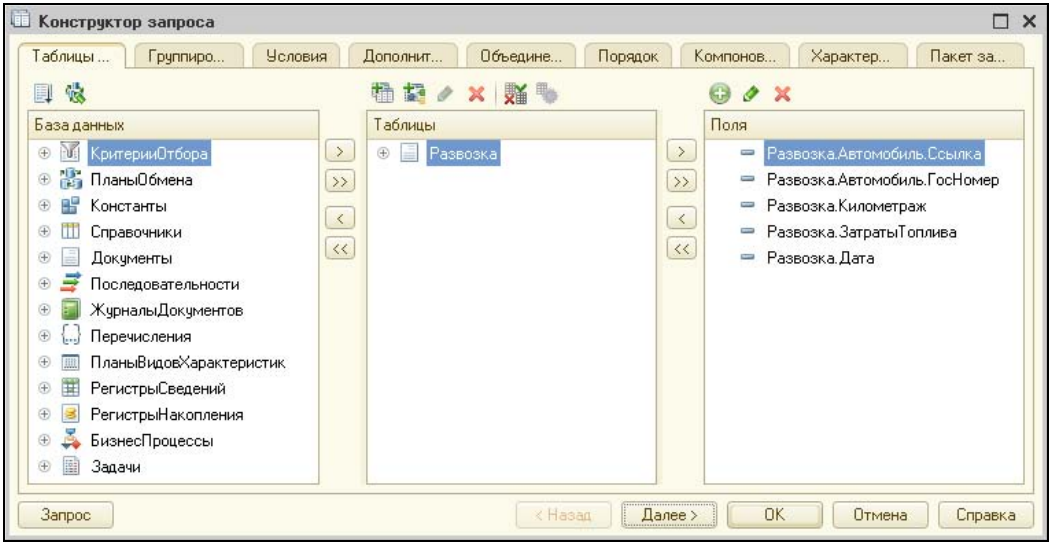



**Рис. 4.29.** Задаем таблицы и поля, которые будут использованы в отчете

Данные в конструкторе заданы, нажмем кнопку **OK** и получим запрос, автоматически сформированный на основе нашего отбора данных (рис. 4.30).

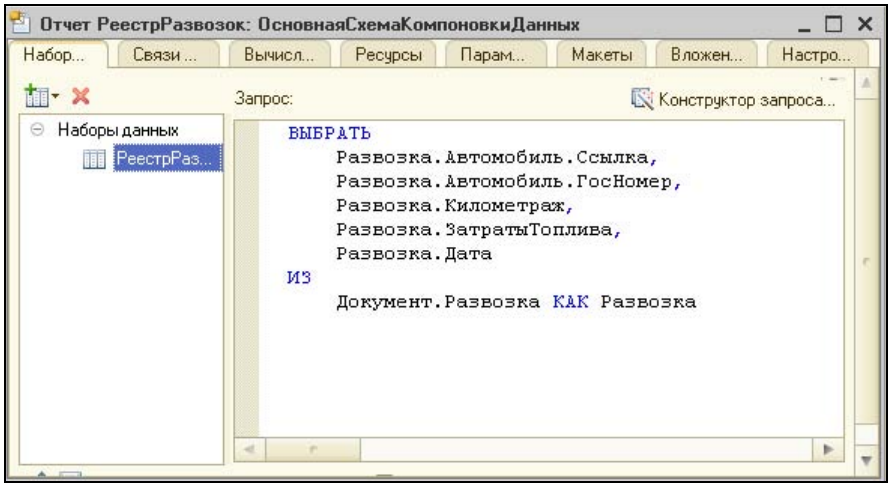



**Рис. 4.30.** На основе отбора в конструкторе запрос создается автоматически

Синтаксиса языка запросов мы коснемся в следующем разделе этой главы, а пока просто продолжим конструирование отчета. Наш набор данных теперь выглядит так, как представлено на рис. 4.31.

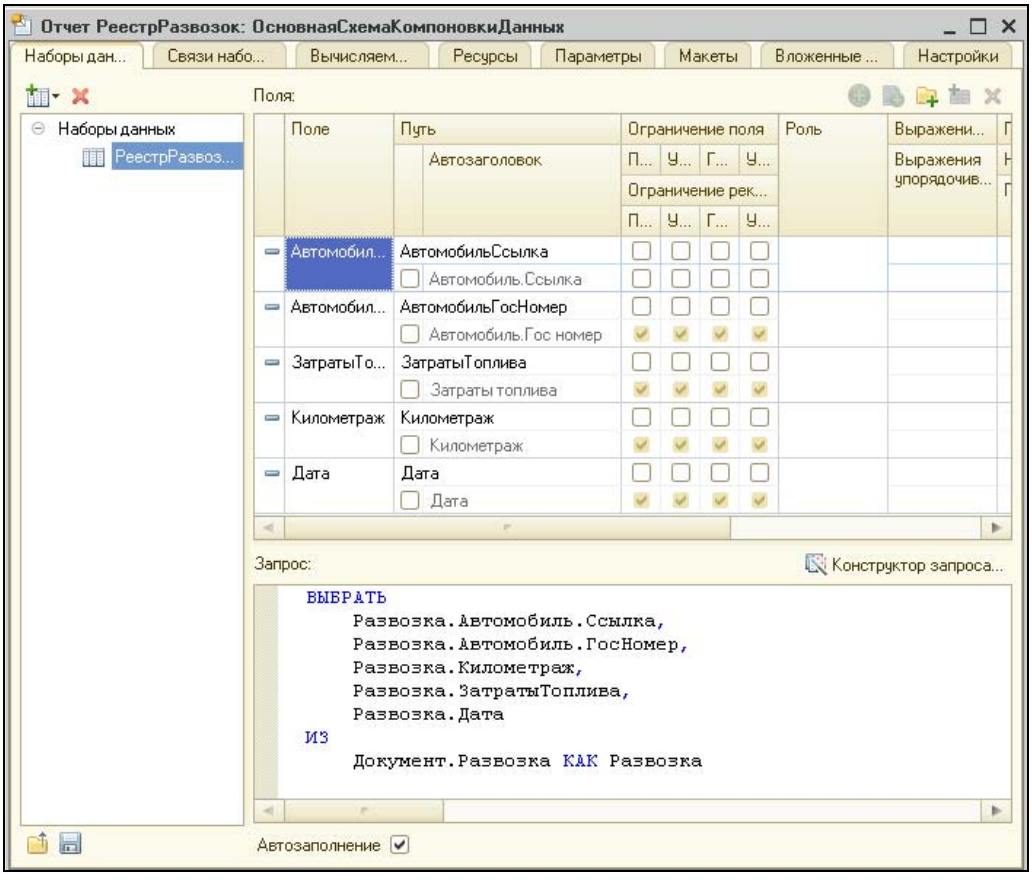



**Рис. 4.31.** Схема компоновки запроса, вкладка **Наборы данных**

Теперь перейдем на вкладку **Ресурсы** и из списка доступных полей, определенных нами ранее, перенесем кнопками со стрелками в правую часть окна те поля, по которым будет производиться суммирование (рис. 4.32).

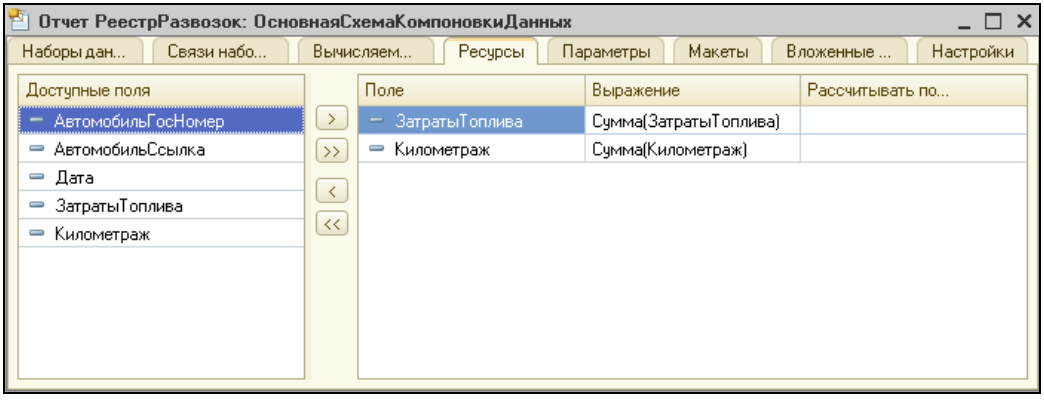



**Рис. 4.32.** Схема компоновки запроса, вкладка **Ресурсы**

Вообще-то здесь можно определить не только суммирование, если щелкнуть мышью на поле **Выражение** для любой из строк, можно увидеть несколько предложенных вариантов: минимум, максимум, количество, суммирование и т. д. Сейчас нас интересует именно суммирование.

Далее нам нужно задать порядок группировки данных, выводимых в отчете. Для этого перейдем на вкладку **Настройки** (рис. 4.33).

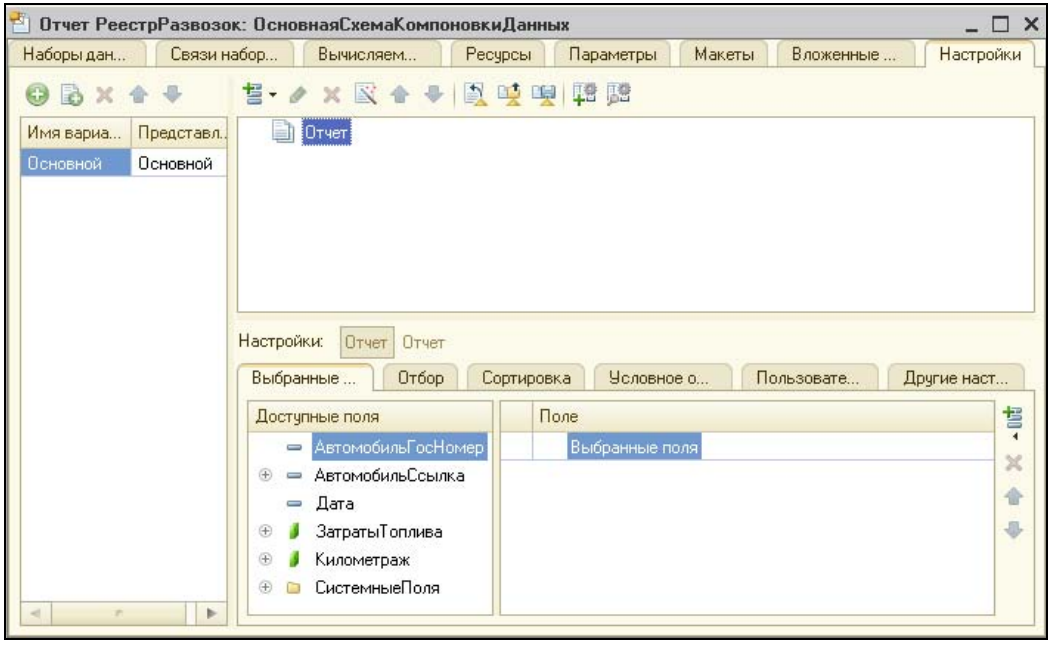



**Рис. 4.33.** Схема компоновки запроса, вкладка **Настройки**

Щелкнем правой кнопкой мыши по заголовку **Отчет** и в контекстном меню выберем пункт **Новая группировка**. Откроется окно выбора группировки (рис. 4.34).

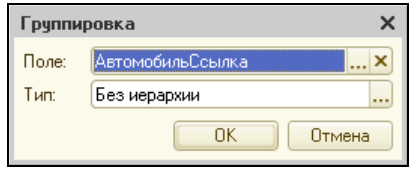



**Рис. 4.34.** Создаем новую группировку

**Поле** — это поле данных, по которым будет (или наоборот, не будет) производиться группировка. А **Тип** — это вариант построения иерархии. Может быть трех типов.

- **Без иерархии.** Записи выводятся без иерархии, списком.
- **Иерархия.** Записи выводятся в иерархии. Например, если бы мы формировали отчет по оборотам с контрагентами, то у нас была бы выведена иерархия данных

в зависимости от построения справочника контрагентов. В первом уровне иерархии выводились бы группы, например, "Поставщики" и "Покупатели", а для каждой группы выводились бы иерархические данные второго уровня — собственно обороты по каждому контрагенту.

- **Только иерархия.** А в этом случае в отчет выводилась бы только иерархия, без данных низшего уровня. Для приведенного ранее отчета по оборотам это означало бы вывод данных о продажах только по группам, без детализации по каждому конкретному контрагенту.

В нашем случае иерархические справочники не используются (автомобили у нас не делятся ни по автопаркам, ни по отделам, так что иерархия нам в данном отчете неинтересна), поэтому выберем вариант **Без иерархии** (см. рис. 4.34). Затем щелкнем правой кнопкой мыши по созданному нами элементу **Автомобиль.Ссылка** и добавим подчиненную ей группировку **Автомобиль.ГосНомер**, тоже без иерархии. Результат должен выглядеть так, как показано на рис. 4.35.

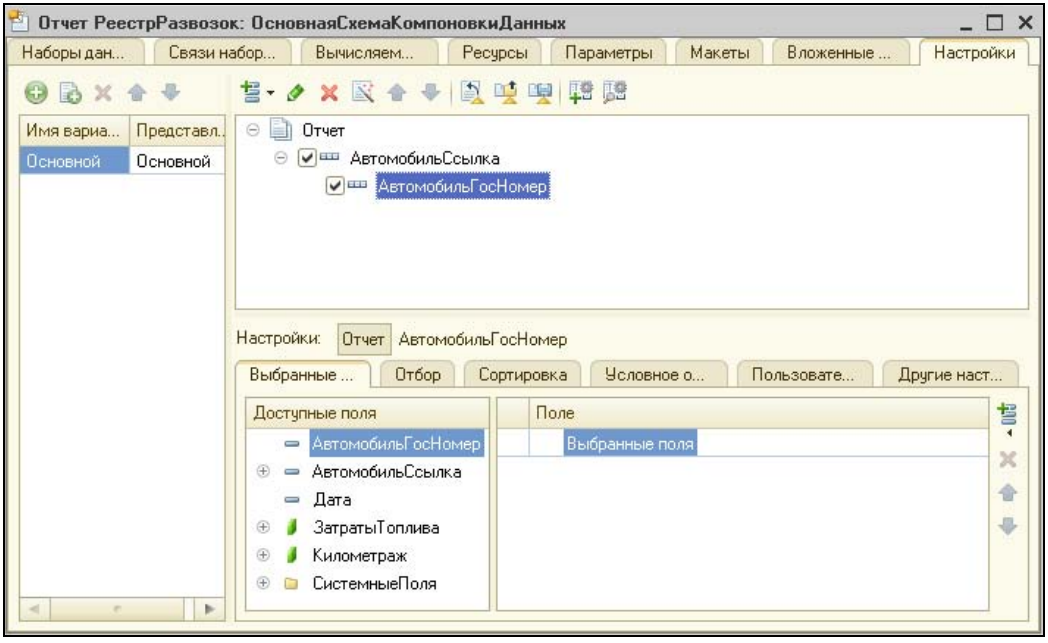



**Рис. 4.35.** Схема компоновки запроса, вкладка **Настройки**. Иерархическая группировка

С группировкой определились, теперь в нижней части этого же окна добавим поля (рис. 4.36).

Собственно, на этом отчет практически завершен. Разумеется, мы не использовали еще много различных возможностей компоновки данных, которые можно найти, если пройтись по вкладкам. Но поставленной цели мы уже достигли — сформировали отчет, идентичный тому, который создали в начале главы, не написав при этом ни строчки программного кода. Осталось подправить еще некоторые детали.

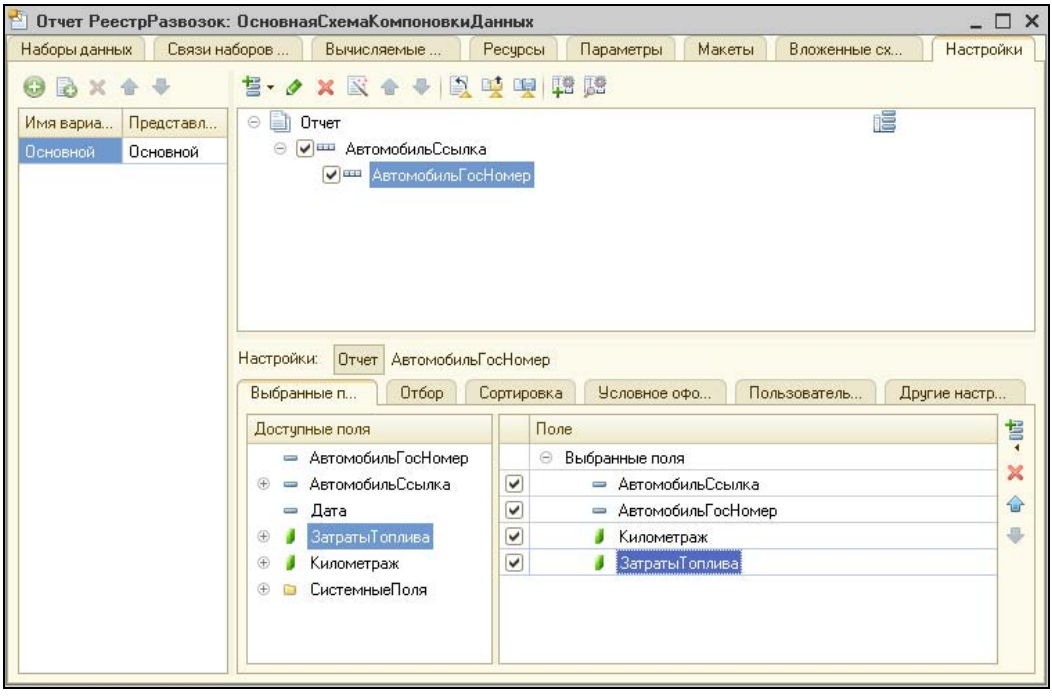



**Рис. 4.36.** Схема компоновки запроса, вкладка **Настройки**. Выбираем поля

Перейдем обратно на вкладку **Наборы данных**. Обратите внимание на поле **Заголовок**. Все, что указано в этом поле, в таком же виде попадет в отчет. **ЗатратыТоплива**, **Километраж** и **Дата** нареканий не вызывают, но **Автомобиль.Ссылка** и **Автомобиль.ГосНомер** в качестве заголовков колонок выглядят явно некрасиво. Поэтому мы установим рядом с этими полями флажки и вручную впишем свой вариант заголовков — **Автомобиль** и **Номер** (рис. 4.37).

В общем-то, и все. Сохраним конфигурацию, перейдем в режим Предприятия и запустим отчет. Поскольку в интерфейс пользователя мы наш новый отчет не добавляли, на панелях и в меню **Отчеты** его не будет, но мы всегда можем запустить новый отчет из пункта меню **Операции | Отчет**, где находится полный список отчетов конфигурации, есть там и наш новый "Реестр развозок". Запустим его и посмотрим на результат. Заодно можно в соседнем окне запустить для сравнения тот вариант, который мы делали вручную. Сравнение — на рис. 4.38.

Данные получаем те же, принцип тот же. Немножко другой макет (макет при желании можно поправить в схеме компоновки, на вкладке **Макеты**), добавлено оформление в виде плюсиков с разворачивающейся группировкой. (Помните, мы сделали подчиненные группировки, на рис. 4.35?) В остальном суть та же, только вот мы сделали этот отчет без единой строчки кода, одним конструированием.

В нашем отчете еще не хватает отбора по датам, пока выборка производится по всему возможному отрезку времени. Добавим пару полей отбора, для этого нам даже нет необходимости заходить в Конфигуратор. Прямо в режиме Предприятия,

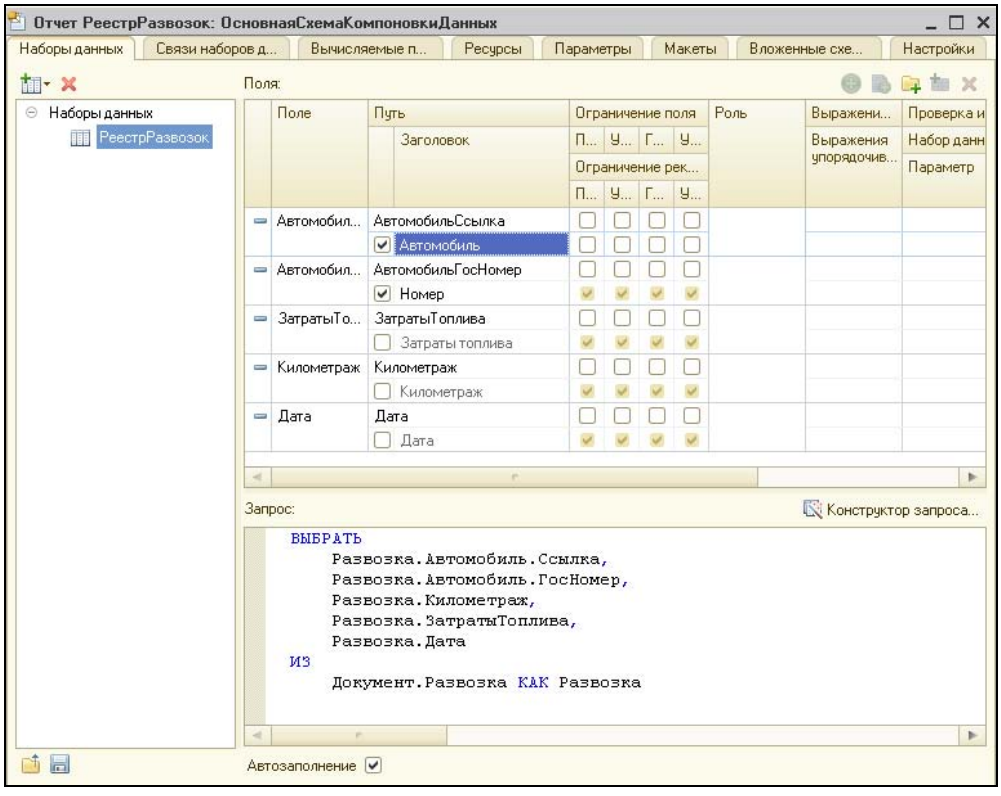



**Рис. 4.37.** Отчет готов. Заголовки **Автомобиль** и **Номер** задаем вручную

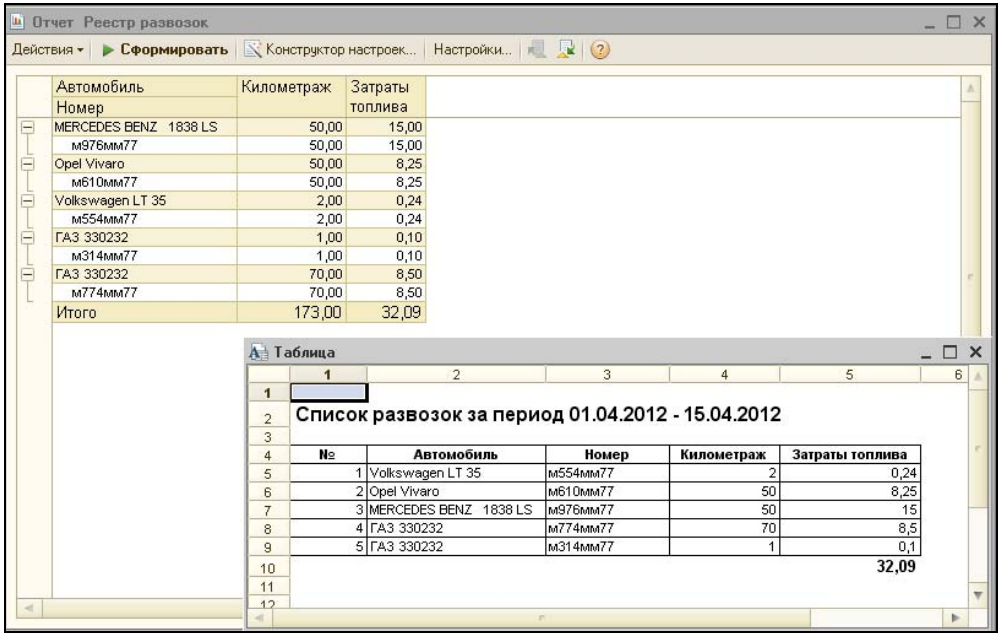



**Рис. 4.38.** Сформированный отчет в сравнении с версией, созданной вручную в предыдущем разделе

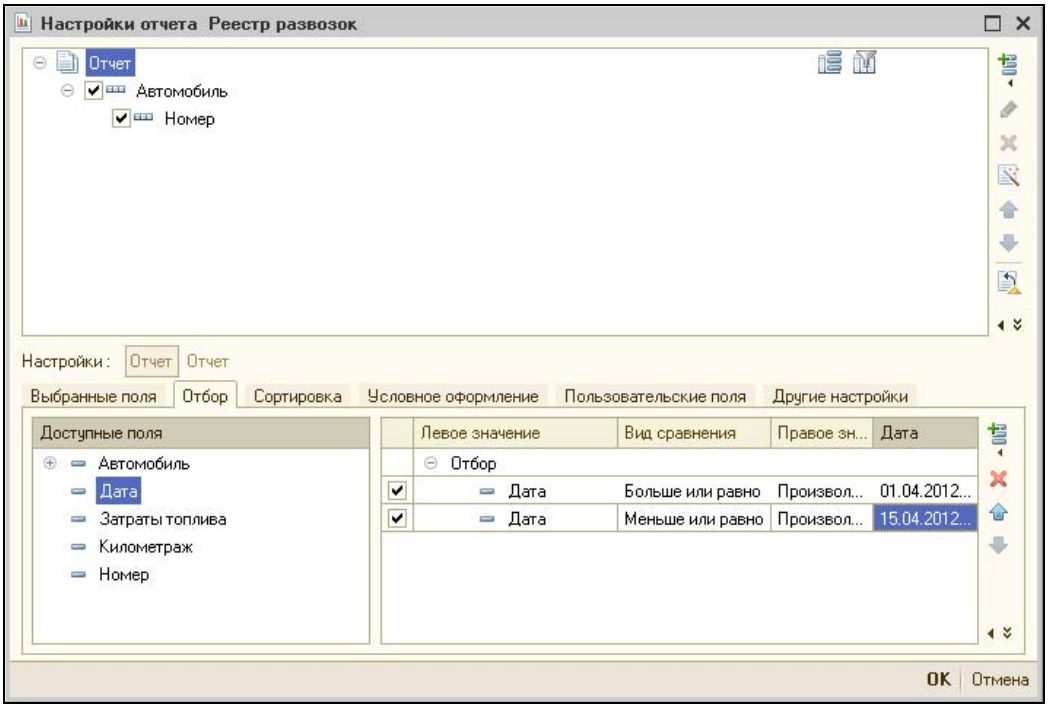



**Рис. 4.39.** В настройках задаем период отбора

в верхней части окна отчета, изображенного на рис. 4.38, нажмем кнопку **Настройки**. Откроется окно настройки нашего отчета (рис. 4.39).

И вот здесь-то, на вкладке **Отбор**, мы и добавляем два параметра типа "Дата" и сравниваем с конкретной датой. Один должен быть больше или равен 01.04.2012 (этот параметр в нашем "ручном" отчете носил название `НачДата`), второй — меньше или равен 15.04.2012 (этот параметр в нашем "ручном" отчете носил название `КонДата`). Нажмем кнопку **OK** и кнопку **Сформировать**. Теперь отчет у нас делает выборку в пределах двух заданных дат (рис. 4.40).

Цифры не изменились, но это потому, что все развозки в пробной базе данных, по которой я привожу примеры, и так попадают в заданный интервал дат.

Теперь, раз уж мы заговорили о настройках отчета в режиме Предприятия, рассмотрим заодно и вторую кнопку, находящуюся рядом с кнопкой **Настройки**. Это кнопка **Конструктор настроек**. Благодаря конструктору настроек, мы можем данные, выбранные одним и тем же отчетом, вывести совершенно в разных вариантах. При нажатии на кнопку **Конструктор настроек** открывается окно с выбором этих вариантов (рис. 4.41).

Это **Список**, **Таблица** и **Диаграмма**. Отчет, который мы только что рассматривали, выводился в форме списка, по умолчанию. Теперь давайте выберем вариант **Таблица** и нажмем кнопку **Далее**. Откроется окно конструктора настроек компоновки (рис. 4.42).

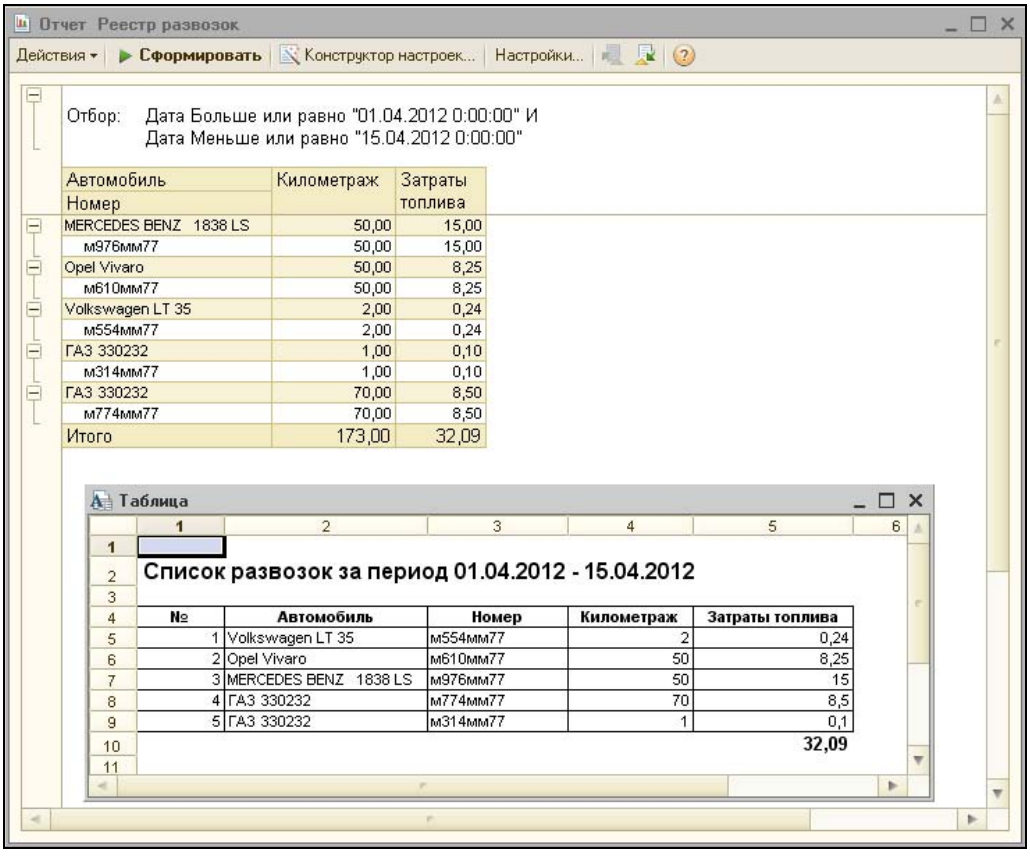



**Рис. 4.40.** Данные идентичны. Но с помощью компоновки куда быстрее

Поля должны быть добавлены так, как показано на рис. 4.42. Нажмем кнопку **Далее** и попадем на следующую страницу, где данные следует распределить по строкам и столбцам (рис. 4.43).

Мы задали, что в строках у нас будут выводиться автомобили, в столбцах — номера, данные же по затратам топлива будут выводиться на пересечении соответствующей строки (автомобиль) и столбца (номер). Результат будет такой, как на рис. 4.44.

Ну и, конечно, мы можем вывести данные в виде диаграммы, только в этом случае отбор будет вестись по единственному ресурсу — затратам топлива (рис. 4.45).

Отчет в этом случае будет выглядеть так, как показано на рис. 4.46.

В общем, один и тот же отчет, а вариантов отображения данных — много.

Один отчет мы уже создали, но такой отчет был нами сформирован и ранее. Давайте создадим еще один отчет, новый. Значительная часть отчетов выбирает данные из регистров, по остаткам ли, по движениям ли, по ценам ли... Вот с остатками товаров мы и поработаем. А именно — самостоятельно создадим отчет по остаткам товаров. Поработаем с данными из регистров и закрепим только что пройденный материал.

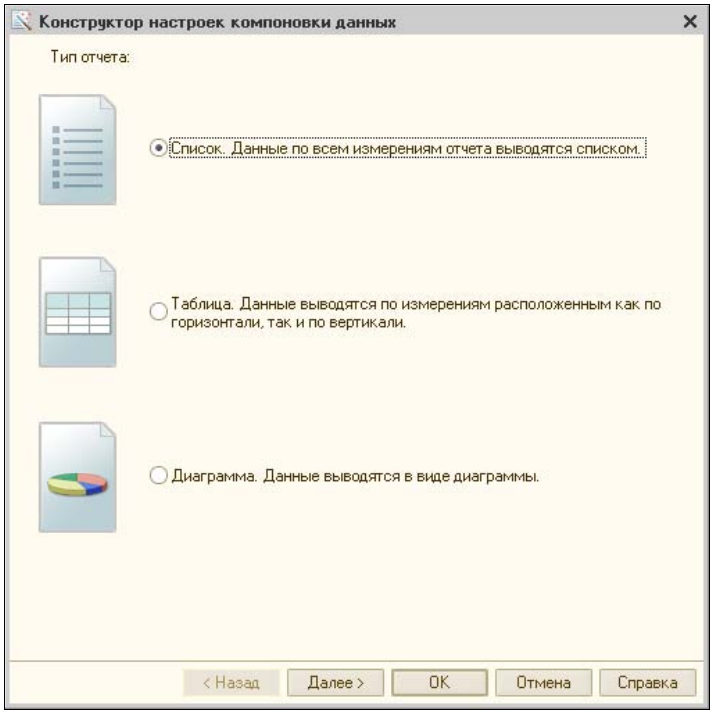



**Рис. 4.41.** Один и тот же отчет может выводиться по-разному

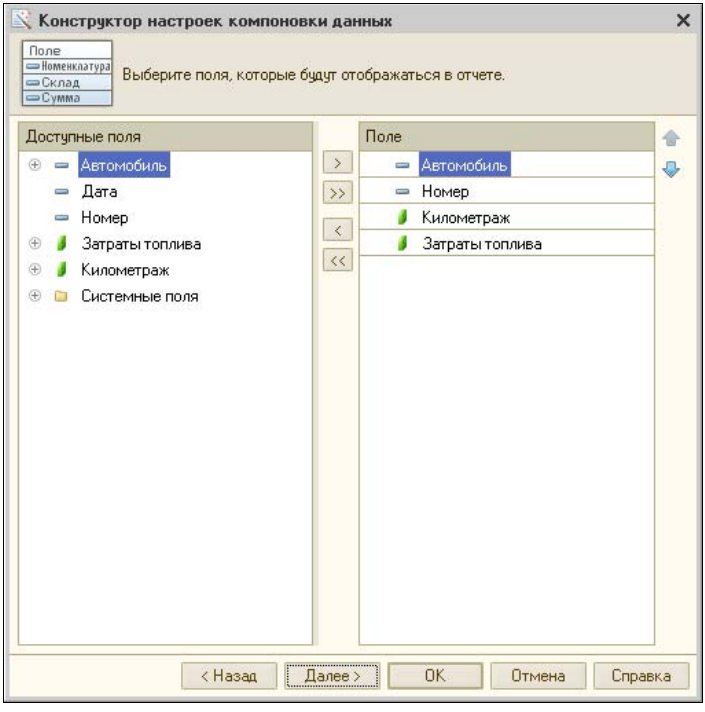



**Рис. 4.42.** Отбор полей отчета

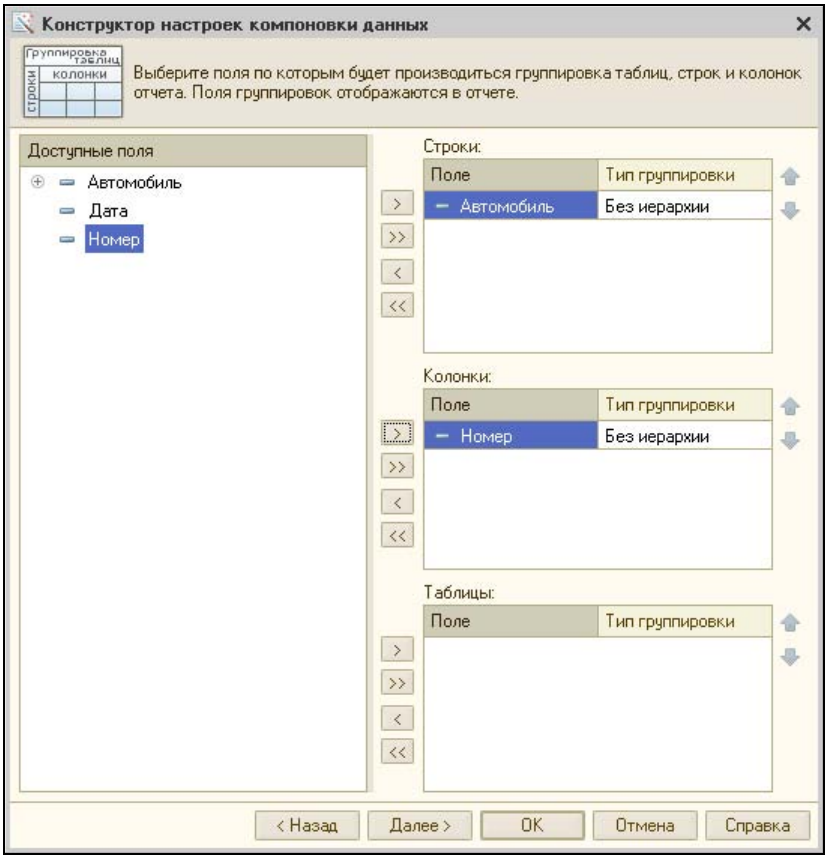



**Рис. 4.43.** Формируем структуру таблицы

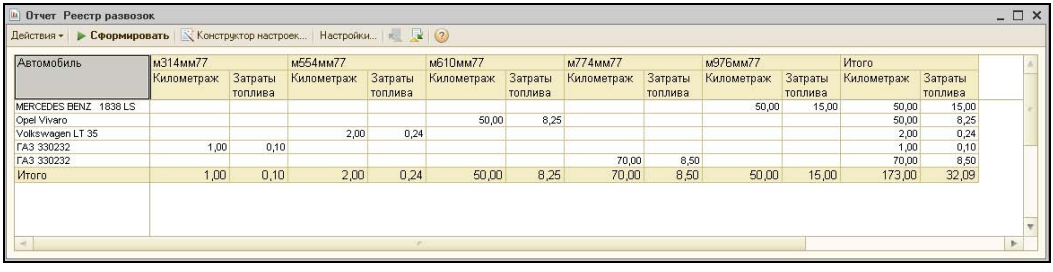



**Рис. 4.44.** Тот же самый отчет в виде таблицы

Точно так же, как мы это делали с реестром развозок, добавим в дерево конфигурации новый отчет, поля **Имя** и **Синоним** заполним так, как показано на рис. 4.47.

Теперь откроем схему компоновки данных одноименной кнопкой (рис. 4.48).

В окне схемы компоновки переименуем набор данных и нажмем кнопку **Конструктор запроса** (рис. 4.49).

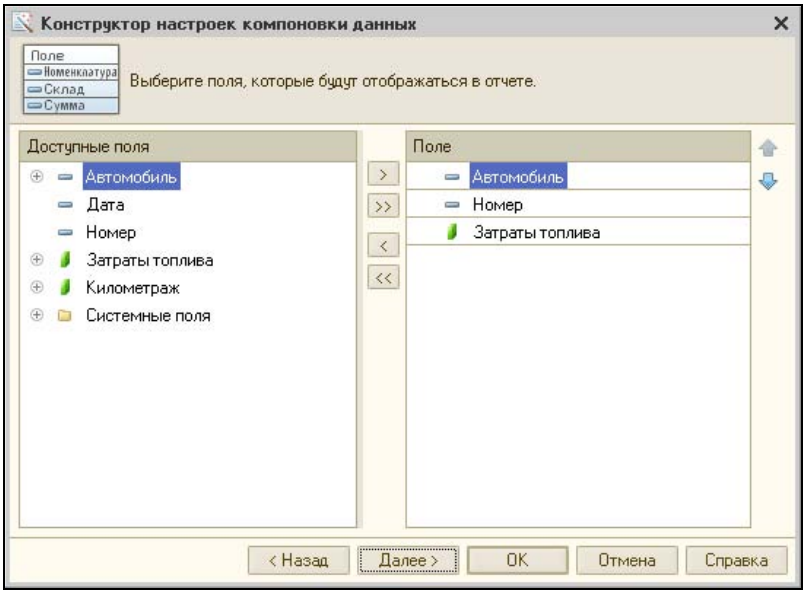



**Рис. 4.45.** Компоновка данных для диаграммы

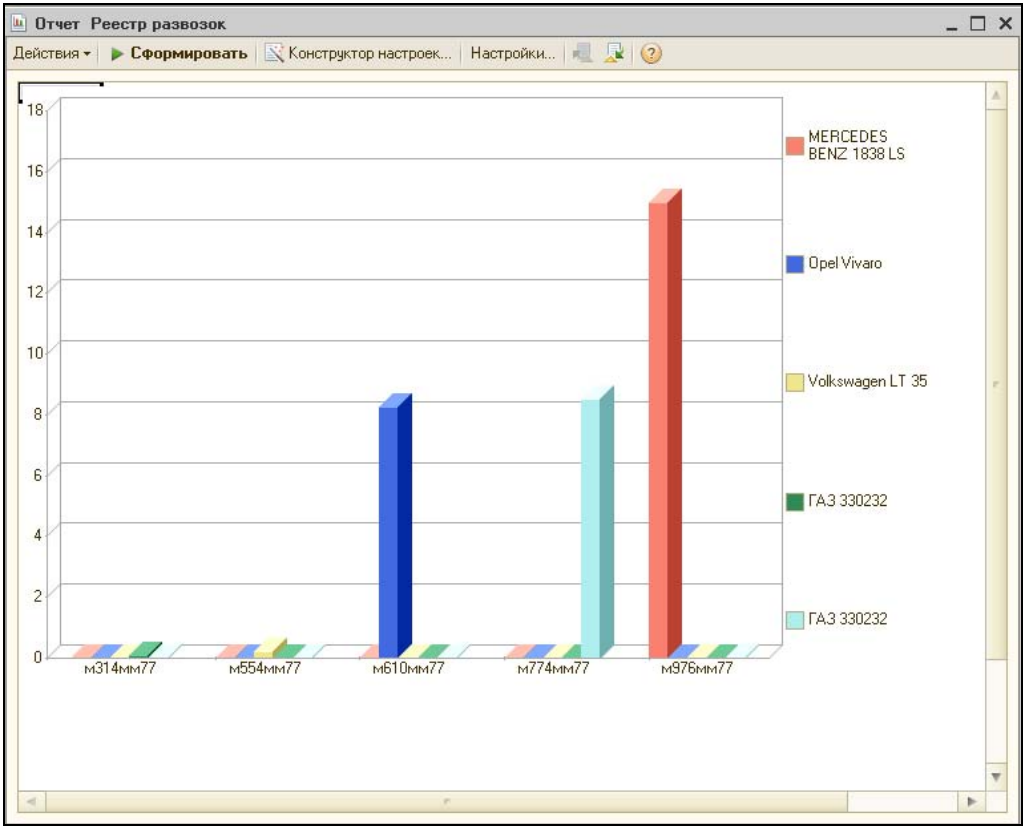



**Рис. 4.46.** Затраты топлива в виде диаграммы

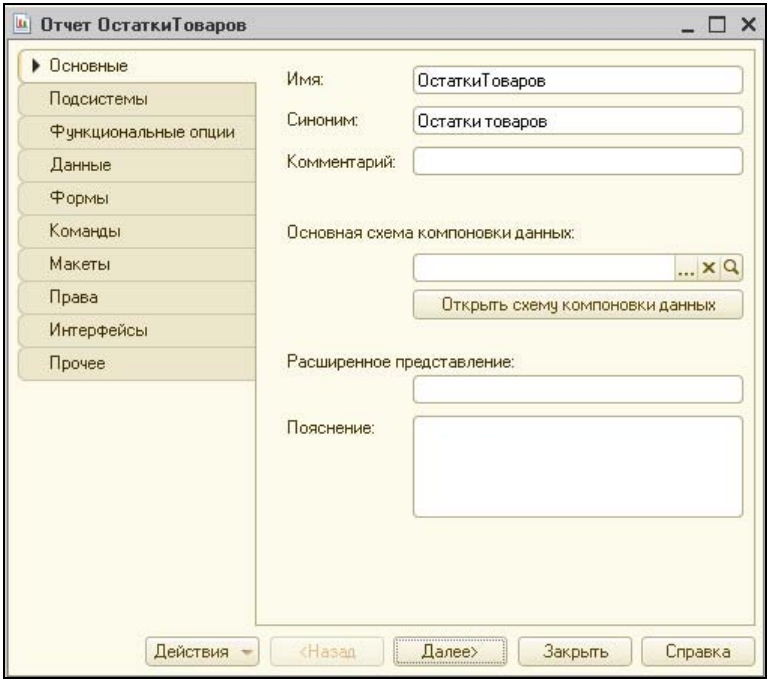



**Рис. 4.47.** Создаем отчет по остаткам товаров

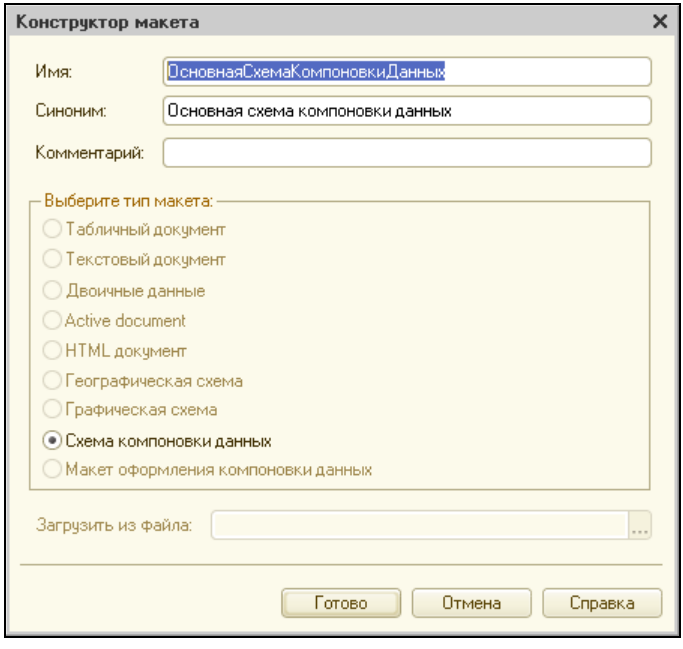



**Рис. 4.48.** Новая схема компоновки данных

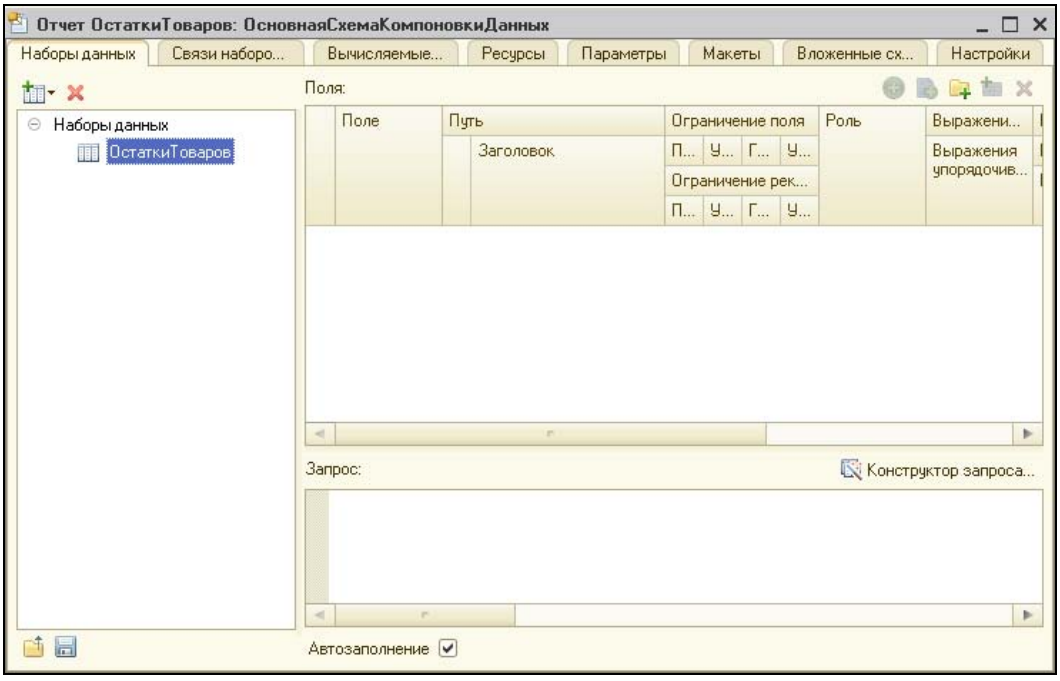



**Рис. 4.49.** Новый набор данных для компоновки

Интересующие нас данные об остатках товаров в конфигурации "Управление торговлей" находятся в регистре накопления "ТоварыНаСкладахОстатки". Вот его мы и добавляем в список таблиц. А в список полей мы добавляем измерения "Номенклатура" и "Склад" (последнее будет использоваться для отбора и детализации по складам), а также ресурс "Количество" (собственно, это и есть искомый остаток товара). В результате окно конструктора будет выглядеть так, как на рис. 4.50.

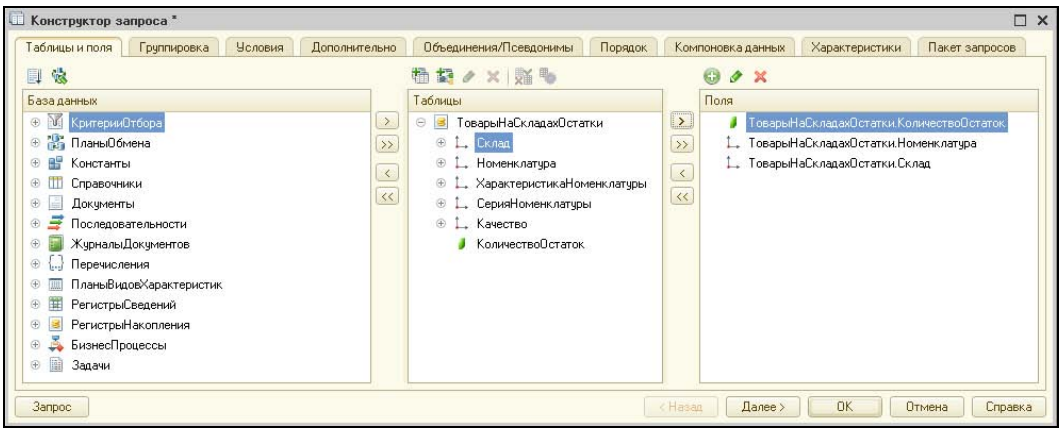



**Рис. 4.50.** Список таблиц и полей для использования в отчете. Данные получаем из регистров

Теперь окно наборов данных выглядит так, как показано на рис. 4.51. Сформировался и запрос, опять-таки автоматически, как результат нашей работы с конструктором запросов.

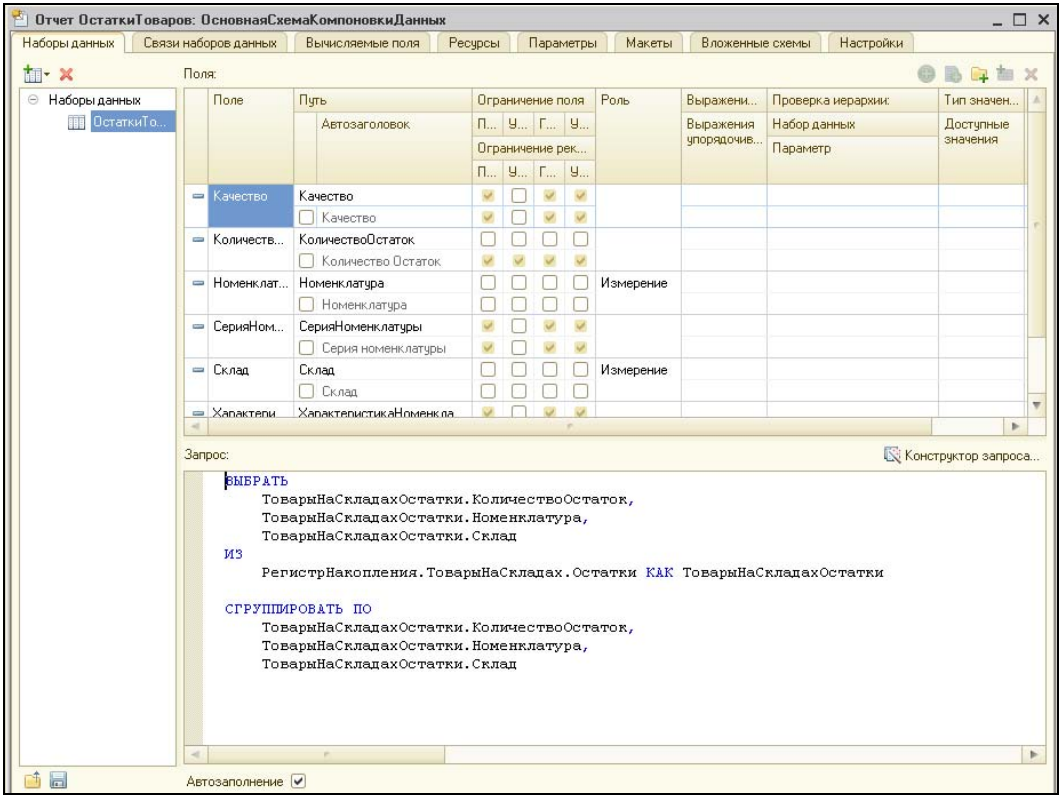



**Рис. 4.51.** Компоновка данных и запрос

Теперь перейдем на вкладку **Настройки** и зададим группировку (рис. 4.52).

Так же, как мы это делали в предыдущем отчете, добавим группировку (щелчок правой кнопкой мыши и выбор пункта **Новая группировка**). Группировку выбираем, как показано на рис. 4.53.

Группировать будем по складам, причем, в отличие от предыдущего отчета, будем использовать группировку в иерархии. Затем создадим подчиненную группировку **Номенклатура**. Поля отчета в нижней части окна выбираем следующие: **Склад**, **Номенклатура** и **КоличествоОстаток**. Результат должен быть такой, как показано на рис. 4.54.

Также давайте отсортируем данные по номенклатуре, для этого перейдите на вкладку **Сортировка**, добавьте поле и способ сортировки (рис. 4.55).

Отчет готов. Содержимое вкладки **Набор данных** должно соответствовать тому, что показано на рис. 4.56.

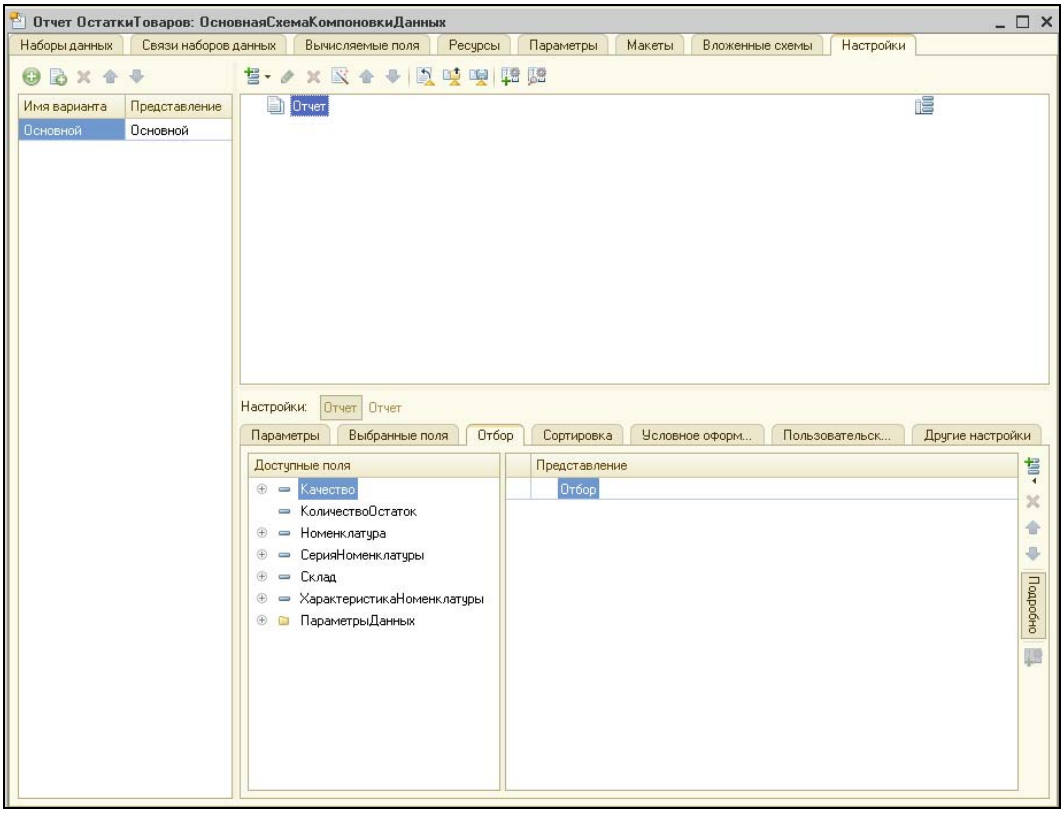



**Рис. 4.52.** Задаем группировки отчета

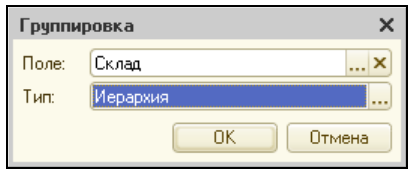



**Рис. 4.53.** Иерархическая группировка по складам

Сохраните конфигурацию, перейдите в режим Предприятия и запустите наш новый отчет через пункт меню **Операции | Отчет**. Результат должен выглядеть так, как показано на рис. 4.57.

Нам показаны остатки товаров с иерархической группировкой по складам и общим итогом. Теперь нажмем кнопку **Настройки** и осуществим отбор по выбранному складу (рис. 4.58).

Сделаем отбор по **Склад-холодильник**, нажмем кнопку **OK** и сформируем отчет заново. На этот раз отбор идет только по выбранному складу (рис. 4.59).

Теперь с помощью конструктора настроек выведем тот же отчет в форме таблицы. Выберем данный способ в конструкторе настроек (рис. 4.60).

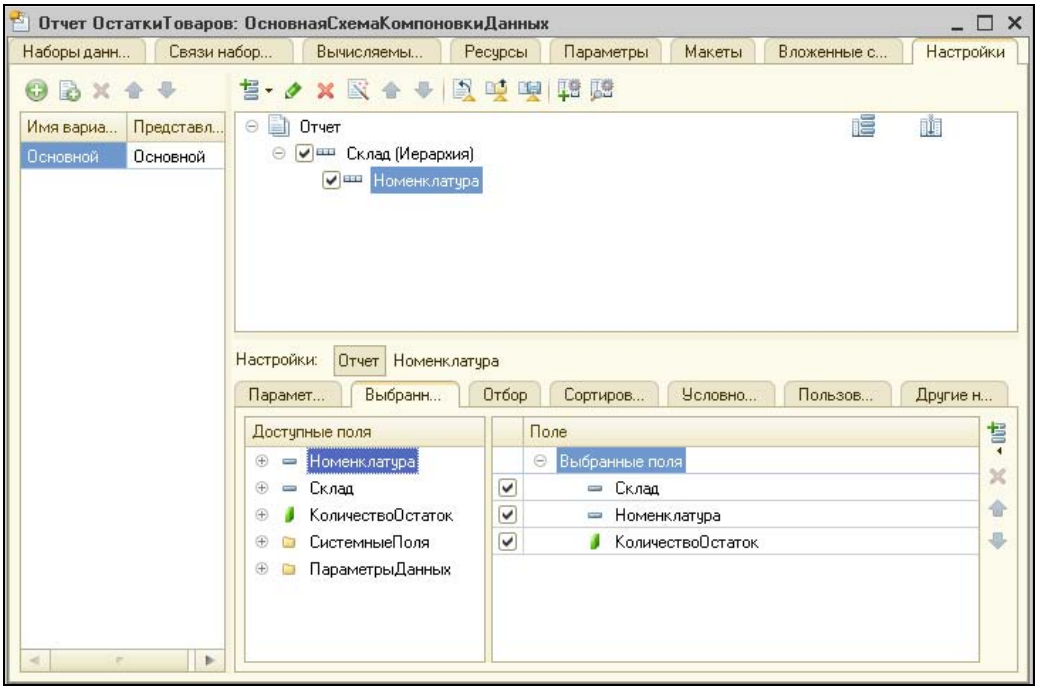



**Рис. 4.54.** Задаем группировку и поля для отчета

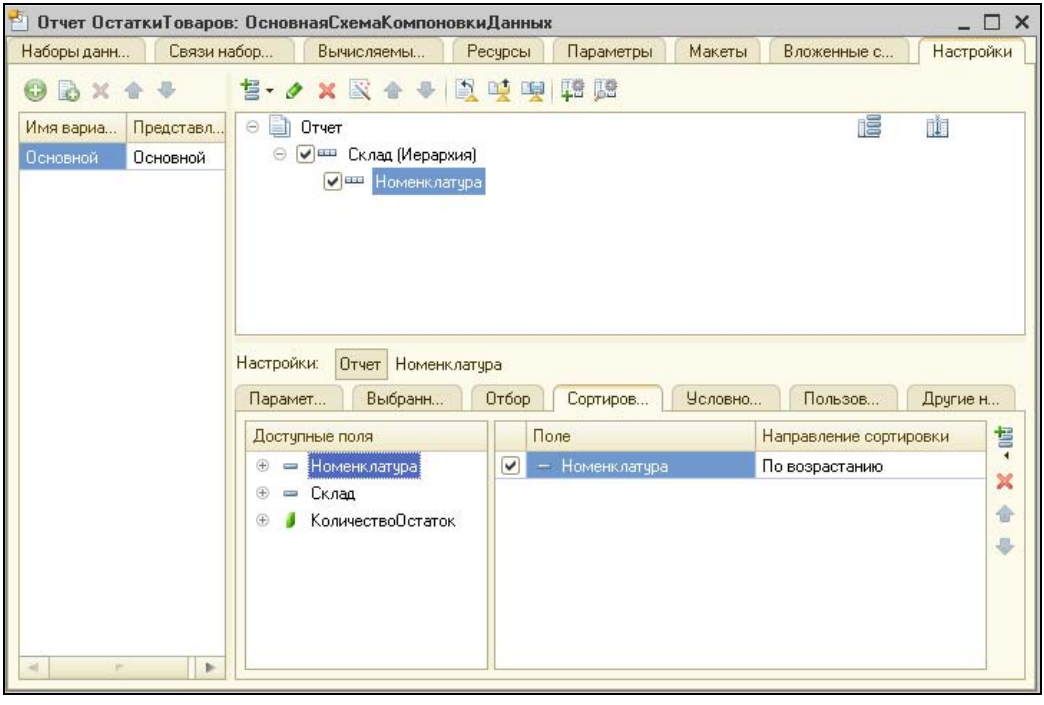



**Рис. 4.55.** Задаем сортировку по номенклатуре

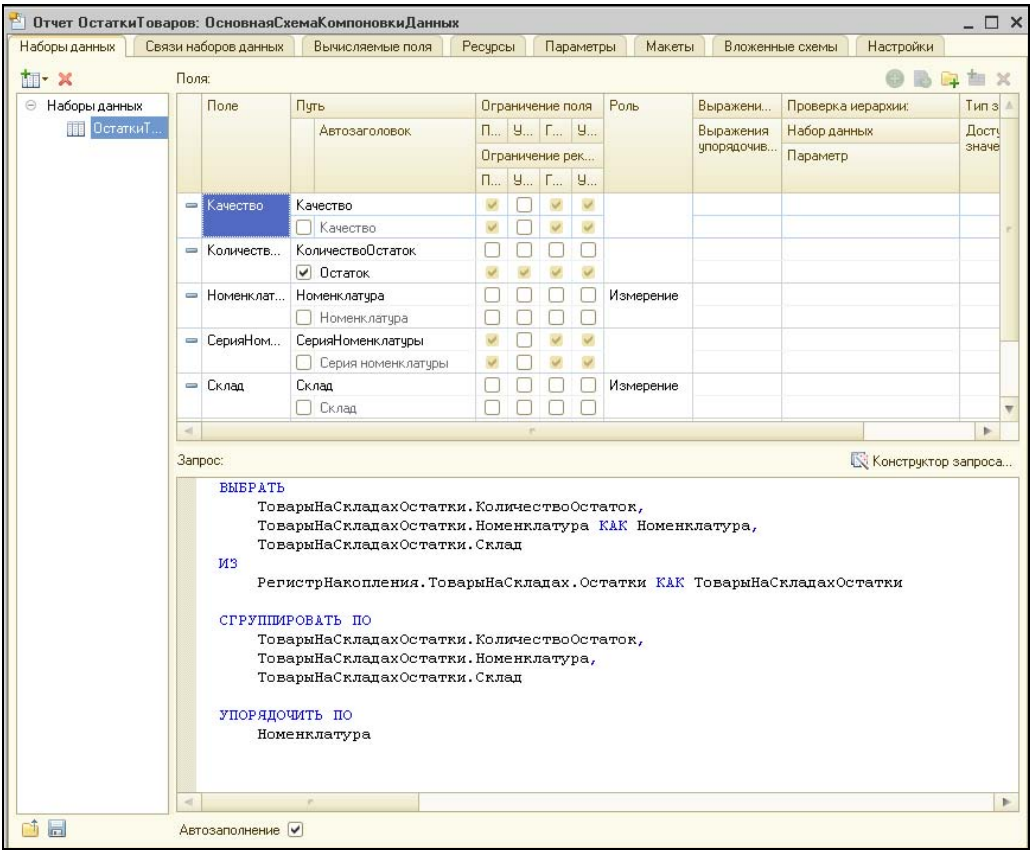



**Рис. 4.56.** Готовая схема компоновки данных. Запрос создан полностью автоматически

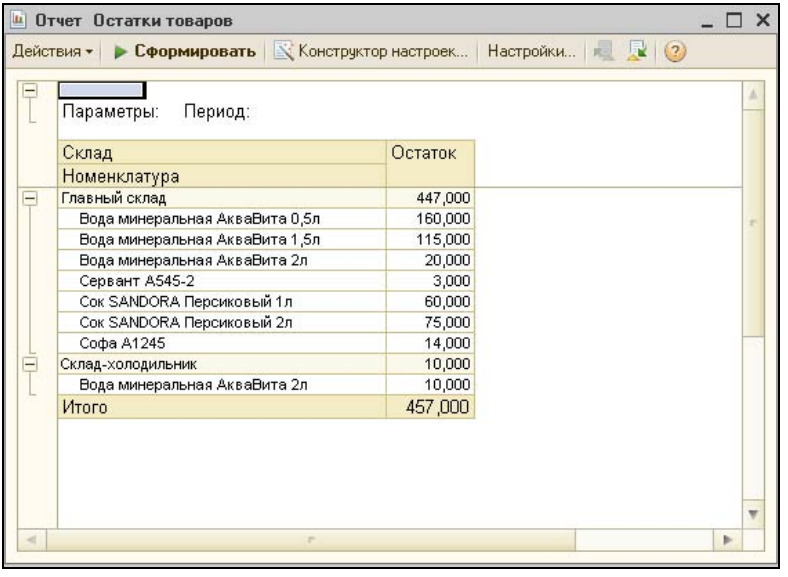



**Рис. 4.57.** Так выглядит сформированный отчет

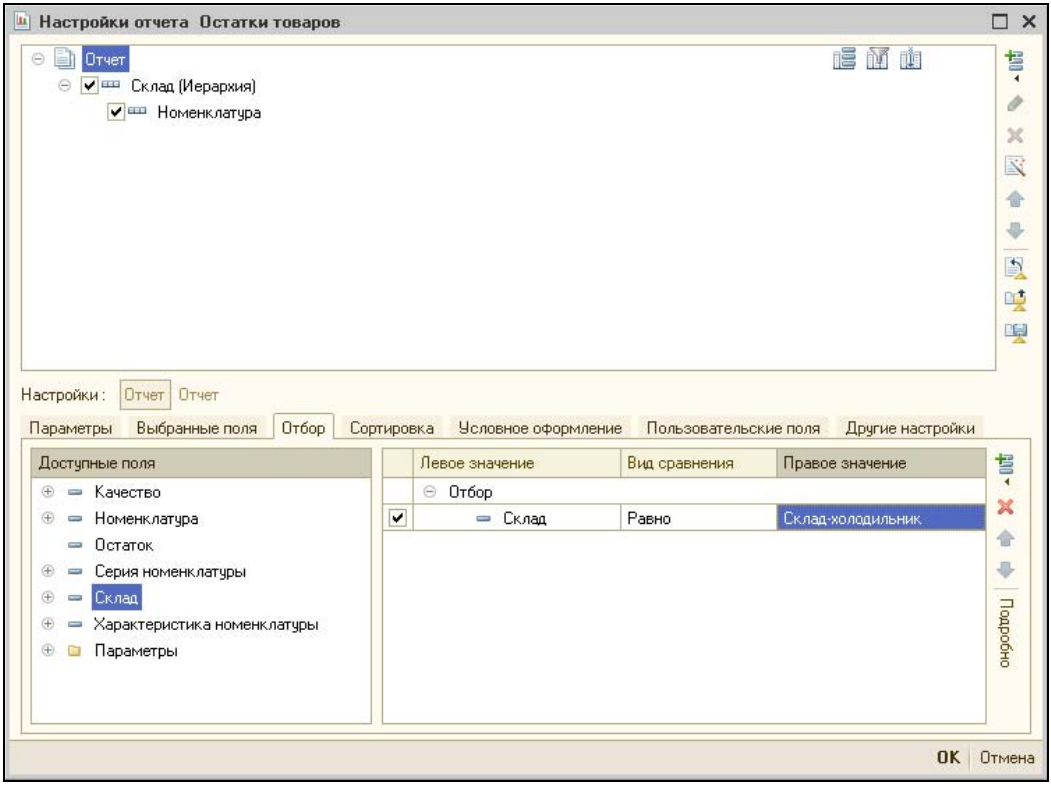



**Рис. 4.58.** В настройках добавляем отбор по складу

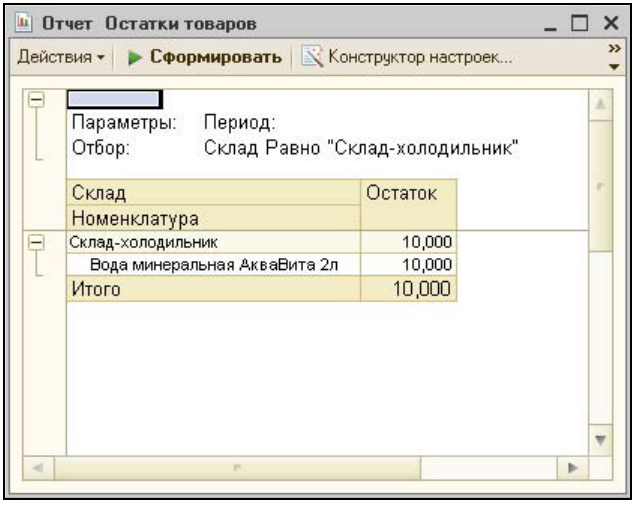



**Рис. 4.59.** Отчет по остаткам с отбором по складу

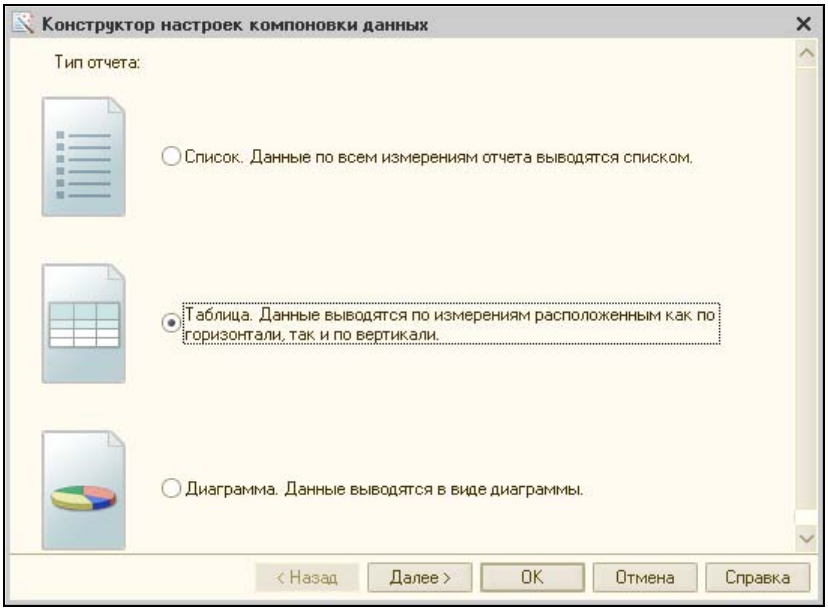



**Рис. 4.60.** Пересчитываем тот же отчет в форме таблицы

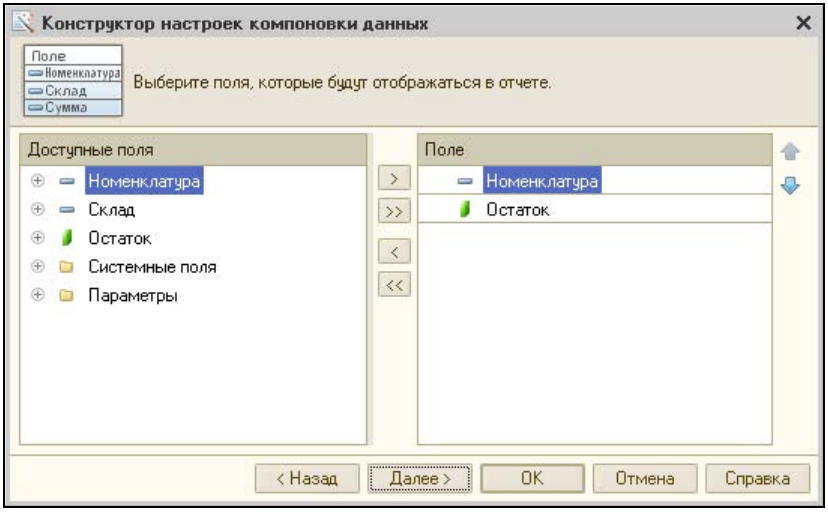



**Рис. 4.61.** Задаем табличные поля

Нажмем кнопку **Далее** и выберем поля отбора так, как показано на рис. 4.61.

Теперь снова нажмем кнопку **Далее** и настроим отображение данных в таблице: что отображать в строках, что в столбцах (рис. 4.62).

Теперь отчет сгруппирован уже иначе, хотя данные те же самые (рис. 4.63).

Работа с отчетом завершена. Много времени на него ушло? Совсем немного. И уж конечно меньше, чем ушло бы, если бы мы формировали этот отчет вручную. Мог-

ли бы мы его сделать вручную? Запросто. И иногда это делать приходится. Но всегда стоит помнить, какое мощное средство конструирования отчетов есть у нас в арсенале.

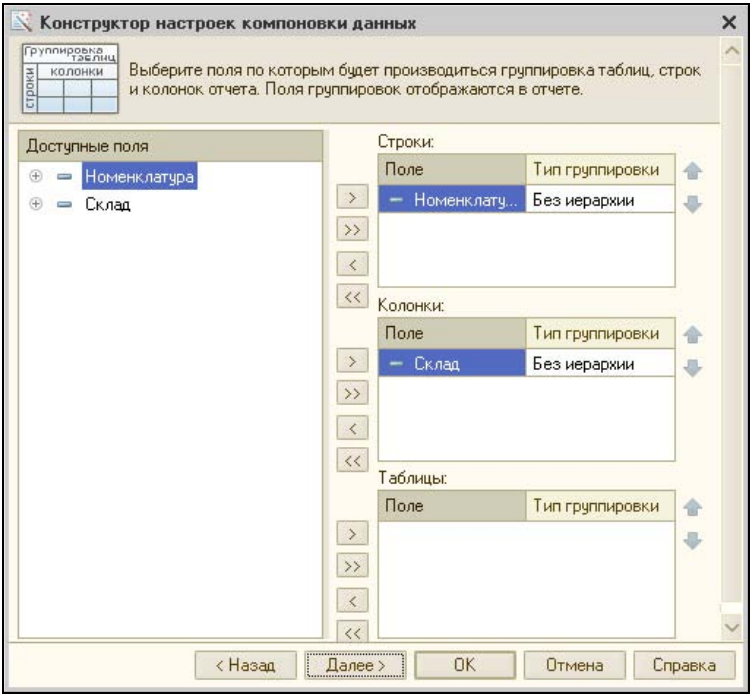
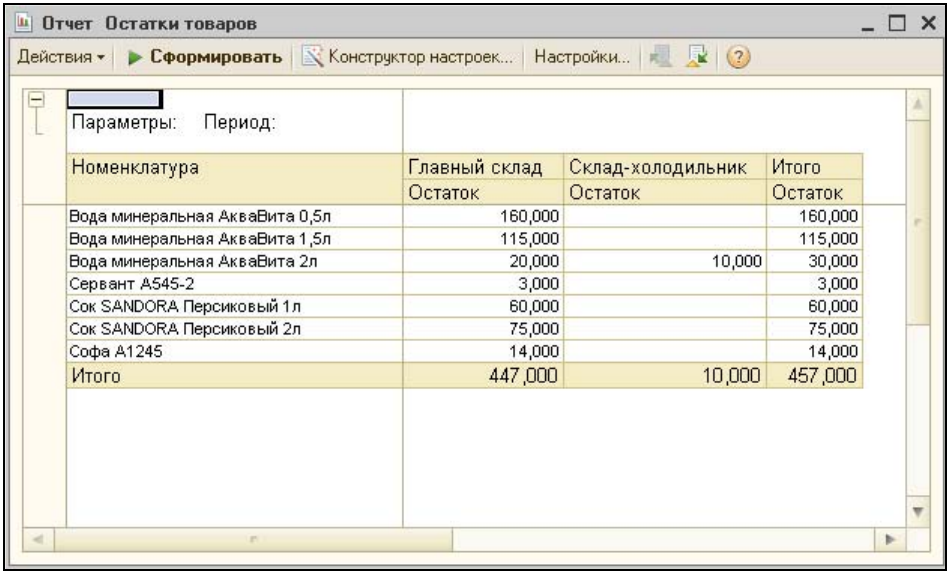



**Рис. 4.62.** Задаем структуру таблицы




| Номенклатура | Главный склад | Склад-холодильник | Итого |
|---|---|---|---|
| | Остаток | Остаток | Остаток |
| Вода минеральная АкваВита 0,5л | 160,000 | | 160,000 |
| Вода минеральная АкваВита 1,5л | 115,000 | | 115,000 |
| Вода минеральная АкваВита 2л | 20,000 | 10,000 | 30,000 |
| Сервант А545-2 | 3,000 | | 3,000 |
| Сок SANDORA Персиковый 1л | 60,000 | | 60,000 |
| Сок SANDORA Персиковый 2л | 75,000 | | 75,000 |
| Софа А1245 | 14,000 | | 14,000 |
| Итого | 447,000 | 10,000 | 457,000 |

**Рис. 4.63.** Отчет по остаткам на главном складе в форме таблицы

# Основные операторы встроенного языка запросов

В предыдущем разделе мы рассматривали автоматическое конструирование отчетов. На основании визуальных настроек в конструкторе автоматически формировался запрос на встроенном языке для работы с данными в базе. Однако работать с запросами можно и без конструктора, мало того, нередко это гораздо удобнее и продуктивнее, чем работа с использованием встроенного языка 1С, а работа с компоновкой данных и конструкторами не всегда представляется возможной. В этом разделе мы рассмотрим назначение и использование основных и наиболее употребимых операторов встроенного языка запросов.

Для работы с запросами создайте новую внешнюю обработку и назовите ее "РаботаСЗапросами". Форму создайте по умолчанию, самую обычную. Напомню, поэтапное создание внешней обработки мы рассматривали в *разд. "Наша первая обработка" главы 2*.

Итак, у нас есть пустая обработка, с одной-единственной пустой процедурой в модуле. Дальше будут приводиться примеры запросов, которые нужно будет поочередно прописывать в эту процедуру, после чего обработку можно запустить на выполнение и наглядно убедиться в результате работы каждого из рассматриваемых запросов.

## Выборка данных. *ВЫБРАТЬ... ИЗ... ГДЕ*

Запрос такого типа выбирает данные по указанному критерию из указанного места и при выполнении определенных условий.

Иначе говоря:

**ВЫБРАТЬ** *некоторые данные* **ИЗ** *размещение данных* **ГДЕ** *условие, при котором происходит выборка*

Если условие ГДЕ не проставлено, то выборка производится по всем данным отбираемого типа.

Следующий запрос выбирает все документы "РеализацияТоваровУслуг" (ВЫБРАТЬ *ссылки на документы ИЗ документов РеализацияТоваровУслуг*) и загружает их в динамически созданную таблицу значений. Таблица выводится на экран. При желании мы можем выбрать одну из записей и получить сообщение о том, какую именно запись выбрали.

```
ТекстЗапрос = Новый Запрос;
ТекстЗапрос.Текст =
  "ВЫБРАТЬ
  |Документ.РеализацияТоваровУслуг.Ссылка
  |ИЗ Документ.РеализацияТоваровУслуг";
ТЗ = ТекстЗапрос.Выполнить().Выгрузить();
ВыбранныйЭлемент = ТЗ.ВыбратьСтроку();
Сообщить("Выбран "+ВыбранныйЭлемент.Ссылка);
```

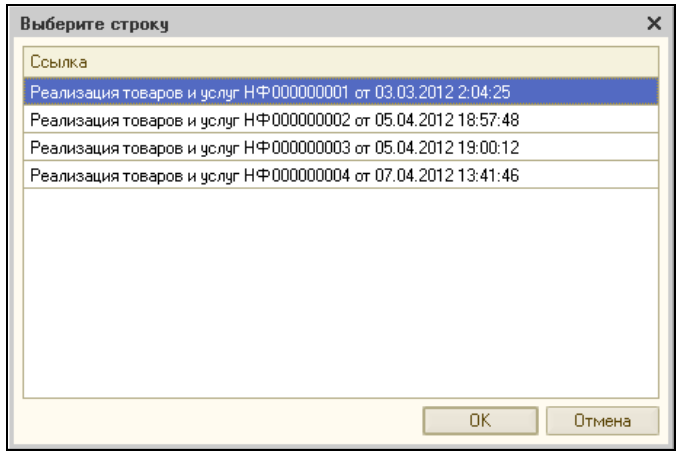



**Рис. 4.64.** Выборка документов "РеализацияТоваровУслуг"

Результат выполнения запроса показан на рис. 4.64.

Следующий запрос работает по тому же принципу, что и предыдущий, но выбирает не сами документы, а товары из таблицы "Товары" каждого из документов "РеализацияТоваровУслуг".

```
ТекстЗапрос = Новый Запрос;
ТекстЗапрос.Текст =
  "ВЫБРАТЬ
  |Документ.РеализацияТоваровУслуг.Товары.Номенклатура
  |ИЗ Документ.РеализацияТоваровУслуг.Товары";
ТЗ = ТекстЗапрос.Выполнить().Выгрузить();
ВыбранныйЭлемент = ТЗ.ВыбратьСтроку();
Сообщить("Выбран "+ВыбранныйЭлемент.Номенклатура);
```

Результат выполнения запроса показан на рис. 4.65.

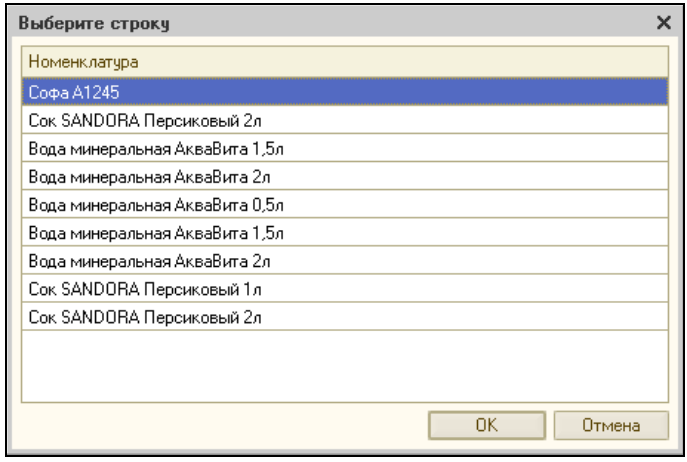



**Рис. 4.65.** Выборка товарного состава из документов "РеализацияТоваровУслуг"

Товары мы выбрали, но, как можно заметить, выбраны ВСЕ товары, которые проходили по ВСЕМ документам реализации, поэтому многие позиции повторяются. Если бы в реальной рабочей ситуации мы ставили целью получить список продаваемых товаров, такое нас никак бы не устроило, нам ведь интересен продаваемый ассортимент, а не масса повторяющихся строк. Поэтому в следующем запросе мы используем ключевое слово РАЗЛИЧНЫЕ и выберем только неповторяющиеся позиции.

```
ТекстЗапрос = Новый Запрос;
ТекстЗапрос.Текст =
  "ВЫБРАТЬ РАЗЛИЧНЫЕ
  |Документ.РеализацияТоваровУслуг.Товары.Номенклатура
  |ИЗ Документ.РеализацияТоваровУслуг.Товары";
ТЗ = ТекстЗапрос.Выполнить().Выгрузить();
ВыбранныйЭлемент = ТЗ.ВыбратьСтроку();
Сообщить("Выбран "+ВыбранныйЭлемент.Номенклатура);
```

Результат выполнения запроса показан на рис. 4.66.

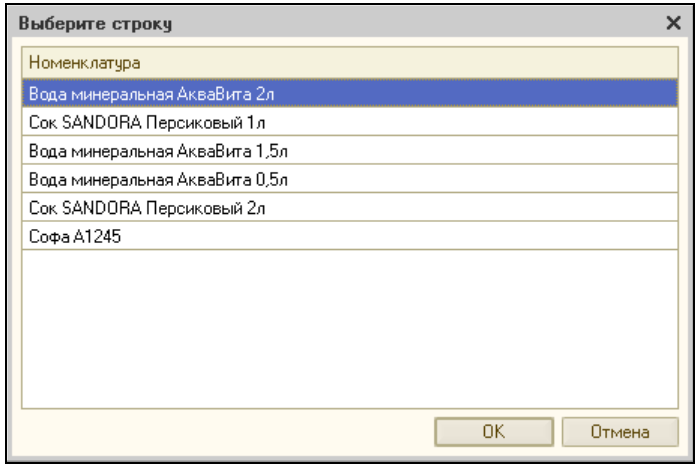



**Рис. 4.66.** Выборка уникальных товарных позиций из документов "РеализацияТоваровУслуг"

Разумеется, данные можно выбирать не по одному полю. В следующем запросе сделаем нормальную выборку данных из накладных, с количеством, ценами и суммами, выбираемые поля перечисляем через запятую.

```
ТекстЗапрос = Новый Запрос;
ТекстЗапрос.Текст =
  "ВЫБРАТЬ
  |Документ.РеализацияТоваровУслуг.Товары.Номенклатура,
  |Документ.РеализацияТоваровУслуг.Товары.ЕдиницаИзмерения,
  |Документ.РеализацияТоваровУслуг.Товары.Количество,
  |Документ.РеализацияТоваровУслуг.Товары.Цена,
  |Документ.РеализацияТоваровУслуг.Товары.Сумма
  |ИЗ Документ.РеализацияТоваровУслуг.Товары";
```

```
ТЗ = ТекстЗапрос.Выполнить().Выгрузить();
ВыбранныйЭлемент = ТЗ.ВыбратьСтроку();
Сообщить("Выбран "+ВыбранныйЭлемент.Номенклатура);
```

Результат выполнения запроса показан на рис. 4.67.

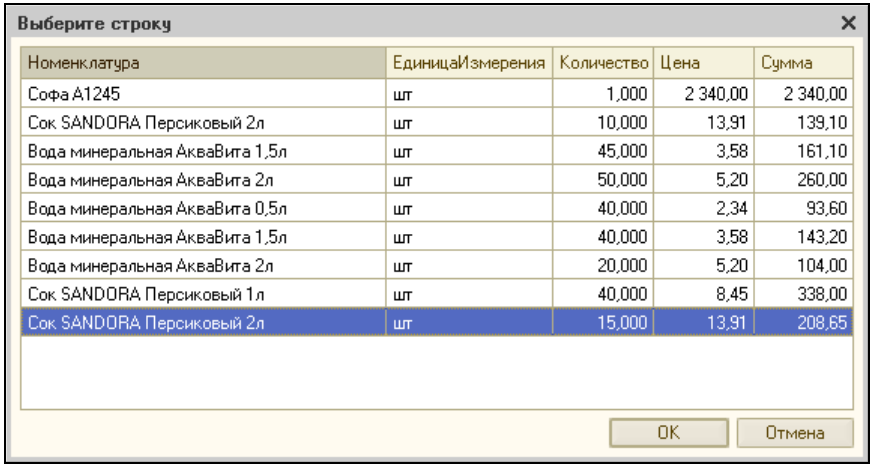



**Рис. 4.67.** Выборка в виде накладной

Неудобно все время обращаться по полному имени, как например, `Документ.РеализацияТоваровУслуг.Товары.ЕдиницаИзмерения`. Да и в шапке полученной таблицы заголовок **ЕдиницаИзмерения** не стильно смотрится. В следующем запросе мы повторим предыдущий, но начнем использовать *псевдонимы*. При объявлении псевдонима в запросе уже можно обращаться к нему, а не к имени.

```
ТекстЗапрос = Новый Запрос;
ТекстЗапрос.Текст =
  "ВЫБРАТЬ
  |СписокТоваров.Номенклатура КАК Товар,
  |СписокТоваров.ЕдиницаИзмерения КАК Ед,
  |СписокТоваров.Количество КАК Колво,
  |СписокТоваров.Цена,
  |СписокТоваров.Сумма
  |ИЗ Документ.РеализацияТоваровУслуг.Товары КАК СписокТоваров";
ТЗ = ТекстЗапрос.Выполнить().Выгрузить();
ВыбранныйЭлемент = ТЗ.ВыбратьСтроку();
Сообщить("Выбран "+ВыбранныйЭлемент.Товар);
```

Документу "Документ.РеализацияТоваровУслуг.Товары" мы присвоили псевдоним `СписокТоваров`. И можем писать в списке отбираемых значений `СписокТоваров.Номенклатура` вместо длинного `Документ.РеализацияТоваровУслуг.Товары.Номенклатура`. Некоторым данным выборки мы псевдонимы не присваиваем, например `СписокТоваров.Сумма`. Но тут имя и так состоит из одного слова "Сумма", поэтому без псевдонима в этом случае мы обошлись.

Результат выполнения запроса показан на рис. 4.68.

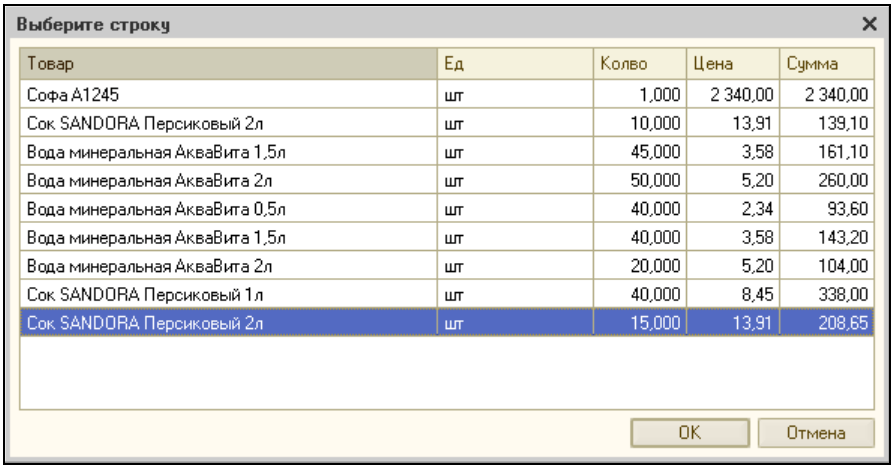



**Рис. 4.68.** Та же выборка в виде накладной, но с использованием псевдонимов полей

В следующем запросе сделаем ту же выборку, но уже с условием: отбирать только те позиции, где сумма больше либо равна 200.

```
ТекстЗапрос = Новый Запрос;
ТекстЗапрос.Текст =
  "ВЫБРАТЬ
  |СписокТоваров.Номенклатура КАК Товар,
  |СписокТоваров.ЕдиницаИзмерения КАК Ед,
  |СписокТоваров.Количество КАК Колво,
  |СписокТоваров.Цена,
  |СписокТоваров.Сумма
  |ИЗ Документ.РеализацияТоваровУслуг.Товары КАК СписокТоваров
  |ГДЕ Сумма >= 200";
ТЗ = ТекстЗапрос.Выполнить().Выгрузить();
ВыбранныйЭлемент = ТЗ.ВыбратьСтроку();
```

Результат выполнения запроса показан на рис. 4.69.

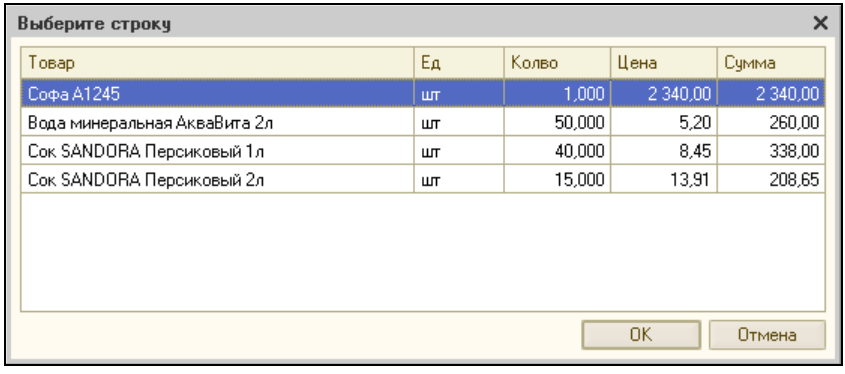



**Рис. 4.69.** Выборка с условием: сюда попадают только записи с суммой, большей либо равной 200

Условие может быть не одно, в запросах могут использоваться ключевые слова И, ИЛИ и НЕ. Например, в следующем запросе мы сделаем выборку по крупным покупкам, где сумма больше либо равна 1000 или количество больше либо равно 40.

```
ТекстЗапрос = Новый Запрос;
  ТекстЗапрос.Текст =
  "ВЫБРАТЬ
  |СписокТоваров.Номенклатура КАК Товар,
  |СписокТоваров.ЕдиницаИзмерения КАК Ед,
  |СписокТоваров.Количество КАК Колво,
  |СписокТоваров.Цена,
  |СписокТоваров.Сумма
  |ИЗ Документ.РеализацияТоваровУслуг.Товары КАК СписокТоваров
  |ГДЕ Сумма >= 1000 ИЛИ Количество >= 40";
ТЗ = ТекстЗапрос.Выполнить().Выгрузить();
ВыбранныйЭлемент = ТЗ.ВыбратьСтроку();
```

Результат выполнения запроса показан на рис. 4.70.

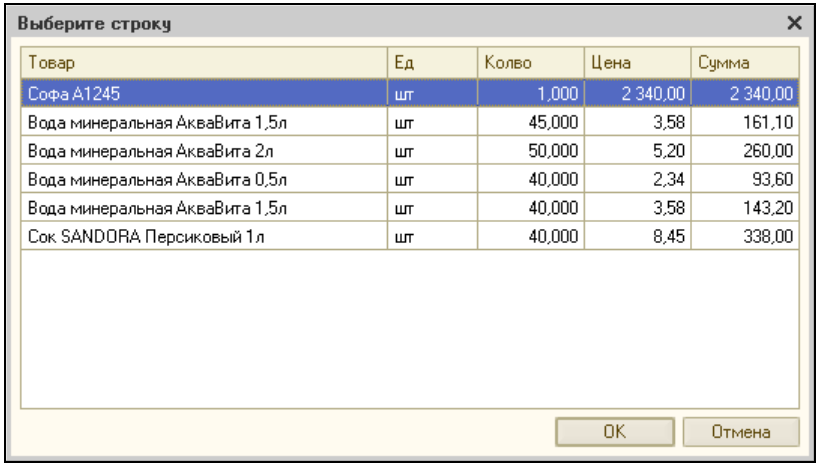



**Рис. 4.70.** Здесь выборка производится или при сумме, большей либо равной 1000, или при количестве, большем либо равным 40

Бывает так, что нужно отобрать данные по части строки. Например, если вернуться к созданному ранее документу "Развозка", нам может быть интересно, какие развозки и по какому адресу произвел автомобиль Opel Vivaro. Но писать наименование в запросе полностью неудобно, особенно если оно длинное, присутствует модель, а еще возможны и пробелы. Так что хорошо бы иметь возможность делать выборку по некоторому вхождению в строку. Такая возможность есть. Выполним следующий запрос, который позволяет выбрать данные из документов "Развозка", где в наименовании автомобиля встречается слово "Opel":

```
ТекстЗапрос = Новый Запрос;
Текст = "Opel";
ТекстЗапрос.УстановитьПараметр("Подстрока", "%" + Текст + "%");
```

```
ТекстЗапрос.Текст =
  "ВЫБРАТЬ
  |Развозка.Номер,
  |Развозка.Дата,
  |Развозка.Адрес
  |ИЗ Документ.Развозка КАК Развозка
  |ГДЕ Развозка.Автомобиль.Наименование ПОДОБНО &Подстрока";
ТЗ = ТекстЗапрос.Выполнить().Выгрузить();
ВыбранныйЭлемент = ТЗ.ВыбратьСтроку();
```

Как мы построили запрос? Мы использовали параметр. Параметр запроса задается перед текстом выборки и может быть в дальнейшем использован в запросе.

В нашем примере использовалась процедура `УстановитьПараметр`, как раз и устанавливающая параметр, который впоследствии может использоваться в тексте запроса.

Синтаксис:

**УстановитьПараметр(*Имя*, *Значение*);**

Здесь: *Имя* — имя параметра; *Значение* — значение устанавливаемого параметра.

В значении могут быть использованы следующие служебные символы:

- `%` (знак процента) — последовательность, содержащая любое количество любых символов;
- _ (подчеркивание) — один произвольный символ;
- `[...]` (один или несколько символов в квадратных скобках) — любой одиночный символ из перечисленных внутри квадратных скобок;
- `[^...]` (значок отрицания, за которым следует один или несколько символов, заключены в квадратные скобки) — любой одиночный символ, кроме тех, которые перечислены следом за значком отрицания.

В нашем примере `"Подстрока"` — это имя параметра, переменную `Текст` мы определили заранее. Это `"Opel"`. Но кроме этого, перед словом "Opel" и после него может присутствовать любое количество символов, так что мы не будем зависеть от того, указана марка машины в названии, не указана, есть лишние пробелы или нет.

Обращение к параметру происходит с использованием значка `&`.

Кроме того, мы использовали в запросе ключевое слово `ПОДОБНО`. Собственно, это как раз и есть отбор по подстроке.

Результат выполнения запроса показан на рис. 4.71.

Конечно же, когда мы говорим о выборках данных, мы не можем пропустить выбор всех значений. Например, следующий запрос выберет все поля (даже служебные) из документов "Развозка".

```
ТекстЗапрос = Новый Запрос;
ТекстЗапрос.Текст =
  "ВЫБРАТЬ
  |*
  |ИЗ Документ.Развозка КАК Развозка";
```

```
ТЗ = ТекстЗапрос.Выполнить().Выгрузить();
ВыбранныйЭлемент = ТЗ.ВыбратьСтроку();
```

Результат выполнения запроса показан на рис. 4.72. Как видите, запрос отобрал именно все поля, в том числе "ВерсияДанных" и "ПометкаУдаления".

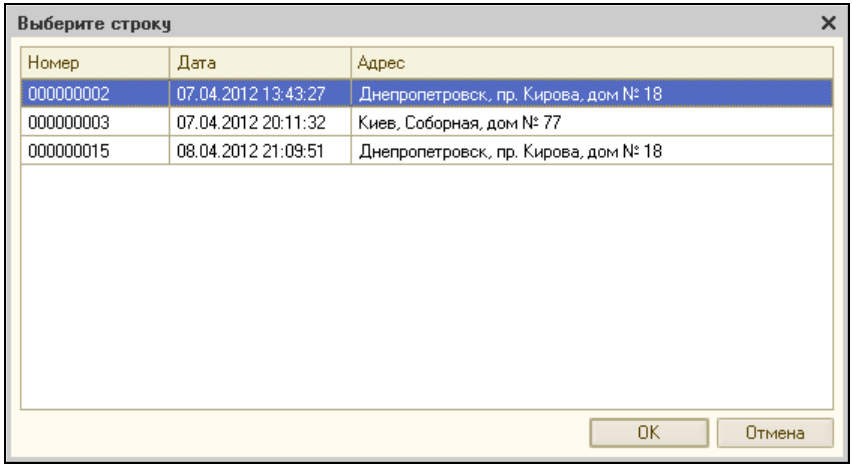



**Рис. 4.71.** Выбираем развозки, сделанные на автомобиле "Opel"

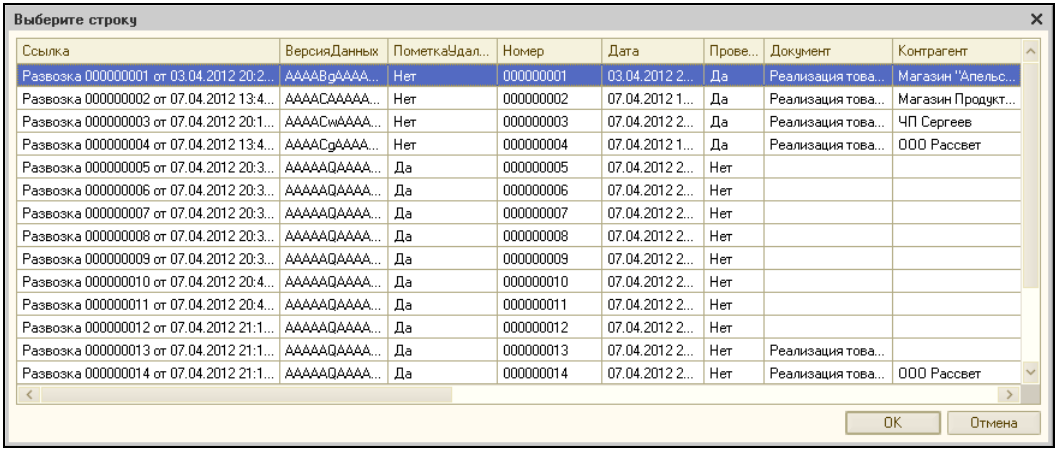



**Рис. 4.72.** А так выглядит результат запроса, если выбрать все (*) поля из документа "Развозка"

Следующий запрос выбирает также все записи, на этот раз из справочника "Автомобили":

```
ТекстЗапрос = Новый Запрос;
ТекстЗапрос.Текст =
  "ВЫБРАТЬ
  |*
  |ИЗ Справочник.Автомобили КАК Авто";
ТЗ = ТекстЗапрос.Выполнить().Выгрузить();
ВыбранныйЭлемент = ТЗ.ВыбратьСтроку();
```

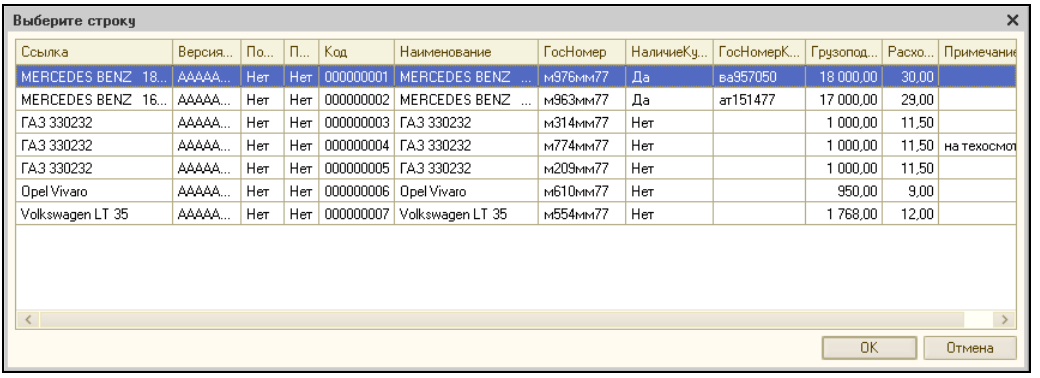



**Рис. 4.73.** Выборка из справочника "Автомобили"

Результат выполнения запроса показан на рис. 4.73.

Выборку данных можно осуществлять из нескольких таблиц сразу. Например, если мы хотим вывести список отгрузок товара с информацией о развозках, то можем выполнить следующий запрос:

```
ТекстЗапрос = Новый Запрос;
ТекстЗапрос.Текст =
  "ВЫБРАТЬ
  |Реализация.Номер,
  |Реализация.Дата КАК Дата,
  |Реализация.Контрагент КАК Покупатель,
  |Реализация.ДоговорКонтрагента КАК Договор,
  |Развозка.Ссылка
  |ИЗ Документ.РеализацияТоваровУслуг КАК Реализация,
                                        Документ.Развозка КАК Развозка
  |ГДЕ Реализация.Номер = Развозка.Документ.Номер";
ТЗ = ТекстЗапрос.Выполнить().Выгрузить();
ВыбранныйЭлемент = ТЗ.ВыбратьСтроку();
```

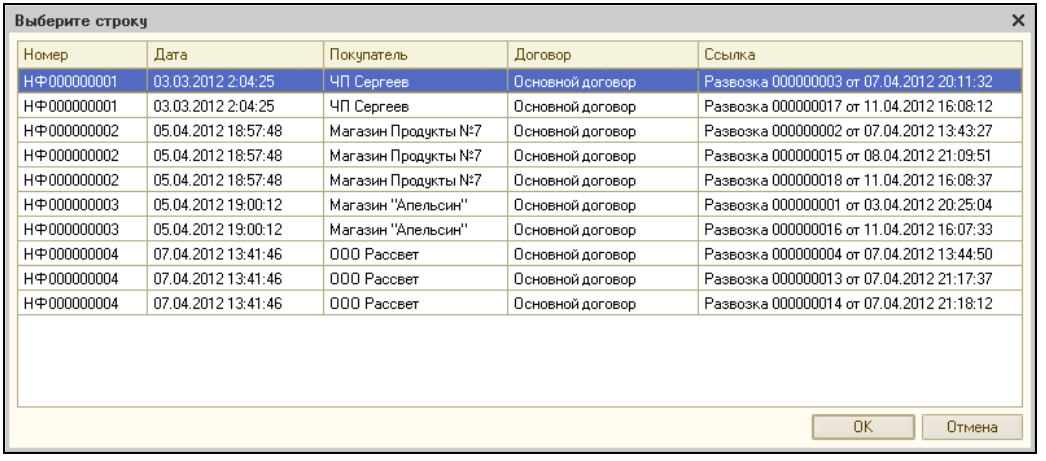



**Рис. 4.74.** Совместная выборка из двух таблиц — список реализаций и связанные с ними развозки

Результат выполнения запроса показан на рис. 4.74. Выборка производится из документов "РеализацияТоваровУслуг" и "Развозка". Для каждого документа реализации мы видим, какой именно документ развозки к нему относится. В данном примере каждая отгрузка товаров развозилась в несколько этапов.

## Сортировка и группировка. *УПОРЯДОЧИТЬ ПО* и *СГРУППИРОВАТЬ ПО*

До сих пор мы делали выборки данных подряд, без всякого упорядочивания. Однако в языке запросов присутствуют возможности группировки и сортировки данных. Например, в следующем запросе мы сделаем выборку по продажам товаров, сразу группируя товары и суммируя данные по количеству отгруженного товара и общей стоимости (функцию суммирования мы рассмотрим далее). Принцип тот же самый, что и при сворачивании таблицы значений во встроенном языке 1С с суммированием числовых значений.

```
ТекстЗапрос = Новый Запрос;
ТекстЗапрос.Текст =
  "ВЫБРАТЬ
  |Док.Номенклатура КАК Товар,
  |СУММА (Док.Количество) КАК Колво,
  |СУММА (Док.Сумма) КАК Сумма
  |ИЗ Документ.РеализацияТоваровУслуг.Товары КАК Док
  |СГРУППИРОВАТЬ ПО Док.Номенклатура";
ТЗ = ТекстЗапрос.Выполнить().Выгрузить();
ВыбранныйЭлемент = ТЗ.ВыбратьСтроку();
```

Результат выполнения запроса показан на рис. 4.75.

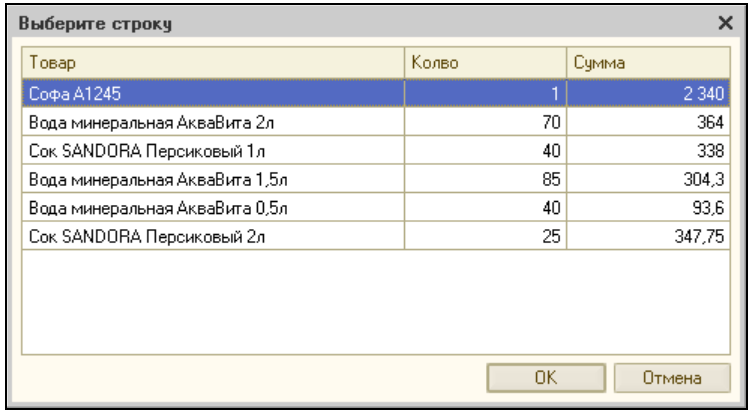



**Рис. 4.75.** Выборка с группировкой по номенклатуре

А в следующем запросе мы выберем данные аналогично тому, как это делали в примере, результат которого показан на рис. 4.68, но отсортируем данные по количеству (УПОРЯДОЧИТЬ ПО). По умолчанию сортировка производится по возрастанию.

```
ТекстЗапрос = Новый Запрос;
ТекстЗапрос.Текст =
  "ВЫБРАТЬ
  |Док.Номенклатура КАК Товар,
  |Док.ЕдиницаИзмерения КАК Ед,
  |Док.Количество КАК Колво,
  |Док.Цена,
  |Док.Сумма
  |ИЗ Документ.РеализацияТоваровУслуг.Товары КАК Док
  |УПОРЯДОЧИТЬ ПО Док.Количество";
ТЗ = ТекстЗапрос.Выполнить().Выгрузить();
ВыбранныйЭлемент = ТЗ.ВыбратьСтроку();
```

Результат выполнения запроса показан на рис. 4.76.

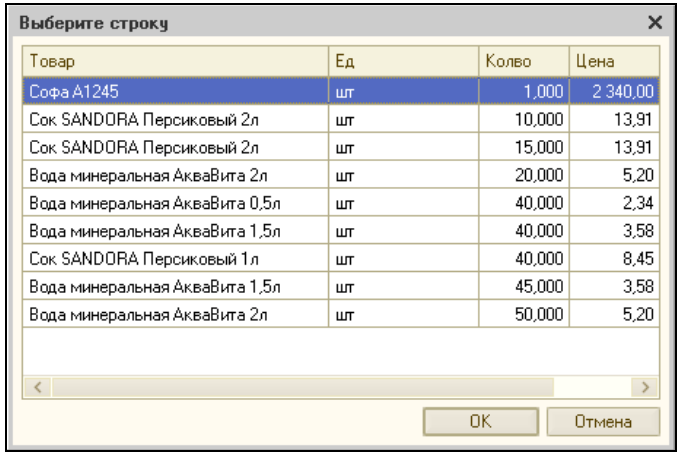



**Рис. 4.76.** Выборка с сортировкой по количеству товара

А теперь мы отберем те же самые продажи товаров, но "свернем" их с суммированием результатов. Отбираются неповторяющиеся (РАЗЛИЧНЫЕ) значения с группировкой по номенклатуре и отсортированные по сумме в порядке убывания (УБЫВ). Получился небольшой топ продаж, по которому мы можем судить, какой товар лучше продается.

```
ТекстЗапрос = Новый Запрос;
ТекстЗапрос.Текст =
  "ВЫБРАТЬ РАЗЛИЧНЫЕ
  |Док.Номенклатура,
  |СУММА (Док.Сумма) КАК Сум
  |ИЗ Документ.РеализацияТоваровУслуг.Товары КАК Док
  |СГРУППИРОВАТЬ ПО Док.Номенклатура
  |УПОРЯДОЧИТЬ ПО Сум УБЫВ";
ТЗ = ТекстЗапрос.Выполнить().Выгрузить();
ВыбранныйЭлемент = ТЗ.ВыбратьСтроку();
```

Результат выполнения запроса показан на рис. 4.77.

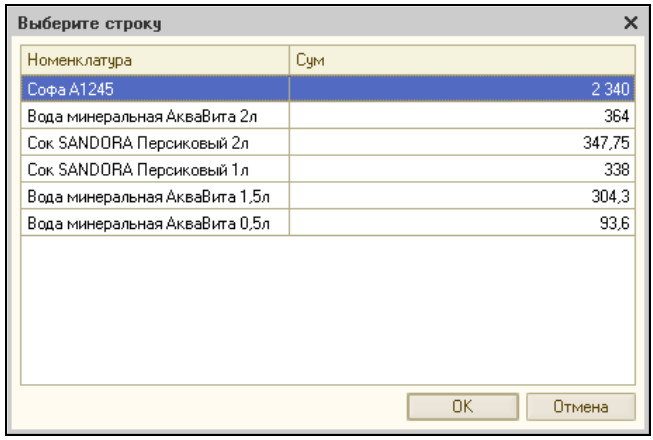



**Рис. 4.77.** Выборка с сортировкой по сумме в порядке убывания

В следующем запросе мы выберем все продажи без группировки, но с сортировкой по количеству в порядке убывания.

```
ТекстЗапрос = Новый Запрос;
ТекстЗапрос.Текст =
  "ВЫБРАТЬ
  |Док.Номенклатура КАК Товар,
  |Док.ЕдиницаИзмерения КАК Ед,
  |Док.Количество КАК Колво,
  |Док.Цена,
  |Док.Сумма
  |ИЗ Документ.РеализацияТоваровУслуг.Товары КАК Док
  |УПОРЯДОЧИТЬ ПО Док.Количество УБЫВ";
ТЗ = ТекстЗапрос.Выполнить().Выгрузить();
ВыбранныйЭлемент = ТЗ.ВыбратьСтроку();
```

Результат выполнения запроса показан на рис. 4.78.

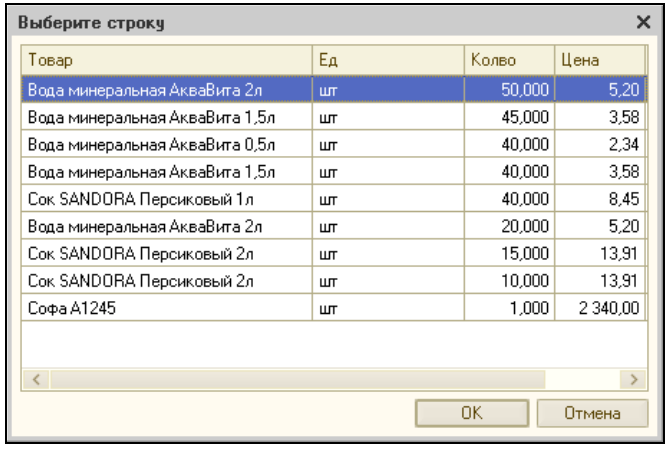



**Рис. 4.78.** Выборка с сортировкой по количеству в порядке убывания

## Агрегатные функции в запросе: *МИНИМУМ*, *МАКСИМУМ*, *СРЕДНЕЕ*, *КОЛИЧЕСТВО*, *СУММА*

Разумеется, в языке запросов имеются функции для расчета минимальных, максимальных и усредненных значений выбираемых величин, а также функция их суммирования. В предыдущем разделе мы уже сталкивались с функцией суммирования.

В следующем запросе мы выберем "Сумму сумм" всех реализаций товаров, т. е. по сути итоговую цифру продаж.

```
ТекстЗапрос = Новый Запрос;
ТекстЗапрос.Текст =
  "ВЫБРАТЬ
  |СУММА (Документ.РеализацияТоваровУслуг.Товары.Сумма)
  |ИЗ Документ.РеализацияТоваровУслуг.Товары";
ТЗ = ТекстЗапрос.Выполнить().Выгрузить();
ВыбранныйЭлемент = ТЗ.ВыбратьСтроку();
```

Произведен выбор суммы по полю `Документ.РеализацияТоваровУслуг.Товары.Сумма`, т. е. получена как раз "сумма сумм".

Результат выполнения запроса показан на рис. 4.79.

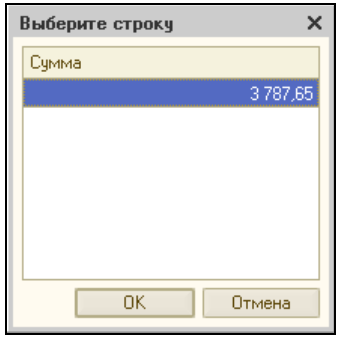



**Рис. 4.79.** Выборка общей суммы по всем документам "РеализацияТоваровУслуг"

А в следующем запросе мы выберем список товаров, которые были отгружены, и три значения по каждому: минимальную сумму отгрузки по данному товару, максимальную и среднее значение.

```
ТекстЗапрос = Новый Запрос;
ТекстЗапрос.Текст =
  "ВЫБРАТЬ
  |Док.Номенклатура КАК Товар,
  |МИНИМУМ (Док.Сумма) КАК Мин_Сумма,
  |СРЕДНЕЕ (Док.Сумма) КАК Средняя_Сумма,
  |МАКСИМУМ (Док.Сумма) КАК Макс_Сумма
  |ИЗ Документ.РеализацияТоваровУслуг.Товары КАК Док
  |СГРУППИРОВАТЬ ПО Док.Номенклатура";
```

```
ТЗ = ТекстЗапрос.Выполнить().Выгрузить();
ВыбранныйЭлемент = ТЗ.ВыбратьСтроку();
```

Результат выполнения запроса показан на рис. 4.80. По первой позиции все три суммы одинаковы, поскольку отгрузка данного товара была всего один раз.

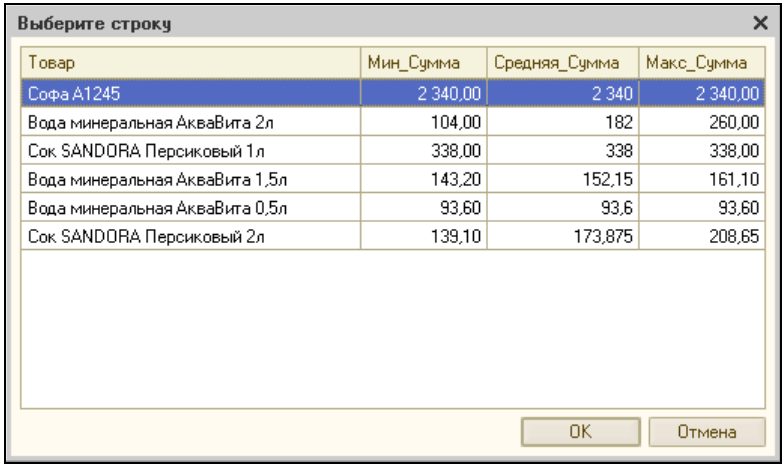



**Рис. 4.80.** Отбор по суммам в накладных — минимальной, максимальной и средней

В следующем запросе мы подсчитаем количество позиций номенклатуры, которые были отгружены. Для этого используется КОЛИЧЕСТВО. Также обязательно ключевое слово РАЗЛИЧНЫЕ, поскольку без него мы выберем не ассортимент, а общее количество строк по всем отгрузкам.

```
ТекстЗапрос = Новый Запрос;
ТекстЗапрос.Текст =
  "ВЫБРАТЬ
  |КОЛИЧЕСТВО (РАЗЛИЧНЫЕ Док.Номенклатура) КАК Ассортимент
  |ИЗ Документ.РеализацияТоваровУслуг.Товары КАК Док";
ТЗ = ТекстЗапрос.Выполнить().Выгрузить();
ВыбранныйЭлемент = ТЗ.ВыбратьСтроку();
```

Результат выполнения запроса показан на рис. 4.81.

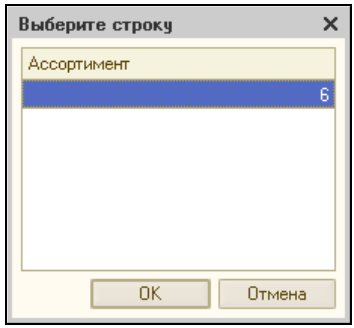



**Рис. 4.81.** Расчет количества уникальных товарных позиций, участвующих в реализациях

## Ключевое слово *ИМЕЮЩИЕ*

Еще один способ задать условие для выборки. Применим только для агрегатных функций (МИНИМУМ, МАКСИМУМ, СРЕДНЕЕ, КОЛИЧЕСТВО, СУММА).

В следующем запросе мы отберем неповторяющиеся значения номенклатуры с суммой, превышающей 200, и отсортируем результат по сумме в порядке убывания.

```
ТекстЗапрос = Новый Запрос;
ТекстЗапрос.Текст =
  "ВЫБРАТЬ РАЗЛИЧНЫЕ
  |Док.Номенклатура,
  |СУММА (Док.Сумма) КАК Сум
  |ИЗ Документ.РеализацияТоваровУслуг.Товары КАК Док
  |СГРУППИРОВАТЬ ПО Док.Номенклатура
  |ИМЕЮЩИЕ СУММА(Док.Сумма) > 200
  |УПОРЯДОЧИТЬ ПО Сум УБЫВ";
ТЗ = ТекстЗапрос.Выполнить().Выгрузить();
ВыбранныйЭлемент = ТЗ.ВыбратьСтроку();
```

Результат выполнения запроса показан на рис. 4.82.

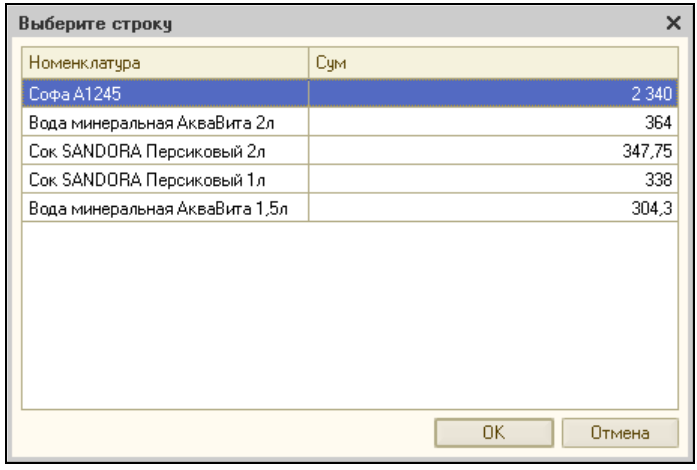



**Рис. 4.82.** Отбор товарных позиций с суммой более 200

## Ключевое слово *МЕЖДУ*

Используется для работы с диапазонами значений. Например, следующий запрос отбирает те же данные, что в предыдущем примере, но не по сумме, большей 200, а по сумме в диапазоне между 300 и 400.

```
ТекстЗапрос = Новый Запрос;
ТекстЗапрос.Текст =
  "ВЫБРАТЬ РАЗЛИЧНЫЕ
```

```
  |Док.Номенклатура,
  |СУММА (Док.Сумма) КАК Сум
  |ИЗ Документ.РеализацияТоваровУслуг.Товары КАК Док
  |СГРУППИРОВАТЬ ПО Док.Номенклатура
  |ИМЕЮЩИЕ СУММА(Док.Сумма) МЕЖДУ 300 и 400
  |УПОРЯДОЧИТЬ ПО Сум УБЫВ";
ТЗ = ТекстЗапрос.Выполнить().Выгрузить();
ВыбранныйЭлемент = ТЗ.ВыбратьСтроку();
```

Результат выполнения запроса показан на рис. 4.83. Как видим, в сравнении с предыдущим примером отгрузка на сумму 2340 в выборку не попала, поскольку удовлетворяет условию `>200`, но не удовлетворяет условию `МЕЖДУ 300 и 400`.

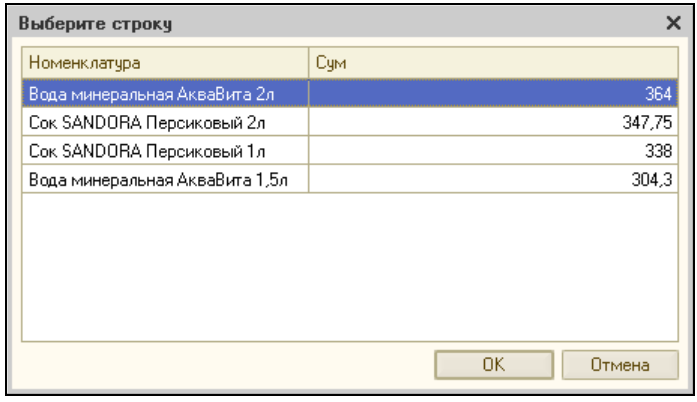



**Рис. 4.83.** Отбор товарных позиций с суммами из промежутка от 300 до 400

## Формирование итоговой строки. Операция *ИТОГИ*

Операция `ИТОГИ` позволяет рассчитать итоги по полям запроса.

Например, в следующем запросе из реализации производится выборка значений с расчетом итогов по сумме.

```
ТекстЗапрос = Новый Запрос;
ТекстЗапрос.Текст =
  "ВЫБРАТЬ
  |Док.Номенклатура КАК Товар,
  |Док.Количество КАК Колво,
  |Док.Цена,
  |Док.Сумма
  |ИЗ Документ.РеализацияТоваровУслуг.Товары КАК Док
  |ИТОГИ СУММА(Док.Сумма) ПО Док.Номенклатура";
ТЗ = ТекстЗапрос.Выполнить().Выгрузить();
ВыбранныйЭлемент = ТЗ.ВыбратьСтроку();
```

Результат выполнения запроса показан на рис. 4.84. Строка итогов по каждой позиции выведена отдельно.

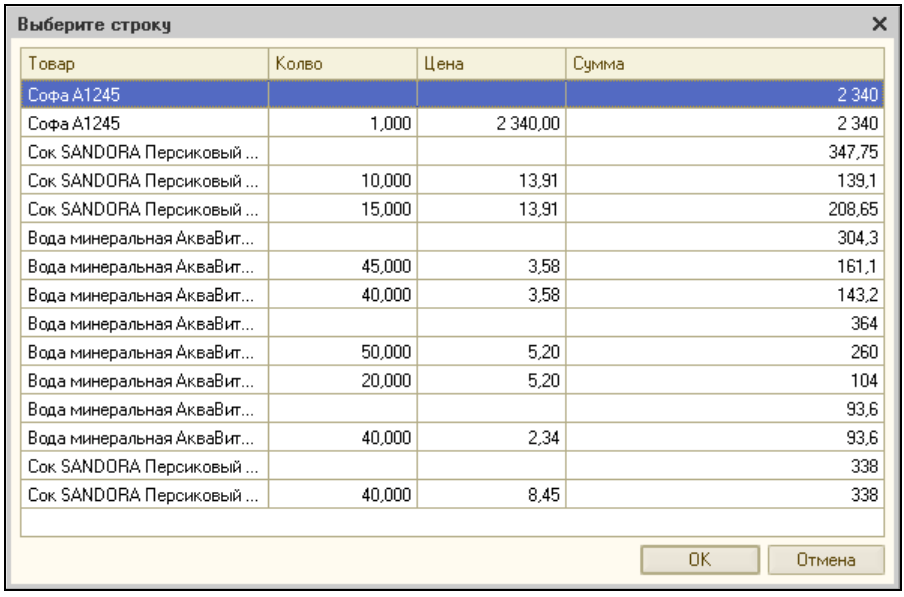



**Рис. 4.84.** Вывод данных по реализациям, отдельной строкой высчитывается общая сумма по позиции

## Объединение результатов нескольких запросов. Операция *ОБЪЕДИНИТЬ*

Запросы можно объединять и выводить на экран (печатную форму, использовать в расчетах) итог работы всех участвующих запросов. Количество полей в объединяемых запросах должно быть одинаковым.

```
ТекстЗапрос = Новый Запрос;
ТекстЗапрос.Текст =
  "ВЫБРАТЬ
  |Документ.РеализацияТоваровУслуг.Ссылка,
  |Документ.РеализацияТоваровУслуг.Контрагент КАК Покупатель,
  |Документ.РеализацияТоваровУслуг.ДоговорКонтрагента КАК Договор
  |ИЗ Документ.РеализацияТоваровУслуг
  |ОБЪЕДИНИТЬ
  |ВЫБРАТЬ
  |Документ.ВозвратТоваровОтПокупателя.Ссылка,
  |Документ.ВозвратТоваровОтПокупателя.Контрагент КАК Покупатель,
  |Документ.ВозвратТоваровОтПокупателя.ДоговорКонтрагента КАК Договор
  |ИЗ Документ.ВозвратТоваровОтПокупателя";
ТЗ = ТекстЗапрос.Выполнить().Выгрузить();
ВыбранныйЭлемент = ТЗ.ВыбратьСтроку();
```

В данном примере мы сформировали реестр работы с покупателями — отгрузок и возвратов товаров. Данные эти хранятся в разных документах: "РеализацияТоваровУслуг" и "ВозвратТоваровОтПокупателя", выборка производилась двумя

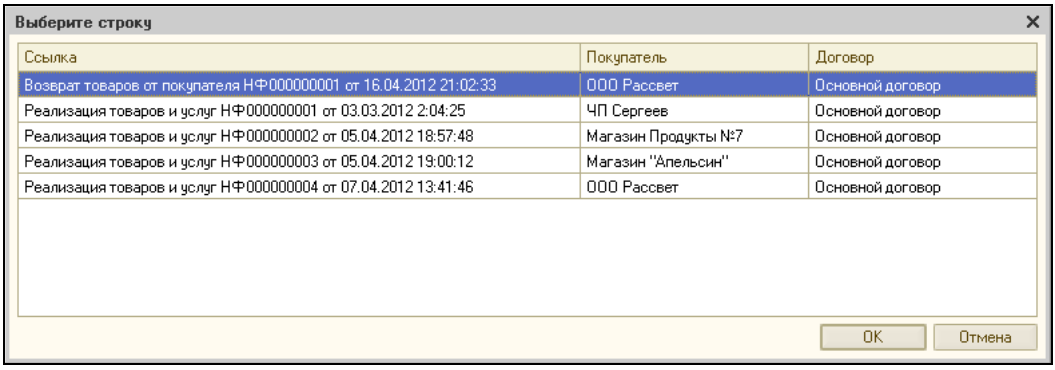



**Рис. 4.85.** Объединение двух запросов — выборка реализаций покупателям и возвратов от покупателей

запросами, которые были объединены. Результат выполнения запроса показан на рис. 4.85.

## Обработка результатов запроса. Выборки из результатов запроса

Разумеется, увидеть результаты запроса на экране — это важно. Но, кроме того, эти результаты надо обрабатывать, выводить на печать и т. д. Как обратиться к результатам запроса? Простейший пример был приведен в самом начале главы — результат запроса выводился в таблицу значений, мы выбирали нужную строку и получали ссылку на выбранную строку (а значит, точно так же могли написать любой обработчик полученного результата). Во всех разобранных нами примерах результат мы выгружаем в таблицу значений, из которой, в свою очередь, можно выбрать данные для обработки или просто поместить их в печатную форму (перебором таблицы значений и присваиванием переменным в печатной форме значений переменных в таблице значений).

Приведем пример обработки результатов запроса. Для этого выберем данные по реализациям вместе с соответствующими им документами отгрузок, а затем пометим на удаление отгрузки, относящиеся к контрагенту "ЧП Сергеев".

```
ТекстЗапрос = Новый Запрос;
ТекстЗапрос.Текст =
  "ВЫБРАТЬ
  |Реализация.Номер,
  |Реализация.Дата КАК Дата,
  |Реализация.Контрагент КАК Покупатель,
  |Реализация.ДоговорКонтрагента КАК Договор,
  |Развозка.Ссылка КАК ССылкаНаДокументРазвозки
  |ИЗ Документ.РеализацияТоваровУслуг КАК Реализация,
                                        Документ.Развозка КАК Развозка
  |ГДЕ Реализация.Номер = Развозка.Документ.Номер";
```

```
РезультатыЗапроса = ТекстЗапрос.Выполнить().Выбрать();

Спр = Справочники.Контрагенты.НайтиПоНаименованию(СокрЛП("ЧП Сергеев"));
Пока РезультатыЗапроса.Следующий() Цикл
  Если РезультатыЗапроса.Покупатель = Спр Тогда
    Сообщить("Имеется документ по этому контрагенту");
    Сообщить(РезультатыЗапроса.Номер);
    Сообщить(РезультатыЗапроса.Дата);
    ДокументРазвозки =
РезультатыЗапроса.ССылкаНаДокументРазвозки.ПолучитьОбъект();
    ДокументРазвозки.УстановитьПометкуУдаления(Истина);
  КонецЕсли;
КонецЦикла;
```

Что мы сделали? Сначала сделали выборку, результат которой представлен на рис. 4.86.

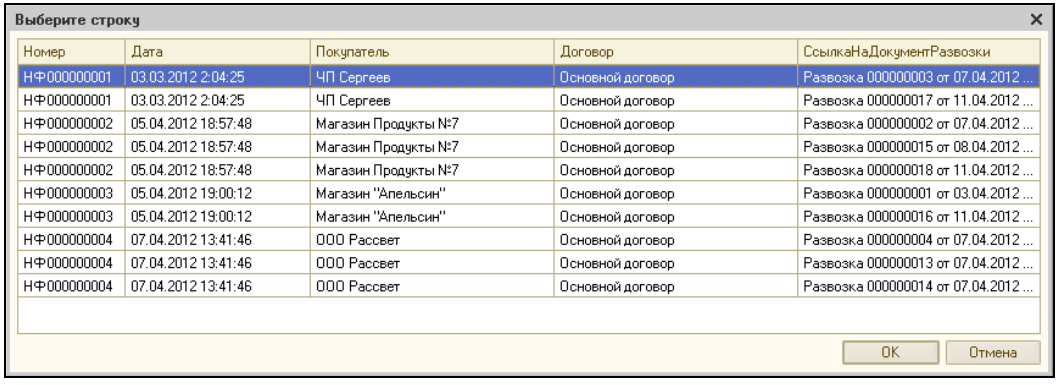



**Рис. 4.86.** Результат запроса для дальнейшей обработки

В отличие от приведенных ранее примеров, результаты этого запроса мы на экран не выводим, иллюстрация просто должна продемонстрировать читателю, какие данные мы отобрали запросом. Результаты запроса мы никуда не выгружаем, поэтому не используем метод `Выгрузить()`.

Результаты запроса будут обработаны командой `Выбрать()`. Если только поле `Покупатель` из запроса совпадет со значением `"ЧП Сергеев"`, спозиционированным нами в переменную `Спр`, выполнится следующий код:

```
Сообщить("Имеется документ по этому контрагенту");
Сообщить(РезультатыЗапроса.Номер);
Сообщить(РезультатыЗапроса.Дата);
ДокументРазвозки = РезультатыЗапроса.ССылкаНаДокументРазвозки.ПолучитьОбъект();
ДокументРазвозки.УстановитьПометкуУдаления(Истина);
```

Мы выводим сообщение пользователю о том, что документ найден, демонстрируем его номер и дату, затем получаем объект для последующих с ним действий и помечаем на удаление.

На этом мы завершим главу, посвященную запросам и отчетам. Надеюсь, вы почерпнули в ней немало полезного. В этой главе не ставилась цель показать всю мощь запросов и отчетов, чтобы не нагружать читателя обилием новой информации, поэтому они не были очень сложными (в реальной работе иногда приходится писать очень громоздкие и сложные запросы). Однако это был базис, который читатель сможет использовать в работе. А дальше придет время и для более сложных отчетов.

ГЛАВА 5

# Пользователи, интерфейсы, права

Сейчас мы поговорим об очень важном аспекте работы в системе "1С:Предприятие", а именно о пользователях, их правах и возможностях. Каждый пользователь в системе работает под своим именем, что-то добавляет в базу данных, что-то изменяет. Но возможности у всех разные. У системного администратора они максимальные, у менеджера — меньше, у кладовщика или продавца — еще меньше. И это правильно, каждый должен заниматься своим делом. Если взять продавца (менеджера, кладовщика, да даже руководителя предприятия) и дать ему доступ ко всем возможностям системы, это будет хорошо? Нет. Может навредить, пусть не специально, по незнанию, но результат один. С другой стороны, продавцу не нужно знать многие аспекты работы предприятия, которые положено знать директору или главному бухгалтеру: сколько осталось товара на всех складах, от кого он пришел, по какой цене, какова выручка с продаж. Не нужны эти данные продавцу. Опять же, коммерческая тайна...

Вот для подобного разграничения доступа и придумана такая замечательная штука, как права пользователей. Итак, по порядку.

Под *пользователем* в системе "1С:Предприятие" понимается учетная запись, обладающая собственным паролем, наборами прав доступа к объектам метаданных и пользовательским интерфейсом.

*Интерфейс пользователя* — это набор визуальных компонентов (панели, кнопки, меню), доступных пользователю при работе с программой. Для разных пользователей он может значительно отличаться.

*Права* — набор разрешений, ограничений и запретов на использование того или иного объекта метаданных.

Для того чтобы увидеть список пользователей (рис. 5.1), зарегистрированных в системе, нужно войти в режиме Конфигуратора и нажать пункт меню **Администрирование | Пользователи** (с изображением человеческой головы) на панели инструментов.

Список пользователей по умолчанию показывает нам следующее:

- **Имя** — некоторый псевдоним, под которым пользователь входит в систему. При запуске системы может быть выбран в выпадающем списке **Пользователь;**

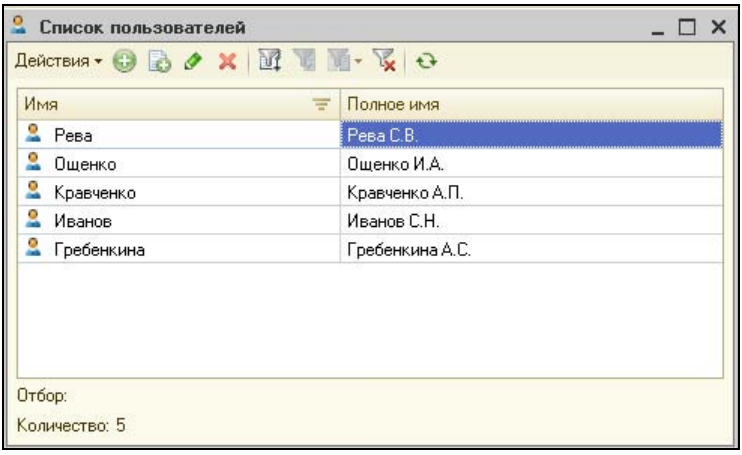



**Рис. 5.1.** Так может выглядеть список пользователей системы

- Полное имя — реальное имя пользователя, которое должно отображаться в документах.

При необходимости мы можем настроить список пользователей, выведя дополнительную информацию. Для этого щелкните правой кнопкой мыши по окну со списком пользователей и в контекстном меню выберите пункт **Настройка списка**. Откроется окно, как на рис. 5.2.

Как видите, мы можем настроить состав, последовательность и параметры колонок. Изначально у нас есть только **Имя** и **Полное имя**, добавим также **Роли** и **Основ-**

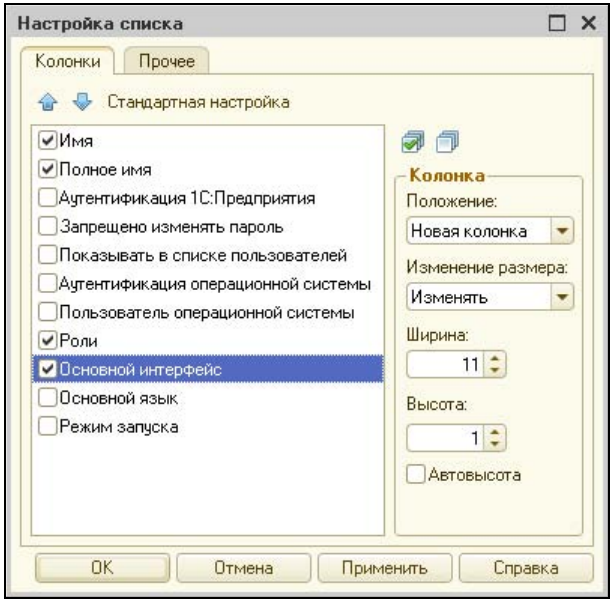



**Рис. 5.2.** Модификация списка пользователей

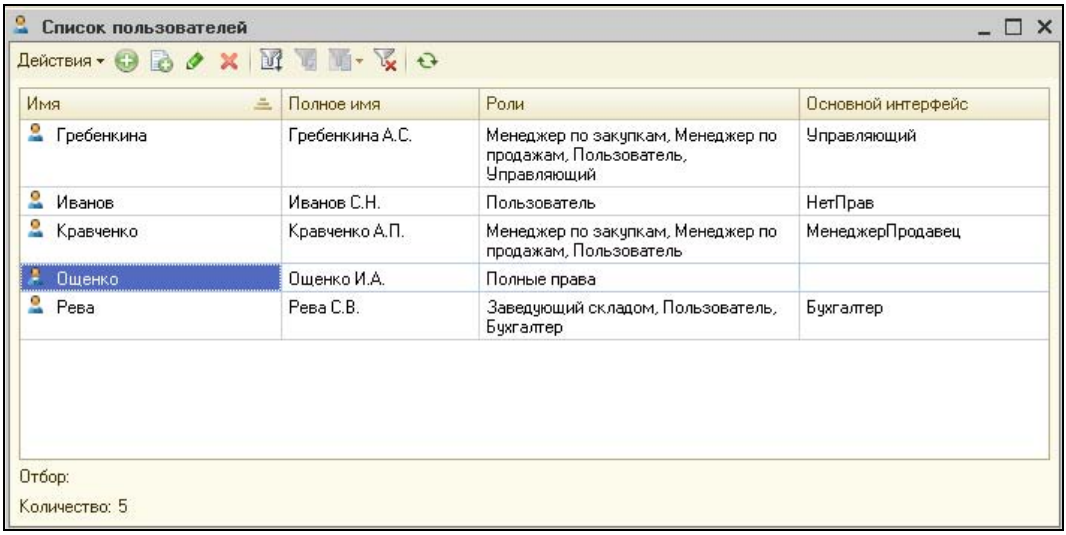



**Рис. 5.3.** Модифицированный список пользователей системы

**ной интерфейс** (см. на рис. 5.2), а затем нажмем кнопку **OK**. В результате окно списка пользователей уже будет выглядеть так, как показано на рис. 5.3.

У нас добавились еще две колонки.

- **Роли.** Представляют собой сформированные наборы пользовательских прав на использование тех или иных объектов метаданных, поименованные некоторым общим названием. Одному пользователю может принадлежать сразу несколько ролей. Например, часть сотрудников может выполнять обязанности "Менеджер по закупкам", часть сотрудников — обязанности "Менеджер по продажам", а один из сотрудников, начальник отдела, может иметь сразу обе данных роли и вдобавок специальную роль "Управляющий", дающую доступ к дополнительным отчетам.
- **Основной интерфейс.** Представляет собой совокупности пользовательских компонентов, таких как меню, панели инструментов и пр., поименованные некоторым общим названием. Конечно, можно для каждого из пользователей вывести полный интерфейс программы, но зачем перегружать рабочее окно уймой неиспользуемых кнопок и меню, к части из которых, к тому же, пользователь не имеет прав доступа? Поэтому интерфейсы формируются из тех объектов, с которыми пользователь, выполняющий соответствующую роль, обычно работает.

Список ролей работающей на предприятии конфигурации может выглядеть, например, так, как показано на рис. 5.4.

За основу взяты стандартные роли конфигурации "Управление торговлей", но с учетом специфики конкретного предприятия немного переделаны, а также добавлено несколько новых ролей. Для того чтобы изменить какую-либо из ролей, нужно дважды щелкнуть по ней мышью. Например, откроем таким образом роль "МенеджерПоПродажам" (рис. 5.5).

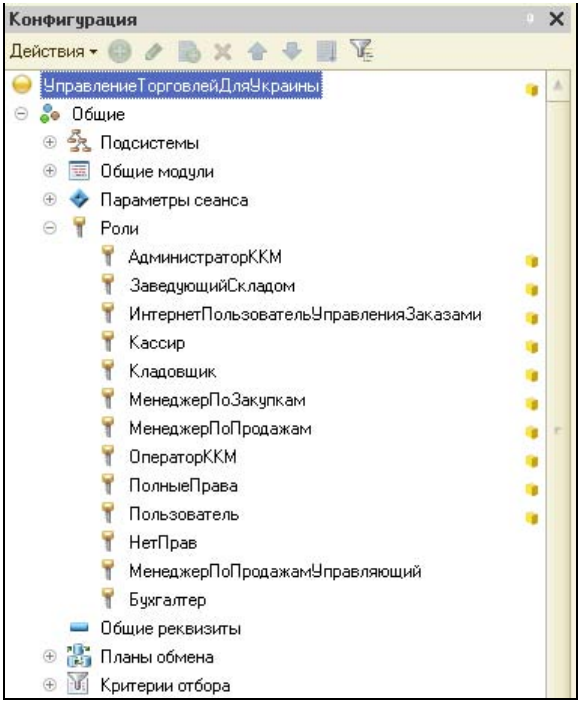



**Рис. 5.4.** Список ролей в дереве метаданных

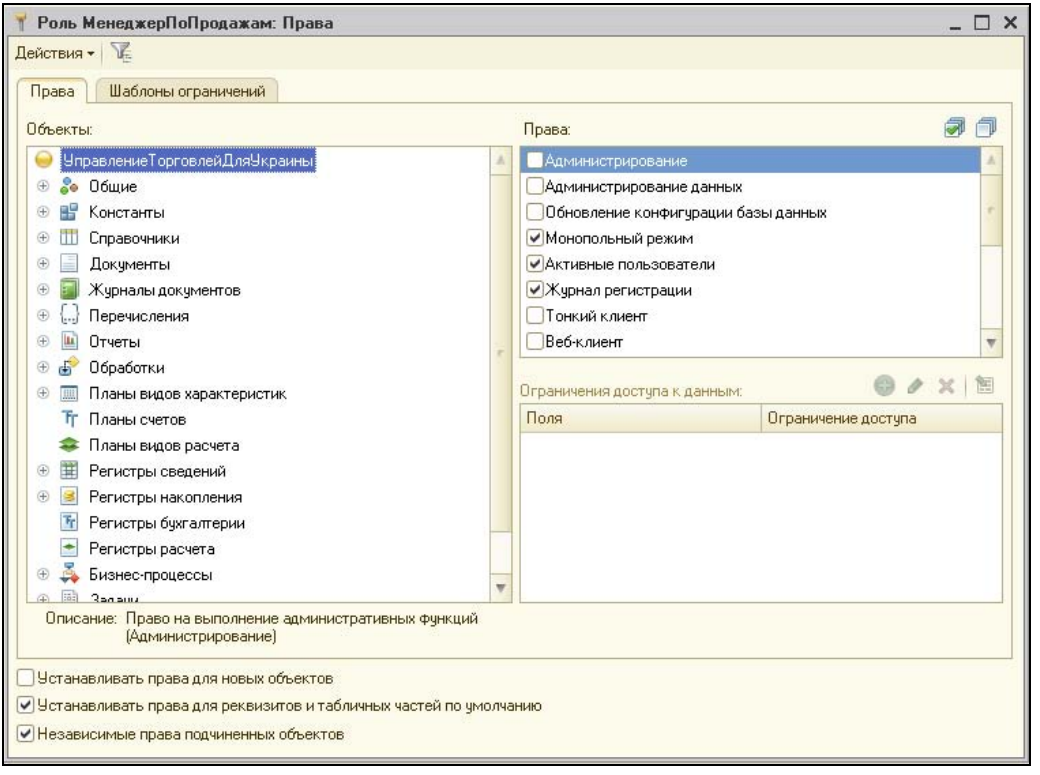



**Рис. 5.5.** Назначение прав для выбранной роли

Окно редактирования роли разделено на три части. Слева показано дерево конфигурации. Справа, в верхней части — права, которые можно назначить для элемента метаданных, выбранного в левой части окна. На данный момент выбран заголовок конфигурации, т. е. по сути сама конфигурация, в которую пользователь роли "МенеджерПоПродажам" может заходить монопольно, просматривать список активных пользователей (монитор пользователей) и журнал регистрации. А вот администрировать конфигурацию менеджер по продажам не может. И обновлять конфигурацию тоже ему запрещено.

Если бы мы выбрали в левой части окна какой-нибудь другой элемент метаданных, то и права были бы другие. Например, для документа "ПриходныйКассовыйОрдер" это было бы **Чтение**, **Добавление**, **Изменение**, **Удаление**, **Проведение**, **Отмена проведения** и т. д. Нужно учесть, что некоторые функции взаимосвязаны между собой. Если, например, снять флажок **Чтение**, доступ к этому документу вообще будет закрыт для этой группы пользователей. В самом деле, ведь нельзя корректировать или проводить накладную, если даже нет прав на ее чтение.

Для справочника контрагентов права доступа роли "МенеджерПоПродажам" могут выглядеть так, как показано на рис. 5.6.

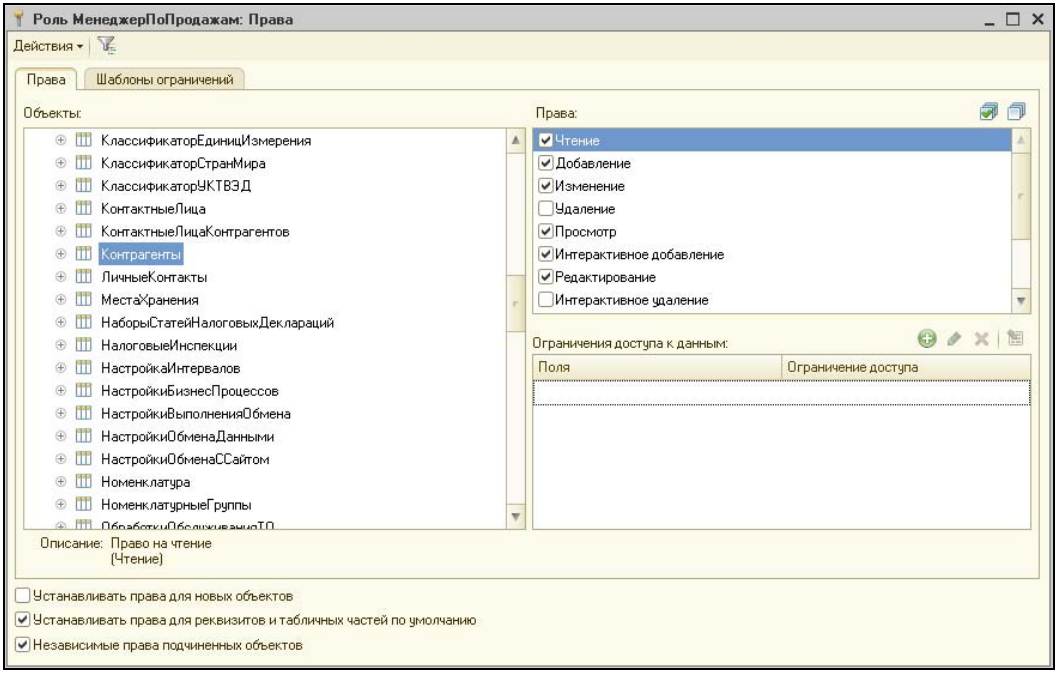



**Рис. 5.6.** Права на справочник "Контрагенты" для выбранной роли

Кроме того, можно добавлять различные ограничения доступа к данным. Это делается в правой нижней части окна редактирования роли (см. рис. 5.6). Условия ограничения добавляются зеленой кнопкой с плюсом, после чего задается, по каким полям будет производиться ограничение и в чем оно состоит. Причем ограничение

будет относиться именно к тому действию, которое выбрано в правой верхней части окна, в нашем случае это **Чтение**. Выбор списка полей выглядит так, как показано на рис. 5.7. Там выбраны все поля.

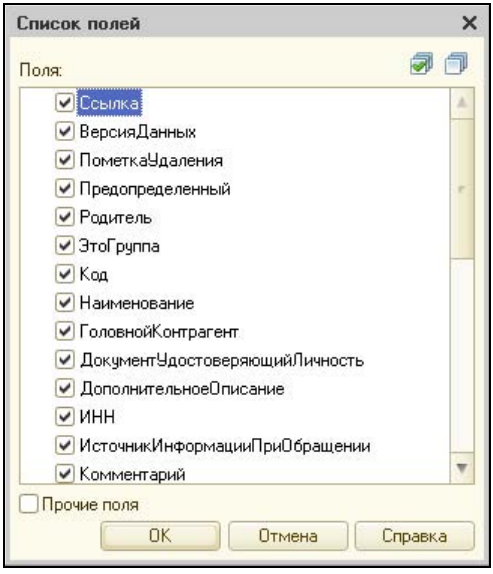



**Рис. 5.7.** Выбираем поля для ограничения доступа

А условие пишется в отдельном поле ввода, которое открывается по щелчку мышью в строке условия. Используется внутренний язык запросов.

Например, так нужно написать, если мы хотим добавить ограничение для контрагента с названием "Мебельный комбинат №7" (рис. 5.8).

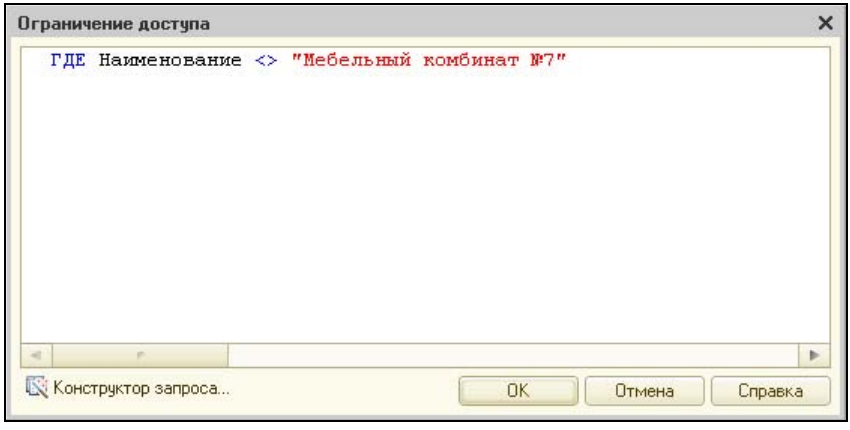



**Рис. 5.8.** Условие ограничения доступа для определенной карточки контрагента

В совокупности с тем, что в окне, изображенном на рис. 5.7, мы выбрали все поля, а на рис. 5.6 выбрали параметр **Чтение**, то получается, что мы полностью закрыли для роли "МенеджерПоПродажам" любые действия с карточкой контрагента

"Мебельный комбинат №7". Ведь если нет доступа на чтение, то соответственно ни редактировать, ни удалять эту карточку тем более нельзя.

Нижняя правая часть окна с ограничением данных будет выглядеть так, как показано на рис. 5.9.


**Рис. 5.9.** Список ограничений доступа к данным

Теперь, если мы зайдем в режим Предприятия как пользователь с ролью "МенеджерПоПродажам" и откроем справочник контрагентов, то записи "Мебельный комбинат №7" мы там не увидим, хотя она и была заведена. Если же зайдем, предположим, как пользователь с ролью "ПолныеПрава", то запись увидим и сможем с ней работать.

При настройке ролей также удобно использовать окно **Все роли** (рис. 5.10), позволяющее видеть права на объекты сразу у всех ролей системы. Открывается оно

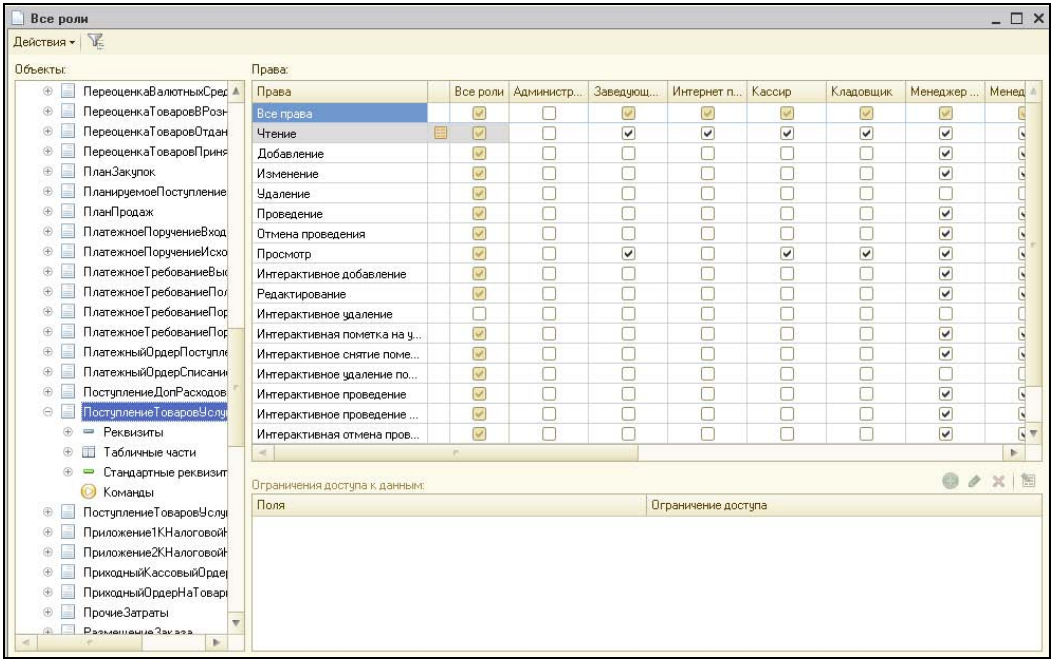



**Рис. 5.10.** Права по всем ролям

щелчком правой кнопки мыши на заголовке **Роли** в дереве метаданных и выбором пункта **Все роли**.

Отмеченный флажок означает разрешение, сброшенный — запрет, затененный — для строк и столбцов и означает, что для данной роли возможны не все действия или данное действие могут выполнять не все роли.

Также полезно использовать окно **Все ограничения доступа**, оно открывается там же, где и окно **Все роли** и показывает все прописанные нами ограничения. Иначе можно забыть, что и для кого мы прописывали, особенно если база большая и делали мы это не сегодня (рис. 5.11).

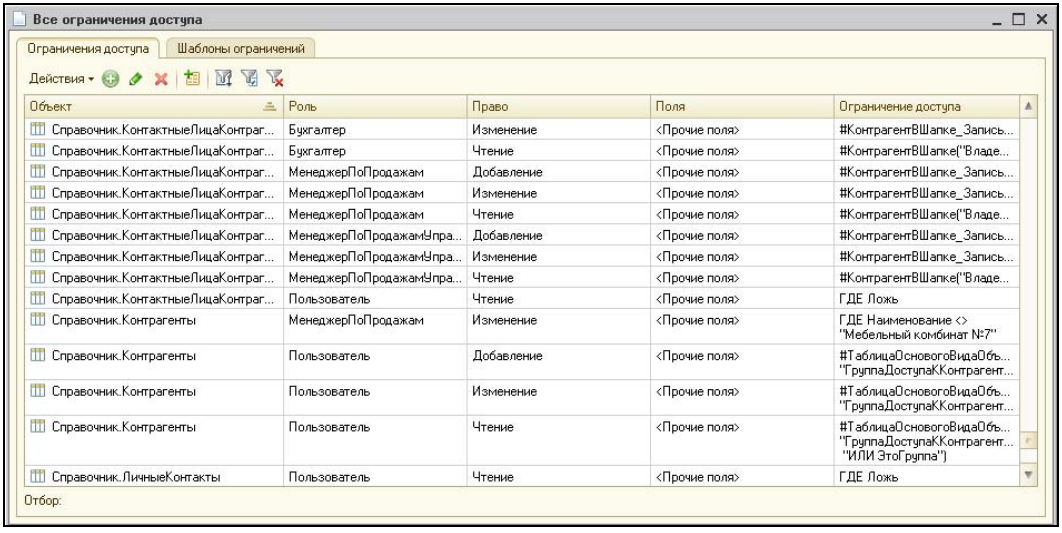



**Рис. 5.11.** Все ограничения доступа

Теперь, когда о ролях и их правах мы поговорили, обсудим немного интерфейсы. Список интерфейсов, так же как и список ролей, находится в дереве конфигурации (рис. 5.12).

Некоторые интерфейсы стандартные, а другие сделаны самостоятельно под специфику предприятия (стандартные отмечены желтыми значками). Для того чтобы изменить один из интерфейсов, необходимо сделать по нему двойной щелчок мышью. Вот так может выглядеть интерфейс "Управляющий" (рис. 5.13).

В верхней части окна добавляем панели интерфейса, ниже правим меню. В меню имеются как стандартные пункты, так и пункт **Новая** для создания новых пунктов меню. Имеющиеся пункты меню можно менять местами, перетаскивая мышью, либо удалять вовсе. Также их состав может быть модифицирован. Каждый из пунктов меню может быть открыт (см. рис. 5.13), а затем с помощью пункта **Новая** туда добавляются нужные нам команды. При создании нового пункта меню нам надо задать его имя, вариант отображения пункта (текст, картинка, текст с картинкой, выводить ли подсказку), а также назначить действие, которое будет связано с этим пунктом меню. На рис. 5.14 мы привязываем к новому пункту отчет "АнализЗаказовПокупателей".

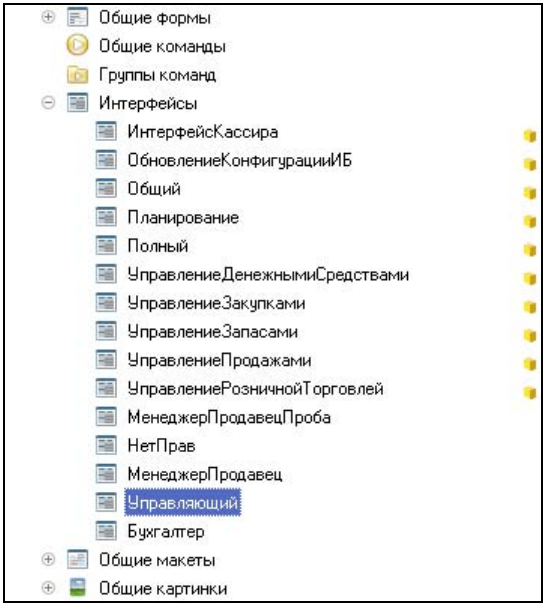



**Рис. 5.12.** Список интерфейсов в дереве метаданных

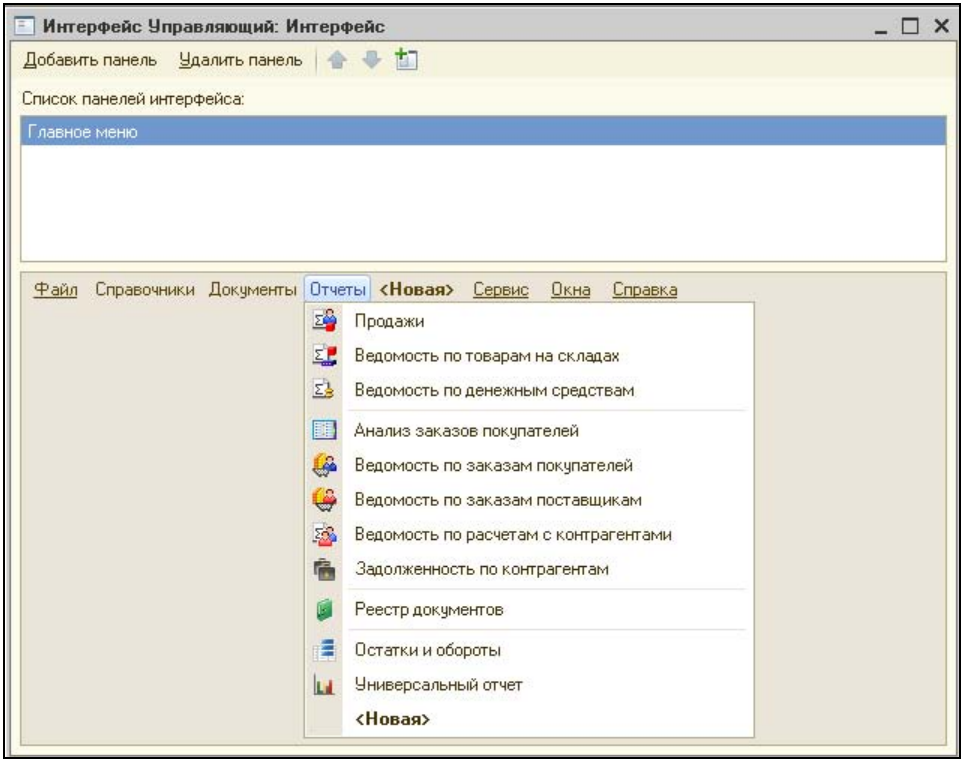



**Рис. 5.13.** Настраиваем меню для выбранного интерфейса

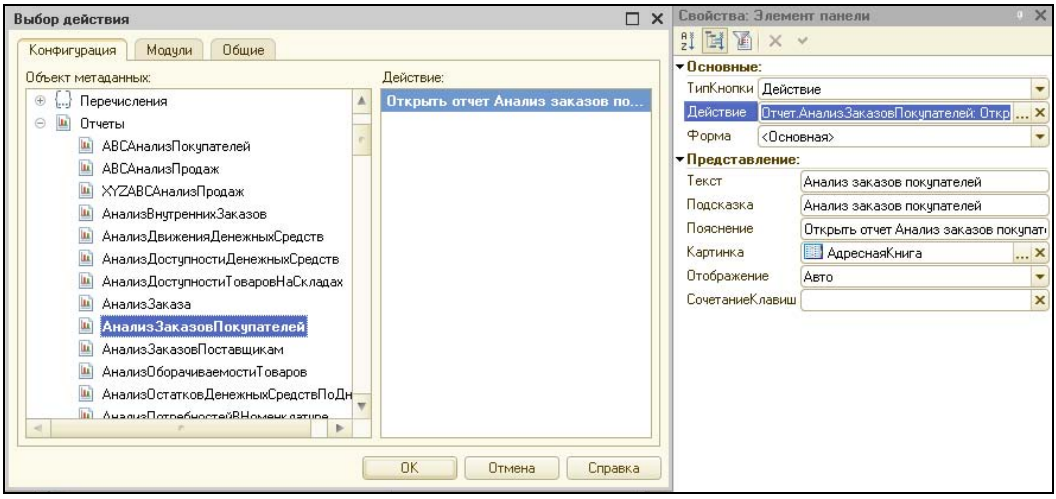



**Рис. 5.14.** Так настраивается новый пункт меню

Теперь, вооружившись полученными знаниями, попробуем создать нового пользователя системы. В режиме Конфигуратора зайдем в пункт меню **Администрирование | Пользователи** и нажмем зеленую кнопку с плюсом — **Добавить**.

В окне добавления нового пользователя для него нужно задать имя и полное имя, указать пароль (при необходимости) и повторить пароль еще раз, чтобы исключить ошибку ввода (рис. 5.15).

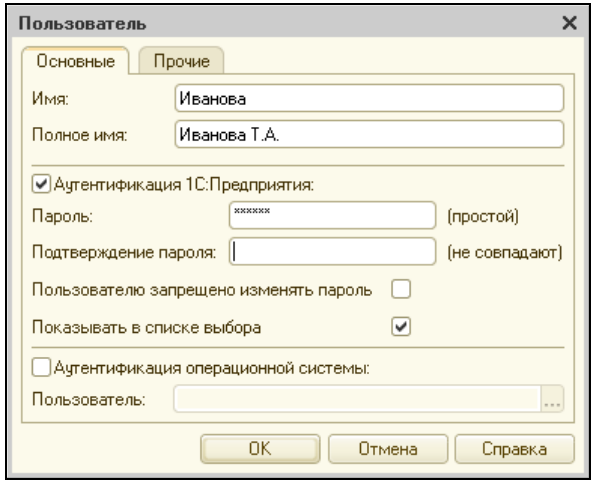



**Рис. 5.15.** Добавляем нового пользователя

На рис. 5.15 подтверждение пароля еще не введено, поэтому напротив поля ввода подтверждения указано, что пароли не совпадают. А для поля **Пароль** указано, что пароль слишком простой и желательно бы его сделать посложнее (например, буквенно-цифровой).

В том же окне на вкладке **Прочие** нам нужно назначить пользователю его роль (или роли, если предполагается несколько разных ролей). Предположим, это будет "Заведующий складом". Интерфейс назначаем "УправлениеЗапасами", язык — русский (рис. 5.16).

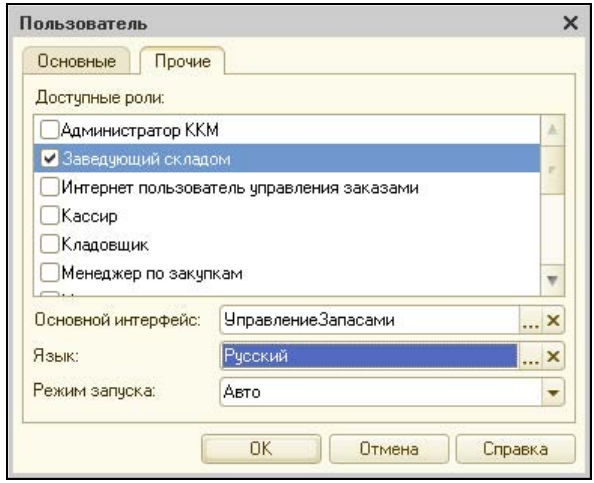



**Рис. 5.16.** Задаем роли нового пользователя

Нажмем кнопку **OK**, сохраним конфигурацию и пробуем зайти как новый пользователь. Не факт, что получится с первого раза. Зависит от того, какая конфигурация, стандартная она или уже изменялась, и как изменялась. В общем, не исключены ошибки с правами доступа, а наша задача — настроить права доступа, чтобы ошибки не возникали. Может произойти такая ошибка (рис. 5.17).

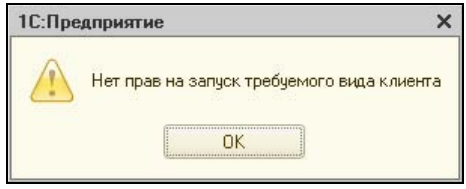



**Рис. 5.17.** Недостаточно прав!

Под видом клиента понимается "толстый" клиент, "тонкий" клиент или веб-клиент. В данном случае мы работаем под "толстым" клиентом, если на него нет прав, то добавляем (рис. 5.18).

Довольно распространена следующая ошибка (рис. 5.19).

Для запуска информационной базы также нужна роль, а роль нашего пользователя ею, похоже, не обладает. Программный код, ответственный за вышеуказанное сообщение, находится в общем модуле "УправлениеПользователями" и выглядит примерно так, как показано на рис. 5.20 (напоминаю, примеры мы разбираем на конфигурации "Управление торговлей").

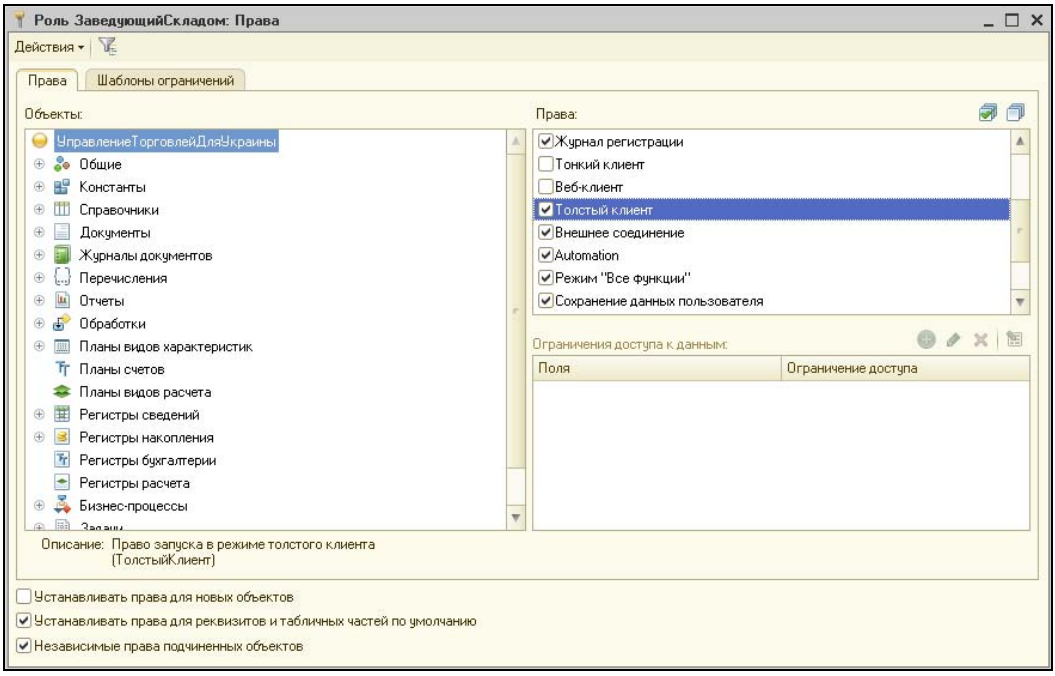



**Рис. 5.18.** Даем доступ на работу с "толстым" клиентом

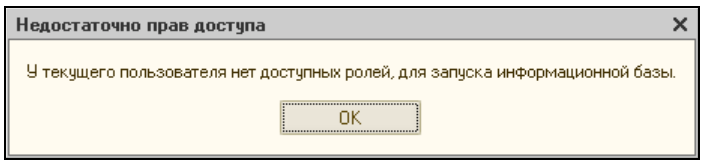



**Рис. 5.19.** Известная ошибка, нет доступных ролей для запуска информационной базы

Без полных прав нас не пускают, что нехорошо. Конечно, можно просто закомментировать содержание данной процедуры, но это неправильно, и мы так делать не будем. Просто немножко подправим ее (рис. 5.21).

Добавили роль "ЗаведующийСкладом", которую имеет наш новый пользователь.

Тем не менее, не исключено, что при входе в режим Предприятия опять будут появляться ошибки. Чаще всего это связано с перенастройкой стандартных ролей и прав доступа. Итак, нарушение прав доступа (рис. 5.22).

Для такой ошибки мы уже вполне можем увидеть, а к чему, собственно, у нас нет права доступа? Нажмем кнопку **Подробно** и увидим подробности о причине ошибки (рис. 5.23).

В данном случае у нашей роли нет прав на использование константы `ЗаголовокСистемы`. Зайдем в настройки роли и дадим права на чтение этой константы (рис. 5.24).

В случае базы данных, на основе которой я приводил примеры, было еще две ошибки: ошибка доступа к обработке "ТОСервер" и к справочнику "Регламентиро-

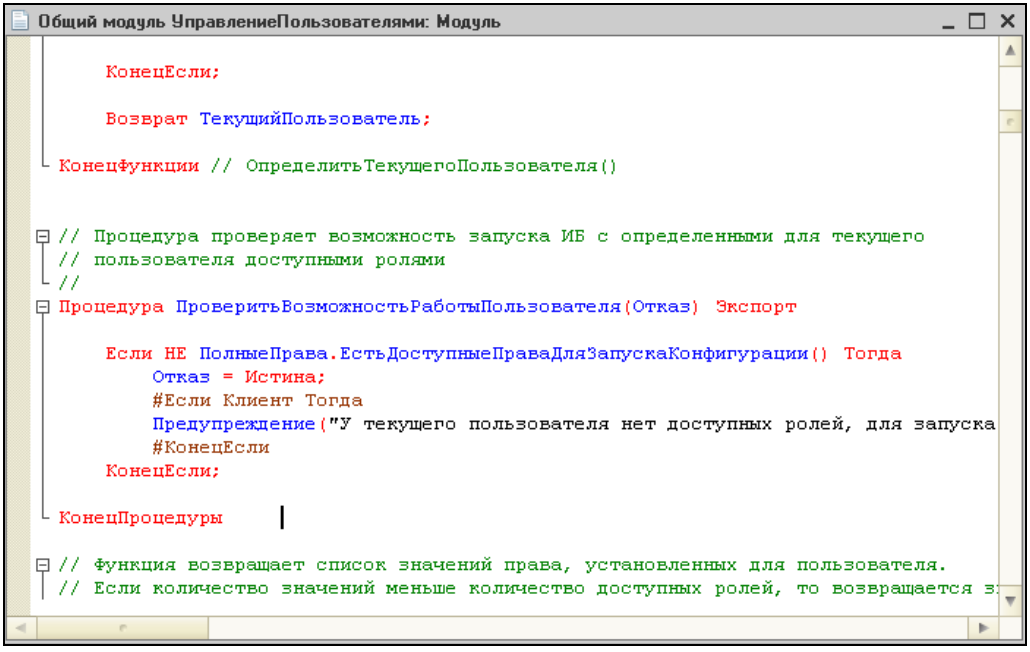



**Рис. 5.20.** Проверка на наличие ролей производится здесь

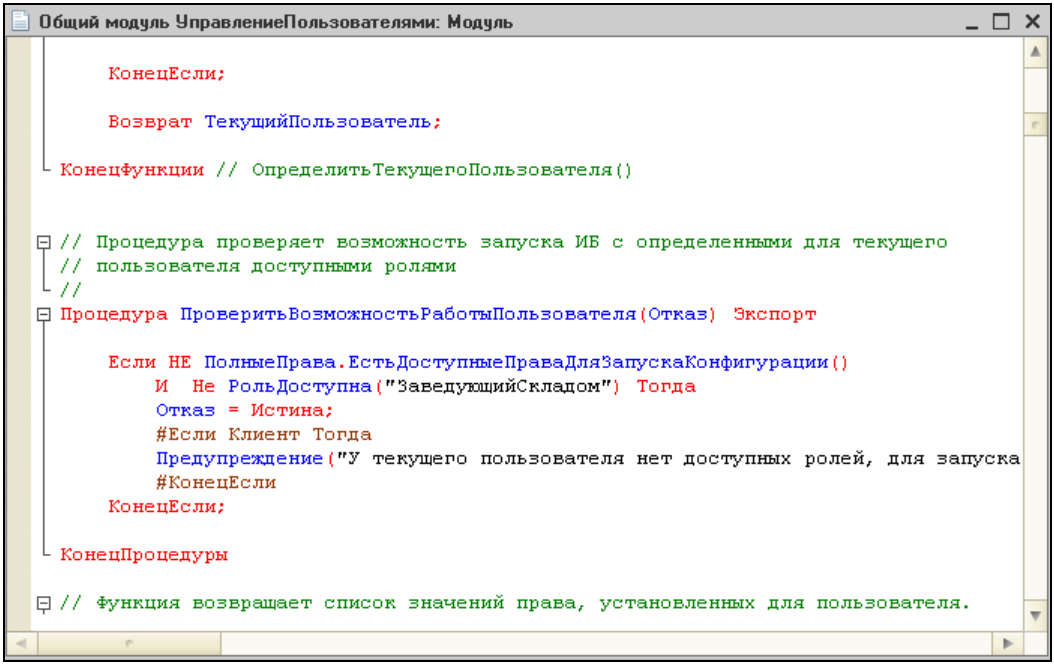



**Рис. 5.21.** Правим модуль для роли нового пользователя

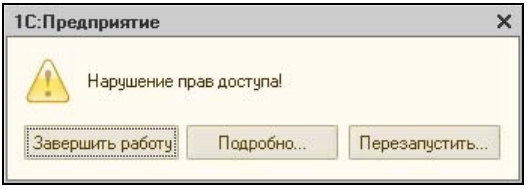



**Рис. 5.22.** И все равно ошибка прав доступа

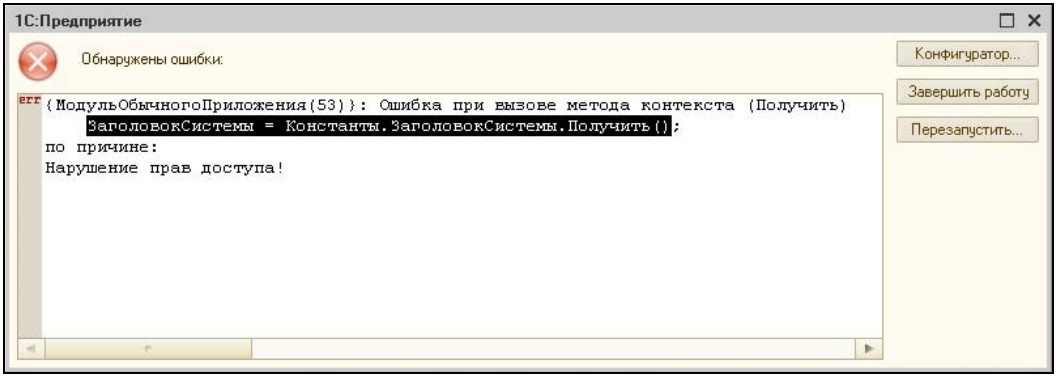



**Рис. 5.23.** С помощью кнопки **Подробно** смотрим причину ошибки

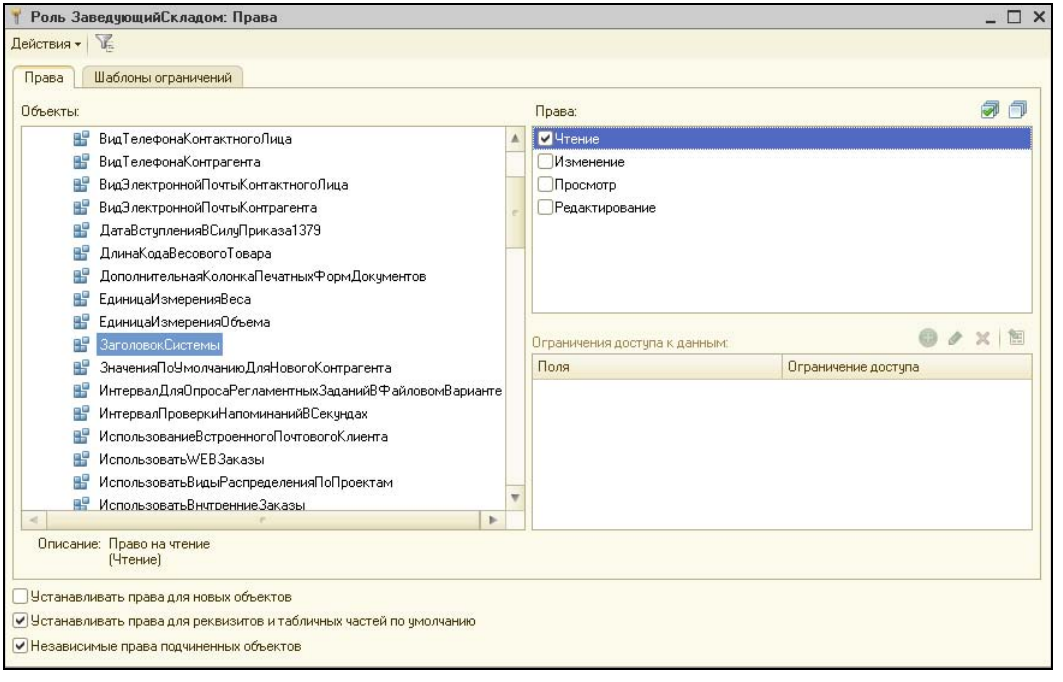



**Рис. 5.24.** Доступ на использование нужной константы

ванныеОтчеты". У читателя, возможно, будет по-другому, а может, и никаких ошибок не будет. Главное, чтобы он сумел сориентироваться, если ошибки-таки проявятся. Посмотрим описание ошибки и дадим соответствующий доступ к нужному объекту. Или не дадим, а подумаем, что мы настроили не так, особенно, если до того мы экспериментировали с правами и ролями. Еще может помочь назначение второй роли, в которой на объект есть права, отсутствующие у первой роли пользователя (например, для пользователя ЗаведующийСкладом вполне подошла бы вторая роль — Пользователь). Но тут главное — не дать лишнего, поэтому надо четко представлять, на что еще пользователь получит доступ, если дать ему дополнительную роль, можно ли это делать. Проще ведь вообще всем назначить администраторские права, только это будет неправильно.

ГЛАВА 6

# Работа с отладчиком

При написании программных модулей бывает так, что вроде бы модуль написан правильно... но вот работает неверно. Речь не об ошибках в синтаксисе, которые легко обнаруживаются еще при сохранении модуля или нажатии кнопки **Синтаксический контроль**, когда программа сигнализирует нам об ошибке, да еще и показывает ее местонахождение и в чем она заключается. Речь о неявных ошибках в логике работы модуля, которые в больших и сложных модулях отловить не так просто. Можно перечитывать текст модуля (или модулей, ведь в серьезных рабочих конфигурациях модули обычно взаимосвязаны друг с другом), следовать за логикой выполнения, пытаясь найти неверный алгоритм. Можно выводить системные сообщения со значениями ключевых переменных модуля на разные моменты выполнения программы. А можно воспользоваться отладчиком.

## Назначение отладчика, отладка программного кода

*Отладчик* — это встроенный в конфигуратор инструмент, предназначенный для проверки и анализа выполнения программных алгоритмов. С его помощью мы можем проанализировать работу программного алгоритма, определить значения переменных в указанный момент выполнения программы и обнаружить логические ошибки. В этой главе в качестве примера работы отладчика будут взяты программные модули, уже рассмотренные в данной книге, мы обсудим их пошаговое выполнение до указанных точек останова и вычисление значений переменных в указанной точке.

Что такое *точка останова*? Это своего рода маркер, который мы устанавливаем на полях программного модуля и по достижении которого выполнение программного модуля останавливается. Именно благодаря точкам останова мы можем выполнять модуль по частям, замеряя интересующие нас значения переменных и реквизитов объектов.

Напишем простенькую обработку. Затем зайдем в ее модуль и сделаем двойной щелчок на поле слева от программного кода. На поле появится маркер (рис. 6.1).

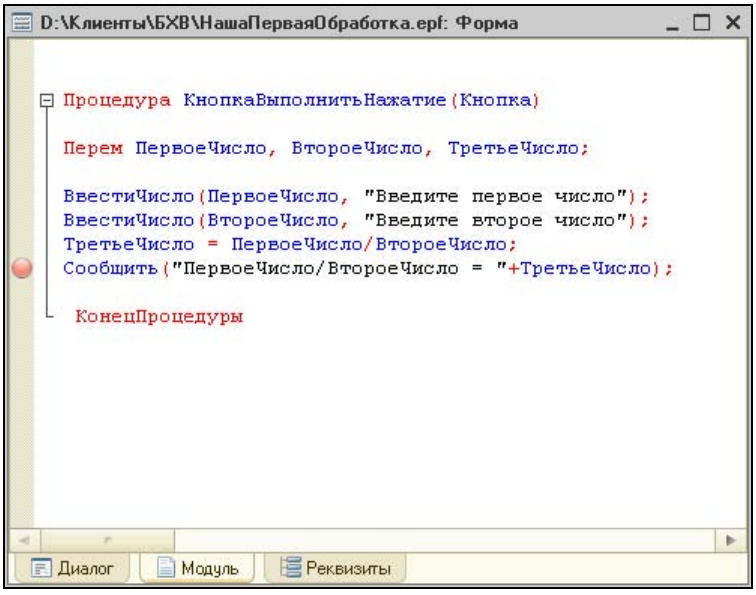



**Рис. 6.1.** Так выглядит точка останова

Теперь зайдем в меню **Отладка** и выберем пункт **Начать отладку** или же нажмем кнопку **Начать отладку** на панели инструментов. Запустится режим Предприятия, в котором нужно открыть обработку и запустить ее на выполнение. Обработка выполнится до точки останова, после чего ее выполнение будет прекращено. На изображении точки останова появится стрелка. Здесь произошла остановка выполнения модуля (рис. 6.2).

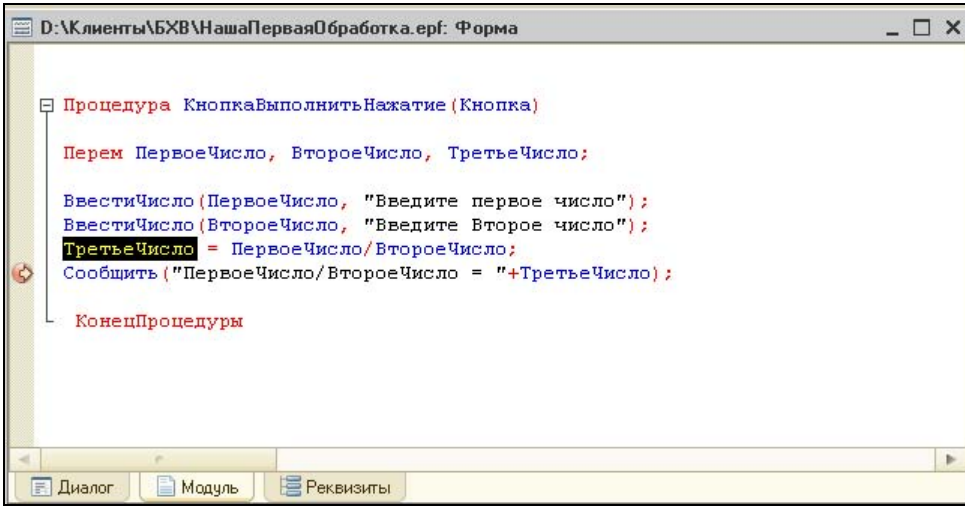



**Рис. 6.2.** Точка останова сработала

Выделим мышью переменную `ТретьеЧисло` в модуле и нажмем кнопку **Вычислить выражение** на панели инструментов. При выполнении программы первое число я задал равным 5, второе — равным 4. В итоге `ТретьеЧисло` равно 1,25, что мы и видим в окне расчета выражения (рис. 6.3).

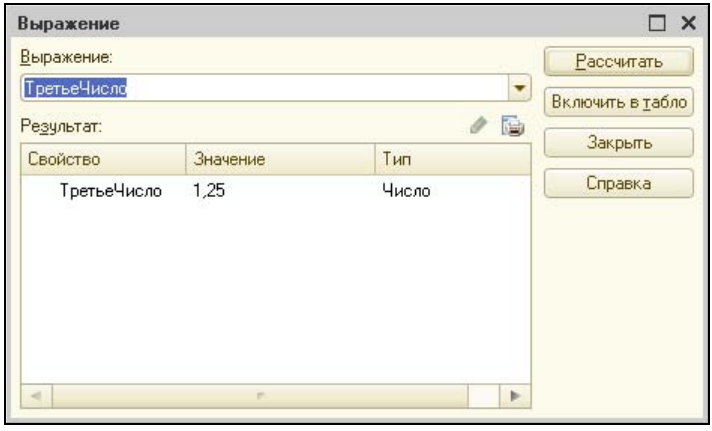



**Рис. 6.3.** Так выглядит вычисление значения в отладчике

Таким образом, при отладке программного кода мы можем смотреть, какое значение принимают те или иные переменные или реквизиты метаданных в процессе работы модуля. Большое количество таких значений удобно просматривать в табло, которое располагается в нижней части экрана. Чтобы перенести данные из окна вычисления выражения, достаточно нажать кнопку **Включить в табло**. Результат мы можем видеть на рис. 6.4.

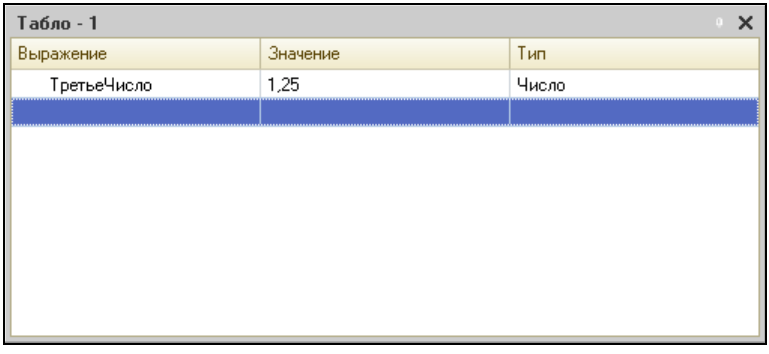



**Рис. 6.4.** Та же информация, но перенесенная в табло

Кроме значений переменных при отладке также можно видеть возвращаемые значения функций. Возьмем простой пример — процедуру, в которой нам будет предложено ввести код товара, а затем передающую этот код функции, в которой по переданному коду производится поиск. Если товар найден, тогда возвращается ссылка на него, если нет, то функция вернет строку "Нет такого товара". Точку останова установите на следующей строке после обращения к функции (рис. 6.5).

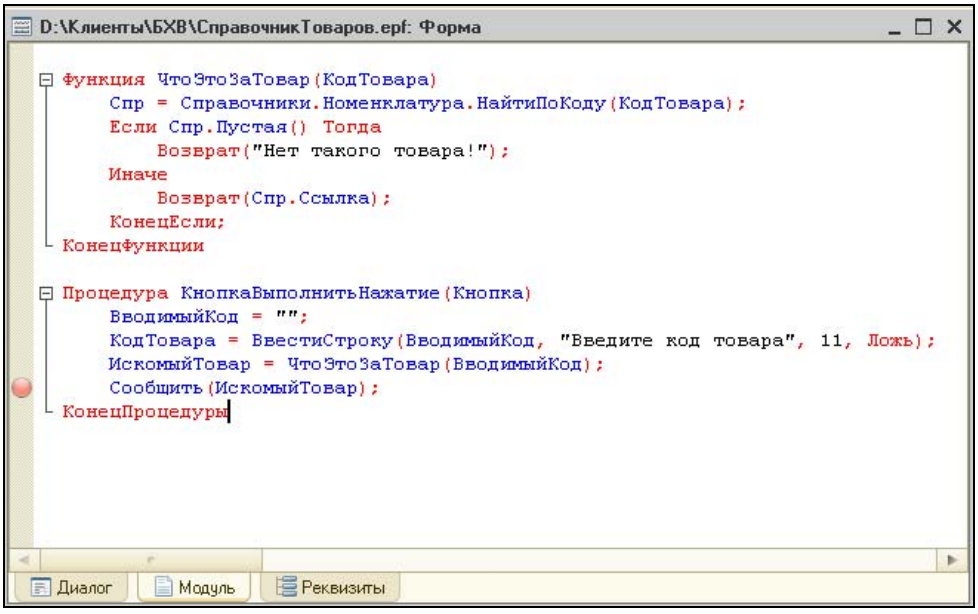



**Рис. 6.5.** Точка останова после отработки функции

Запустим отладку, точка останова срабатывает. Выделим переменную `ИскомыйТовар` и нажмем кнопку **ВычислитьВыражение**  . Код товара я ввел произвольный, так что неудивительно, что функция вернула значение "Нет такого товара". Вычисленное значение мы видим на рис. 6.6.

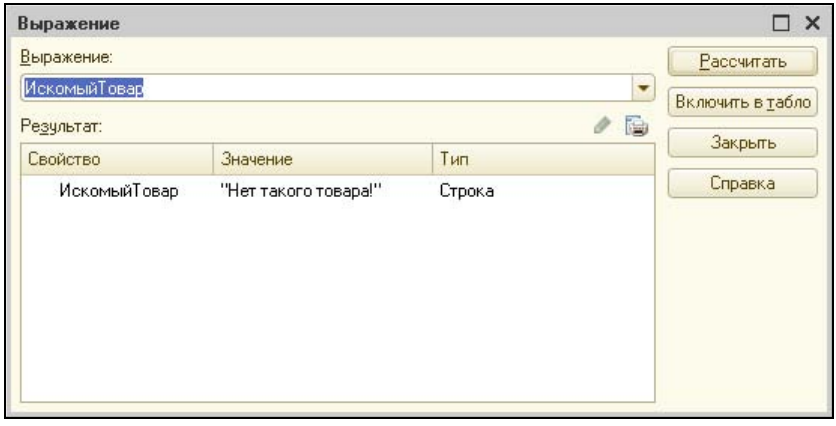



**Рис. 6.6.** Результат, который вернула функция. Товар не найден

В меню **Отладка** выберем пункт **Продолжить отладку** или нажмем ту же кнопку на панели инструментов  . На этот раз введем существующий код товара, вычислим, чему теперь равно значение, возвращаемое функцией. Другое дело!

Вот так выглядит развернутая информация по элементу справочника "Номенклатура" (рис. 6.7).

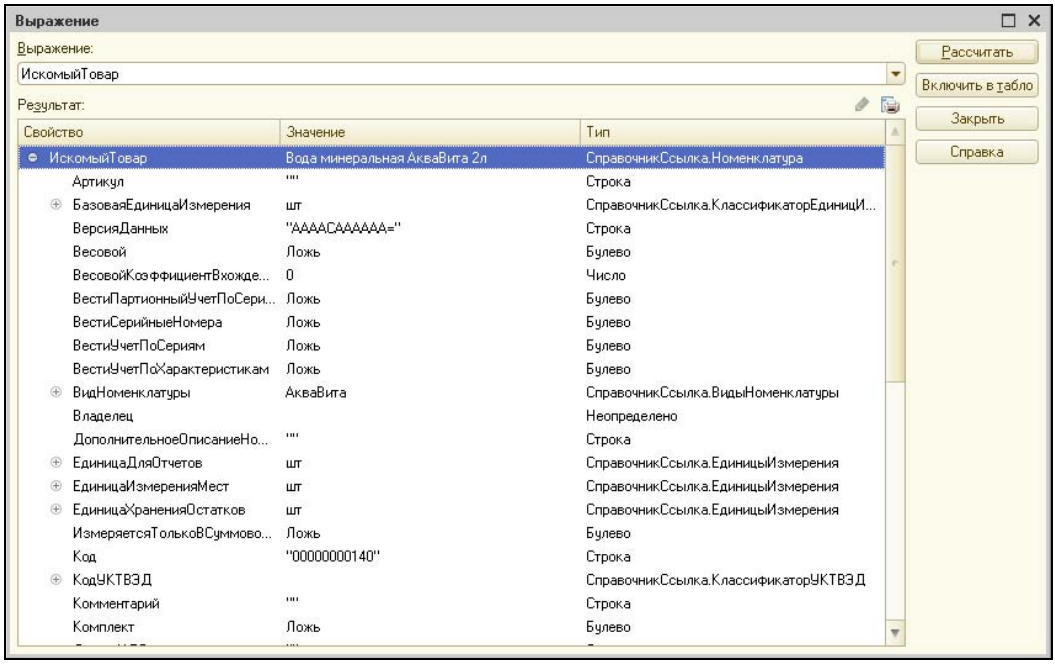



**Рис. 6.7.** Еще один результат работы функции.
Развернутая информация по элементу справочника товаров

Перенесем результат в табло, где уже собраны различные результаты отработки программных модулей. Кстати, ненужные уже можно удалить, выделив их мышью и нажав клавишу <Delete>.

Переменные и элементы, имеющие свойства, доступны к открытию с помощью значка "+" возле переменной (рис. 6.8).

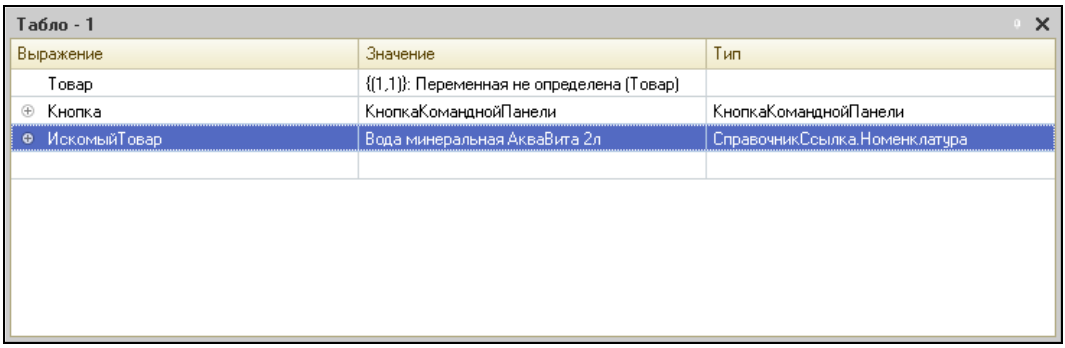



**Рис. 6.8.** В табло можно добавить много различных отладочных данных

В самом деле, на рис. 6.8 "ИскомыйТовар" — это элемент справочника товаров, он обладает какими-либо реквизитами и свойствами, которые имеются у этого справочника, а кнопка, как элемент формы, опять-таки имеет набор свойств.

Теперь напишем простую обработку, перебирающую в цикле справочник номенклатуры и присваивающую переменной `Товар` ссылку на каждый перебранный элемент справочника. Точку останова ставим уже после расчета переменной `Товар` (рис. 6.9).

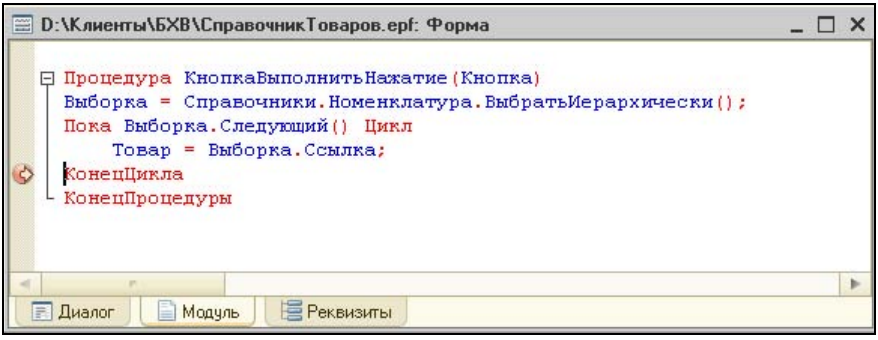



**Рис. 6.9.** Точка останова срабатывает при каждом проходе цикла

Зачем нам понадобился еще один пример на рассмотренную тему? А затем, чтобы продемонстрировать, как при работе цикла меняется значение переменной `Товар`, когда ей по очереди присваиваются все новые элементы. Именно так бы мы искали ошибку в работе цикла. На рис. 6.10 и 6.11 переменная `Товар` изменяется при переборе справочника в цикле.

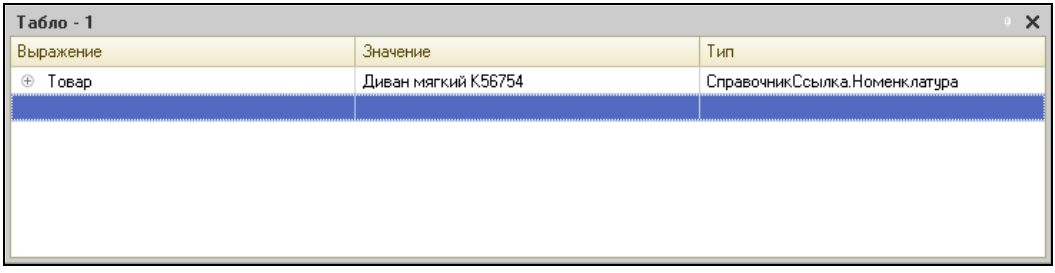



**Рис. 6.10.** Тип и значение переменной `Товар` в табло

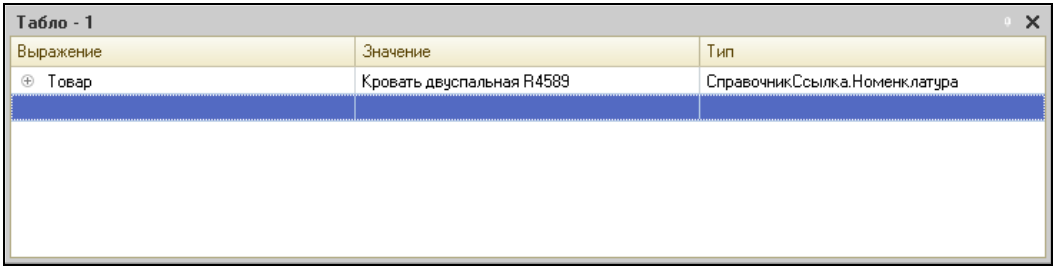



**Рис. 6.11.** При проходе цикла информация в табло изменилась

А теперь установим точку останова в обработке "РеестрРазвозок", созданной нами заранее (рис. 6.12).

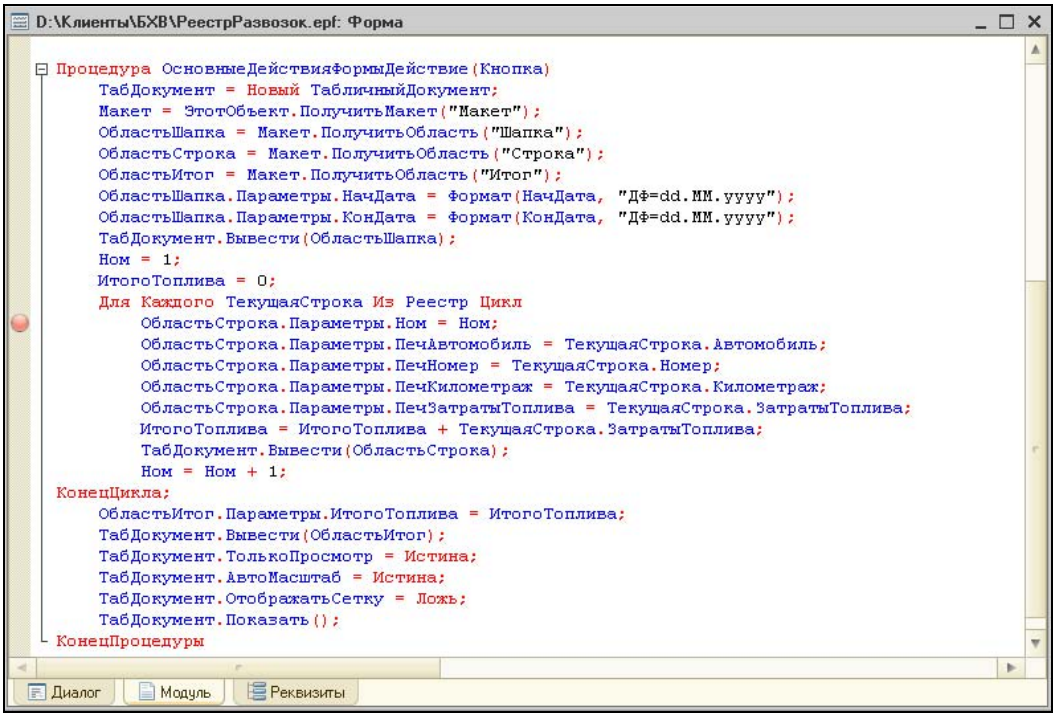



**Рис. 6.12.** Точек останова может быть много. Например, на каждой строке в цикле

При срабатывании точки останова пытаемся определить значение переменной `Ном`, соответствующей номеру строки в печатной форме. Выделяем в коде переменную `Ном`, вычисляем ее... и получаем вообще отсутствие значения (рис. 6.13).

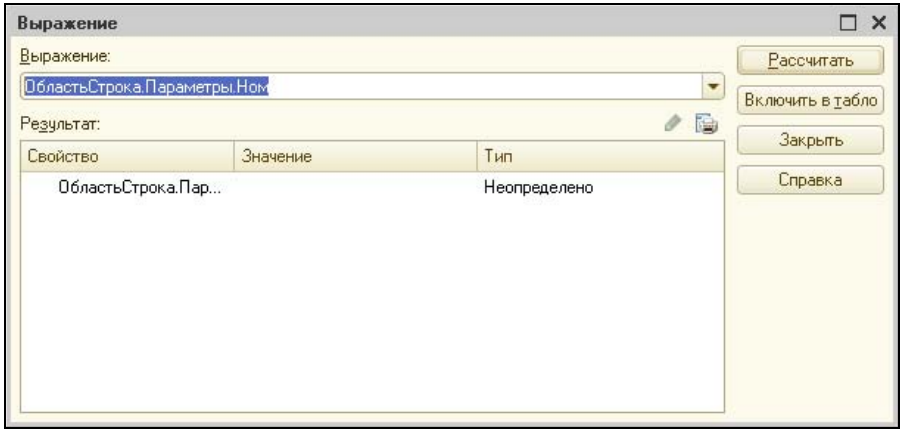



**Рис. 6.13.** Переменная не определена

Однако результат совершенно правильный. `Ном` приобретает значение только после операции присваивания, а при первом проходе цикла эта переменная просто не могла получить определенное значение. Однако при продолжении отладки мы по-

лучаем уже совершенно иные значения. По очереди выделите левую часть каждого выражения в цикле (`ОбластьСтрока.Параметры.Ном`, `ОбластьСтрока.Параметры.ПечАвтомобиль`, `ОбластьСтрока.Параметры.ПечНомер`, `ОбластьСтрока.Параметры.ПечКилометраж`, `ОбластьСтрока.Параметры.ПечЗатратыТоплива`, `ИтогоТоплива`) и для каждого сначала вычислим выражение кнопкой **ВычислитьВыражение**, а потом включим в табло полученный результат кнопкой **ВключитьВТабло**. Должно получиться так, как показано на рис. 6.14.

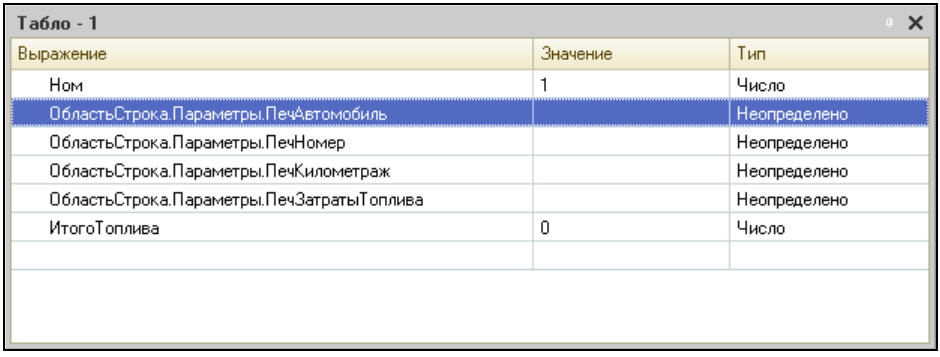



**Рис. 6.14.** Информация о перебираемых элементах в документах "Развозка"…

А теперь несколько раз подряд нажмите клавишу <F10>, что эквивалентно нажатию кнопки **Шагнуть через**  на панели инструментов. Курсор-стрелка при каждом нажатии будет перемещаться на одну строку модуля вниз, при этом соответствующее выражение, которое было вычислено и включено в табло, будет изменяться в зависимости от того, какое значение оно при этом принимает. Это так называемая *пошаговая отладка*, когда мы не запускаем отладку в автоматическом режиме от одной точки останова к другой, а выполняем модуль как бы сами — построчно. Результат первого пошагового прохода цикла можно видеть в табло, изображенном на рис. 6.15. Переменные те же, что и на рис. 6.14, а вот данные уже другие.

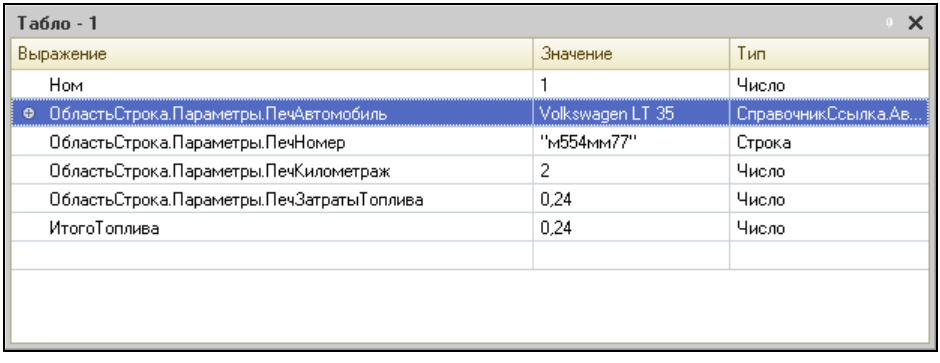
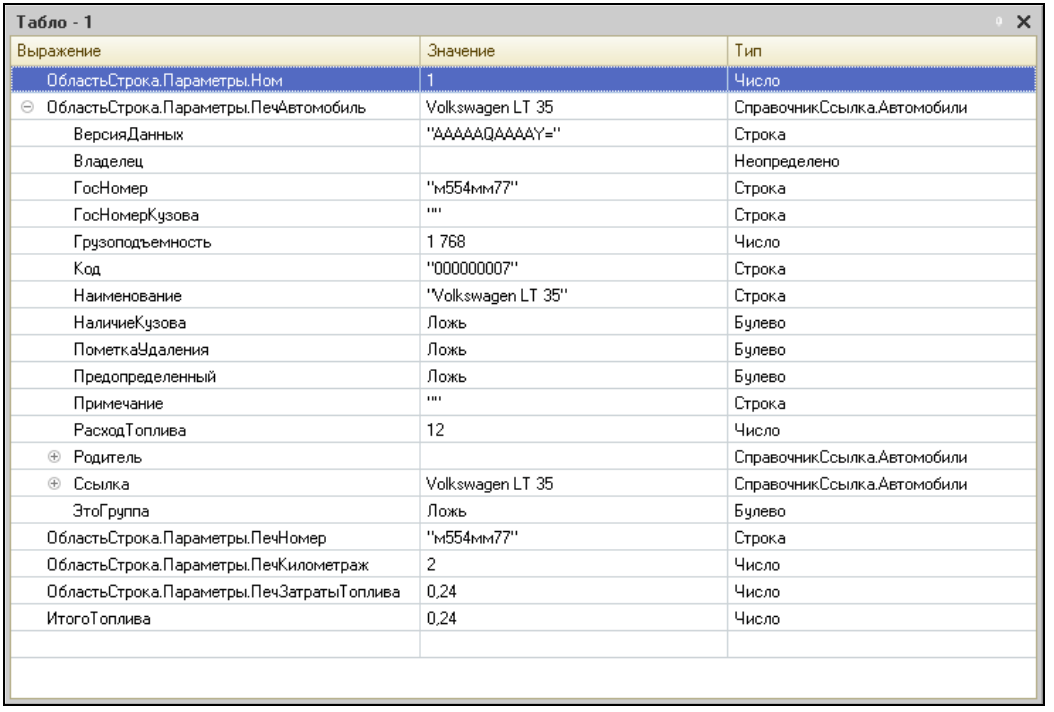



**Рис. 6.15.** ... при выполнении цикла меняется динамически

Обратите внимание, что переменная `ПечАвтомобиль` имеет уровни группировки и может быть "открыта" значком "+". Это правильно, поскольку объект имеет тип "Справочник.Автомобили". В развернутом виде коллекция свойств этой переменной выглядит так, как показано на рис. 6.16.



Табло - 1

| Выражение | Значение | Тип |
|---|---|---|
| ОбластьСтрока.Параметры.Ном | 1 | Число |
| ОбластьСтрока.Параметры.ПечАвтомобиль | Volkswagen LT 35 | СправочникСсылка.Автомобили |
| ВерсияДанных | "AAAAAQAAAAY=" | Строка |
| Владелец | | Неопределено |
| ГосНомер | "м554мм77" | Строка |
| ГосНомерКузова | "" | Строка |
| Грузоподъемность | 1 768 | Число |
| Код | "000000007" | Строка |
| Наименование | "Volkswagen LT 35" | Строка |
| НаличиеКузова | Ложь | Булево |
| ПометкаУдаления | Ложь | Булево |
| Предопределенный | Ложь | Булево |
| Примечание | "" | Строка |
| РасходТоплива | 12 | Число |
| Родитель | | СправочникСсылка.Автомобили |
| Ссылка | Volkswagen LT 35 | СправочникСсылка.Автомобили |
| ЭтоГруппа | Ложь | Булево |
| ОбластьСтрока.Параметры.ПечНомер | "м554мм77" | Строка |
| ОбластьСтрока.Параметры.ПечКилометраж | 2 | Число |
| ОбластьСтрока.Параметры.ПечЗатратыТоплива | 0,24 | Число |
| ИтогоТоплива | 0,24 | Число |

**Рис. 6.16.** Элемент типа "Справочник.Автомобили" имеет свой набор свойств

Таким образом, используя пошаговую отладку, мы прокручиваем выполнение программного модуля в "режиме замедленной съемки", построчно просматривая, как изменяются значения переменных, и можем увидеть, когда значение неверное.

# Команды отладчика, меню и кнопки

Теперь давайте рассмотрим список команд, которые мы можем использовать для отладки программных модулей. Все они доступны на панелях инструментов системы "1С:Предприятие" и в меню **Отладка**:

- **Начать отладку** — начинает отладку модуля;
- **Остановить отладку** — приостанавливает отладку модуля;
- **Продолжить отладку** — продолжает остановленную отладку модуля;
- **Завершить отладку** — завершает отладку модуля;

- **Вычислить выражение** — вычисляет текущее значение реквизита, переменной или результата обращения к функции;
- **Табло** — открывает табло;
- **Замер производительности** — позволяет разработчику оценивать скорость работы как всей конфигурации в целом, так и отдельной ее части. В этом режиме измеряется частота использования конкретных участков кода и скорость их выполнения. Для приведенного на рис. 6.12 примера окно замера производительности у меня выглядит так, как показано на рис. 6.17;

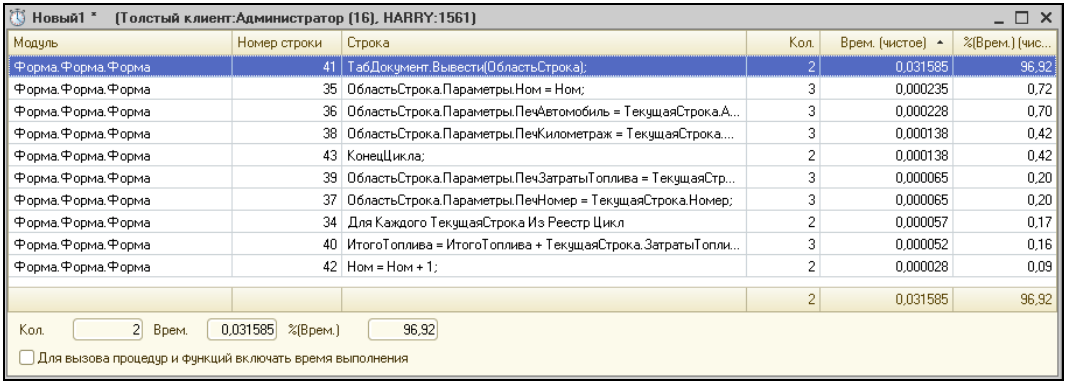



**Рис. 6.17.** Замер производительности для обработки "РеестрРазвозок"

- **Стек вызовов** — показывает стек вызовов, в котором выводится последовательность запуска процедур и функций, приведших к выполнению строки, отлаживающейся в настоящий момент. Для примера, приведенного на рис. 6.12, стек вызовов выглядит так, как представлено на рис. 6.18;

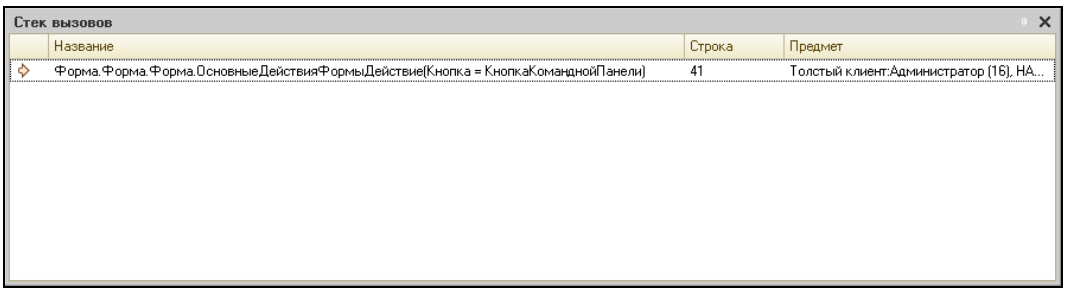



**Рис. 6.18.** Стек вызовов для обработки "РеестрРазвозок". Была выполнена процедура нажатия кнопки

- **Точка останова** — устанавливает точку останова в выбранной строке. В месте установки точки останова начнется отладка приложения. Например, надо начать отладку с пятой строчки своего кода. Ставим на ней точку останова и запускаем программу на выполнение. Как только дойдет очередь до точки оста-

нова, сразу же произойдет переключение в режим пошагового выполнения программы;

- **Точка останова с условием** — устанавливает точку останова с заданным условием. Допустим, при отладке цикла нам необходимо, чтобы отладка остановилась не просто на точке останова, а когда номер строки (переменная `Ном`) будет равняться 3. Создаем точку останова с условием, в открывшемся окне записываем условие (рис. 6.19). Нажимаем кнопку **OK**, после чего окно модуля будет выглядеть так, как показано на рис. 6.20. Как видите, точка останова с условием по внешнему виду отличается от обычной. Если навести на нее курсор мыши, во всплывающей подсказке будет показано условие для точки останова;

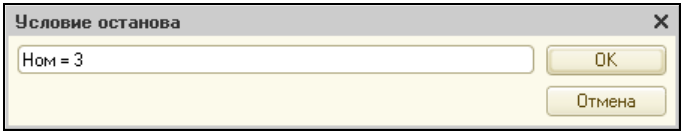



**Рис. 6.19.** Задаем условие для точки останова

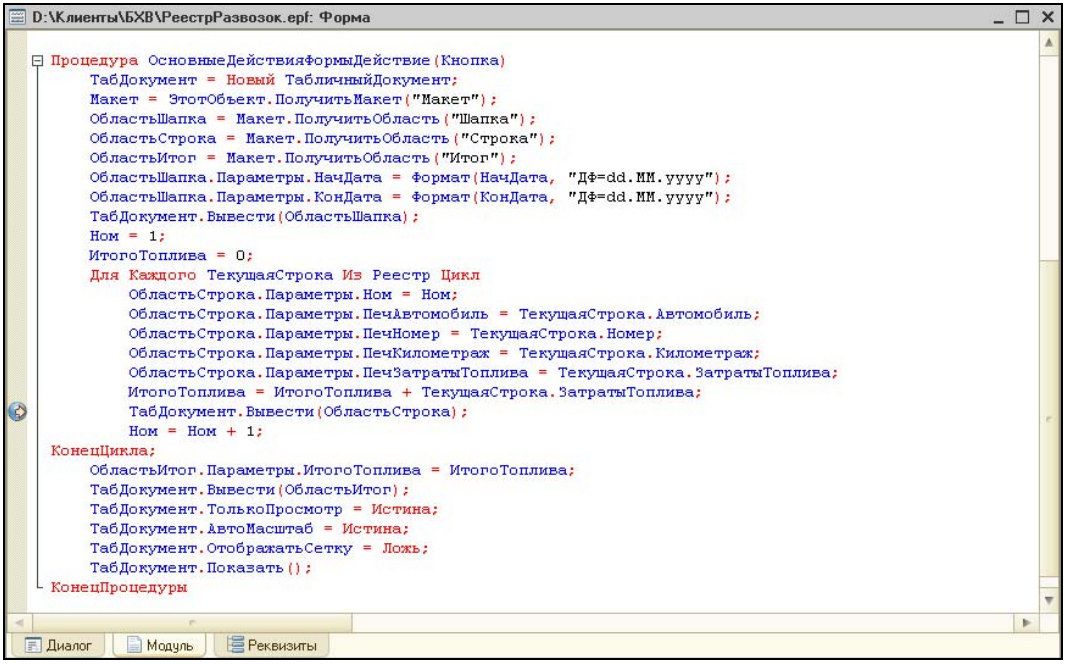



**Рис. 6.20.** Точка останова с условием

- **Отключить точку останова** — отключает точку останова для выбранной строки;
- **Убрать все точки останова** — убирает все точки останова в программном модуле;

-  **Текущая строка** — переход к строке, на которой завершилась текущая отладка. Мы можем переместиться в другое место модуля, но потом при необходимости легко найти, на каком месте завершили отладку;
-  **Идти до курсора** — отладка будет выполняться до места, в котором установлен курсор, после чего она будет завершена;
-  **Шагнуть В** — отладка кода с заходом в процедуру/функцию, к которой идет обращение в отлаживаемом модуле;
-  **Шагнуть Из** — выход из тела функции/процедуры. Применяется, если программист зашел предыдущим пунктом в процедуру/функцию, а затем передумал пошагово просматривать ее выполнение. В этом случае, при нажатии данной кнопки курсор переместится на ту точку, откуда был произведен вход в процедуру/функцию;
-  **Шагнуть Через** — при достижении вызова процедуры/функции — "перешагивание" через нее, без захода внутрь.

# Заключение

Вот и все. Наш экскурс в мир 1С-программирования подошел к своему логическому завершению. Я надеюсь, что теперь читатель сможет без особенного напряжения, удобно и комфортно использовать все те инструменты разработчика и администратора, которые предоставляет нам система "1С:Предприятие". Конечно, с моей стороны было бы слишком самонадеянно считать, что после прочтения этого руководства у читателя не останется абсолютно никаких вопросов в работе с системой. Вопросы будут, вопросов будет много, ведь это только начало. Вы еще не стали специалистом по программированию в системе "1С:Предприятие", однако получили базовые знания, некоторый фундамент, на основе которого вы сможете двигаться дальше. Не останавливайтесь на достигнутом. Читайте специальную литературу, задавайте вопросы на форумах, посвященных 1С, и обязательно практикуйтесь. Если программист регулярно не "набивает руку", он не только не продвигается вперед, но начинает забывать то, что уже знает. Так что пишите обработки, разрабатывайте конфигурации, экспериментируйте, оттачивайте свое мастерство программиста. Искренне желаю вам удачи!

Дорогу осилит идущий!

Может быть, после прочтения книги у читателя появятся вопросы к автору. Кто-то столкнется с проблемой, не описанной в книге, кто-то захочет выразить автору благодарность или, наоборот, обрушить на него град критических замечаний. Буду рад всему, в том числе и последнему, поскольку критика всегда конструктивна, а учиться, в том числе и на собственных ошибках, не поздно никогда.

Удачи!

# Предметный указатель

## П



## Р



## С



## Т



## Ф

## Х



## Ц