Глава 1. Создание новой конфигурации

Предварительная постановка задачи

Мы рассмотрим создание конфигурации на примере склада. При этом мы допустим ряд упрощений, например, не будем вести учет НДС и предположим, что невозможны возвраты от покупателей. Допустим, что у нас имеется гипотетический заказчик конфигурации - компания под названием **TechnoTrade**, занимающаяся оптовой торговлей бытовой техникой и нам нужно автоматизировать учет ее деятельности.

Из разговора с заказчиком выясняется, что в области бытовой техники существует определенная традиция наименования товаров. Типичные товары именуются примерно так:

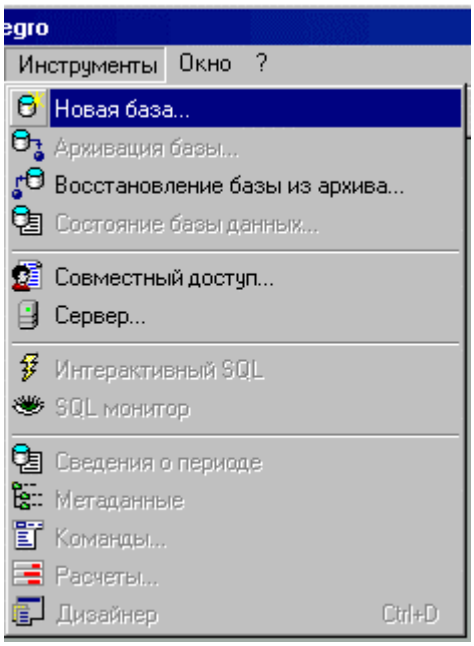
СТИРАЛЬНАЯ МАШИНА BOSCH WFE 2821 EU
ВАРОЧНАЯ ПОВЕРХНОСТЬ BOSCH NKN 646 F
ВАРОЧНАЯ ПОВЕРХНОСТЬ ZANUSSI ZGG 753 Alu
ВЫТЯЖКА ELECTROLUX EFC 939 X

Можно считать за правило, что наименования конкретных товаров состоят из уникальной комбинации трех практически независимых параметров: **вида товара, марки и артикула.** По крайней мере, заказчик уверяет, что в подавляющем большинстве случаев это так…

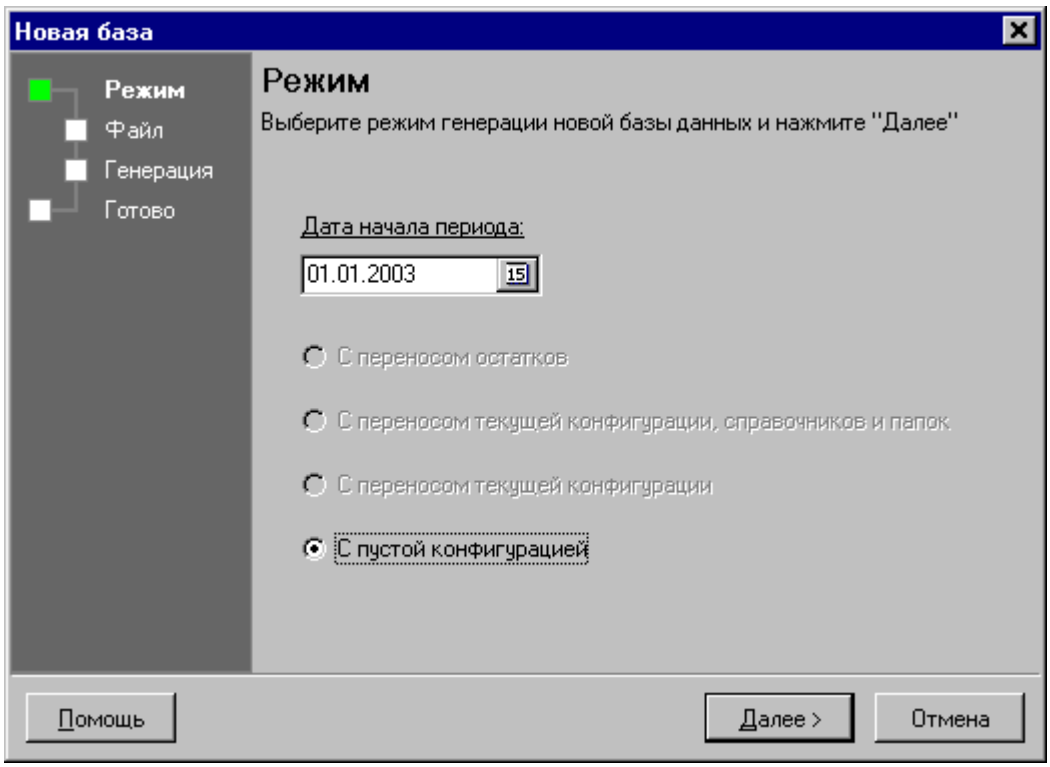
Мы будем постепенно уточнять постановку задачи в процессе проектирования, так как на практике этот процесс часто растягивается во времени и содержит целый ряд этапов.

Создаем новую базу данных

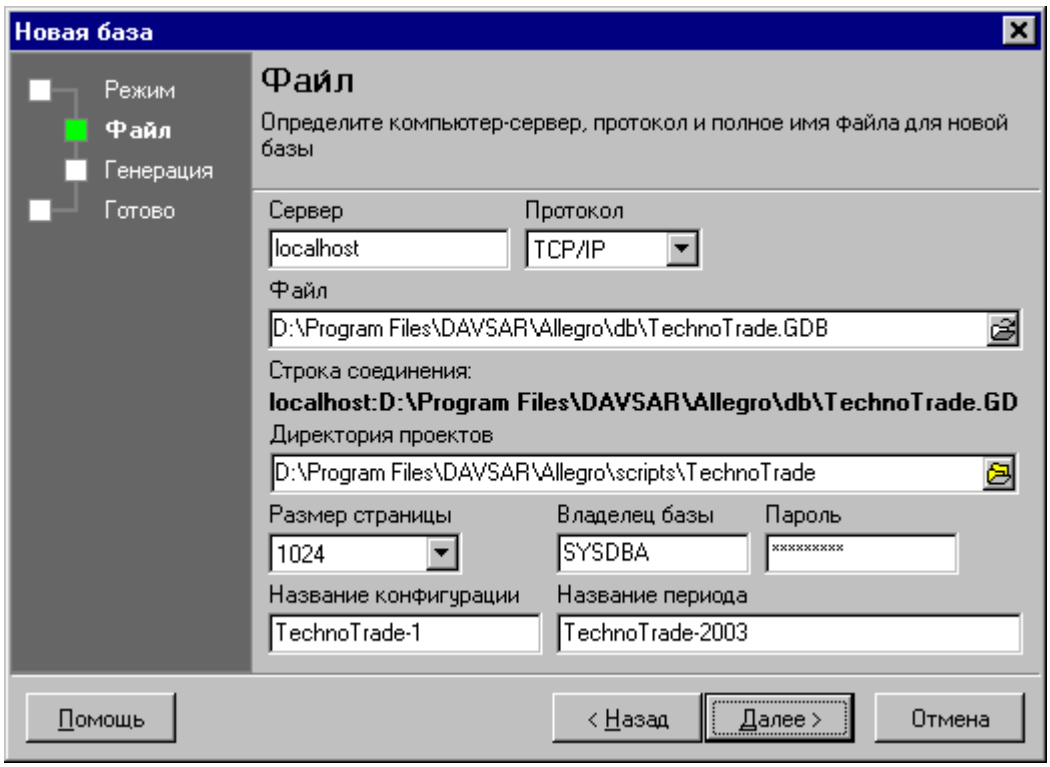
Создадим новую пустую базу под названием **TechnoTrade.gdb**. Для этого воспользуемся пунктом **Инструменты/Новая база** Главного Меню программы:



Появится мастер создания новой базы. Первое, что мы должны ввести, это дату начала нового периода. Пусть это будет 1 Января 2003 года.

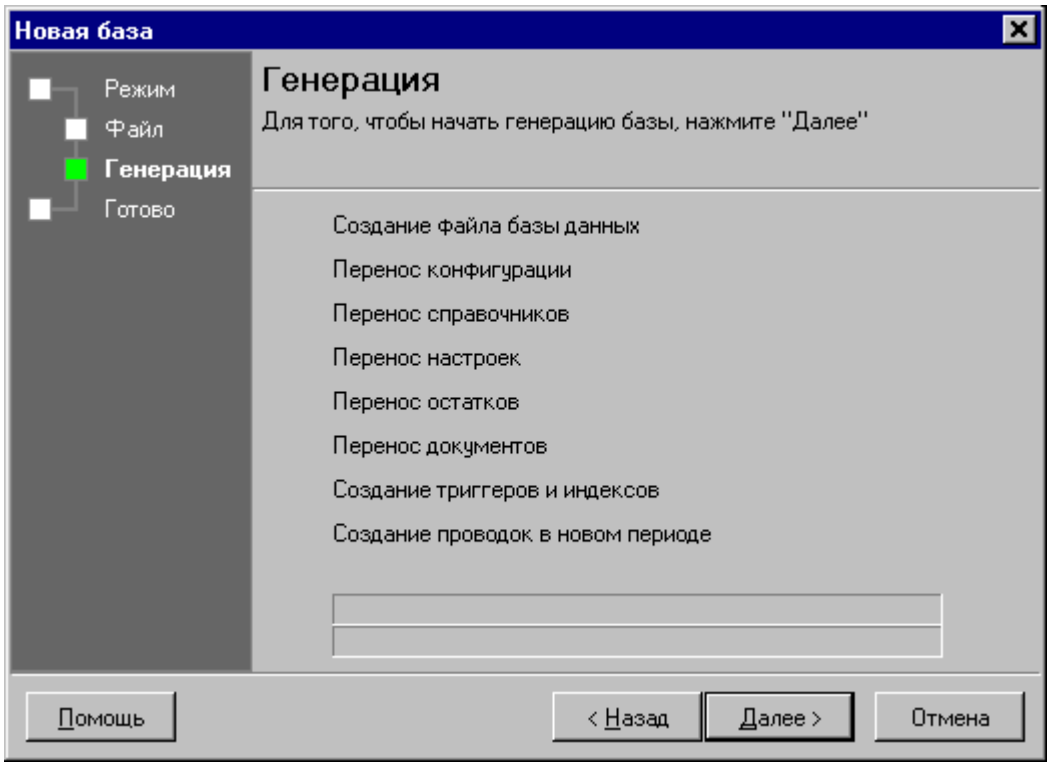


Нажмем кнопку ***Далее.***



Теперь нам нужно указать имя файла для будущей базы данных и название директории, в которой будут находиться файлы скриптовых проектов. Директорию проектов следует создать средствами операционной системы, например, в «Проводнике Windows». Создадим директорию TechnoTrade в подкаталоге Allegro\Scripts и укажем ее в качестве ***Директории проектов***.

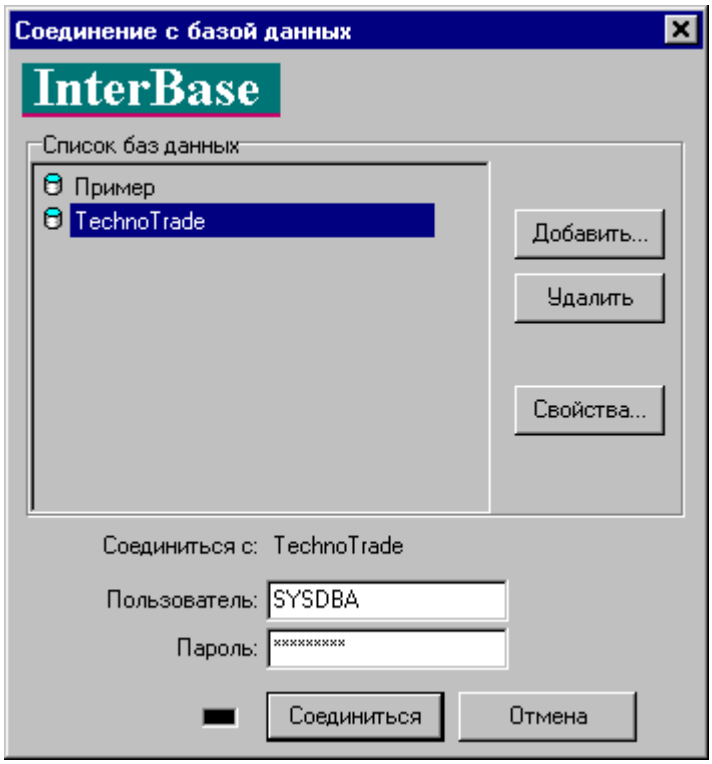
Введем имя будущего ***Владельца базы*** и ***Пароль*** (если они еще не введены). По умолчанию это SYSDBA и masterkey. Введем ***Название конфигурации***, например, TechnoTrade-1 и ***Название периода***, например, TechnoTrade. Нажмем кнопку ***Далее.***



Все готово для генерации новой базы.

Нажмем еще раз кнопку ***Далее***. Программа создаст новую базу данных.

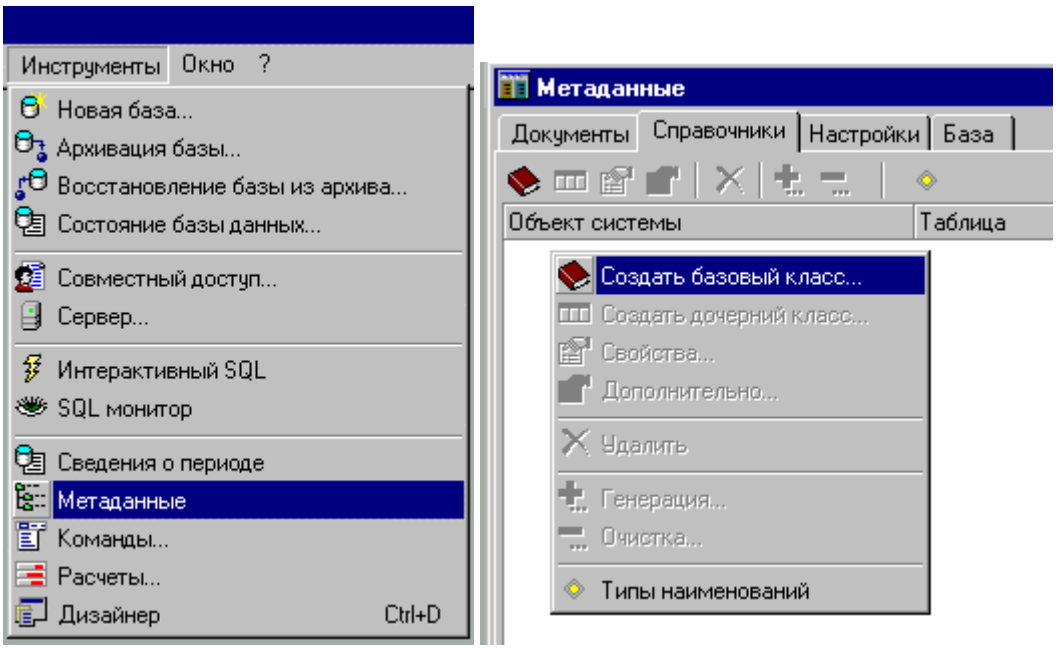
Закроем мастер, нажав на кнопку ***Готово*** и соединимся с новой базой данных с помощью пункта ***База/Соединиться*** Главного Меню. Заметим, что новая база данных попала в список под названием TechnoTrade:



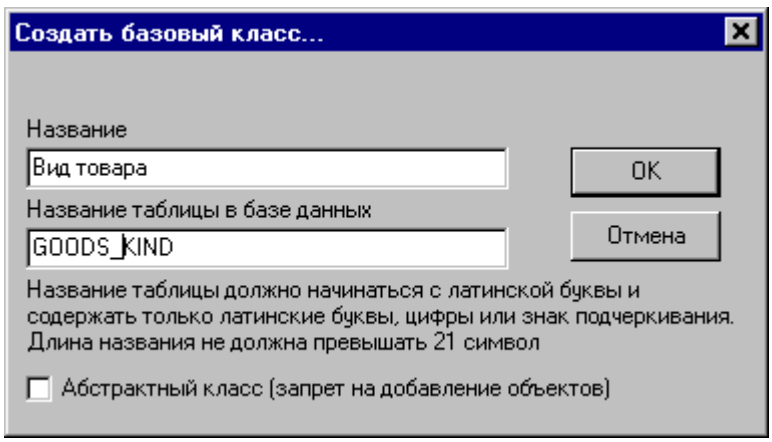
Глава 2. Создание справочной системы

Создаем справочник «Вид товара»

Исходим из того, что все наши товары описываются уникальной комбинацией признаков ***Вид товара*** + ***Марка*** + ***Артикул***. Нам понадобится несколько справочников. Сначала создадим справочник ***Вид товара***. Вызываем окно «Метаданные» через Главное Меню ***Инструменты/Метаданные***. В окне «Метаданные» выбираем закладку «Справочники» и с помощью контекстного меню (вызываемого правой кнопкой мыши) или кнопки с книжечкой на панели инструментов создаем базовый класс.



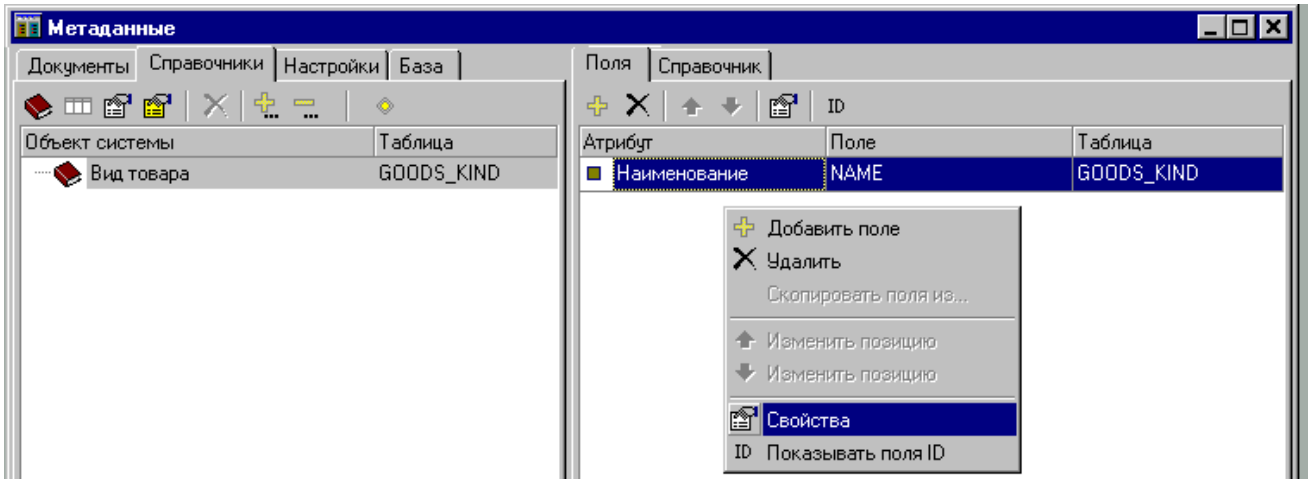
Появится окно диалога, в котором введем название нового справочника «Вид товара» и имя таблицы GOODS_KIND и нажмем кнопку ***OK***



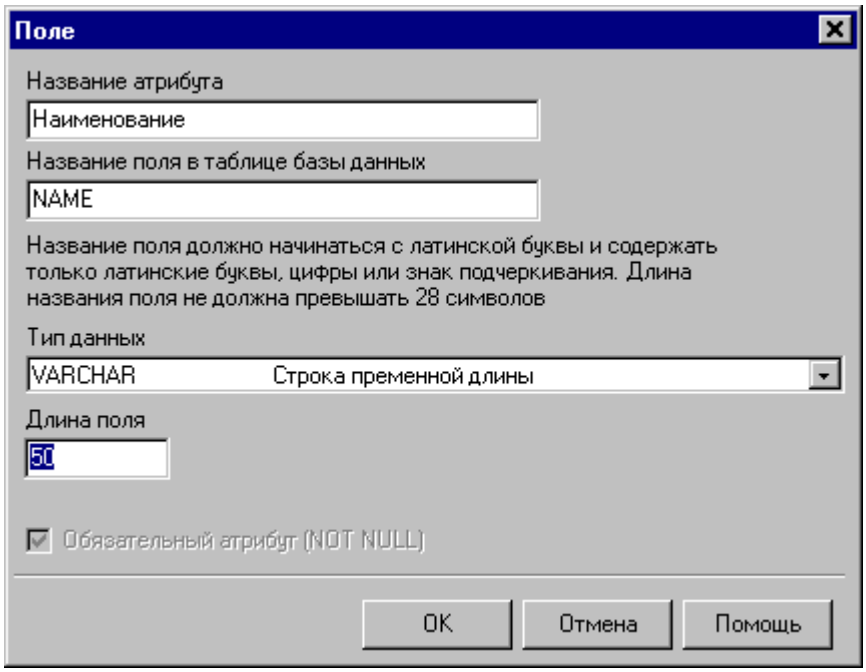
Базовый справочник создается с двумя готовыми полями: ID INTEGER и NAME VARCHAR(20). Поле NAME можно удалить или модифицировать. В нашем случае нужно увеличить длину поля NAME, так как двадцати символов на название вида товара может не хватить.

Изменяем длину поля NAME

Для того чтобы изменить длину поля NAME, щелкнем дважды на нем или вызовем пункт контекстного меню «Свойства»:

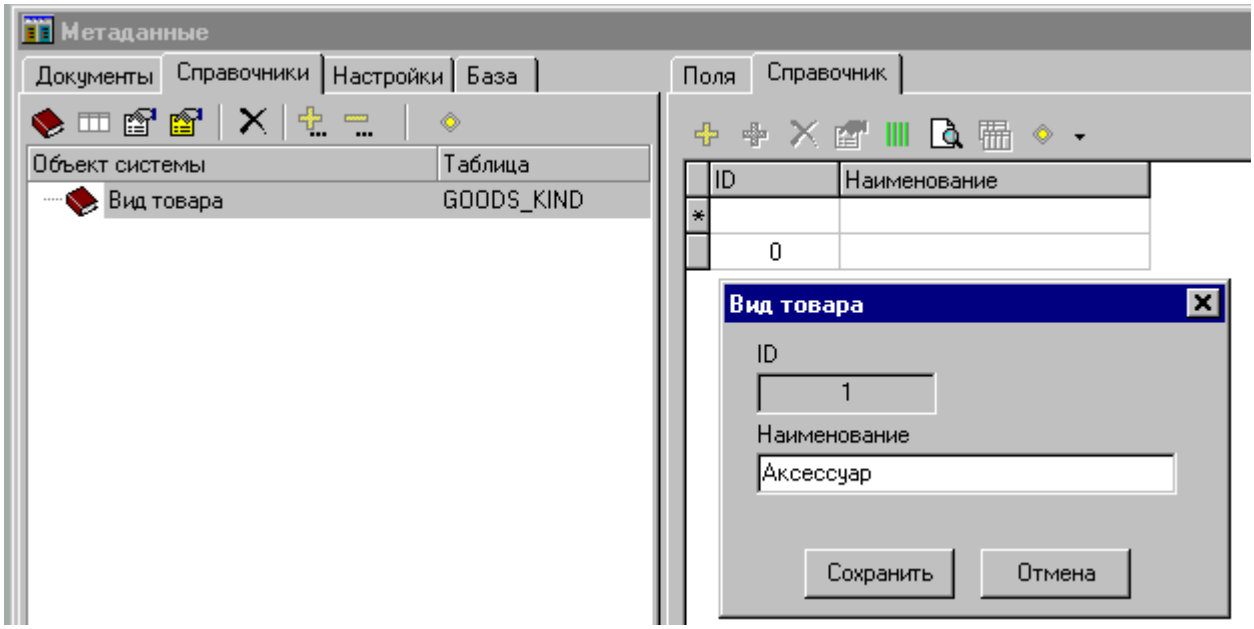


В диалоге свойств поля изменим его длину с 20 на 50 символов:



Добавляем данные в справочник

Выберем закладку «Справочник» в правой стороне окна «Метаданные» и внесем наименования нескольких видов товаров в справочник. Прежде, чем вводить данные в справочник, нужно снять блокировку на ввод данных с помощью пункта контекстного меню «Блокировка справочников» или нажав Ctrl+B. Диалог добавления в справочник вызывается нажатием клавиши Insert с клавиатуры или при помощи кнопки с плюсиком на панели инструментов или с помощью пункта «Добавить» контекстного меню.



Внесем ряд видов товаров, чтобы не иметь дело с пустым справочником:

Аксессуар
Блендер
Варочная поверхность
Вентиляционный канал
Вытяжка
Газовая колонка
Гриль
Кофеварка
Микроволновая печь
Стиральная машина
Сушильная машина
Тостер

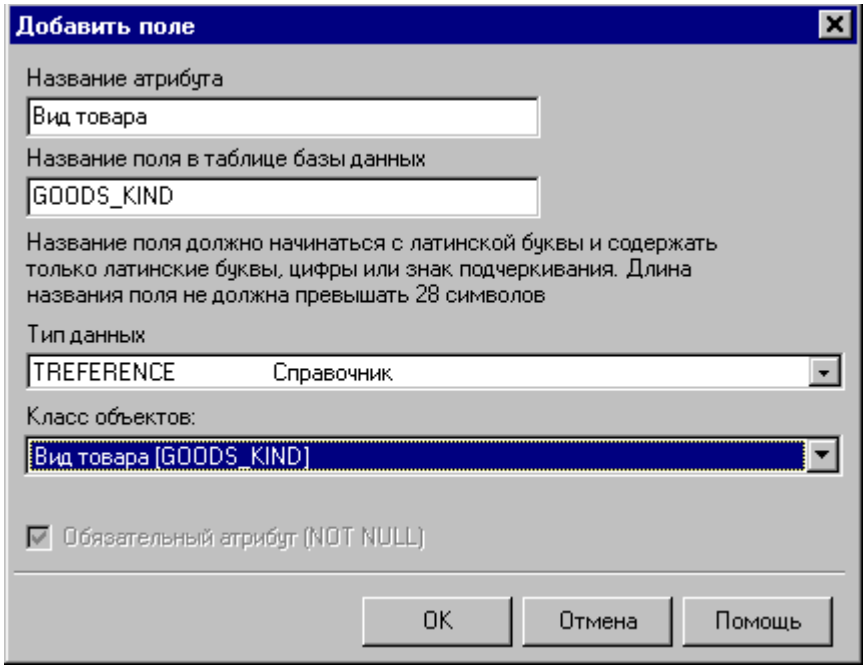
Создаем справочник «Марка товара»

Аналогично создадим базовый класс ***Марка товара***, таблица GOODS_MARK и изменим длину поля NAME с 20 до 30 символов. Введем десять-двадцать марок товаров:

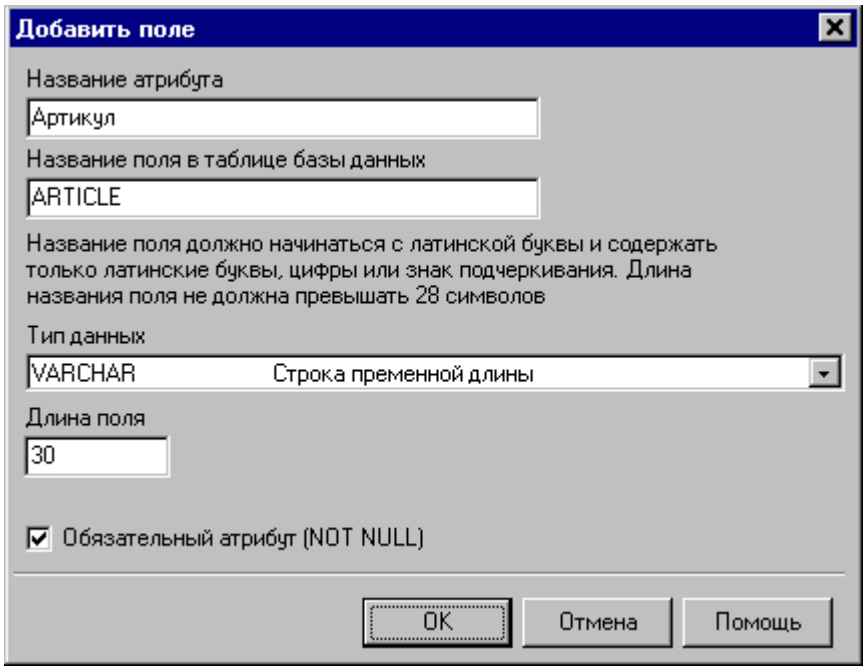
AEG	BOSCH	FABER	SIEMENS	ZEIKO
ARDO	CATA	HACKER	SIRIUS	
ARISTON	DAMIXA	IMPERIAL	SYSTEMAT	
BAUKNECHT	ELECTROLUX	MIELE	ZANUSSI	

Создаем справочник «Товар»

Теперь мы уже можем создать базовый класс ***Товар***, таблица GOODS. После создания класса, выберем закладку «Поля» в окне «Метаданные» и удалим из справочника ***Товар*** поле NAME. Вместо него добавим 2 новые поля: ***Вид товара*** и ***Марка товара***. Эти поля имеют тип данных TREFERENCE (Справочник). Рекомендуется давать полям типа TREFERENCE имена, совпадающие с именами справочников, на которые эти поля ссылаются (во избежание дальнейшей путаницы):



Добавим третье поле: ***Артикул*** (ARTICLE). Выберем тип данных VARCHAR (строка переменной длины) с длиной 30. Оставим галочку «Обязательный атрибут» включенной. Когда пользователи будут вносить данные, сервер не допустит пустого значения в этом поле.

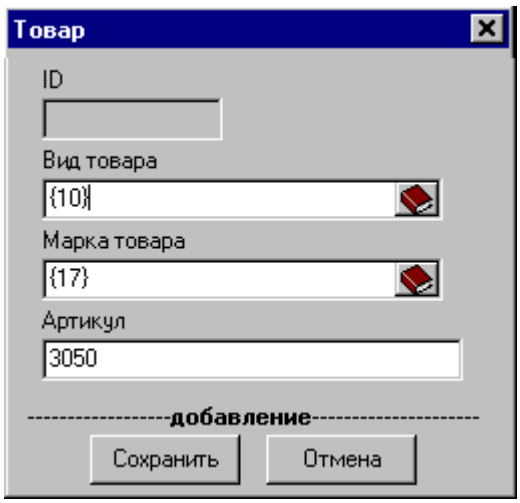


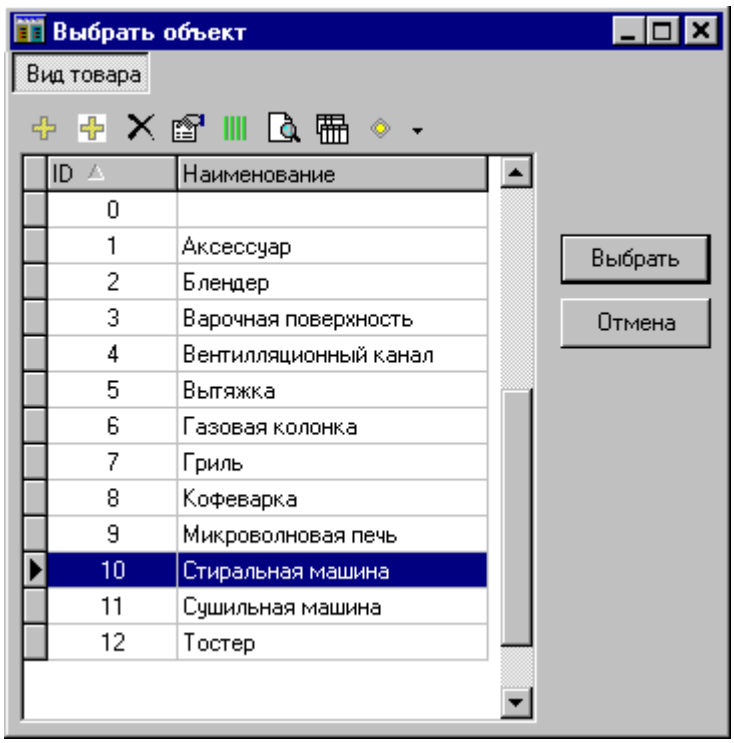
Итак, часть справочной системы, посвященной товарам почти готова. Попытаемся внести несколько товаров и посмотрим, что из этого получится. Выберем закладку «Справочник» и нажмем кнопку с плюсиком

или клавишу Insert. Появится диалог добавления записи в справочник, в котором должны отображаться 4 поля: ***ID*** (невыбираемое), ***Вид товара***, ***Марка товара*** (поля с кнопкой в виде книжки), ***Артикул*** (обычное поле).

Нажимая кнопку с книжкой или комбинацией клавиш ***Alt+Стрелка вниз*** вызовем соответствующие справочники и введем Вид товара и Марку товара.

Попробуем создать товар «Стиральная машина BOSCH 3050».



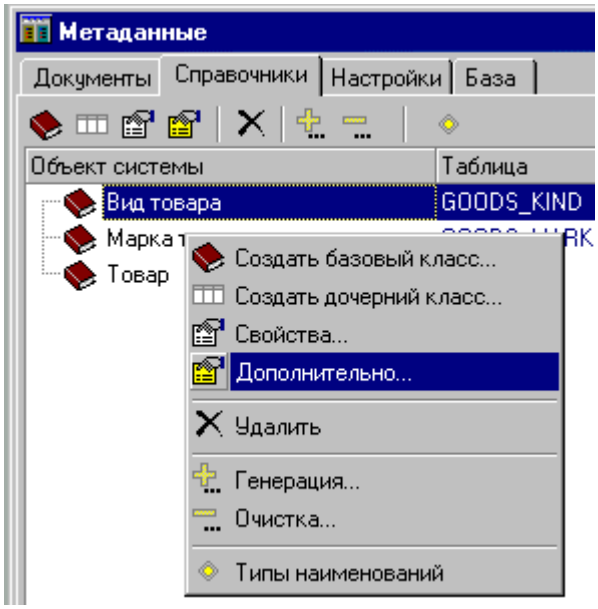


Мы видим, что в двух полях диалога в качестве названий выбранных элементов справочников фигурируют их ID, что, согласитесь не очень удобно для восприятия.

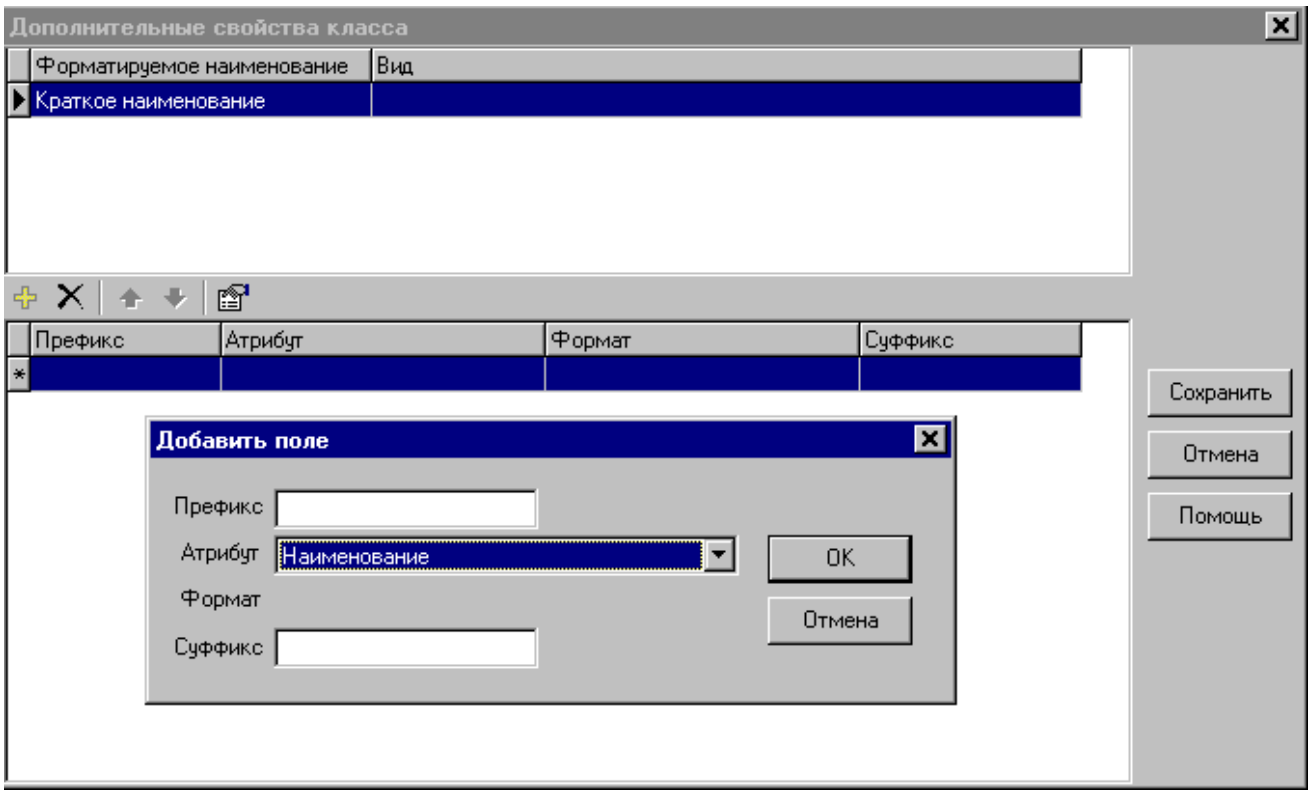
Для того чтобы избавиться от этого явления нам нужно настроить ***способ форматирования кратких наименований*** справочников ***Вид товара*** и ***Марка товара***.

Изменяем форматирование кратких наименований

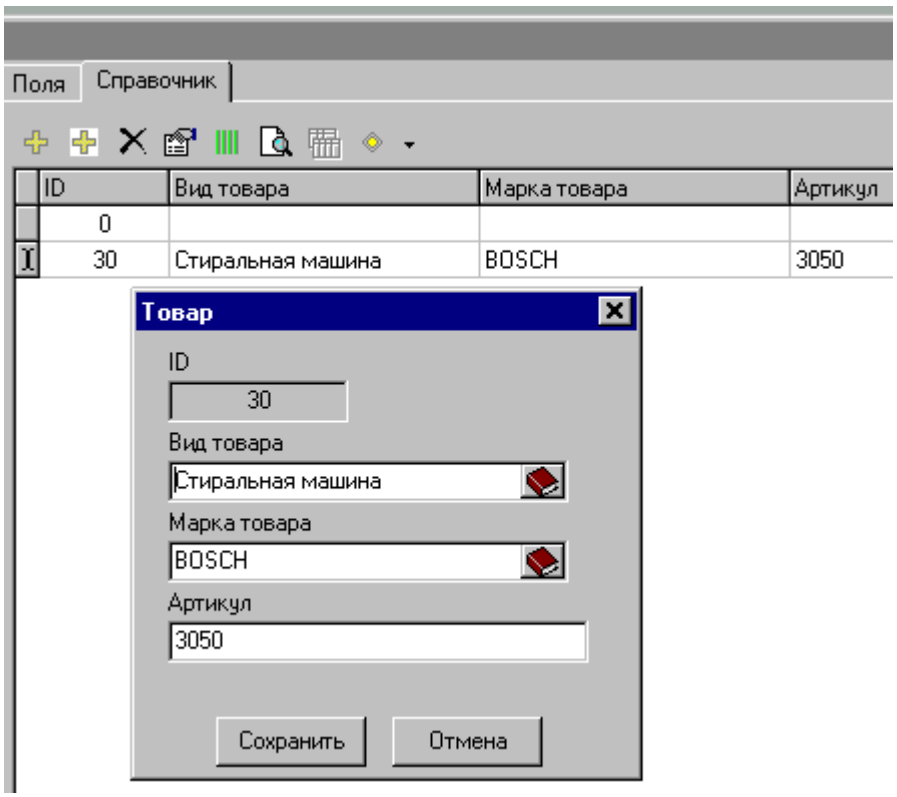
Выберем в окне «Метаданные» слева класс ***Вид товара***. Для изменения способа форматирования кратких наименований вызовем настройку дополнительных свойств класса через пункт контекстного меню «Дополнительно»:



В появившемся окне удалим имеющуюся в нижнем списке строку с атрибутом ID. Вместо нее добавим строку, использующую атрибут «Наименование» без префиксов и суффиксов.



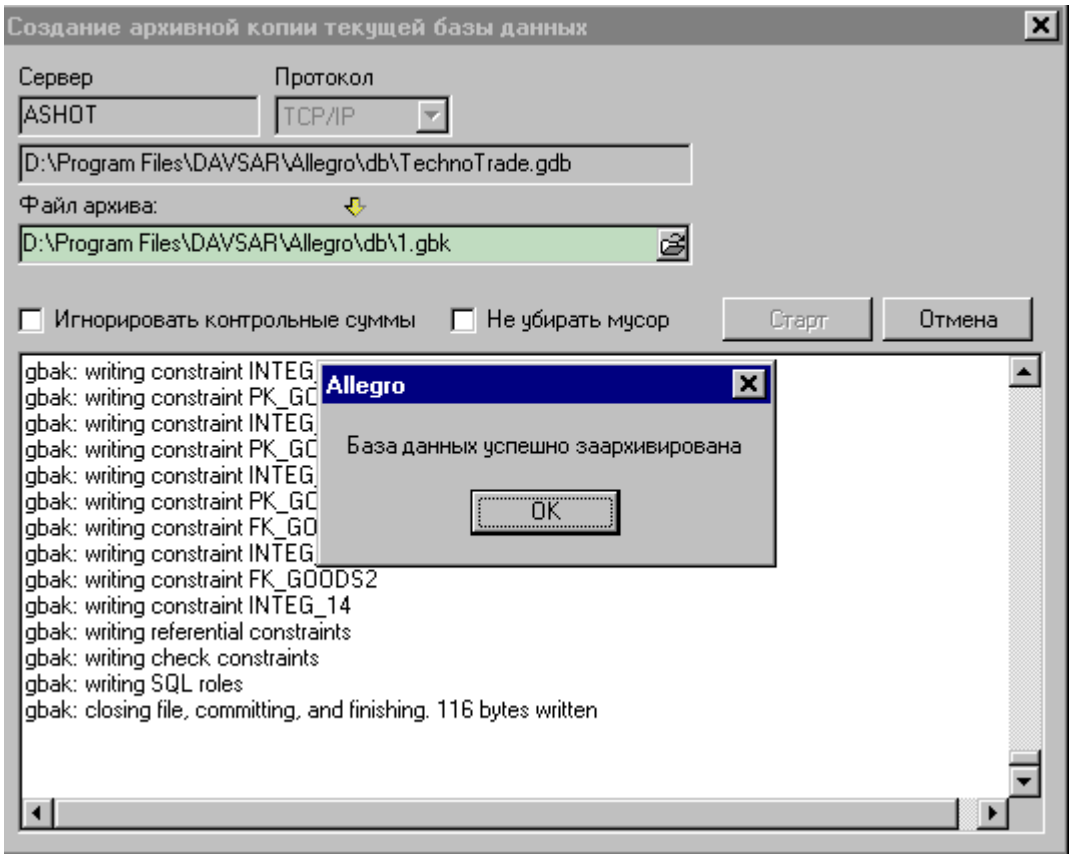
Сохраним этот способ форматирования. Теперь в качестве кратких наименований в справочнике ***Вид товара*** используется поле NAME. Аналогично изменим форматирование кратких наименований справочника ***Марка товара***.



Краткие наименования всегда используются программой при отображении ссылок на объекты справочной системы. Теперь поля ссылок отображаются по-человечески и можно ввести несколько товаров вручную.

Создаем архивную копию базы данных

На всякий случай создаем архивную копию базы данных. Это рекомендуется делать перед каждым существенным изменением метаданных. В качестве названий файлов для архивных копий удобно использовать порядковые номера. Для создания архивной копии (backup) базы, используем пункт ***Инструменты/Архивация базы*** Главного меню:



Создаем ограничение на уникальность

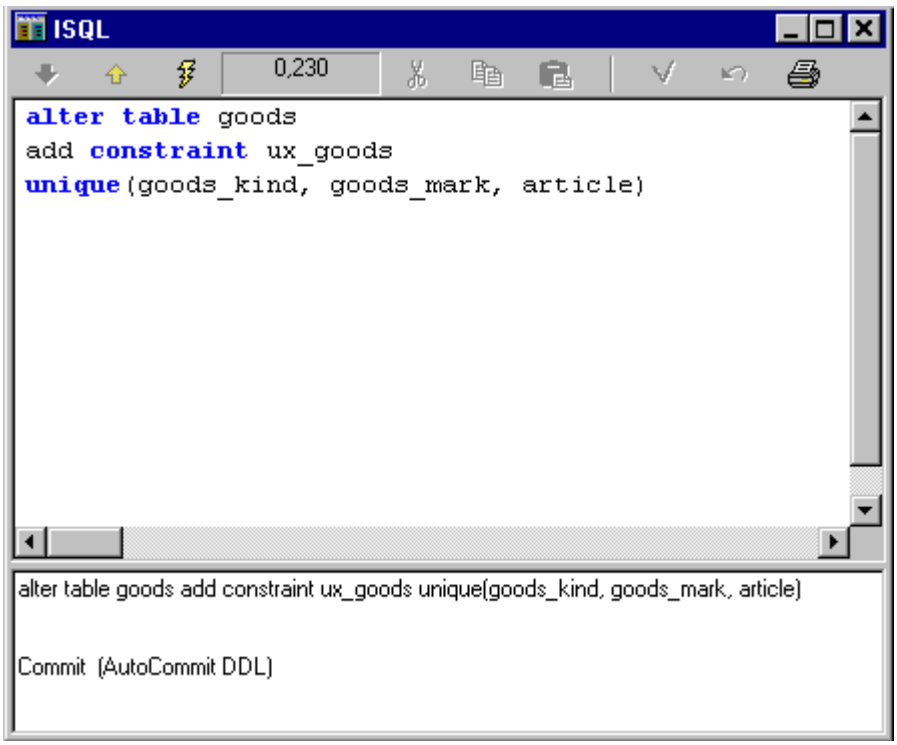
Во избежание таких неприятных инцидентов, как случайное создание товаров-дубликатов, неплохо бы иметь ограничение на уникальность сочетания значений в 3 полях, характеризующих товар. Прежде, чем вводить это ограничение, убедимся, что в справочнике пока нет товаров-дубликатов. Если такие имеются - нужно их удалить.

Теперь создадим уникальный индекс по полям GOODS_KIND, GOODS_MARK и ARTICLE. Делается это вручную при помощи окна ISQL (интерактивного SQL), вызывающегося с помощью пункта ***Инструменты/Интерактивный SQL*** Главного меню. Наберем SQL команду:

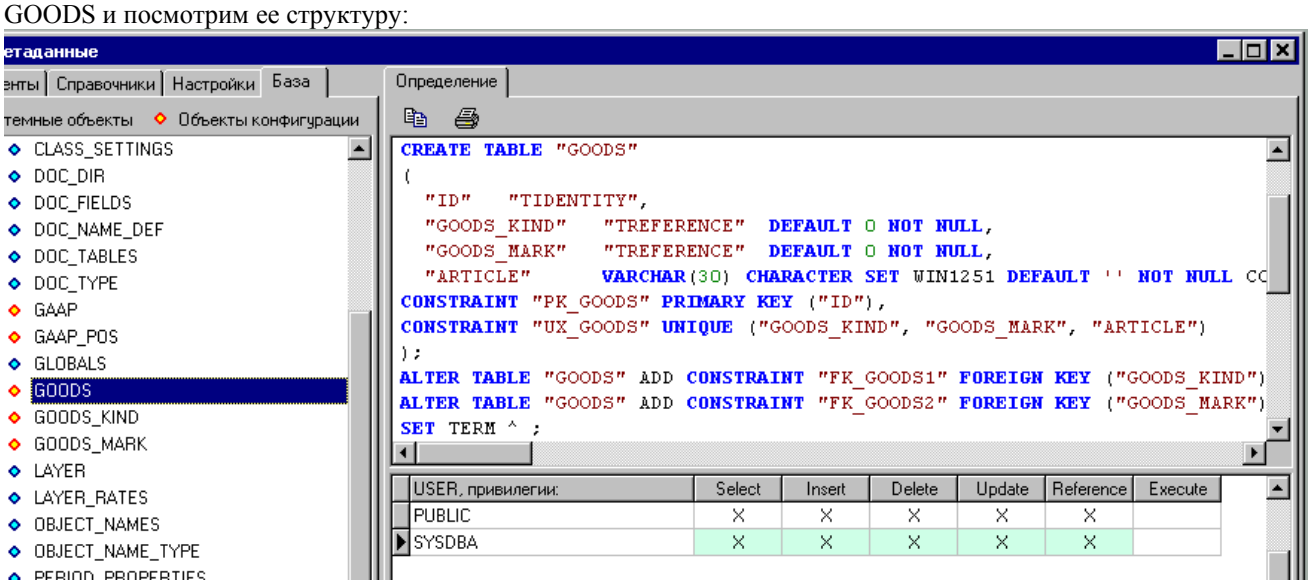
```
alter table goods
add constraint ux_goods
unique(goods_kind, goods_mark, article)
```

и нажмем комбинацию Ctrl+Enter или кнопку ***Выполнить*** на панели инструментов (значок молнии).

Если команда набрана правильно, то она выполнится и ISQL автоматически подтвердит транзакцию, о чем будет свидетельствовать сообщение Commit (AutoCommit DDL) в нижней части окна. Окно ISQL позволяет нам общаться с сервером баз данных напрямую. Создавая ограничение на уникальность, мы назвали его **ux_goods**. Рекомендуется давать названия ограничениям (constraints). Тогда их легче удалить потом в случае необходимости.

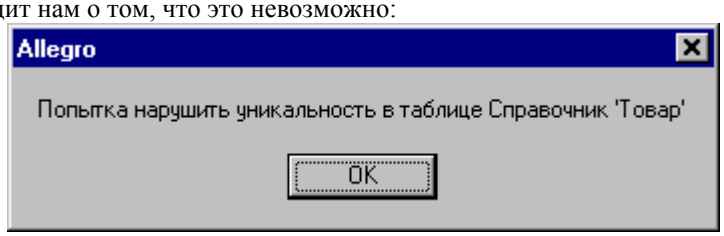


Теперь мы можем посмотреть, как выглядит структура таблицы GOODS в самой базе данных. Для этого в окне «Метаданные» выберем закладку «База» (слева), откроем дерево таблиц, найдем там нашу таблицу товаров GOODS и посмотрим ее структуру:



Мы видим, что таблица GOODS имеет 4 поля, один первичный ключ PK_GOODS и ограничение на уникальность UX_GOODS, которое мы только что создали. Кроме того, таблица имеет два внешних ключа FK_GOODS1 (на таблицу GOODS_KIND) и FK_GOODS2 (на таблицу GOODS_MARK). Внешние ключи служат для обеспечения ссылочной целостности между таблицами справочников и создаются автоматически.

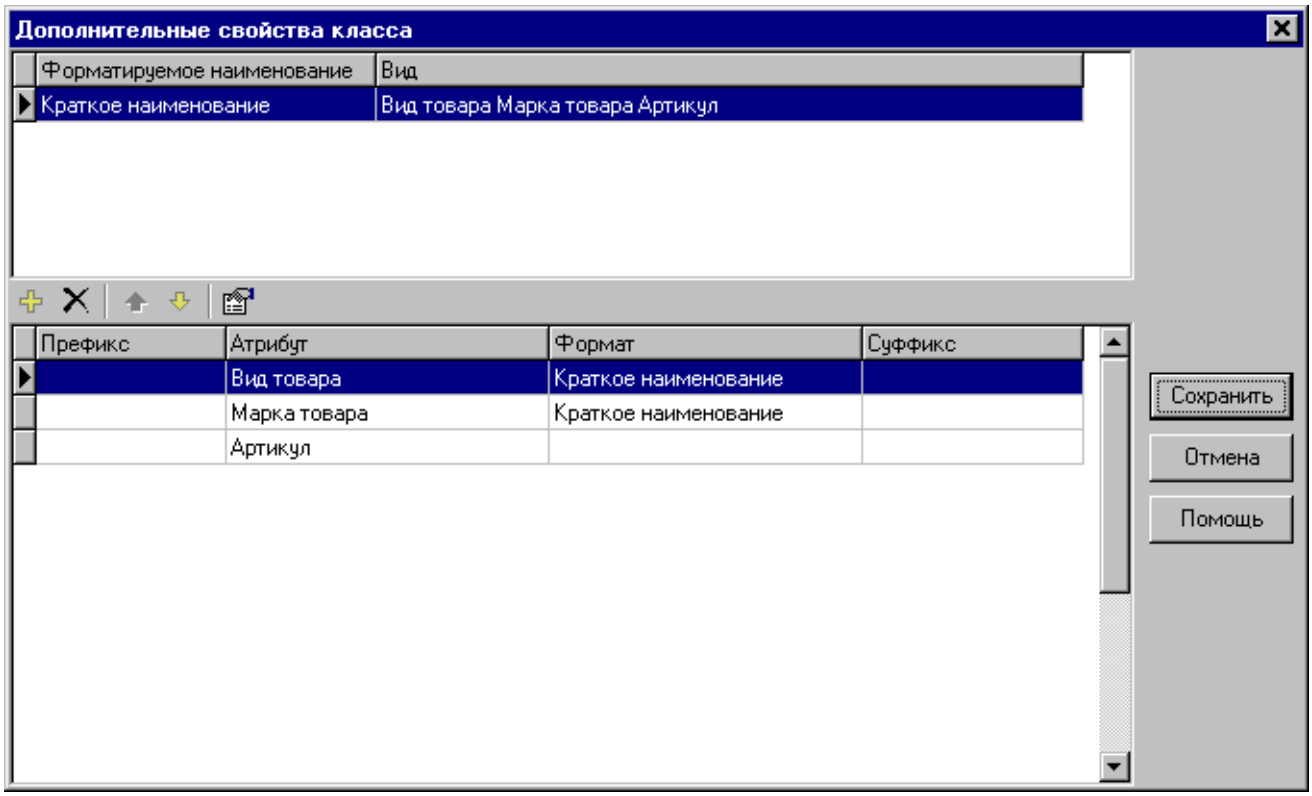
Теперь вернемся к справочнику ***Товар***, выбрав закладку «Справочники» в окне «Метаданные». Попытаемся добавить дубликат в справочник, например ту же «стиральную машину BOSCH 3050». Программа сообщит нам о том, что это невозможно:



Итак, мы защитили справочник от случайного создания товаров-дубликатов. Разумеется, хорошо бы еще ввести ограничение на уникальность наименований в справочниках ***Вид Товара*** и ***Марка товара***. Сделайте это самостоятельно в качестве упражнения.

Форматирование кратких наименований в справочнике «Товар»

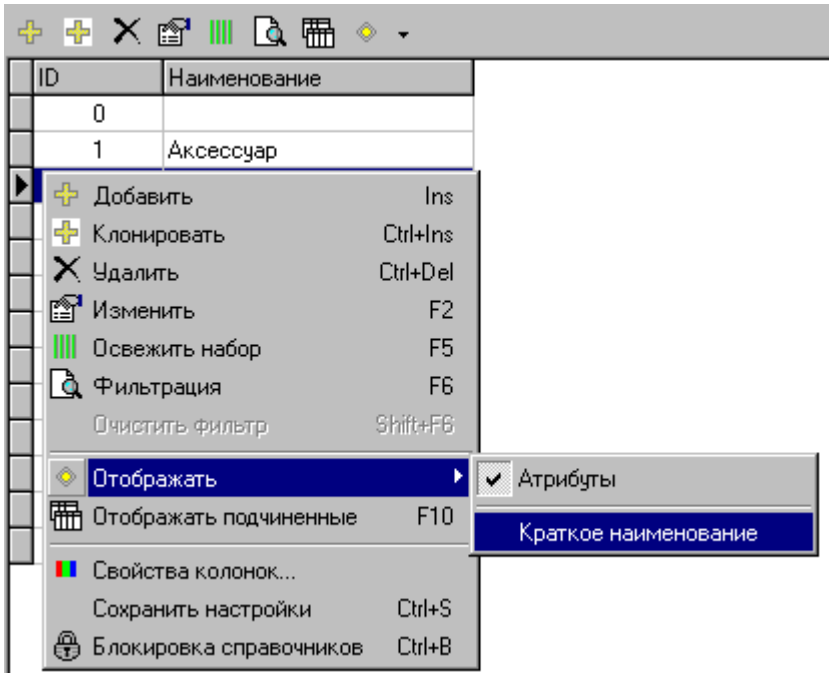
Теперь нам нужно настроить форматирование кратких наименований справочника ***Товар***. Это позволит упростить печатание документов и поиск товаров по наименованию. Мы знаем, что заказчик склоняется называть свои товары в форме: «Вид товара, пробел, Марка товара, пробел, Артикул». Используем пункт контекстного меню «Дополнительно» (слева) для вызова «Дополнительных свойств класса». Удалим из нижнего списка элемент с {ID} и добавим вместо него 3 элемента, не забывая ввести пробел в поле «суффикс»:



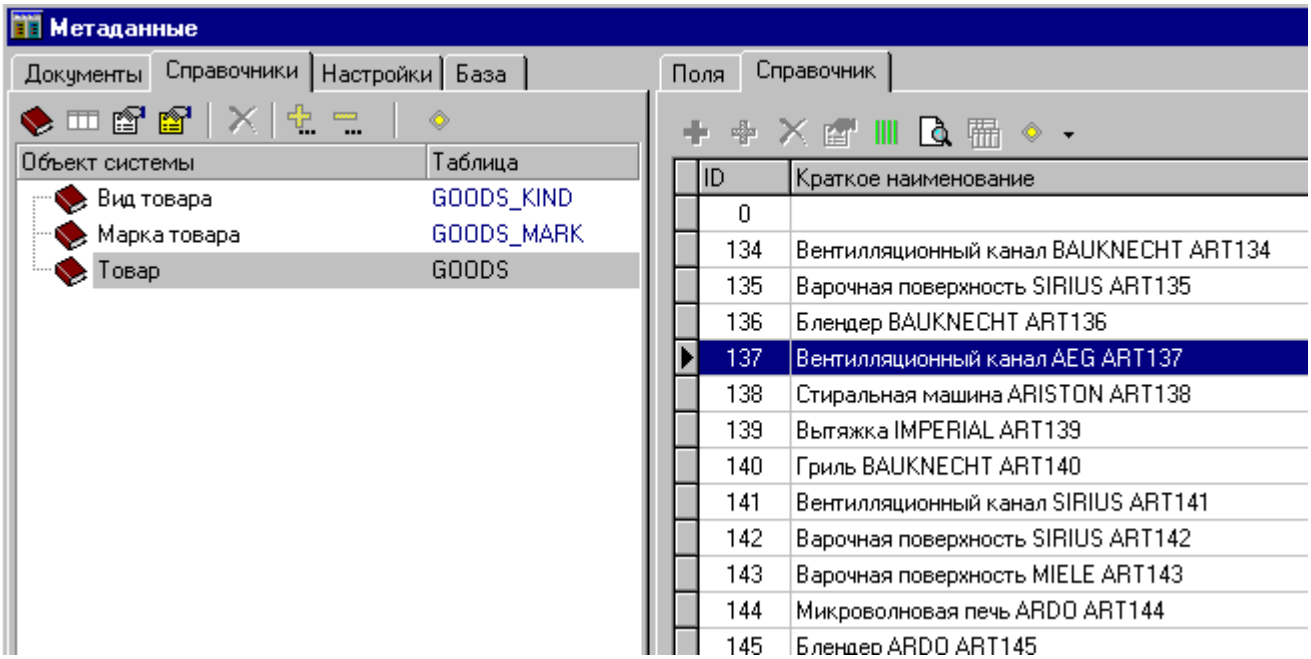
Сохраним этот вид форматирования.

Переключение между режимами отображения справочника

Теперь, просматривая справочник ***Товар*** в режим «Краткое наименование», мы увидим наименование товара, записанное в форме, удобной для печати. Для переключения справочника из режима отображения ***Атрибуты*** в режим отображения ***Краткое наименование***, используем пункт ***Отображать*** контекстного меню справочника:



Итак, перед нами ***краткие наименования*** товаров, искусственно собранные из ***Вида товара, Марки товара*** и ***Артикула***. Эти наименования мы сможем печатать в документах и использовать их для быстрого поиска товара по наименованию.



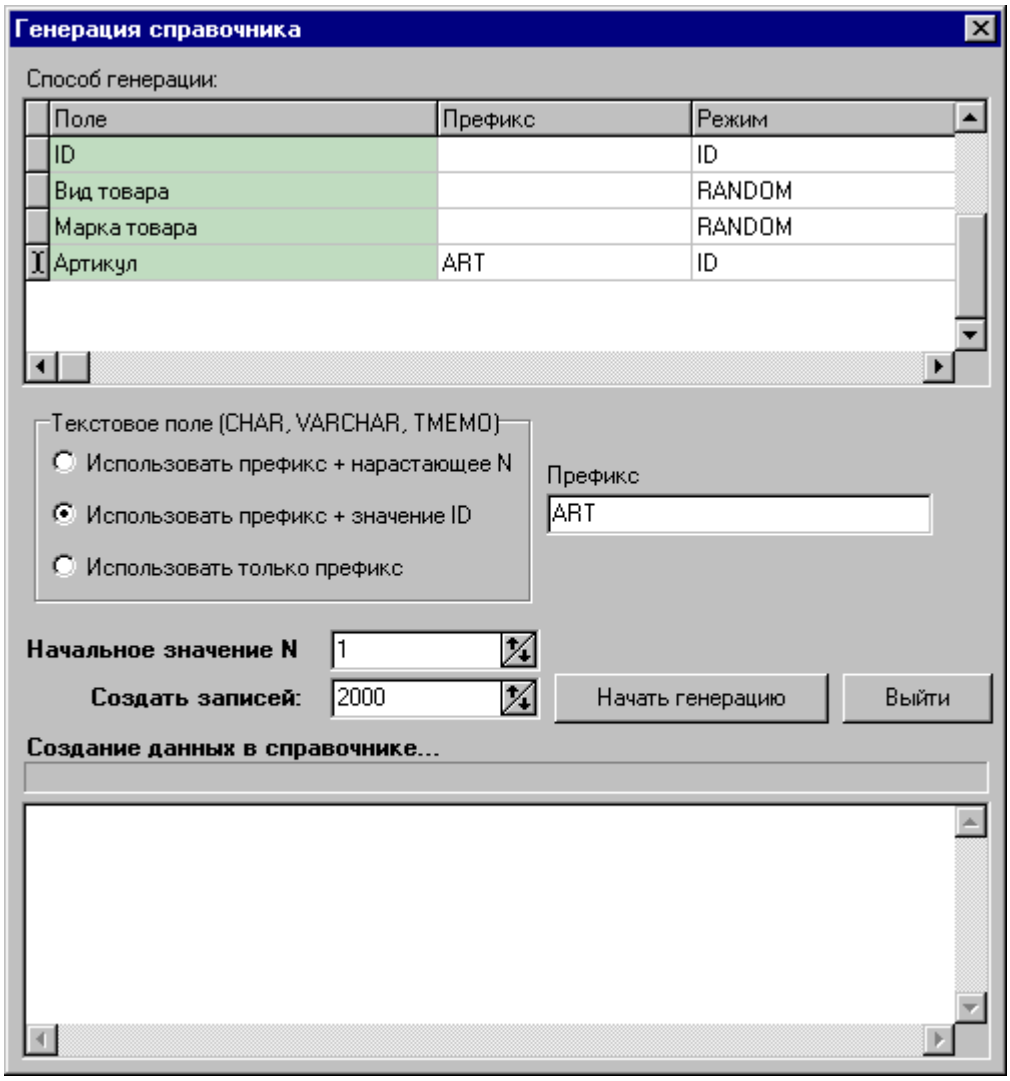
Сами же поля справочника (вид, марка, артикул) будем использовать, когда нам понадобится строгий финансовый анализ, например, анализ продаж товаров по видам или маркам. Их также удобно использовать, если нужно упорядочить или отфильтровать товары по маркам или по видам.

Генерируем 2000 товаров

Для того чтобы не отлаживать конфигурацию на пустых справочниках, мы рекомендуем заполнить справочники данными еще на стадии разработки. Сделав это, мы сможем заранее оценить быстродействие наших SQL-запросов и представить себе, как конфигурация поведет себя у заказчика в реальной работе.

Из бесед с заказчиком ***TechnoTrade*** мы выяснили, что общее ожидаемое количество различных товаров в справочнике - около двух тысяч. Всегда имеет смысл смоделировать ситуацию, максимально близкую к реальной. Поэтому мы создадим две тысячи записей в справочнике ***Товар***. Разумеется, вручную мы этого делать не будем.

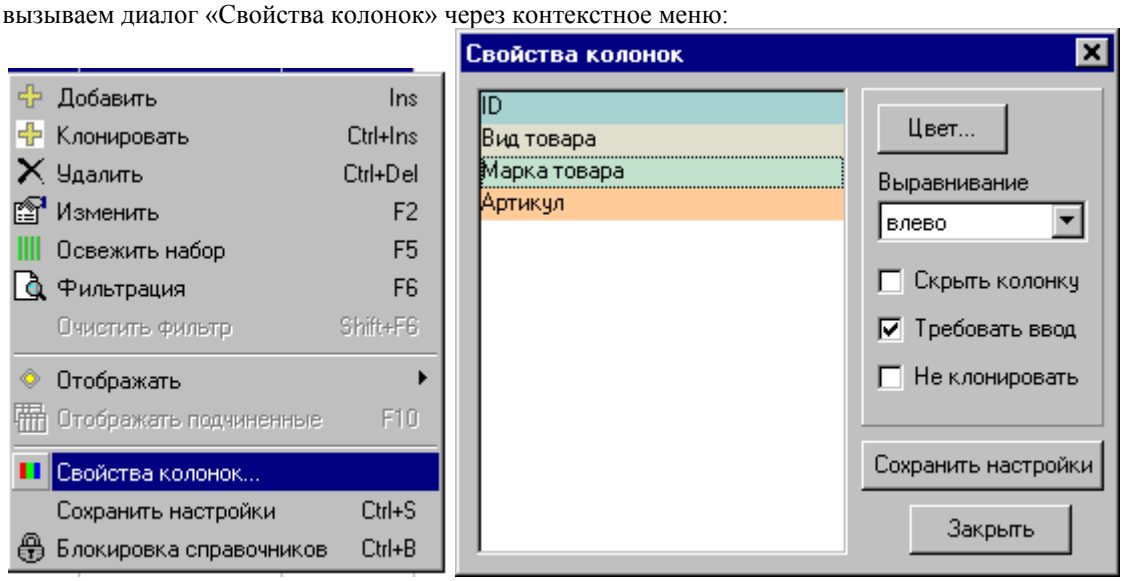
Для генерации нужно использовать пункт контекстного меню «Генерация» (слева). В появившемся окне изменим способ генерации поля Артикул, заменив префикс Артикул на фразу ART (для краткости), и будем использовать ID вместо нарастающего номера N. Также увеличим количество создаваемых записей до двух тысяч и нажмем кнопку ***Начать генерацию***.



Генерация справочника может занять несколько минут. Дождемся завершения этого процесса. В справочнике товаров будет создано 2000 уникальных записей. Нам этого вполне достаточно для отладки конфигурации. Выйдем из окна генерации.

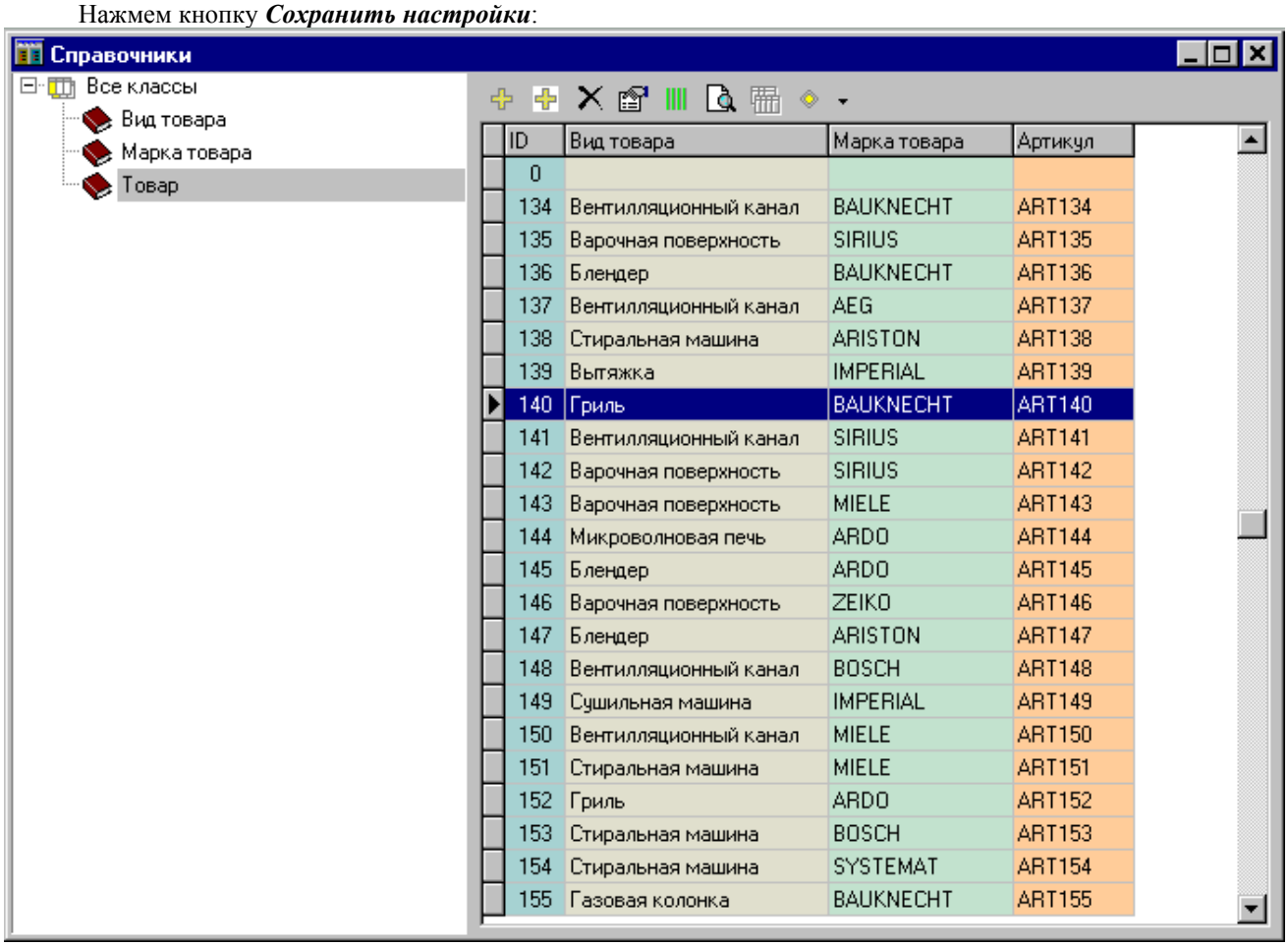
Настраиваем внешний вид справочника «Товар»

Настроим ширину колонок сетки, их цвета и другие свойства в справочнике «Товар». Ширина колонок настраивается с помощью мыши. Для этого нужно попасть курсором в промежуток между заголовками колонок и, удерживая левую кнопу в нажатом положении, растянуть ширину столбца на требуемую величину. Для сохранения настроек используем комбинацию клавиш Ctrl+S. Для изменения цветов и других свойств колонок сетки вызываем диалог «Свойства колонок» через контекстное меню:



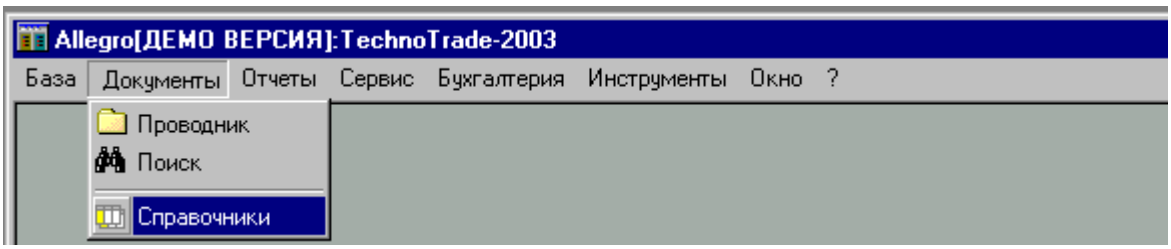
В диалоге, последовательно выбирая поля в левом списке, установим цвета колонок по вкусу. Также для полей ***Вид товара*** и ***Марка товара*** установим галочку «Требовать ввод» для того, чтобы при добавлении новых товаров программа требовала от пользователя обязательного заполнения этих полей.

Нажмем кнопку ***Сохранить настройки***:



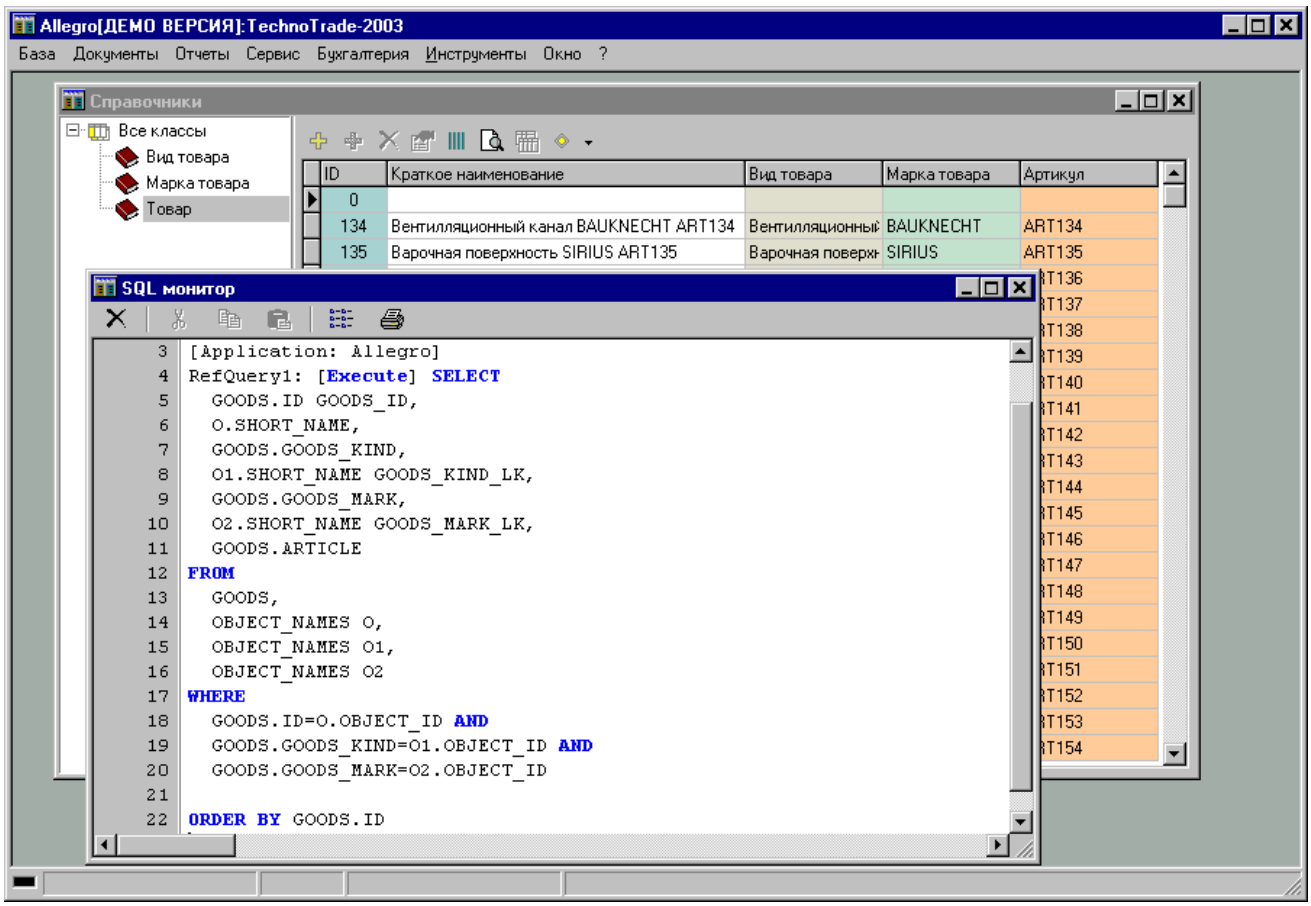
ID	Вид товара	Марка товара	Артикул
0			
134	Вентилляционный канал	BAUKNECHT	ART134
135	Варочная поверхность	SIRIUS	ART135
136	Блендер	BAUKNECHT	ART136
137	Вентилляционный канал	AEG	ART137
138	Стиральная машина	ARISTON	ART138
139	Вытяжка	IMPERIAL	ART139
140	Гриль	BAUKNECHT	ART140
141	Вентилляционный канал	SIRIUS	ART141
142	Варочная поверхность	SIRIUS	ART142
143	Варочная поверхность	MIELE	ART143
144	Микроволновая печь	ARDO	ART144
145	Блендер	ARDO	ART145
146	Варочная поверхность	ZEIKO	ART146
147	Блендер	ARISTON	ART147
148	Вентилляционный канал	BOSCH	ART148
149	Сушильная машина	IMPERIAL	ART149
150	Вентилляционный канал	MIELE	ART150
151	Стиральная машина	MIELE	ART151
152	Гриль	ARDO	ART152
153	Стиральная машина	BOSCH	ART153
154	Стиральная машина	SYSTEMAT	ART154
155	Газовая колонка	BAUKNECHT	ART155

Настраивать внешний вид справочников не обязательно в окне «Метаданные». Это можно делать и в окне «Справочники», вызываемом через пункт ***Документы/Справочники*** Главного Меню:



Переключим справочник в режим отображения «кратких наименований» и настроим поля еще и в этом режиме. Не забываем сохранять настройки при помощи Ctrl+S. Обращаем внимание на то, что цвета колонок не зависят от режима отображения справочника, а ширины колонок – зависят.

Щелкнув на заголовке любой колонки, мы можем ***упорядочить*** справочник по этой колонке. Повторный щелчок изменяет порядок на противоположный. При упорядочивании происходит новый SQL-запрос к серверу баз данных. Для того чтобы увидеть текст этого запроса можно включить SQL-монитор, используя пункт ***Инструменты/SQL-монитор*** Главного меню:



SQL-монитор помогает понять, как программа общается с сервером баз данных. В данном случае мы видим, что программа «собирает» справочник, объединяя несколько таблиц. Для отображения справочника нужно получить наименования ***Вида товара*** и ***Марки товара***. Для этого таблица GOODS объединяется с таблицей OBJECT_NAMES несколько раз, так как в таблице OBJECT_NAMES хранятся все краткие наименования всех объектов справочной системы. В данном случае в SQL-запрос возвращает нам еще и отформатированные краткие наименование самих товаров, которые тоже берутся из таблицы OBJECT_NAMES. Выражение `ORDER BY O2.SHORT_NAME ASC` упорядочивает набор по полю O2.SHORT_NAME в возрастающем алфавитном порядке. В данном случае это поле содержит наименования ***Марок товара***. Рекомендуем скопировать текст этого SQL-запроса (начиная со слова SELECT) через буфер обмена Windows (Ctrl+C, Ctrl+V) в окно ISQL и выполнить его. Мы получим тот же результирующий набор данных.

Создаем справочник контрагентов.

Нам понадобятся справочники для хранения поставщиков и покупателей, а ткже их реквизитов.

Из бесед с заказчиком удалось выяснить, что компания ***TechnoTrade*** не только торгует бытовой техникой на оптовом рынке, но и осуществляет иногда продажу техники частным лицам. При этом атрибутами частных лиц являются Фамилия, Имя и, иногда, Отчество. Все их все можно хранить в одном поле. Еще желательно хранить номера телефонов и адреса покупателей. Для фирм нужно хранить целую кучу атрибутов (Полное название, коды ОКПО, адреса, банковские счета и прочее) для того, чтобы печатать все это в счетах-фактурах. Причем на практике возможны сложные взаимозачеты, когда товар отгружается какой-то одной фирме в счет задолженности перед какой-то другой фирмой.

Проанализировав всю эту информацию, мы пришли к выводу, что ***любой*** контрагент (как фирма, так и частное лицо) имеет следующие атрибуты:

Имя фирмы (или Ф.И.О для частных лиц).,
Физический адрес,
Телефоны

Фирмы, помимо этих атрибутов, еще имеют:

Скидку (в процентах), ***Отсрочку платежа*** (в днях),
Признак «наша фирма» (для самой компании ***TechnoTrade***)
Полное название (в соответствие с законодательством),
Расчетный счет, ИНН, Банк, Город (банка), Корр.счет, БИК,
ОКОНХ, ОКПО, КПП, Юридический адрес,
№ свидетельства о регистрации, Дату регистрации, Директора, Главного бухгалтера

Поэтому мы решили создать класс «Контрагенты» и два дочерних класса: «Фирмы» и «Частные лица».

Вызовем окно «Метаданные» с помощью пункта ***Инструменты/Метаданные*** Главного меню. Выберем закладку «Справочники» и добавим новый базовый класс ***Контрагенты***, таблица CONTRAGENT.

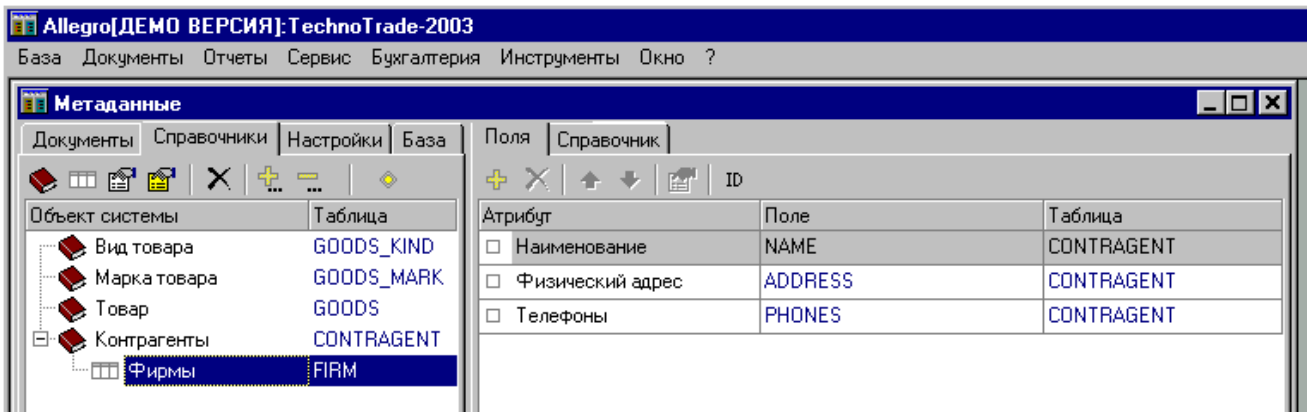
Увеличим длину поля NAME до 50 символов.
Добавим еще 2 поля:

Атрибут	**Поле**	**Тип**	**Обязательный атрибут**
Физический адрес	ADDRESS	VARCHAR(250)	да
Телефоны	PHONES	VARCHAR(50)	да

Справочник ***Контрагенты*** готов.

Создание дочерних классов. Наследование атрибутов.

Теперь создадим дочерний (к классу ***Контрагенты***) класс ***Фирмы***, таблица FIRM.

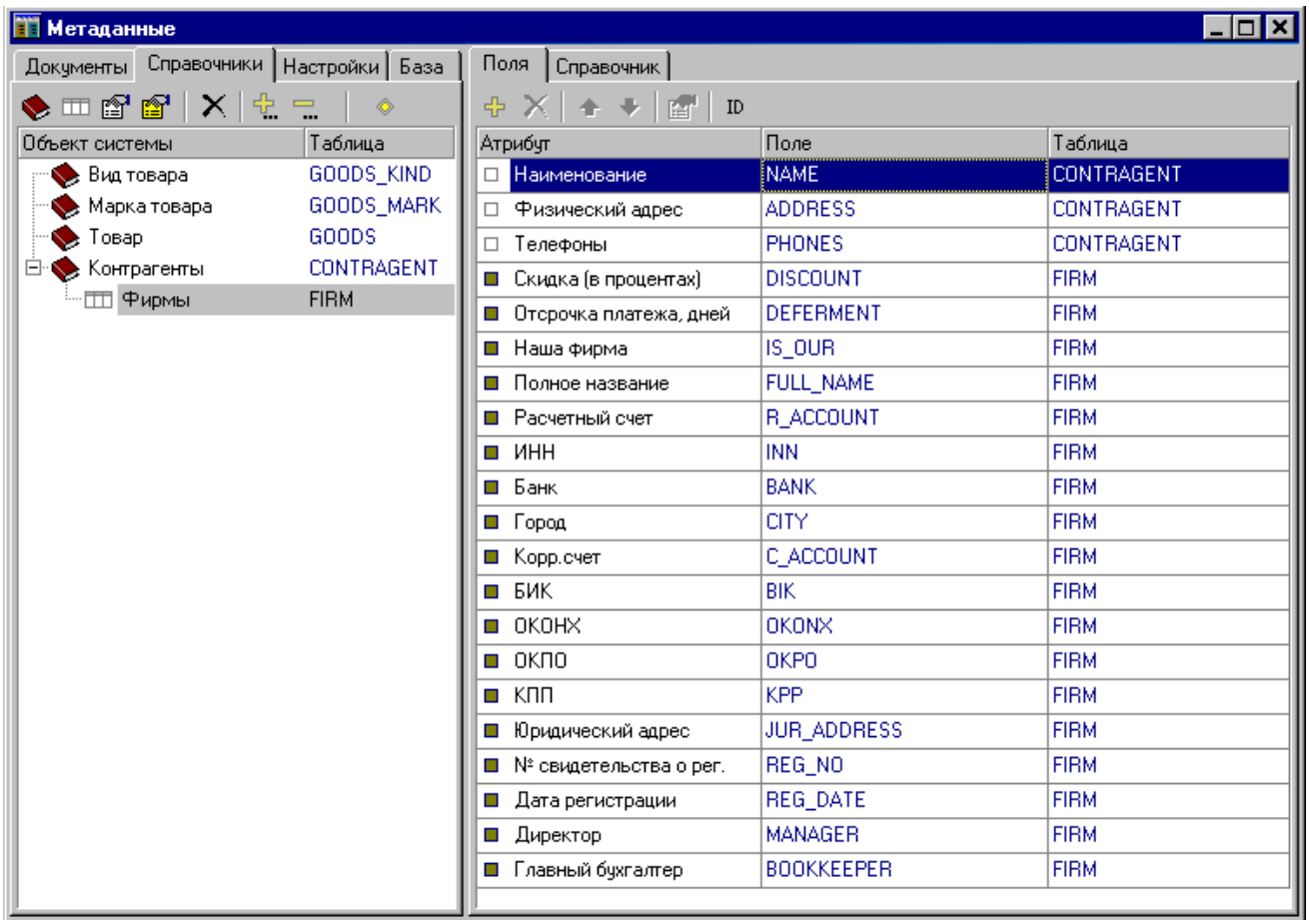


Мы видим, что класс ***Фирмы*** унаследовал все атрибуты класса ***Контрагенты.***

Теперь добавим дополнительные поля в класс ***Фирмы***.

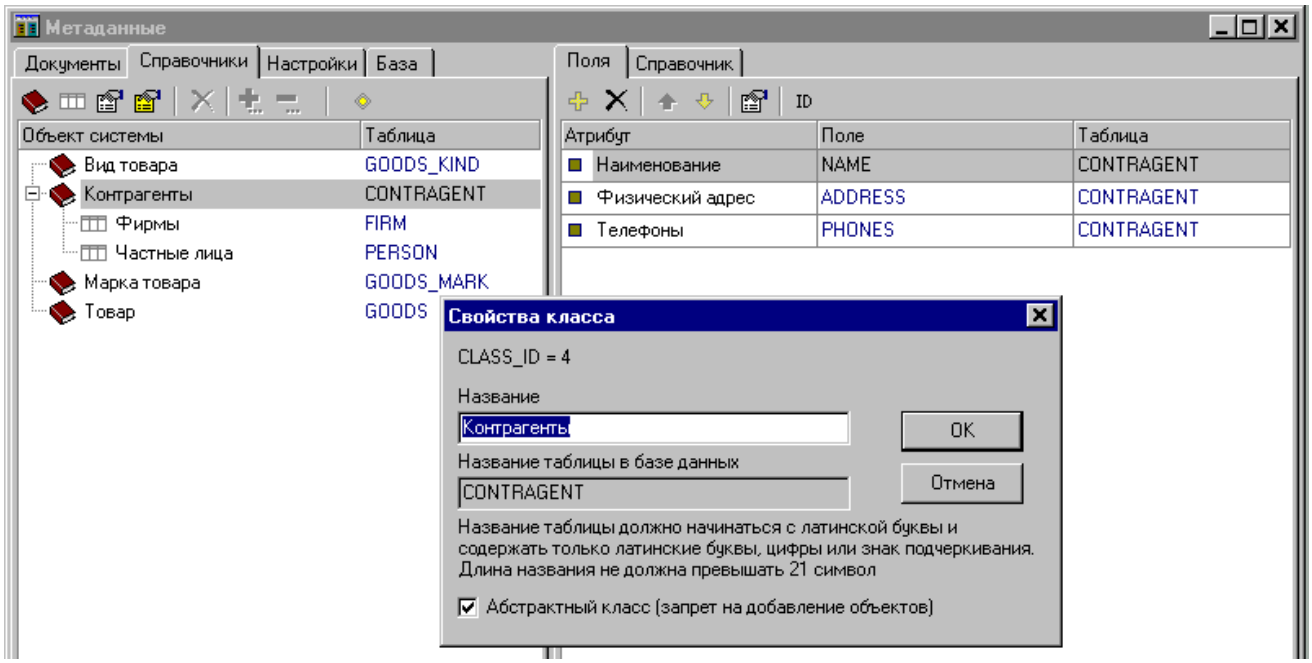
Атрибут	**Поле**	**Тип**	**Обязательный атрибут**
Скидка (в процентах)	DISCOUNT	DECIMAL(5,2)	нет
Отсрочка платежа, дней	DEFERMENT	INTEGER	да
Наша фирма	IS_OUR	TBOOLEAN	да
Полное название	FULL_NAME	VARCHAR(100)	да
Расчетный счет	R_ACCOUNT	VARCHAR(40)	да
ИНН	INN	VARCHAR(30)	да
Банк	BANK	VARCHAR(100)	да
Город	CITY	VARCHAR(40)	да
Корр.счет	C_ACCOUNT	VARCHAR(40)	нет
БИК	BIK	VARCHAR(10)	нет
ОКОНХ	OKONX	VARCHAR(30)	нет
ОКПО	OKPO	VARCHAR(20)	нет
КПП	KPP	VARCHAR(20)	нет
Юридический адрес	JUR_ADDRESS	VARCHAR(250)	да
№ свидетельства о рег.	REG_NO	VARCHAR(20)	нет
Дата регистрации	REG_DATE	DATE	нет
Директор	MANAGER	VARCHAR(30)	нет
Главный бухгалтер	BOOKKEEPER	VARCHAR(30)	нет

В результате класс ***Фирмы*** имеет 3 унаследованных атрибута и 18 собственных:



Создадим еще один дочерний класс к классу ***Контрагенты*** и назовем его ***Частные лица***, таблица PERSON. Новых полей в класс ***Частные лица*** мы добавлять не будем.

Теперь объявим базовый класс Контрагенты «абстрактным» классом, вызвав его «Свойства» и поставив соответствующую галочку. Этим мы запретим пользователям добавлять записи непосредственно в базовый класс:



Для сохранения новых свойств класса нажимаем кнопку ***OK***.

Справочная система. Подведем итоги.

Создадим архивную копию базы данных с помощью пункта ***Инструменты/Архивация базы*** Главного меню. Затем произведем генерацию 100 фирм и 100 частных лиц. Настроим цвета сеток справочников по вкусу. Замечаем, что общие для разных классов атрибуты имеют одинаковые цвета. Настроим также форматирование кратких наименований для справочников ***Контрагенты***, ***Фирмы*** и ***Частные лица***.

Подводя итоги, можно сказать, что мы научились создавать справочники, добавлять в них поля и данные, использовать наследование в случае, когда нам недостаточно простого справочника в виде одной таблицы для описания разнообразных объектов реального мира.

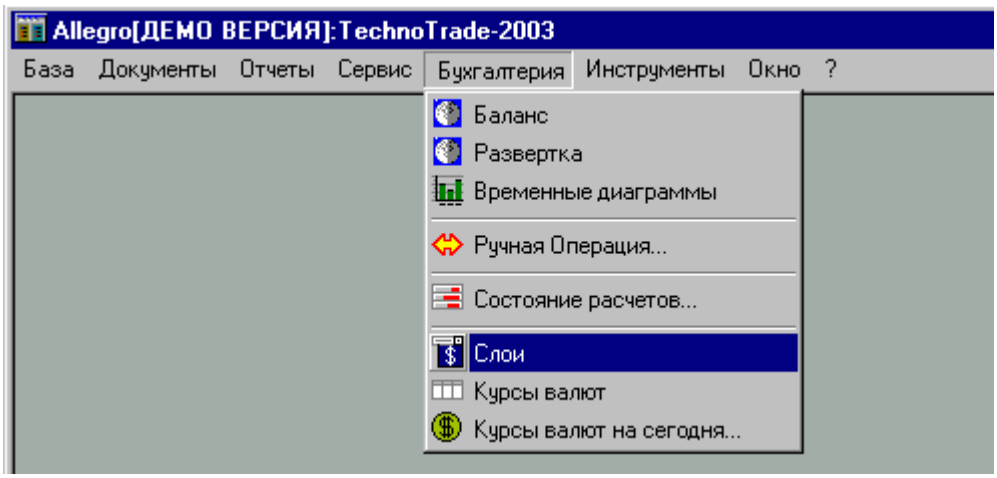
Мы создали все необходимые справочники для складской конфигурации TechnoTrade. Это потребовало создания в общей сложности шести таблиц.

Пора переходить к следующему шагу проектирования - созданию структуры баланса компании TechnoTrade.

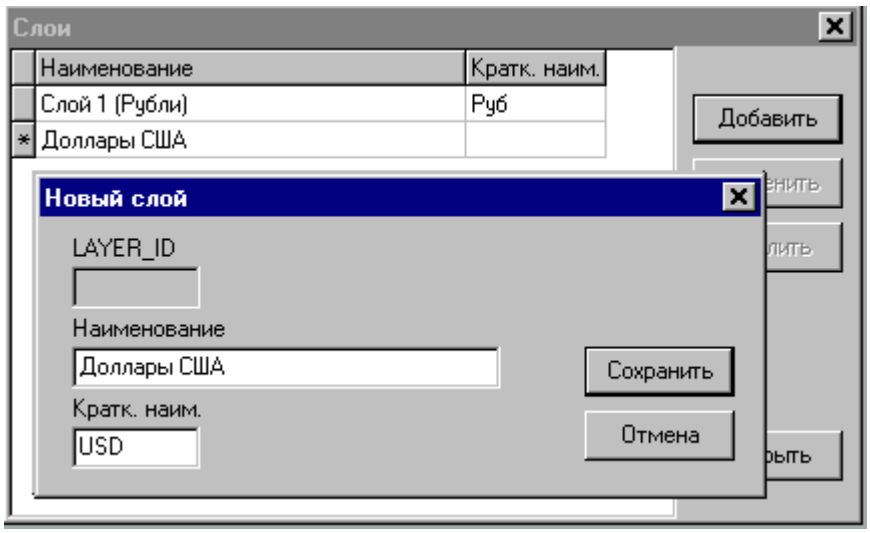
Глава 3. Создание системы бухгалтерских счетов

Создаем новые валютные слои.

Из бесед с заказчиком мы выяснили, что основная масса товаров учитывается в валюте USD. Платежи в основном осуществляются в рублях. Счета поставщиков ведутся часть в EURO, часть в USD, счета покупателей также ведутся в разных валютах. Поэтому первое, что мы сделаем, это заведем два новых валютных слоя. Для этого воспользуемся пунктом ***Бухгалтерия/Слои*** Главного меню:



В окне диалога «Слои», нажимая кнопку ***Добавить***, введем две новые валюты: Доллары США и ЕВРО.



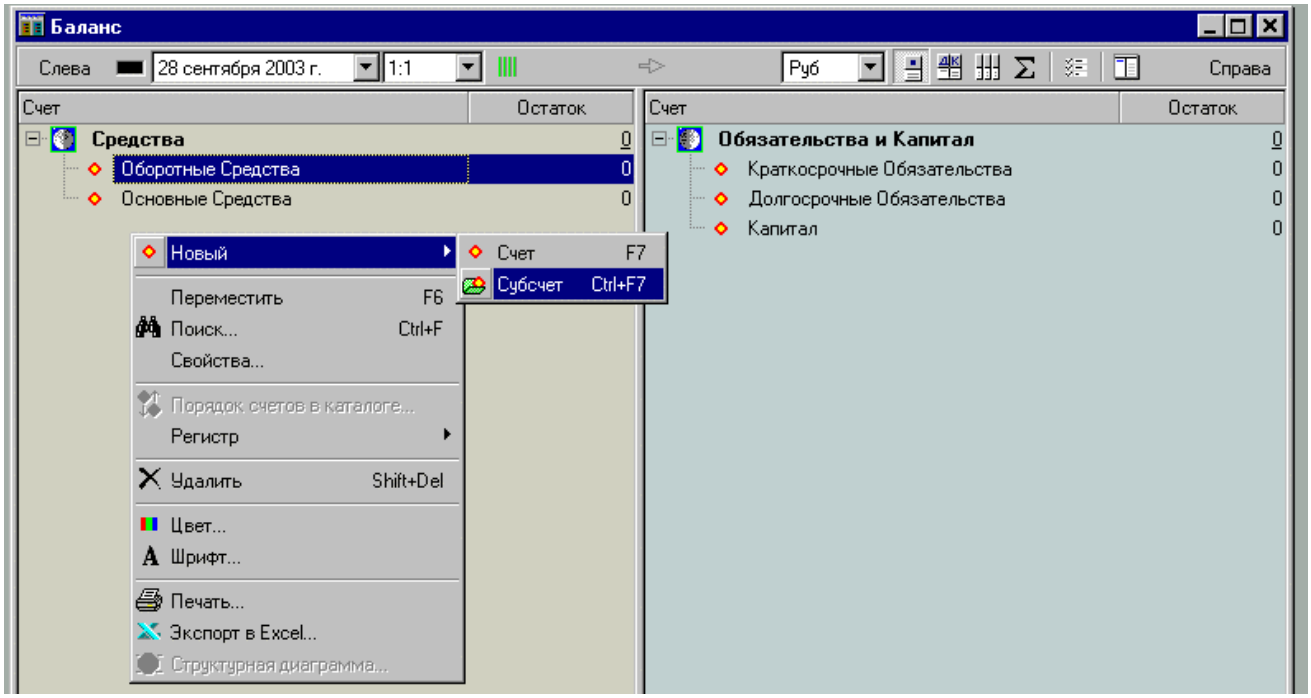
Мы будем вести счета контрагентов в тех валютах, в которых закреплены отношения сторон, независимо от того, в каких валютах производятся платежи. Для конвертации валют при платежах и зачетах будем использовать специальный счет ***Конвертация***.

Создаем счета «Денежных средств»

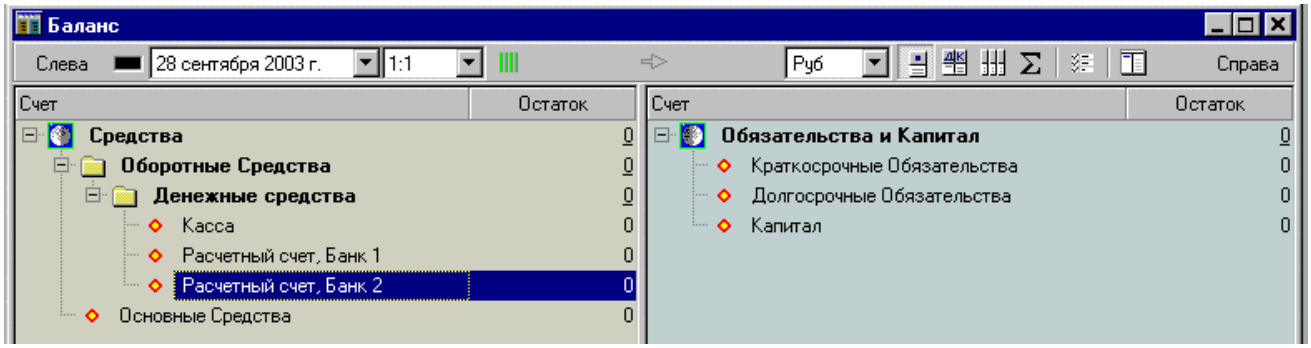
Из бесед с заказчиком выяснилось, что компания ведет несколько счетов ***Денежных средств***:

- ***Касса***
- ***Расчетный счет, Банк 1***
- ***Расчетный счет, Банк 2***

Это простые счета баланса, давайте их создадим.
Вызываем окно «Баланс», используя пункт ***Бухгалтерия/Баланс*** Главного меню.
Выберем счет ***Оборотные средства*** в левой стороне баланса и вызовем контекстное меню правой кнопкой мыши:



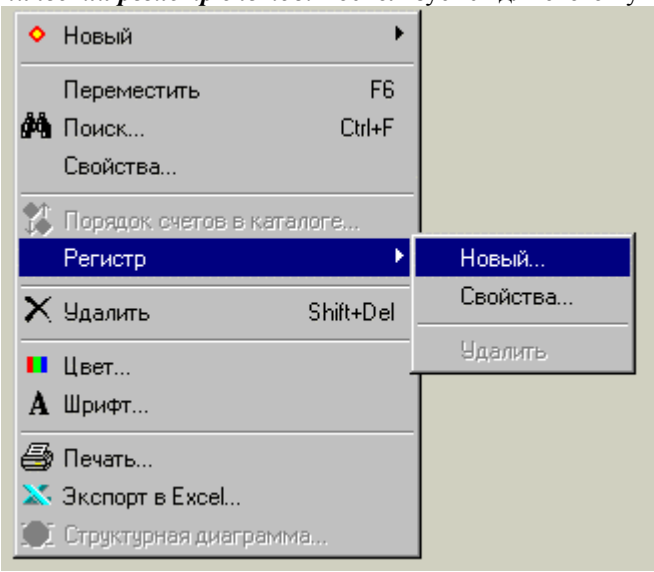
Создадим новый субсчет ***Денежные средства***, а в нем субсчета:
Касса
Расчетный счет, Банк 1,
Расчетный счет, Банк 2



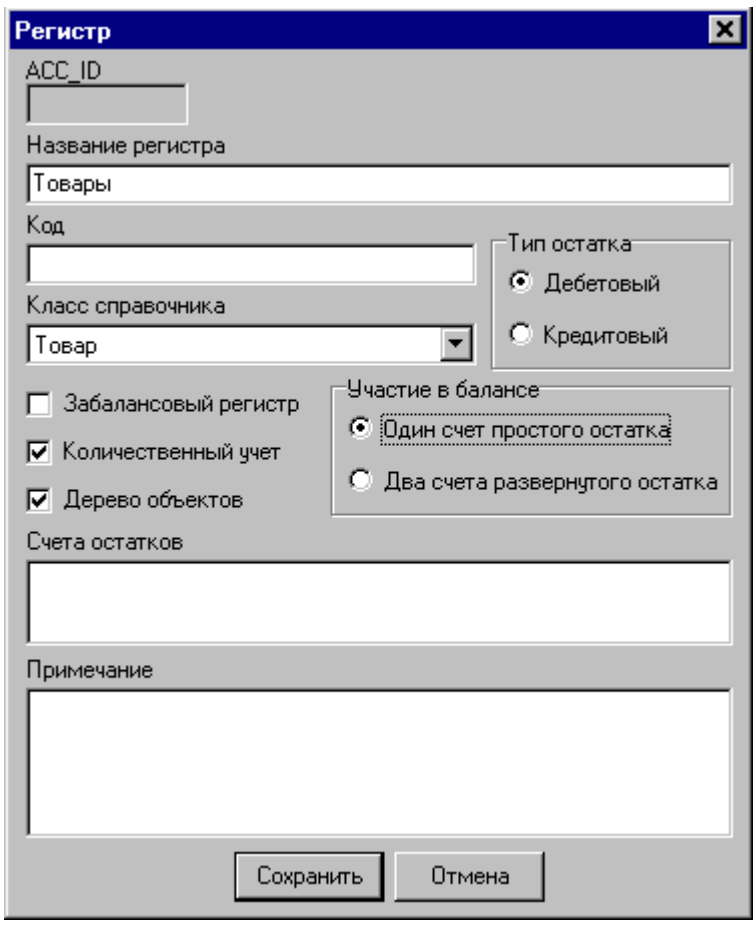
Создавать счета легко с помощью клавиши F7, а субсчета с помощью сочетания клавиш Ctrl+F7. Программа позволяет создавать субсчета любой степени вложенности.

Создаем аналитический регистр «Товары»

Важными счетами оборотных средств являются счета товарных запасов, отражающие стоимость товаров, имеющихся в данный момент на складах компании. Для создания счетов товарных запасов нам нужно создать ***аналитический регистр счетов***. Воспользуемся для этого пунктом контекстного меню ***Регистр/Новый:***

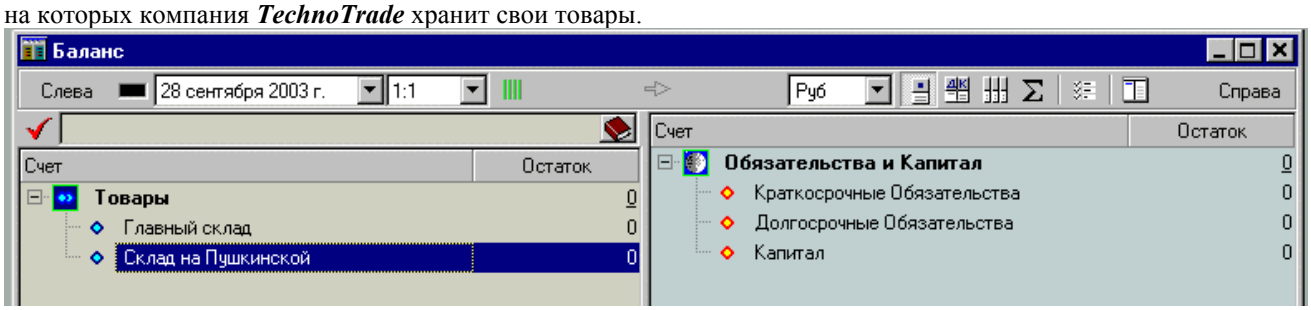


В появившемся диалоге введем название регистра, укажем класс справочника ***Товар***, Поставим галочку ***Количественный учет*** и укажем «Участие в балансе»: ***Один счет простого остатка***.

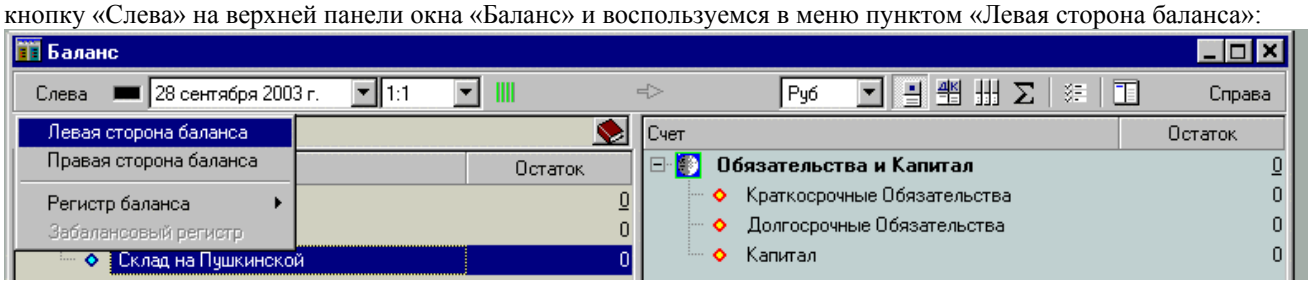


Нажмем кнопку ***Сохранить***.

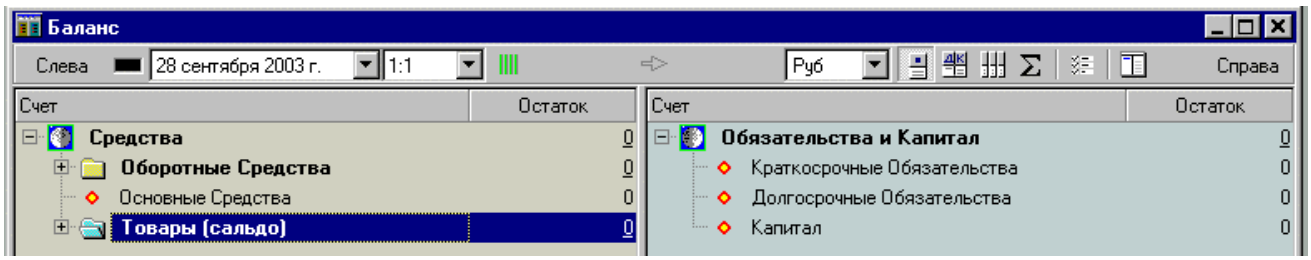
На одной из панелей окна «Баланс» отобразится созданный нами регистр. Выберем корневой счет в регистре и создадим в нем два субсчета: ***Главный склад*** и ***Склад на Пушкинской***. Это фактически два склада, на которых компания ***TechnoTrade*** хранит свои товары.



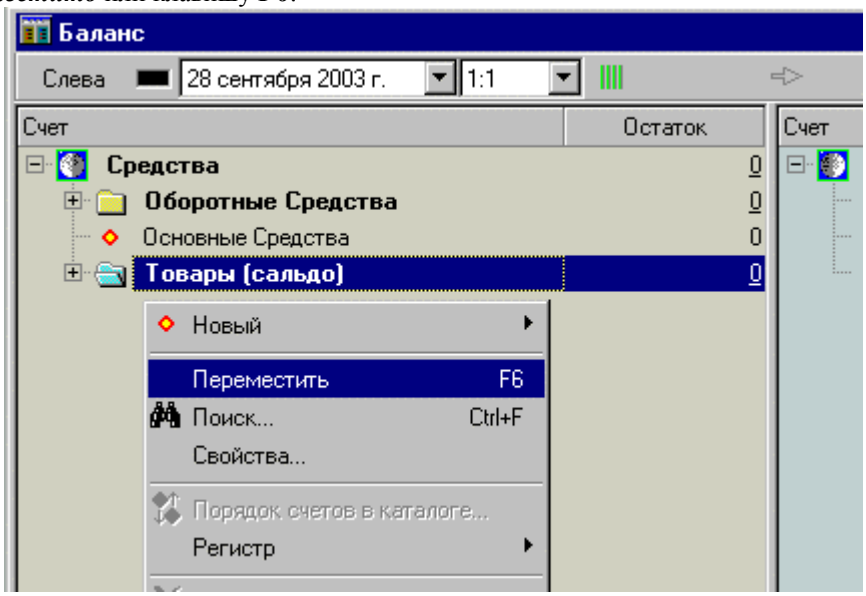
Отобразим в левой стороне окна «Баланс» левую сторону бухгалтерского баланса. Для этого нажмем кнопку «Слева» на верхней панели окна «Баланс» и воспользуемся в меню пунктом «Левая сторона баланса»:



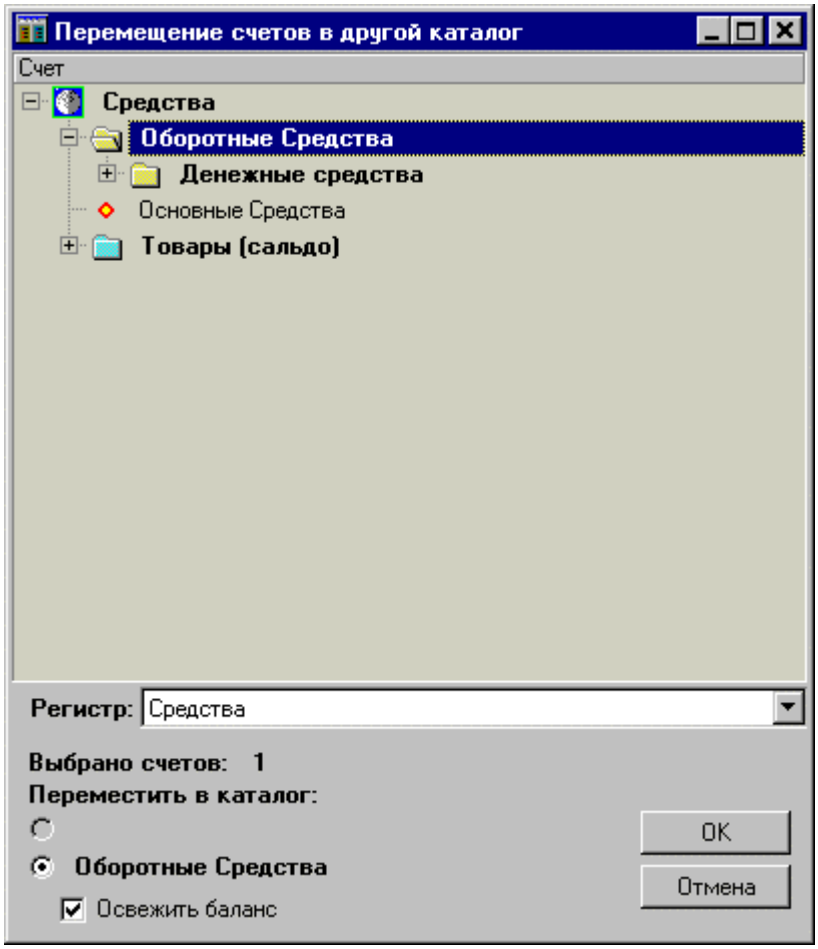
Мы видим, что в левой стороне бухгалтерского баланса возник новый счет ***Товары (сальдо)***. Это и есть то «Участие в балансе», которое принимает созданный нами регистр ***Товары***. Это так называемый счет простого отстатка. Его невозможно удалить из баланса, пока мы не удалим регистр ***Товары***. Однако этот счет можно перемещать и переименовывать.



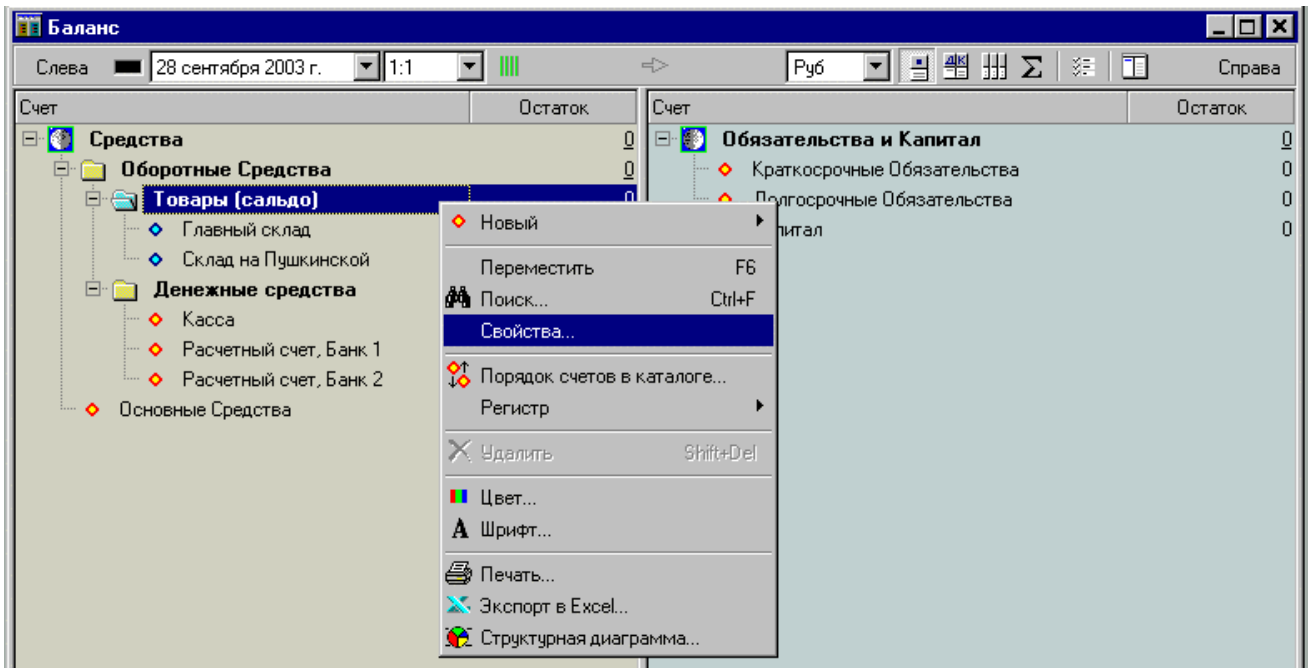
Нам следует переместить этот счет внутрь каталога ***Оборотные средства*** и переименовать его в ***Товарные запасы***. Для перемещения счета нужно его сначала выбрать, а затем использовать пункт контекстного меню ***Переместить*** или клавишу ***F6***:



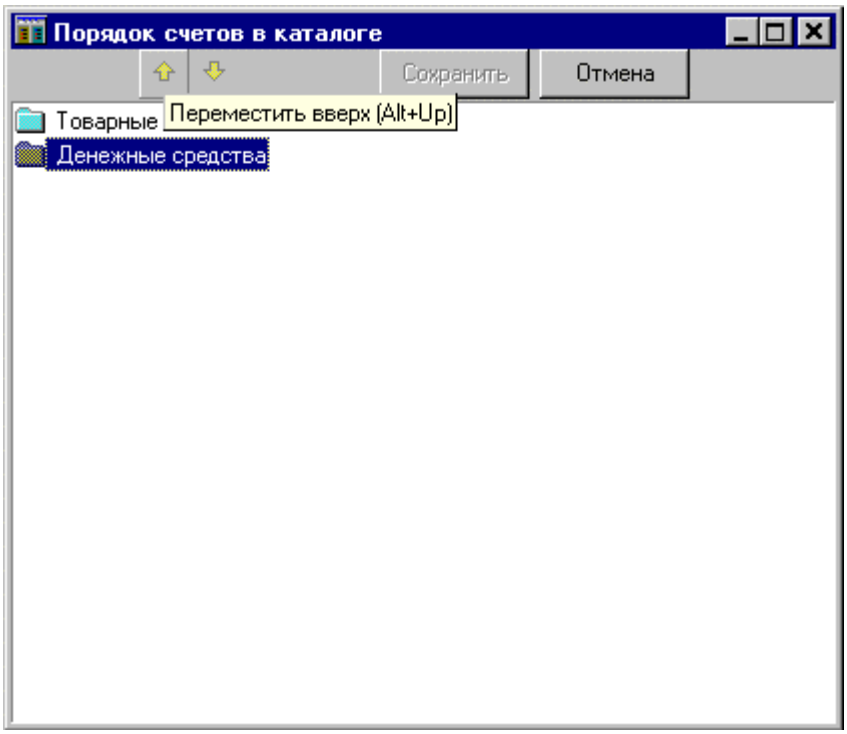
Появится окно, в котором нужно указать, куда перемещать счет. Выберем ***Оборотные средства*** и нажмем ***ОК***:



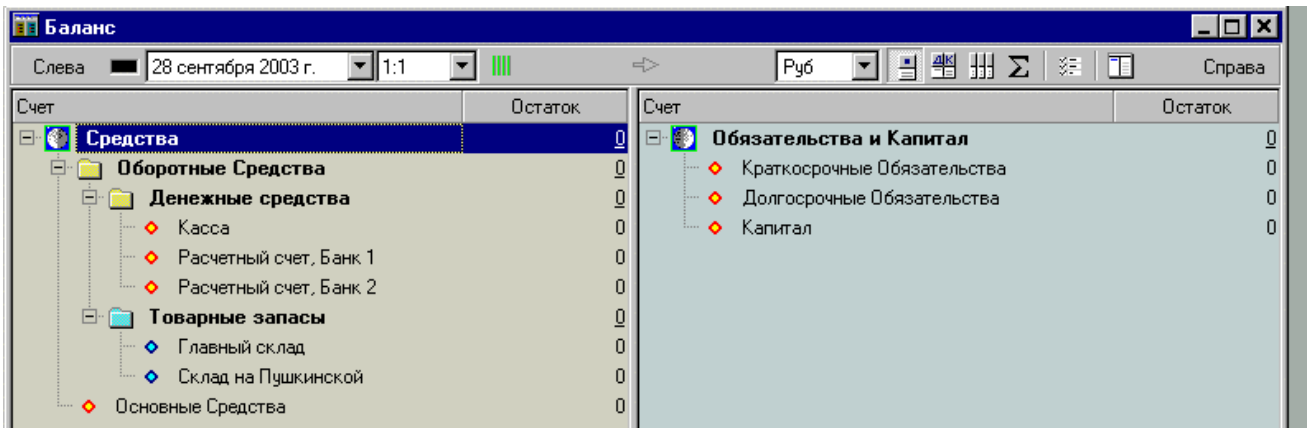
Теперь развернем каталог ***Оборотные средства***. Переименуем счет ***Товары (сальдо)*** в ***Товарные запасы***. Для этого выберем этот счет и используем пункт контекстного меню «Свойства»:



После переименования счета нам осталось изменить порядок счетов в каталоге ***Оборотные средства***. Дело в том, что принято располагать счета в порядке убывания ликвидности активов. Так как ***Денежные средства*** наиболее ликвидны, их счета должны располагаться на первом месте. Для перестановки счетов в каталоге следует выбрать счет-каталог, в данном случае это ***Оборотные средства*** и использовать пункт контекстного меню ***Порядок счетов в каталоге***. В появившемся диалоге с помощью кнопок со стрелками можно изменять порядок счетов в каталоге как угодно:



Переместим ***Денежные средства*** вверх и нажмем кнопку ***Сохранить***.
На данный момент окно «Баланс» выглядит так:



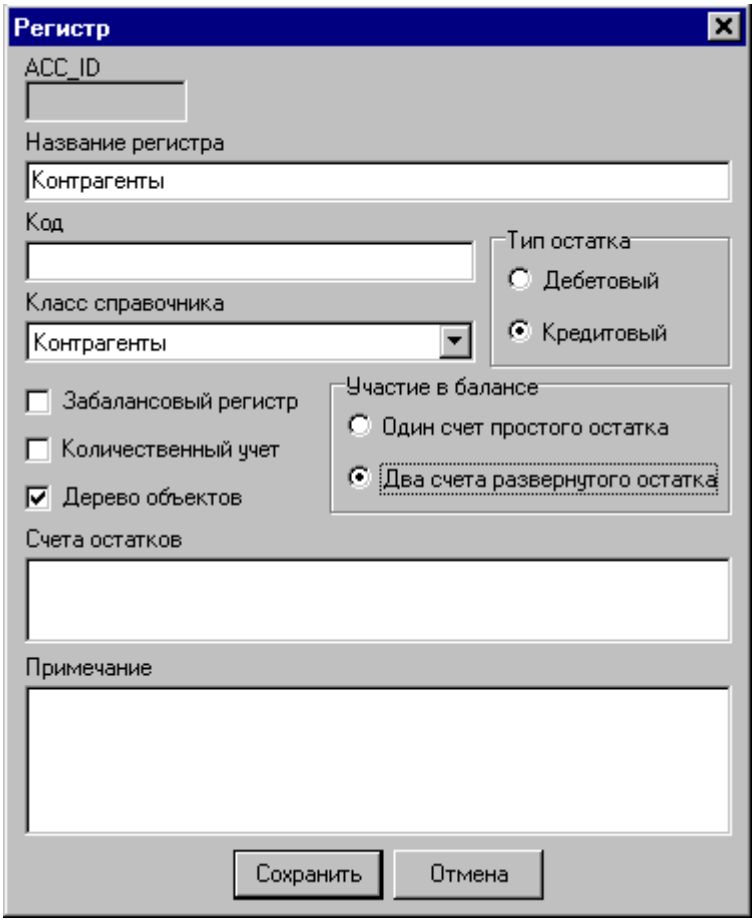
Аналитический регистр Контрагенты. Развернутые остатки.

Компания ведет счета контрагентов (раздельно поставщиков и покупателей). При этом один и тот же контрагент часто выступает как в качестве поставщика, так и в качестве покупателя в разных сделках. И хотелось бы видеть эти отношения как раздельно, так и в виде конечного баланса по каждому контрагенту.

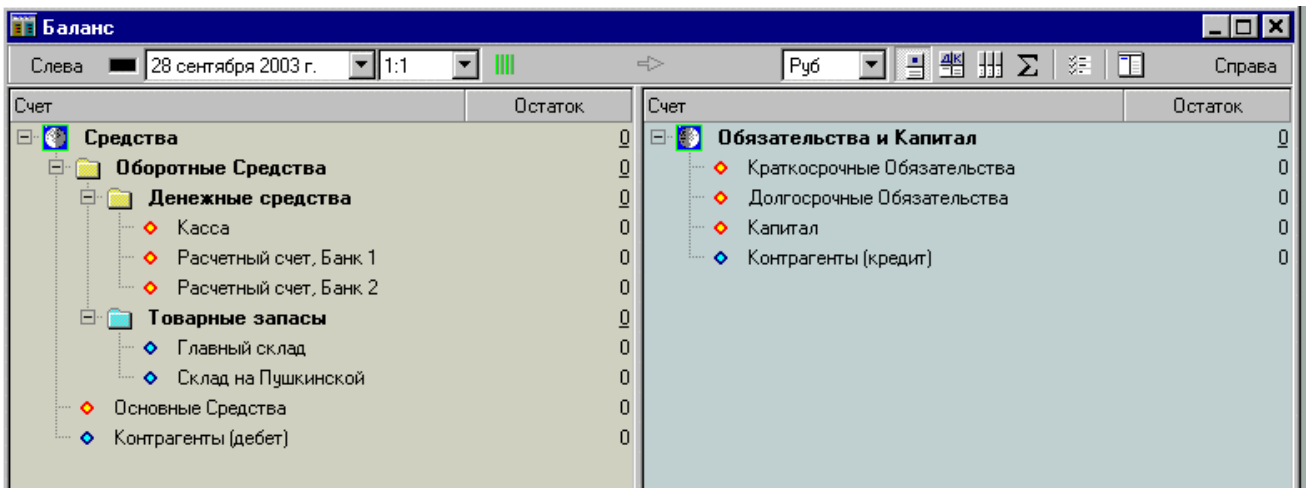
В принципе любой контрагент может в какой-то момент становиться кредитором, а в какой-то другой момент времени, наоборот, дебитором. В балансе компании принято отражать сумму всей дебиторской задолженности слева (в ***Оборотных Средствах***), а сумму всей кредиторской задолженности – справа (в ***Краткосрочных Обязательствах***). Поэтому мы создадим аналитический регистр счетов ***Контрагенты*** с двумя развернутыми остатками (сальдо), которые переименуем ***в Счета дебиторов*** и ***Счета Кредиторов*** и разместим их, соответственно, в левой и в правой стороне баланса. Внутри регистра Контрагенты создадим два счета:

Поставщики и ***Покупатели***. Это позволит нам разделить финансовые отношения с одним и тем же контрагентом, если он выступает в разных ролях и не допускать автоматического встречного взаимозачета сумм дебиторских и кредиторских задолженностей.

Создадим регистр «Контрагенты» с помощью пункта контекстного меню ***Регистр/Новый***. Укажем, что используется ***два счета развернутого остатка*** для участия в балансе, тип остатка регистра пусть будет ***кредитовый***. Регистр должен быть привязан к базовому классу ***Контрагенты***. Тогда в качестве контрагентов смогут выступать объекты как из справочника ***Фирмы***, так и из справочника ***Частные лица***.



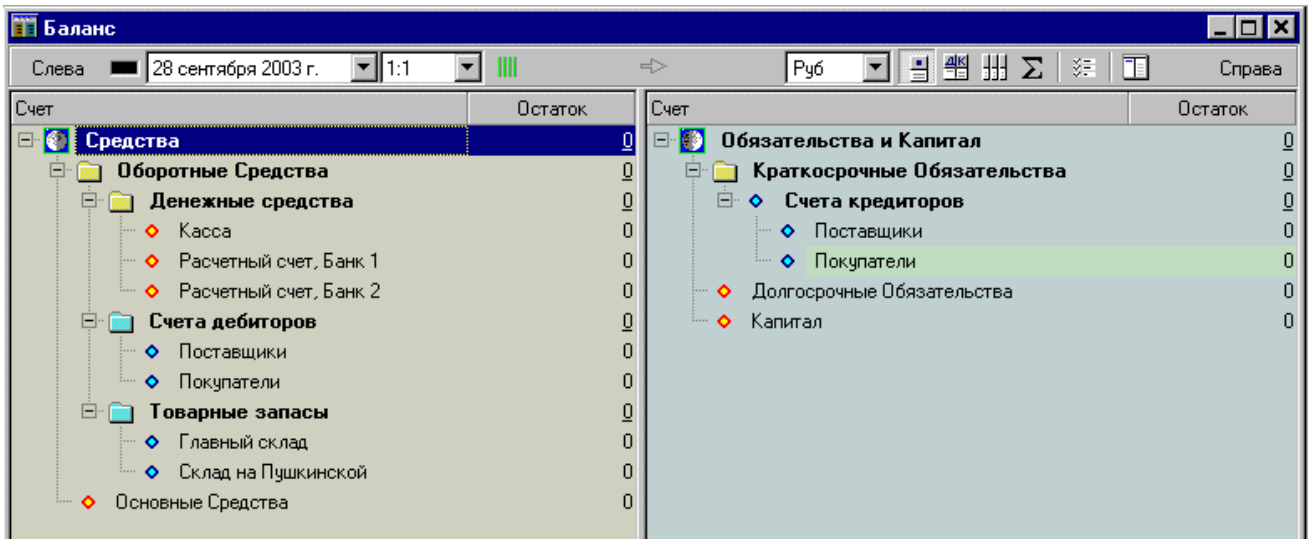
Откроем обе стороны бухгалтерского баланса компании в окне «Баланс»:



Переименуем левый счет развернутого остатка ***Контрагенты (дебет)*** в ***Счета дебиторов*** и переместим его внутрь ***Оборотных средств***, расположив непосредственно под ***Денежными средствами***. Переименуем

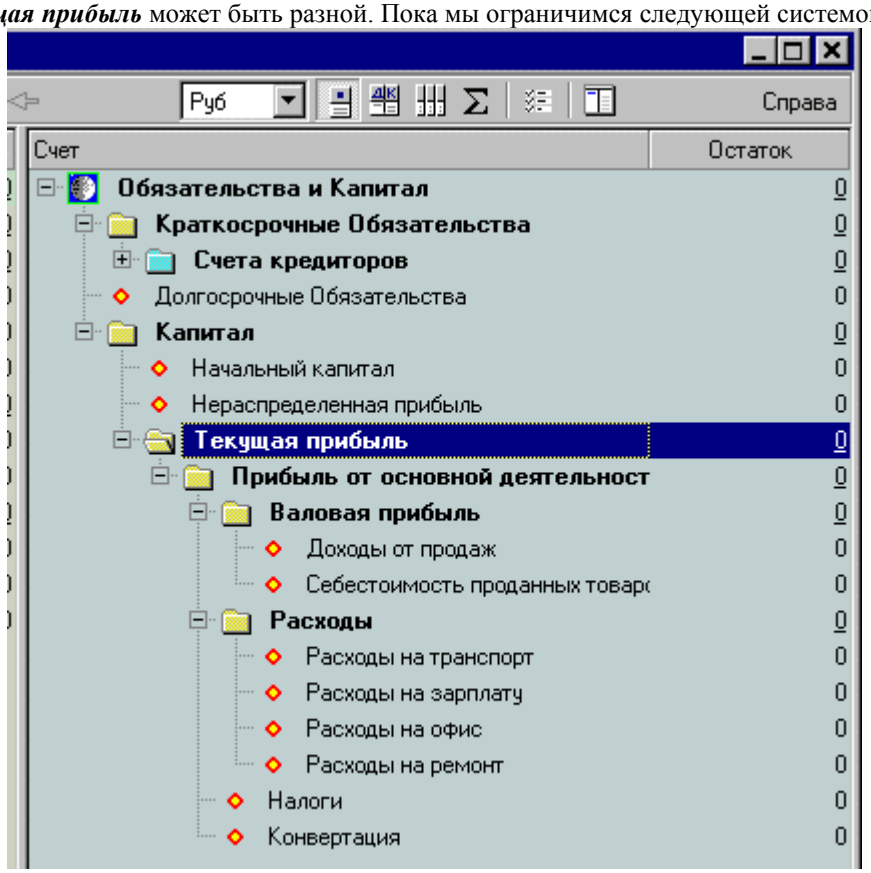
правый счет развернутого остатка ***Контрагенты (кредит)*** в ***Счета кредиторов*** и переместим внутрь счета ***Краткосрочные Обязательства***.

Теперь создадим субсчета ***Поставщики*** и ***Покупатели*** в ***Счетах кредиторов***. Обратим внимание, что два одноименных счета возникли в каталоге ***Счета Дебиторов***. Программа позволяет создавать субсчета как непосредственно в регистре, так и в счетах его остатков. В конечном итоге все равно получаются субсчета, принадлежащие регистру. В результате всех наших действий окно «Баланс» выглядит так:



Создаем счета Капитала и Текущей прибыли.

Нам осталось создать счета Капитала. Любая компания имеет ***Начальный капитал, Нераспределенную прибыль*** и ***Текущую прибыль***. Текущая прибыль, как правило, содержит различные счета доходов и расходов, которые закрываются в конце учетного периода в счет ***Нераспределенной прибыли***. Структура каталога ***Текущая прибыль*** может быть разной. Пока мы ограничимся следующей системой счетов:



В дальнейшем, возможно, заказчик самостоятельно добавит в эту структуру какие-то новые счета.

Глава 4. Создание метаданных документа «Поступление на склад»

Создаем тип документа «Поступление на склад»

Для упрощения системы все виды поступлений на склад мы объединим в один тип документа. Для того чтобы отличать между собой простые приобретения товара за наличные, ввод начальных остатков товара и поступления от поставщиков, мы добавим в документ признак ***вида документа*** и ссылку ***на кредитуемый счет***.

В беседах с заказчиком выяснилось, что счета обязательств перед разными поставщиками ведутся ***в разных валютах***, в зависимости от имеющихся договоренностей. Поэтому мы будем также использовать поле ***валюта документа***, подразумевая под нею валюту самой сделки. Именно в этой валюте будет производиться начисление обязательств, возникающих перед поставщиком в связи с поступлением товара на склад компании **TechnoTrade**.

Обязательным условием заказчика было также хранение в документе рублевой информации для официальной отчетности. В частности, каждая позиция должна иметь, помимо цены сделки, также рублевую цену, сумму без НДС и сумму с НДС, связанную с валютой поставки по курсу, который в каждом документе может быть свой. Следовательно, нам еще понадобится поле ***курса валюты документа***.

В части ведения учета стоимости товаров на складе наш заказчик склоняется к тому, чтобы ***стоимость всех товарных запасов измерять в долларах США***, так как это удобно для оперативного учета. Это означает, что нам потребуется в каждом документе хранить еще и ***курс доллара США***.

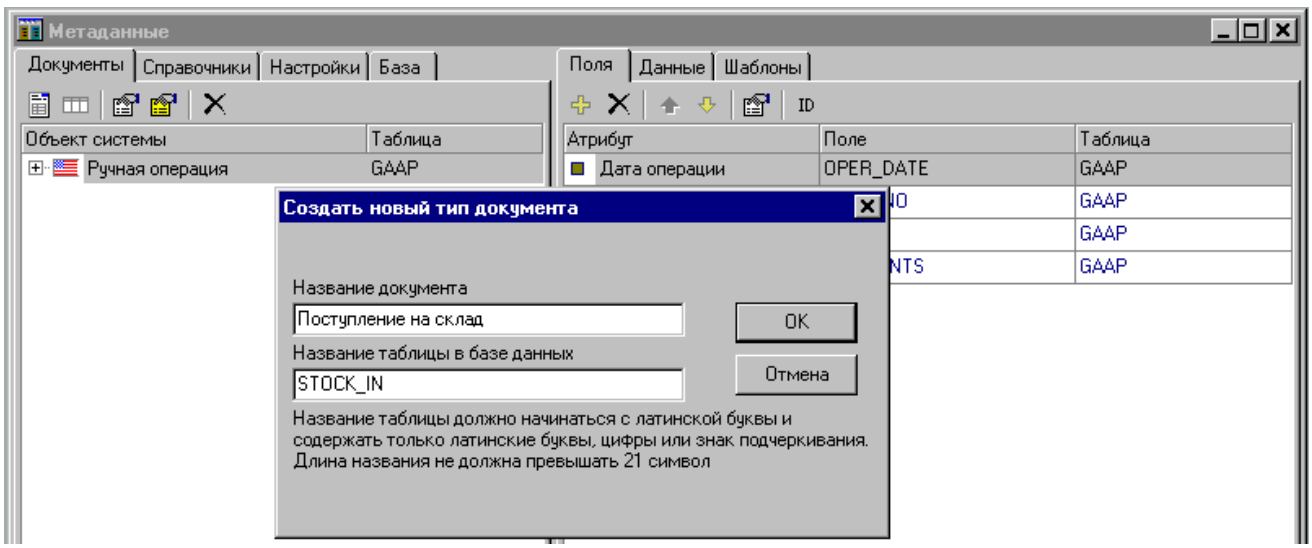
Еще наш заказчик утверждает, что иногда счет-фактура от поставщика присылается на фирму по электронной почте ***до того***, как поступит товар и должна вводиться в систему заранее. То есть нужно иметь признак того, поступил ли товар на склад физически или еще нет. И только когда товар реально поступит, устанавливать «некую птичку» для проведения документа в балансе.

Поразмышляв, мы пришли к выводу, что хорошо бы еще обеспечить на всякий случай возможность при вводе документов ***рассчитывать сумму разными способами***, отталкиваясь либо от цены товара в рублях без НДС, либо от цены с НДС. Дело в том, что при вводе данных персонал компании наверняка будет использовать сумму всего документа для ***контроля правильности ввода цифр***. И если одни поставщики вели расчеты от цены без НДС, а другие от цены с НДС, то, ***в связи с ошибками округления***, сумма введенного в базу данных документа может не совпасть с суммой исходного, что может вызовет осложнения в работе менеджеров. Разумеется, признак того, как рассчитывалась сумма, придется хранить в каждом документе. К тому же потребуется хранить в каждом документе и ставку НДС. Для простоты будем исходить из того, что она одинаковая для всех товаров внутри одного конкретного документа. На практике обычно так и бывает.

Последнее обстоятельство, которое мы должны учесть, это ограничение, накладываемое системой автоматических бухгалтерских операций в Allegro. Это ограничение состоит в том, что в документе ***проводимые суммы должны храниться в полях таблицы*** документа в явном виде. Следовательно, нам, понадобится для каждой позиции товара завести, как минимум, ***два поля суммы***: в валюте документа (для начисления обязательств перед поставщиком) и в долларах США (для начисления стоимости товарных запасов).

Для создания нового типа документа вызовем окно «Метаданные» и, используя контекстное меню или кнопку на панели инструментов на закладке «Документы», создадим новый тип документа:

Поступление на склад, таблица STOCK_IN. Нажмем кнопку ***OK***.



При создании нового типа документа автоматически создаются два целочисленных поля: ID и DIR_ID, которые отображаются в списке полей, если включен режим ***Отображать поля ID***. Таблица STOCK_IN, которую мы создали, предназначена для хранения «шапки» документа, то есть сведений, относящихся к каждому конкретному документу целиком, и которые можно разместить в таблице одной строкой. Такая таблица называется ***главной таблицей***.

Для добавления остальных полей используем пункт ***Добавить поле*** контекстного меню или кнопку с плюсиком на закладке «Поля». Все эти поля будут содержать атрибуты, относящиеся к документу в целом. Для хранения курсов валют будем использовать 5 дробных десятичных знаков, без форматирования чисел, а для хранения номера документа используем строковое поле типа VARCHAR(20), так как нам в принципе неизвестны те замысловатые способы, какими могут нумеровать свои документы будущие поставщики:

Атрибут	**Поле**	**Тип**	**Класс объектов**	**Обязатель-ный атрибут**
Склад	STOCK_ACC	TACCOUNT		да
Дата поступления на склад	ENTRY_DATE	DATE		да
Вид документа	DOC_KIND	INTEGER		да
Кредитуемый счет	ACC_ID	TACCOUNT		да
Контрагент	CONTRAGENT	TREFERENCE	CONTRAGENT	да
Валюта документа	LAYER_ID	TLAYER		да
Способ расчета сумм	CALC_MODE	INTEGER		да
Товар поступил	HAS_ENTRY	TBOOLEAN		да
Дата документа	DOC_DATE	DATE		да
Номер документа	DOC_NO	VARCHAR(20)		да
Ставка НДС	VAT_RATE	DECIMAL(4,2)		да
Курс валюты документа	EXCH_RATE_L	DECIMAL(18,5)		да
Курс доллара США	EXCH_RATE_S	DECIMAL(18,5)		да
Всего, в валюте документа	TATAL_L	DECIMAL(18,2)		да
Всего, в долларах США	TOTAL_S	DECIMAL(18,2)		да
Всего, в рублях	TOTAL_R	DECIMAL(18,2)		да

Итак, главная таблица для типа документа «Поступление на склад» готова. Имеет смысл добавить в нее ограничение на уникальность номера документа. Для предотвращения повторного ввода одного и того же документа мы потребуем уникальности сочетания 3 полей: ***Контрагент, Дата документа, Номер документа***.

Чтобы создать ограничение на уникальность воспользуемся окном ISQL (мы это уже делали раньше), которое вызывается с помощью пункта ***Инструменты/Интерактивный SQL*** Главного меню.

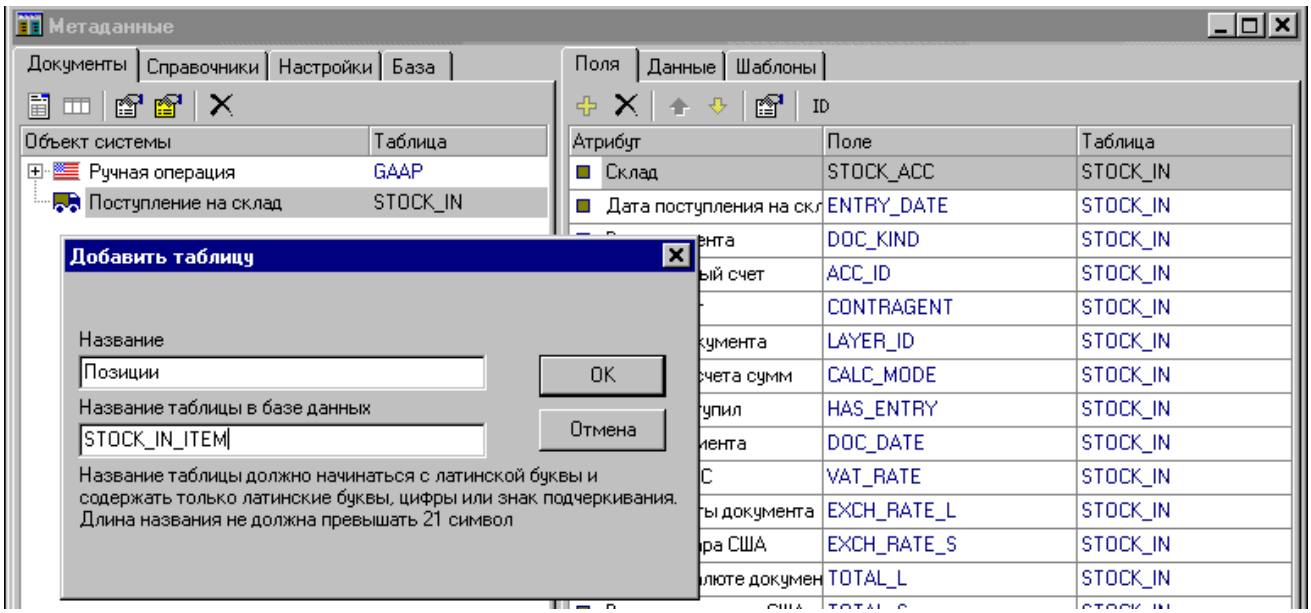
В окне ISQL наберем команду:

```
alter table stock_in
add constraint ux_stock_in
unique(contragent, doc_date, doc_no)
```

и нажмем **Ctrl+Enter**.

Создаем подчиненную таблицу «Позиции» к «Поступлению на склад»

Теперь нам нужно создать так называемую ***подчиненную таблицу*** к типу документов «Поступление на склад». В ней будут храниться сведения о поступивших товарах, по одной строке на каждую поступившую позицию. Для этого на закладке «Документы» используем пункт контекстного меню ***Добавить таблицу***.



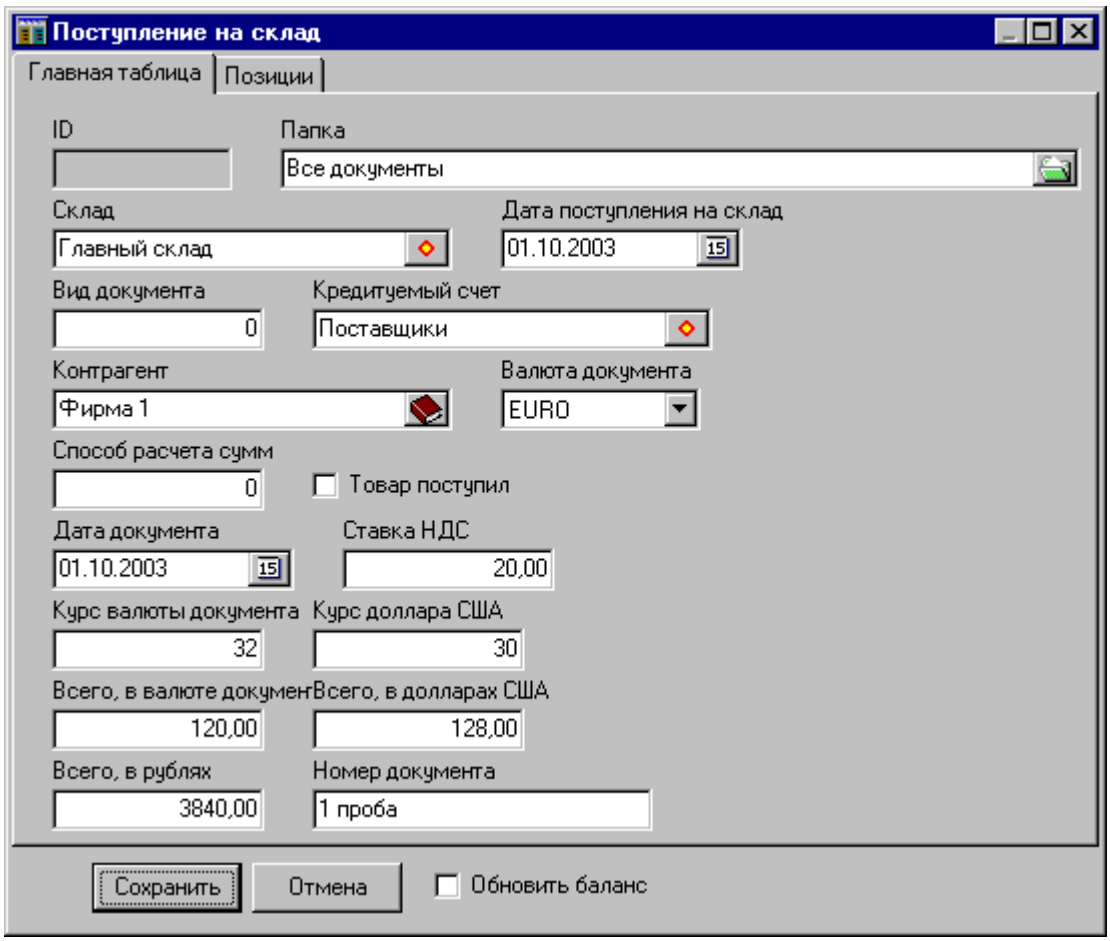
Назовем таблицу ***Позиции***, введем будущее название таблицы в базе данных STOCK_IN_ITEM и нажмем кнопку ***OK***. Подчиненная таблица всегда создается сразу с двумя целочисленными полями: ID и N. Поле ID связывает подчиненную таблицу с главной таблицей документа, а поле N используется в качестве первичного ключа подчиненной таблицы. Одновременно с подчиненной таблицей программа создала в базе данных генератор STOCK_IN_ITEM_N_GEN для генерации новых значений поля N.

Добавим в подчиненную таблицу поля. Для количества используем целочисленное поле без форматирования, так как наш заказчик имеет дело исключительно со штучным товаром. Для цен будем использовать 10 знаков точности, из которых 3 знака дробные. Для этих полей не будем указывать формат чисел. Для полей сумм будем использовать тип DECIMAL(18,2) с форматированием чисел ##0.00.

Атрибут	**Поле**	**Тип**	**Класс объектов**	**Обязательный атрибут**
Товар	GOODS	TREFERENCE	GOODS	да
Количество	QUANTITY	INTEGER		да
Цена в валюте документа, без НДС	PRICE_L_WO_VAT	DECIMAL(10,3)		да
Цена в валюте документа, с НДС	PRICE_L	DECIMAL(10,3)		да
Цена в рублях, без НДС	PRICE_R_WO_VAT	DECIMAL(10,3)		да
Цена в рублях, с НДС	PRICE_R	DECIMAL(10,3)		да
Сумма в валюте документа	AMOUNT_L	DECIMAL(18,2)		да
Сумма в долларах США	AMOUNT_S	DECIMAL(18,2)		да
Сумма в рублях	AMOUNT_R	DECIMAL(18,2)		да

Создаем пробный документ «Поступление на склад»

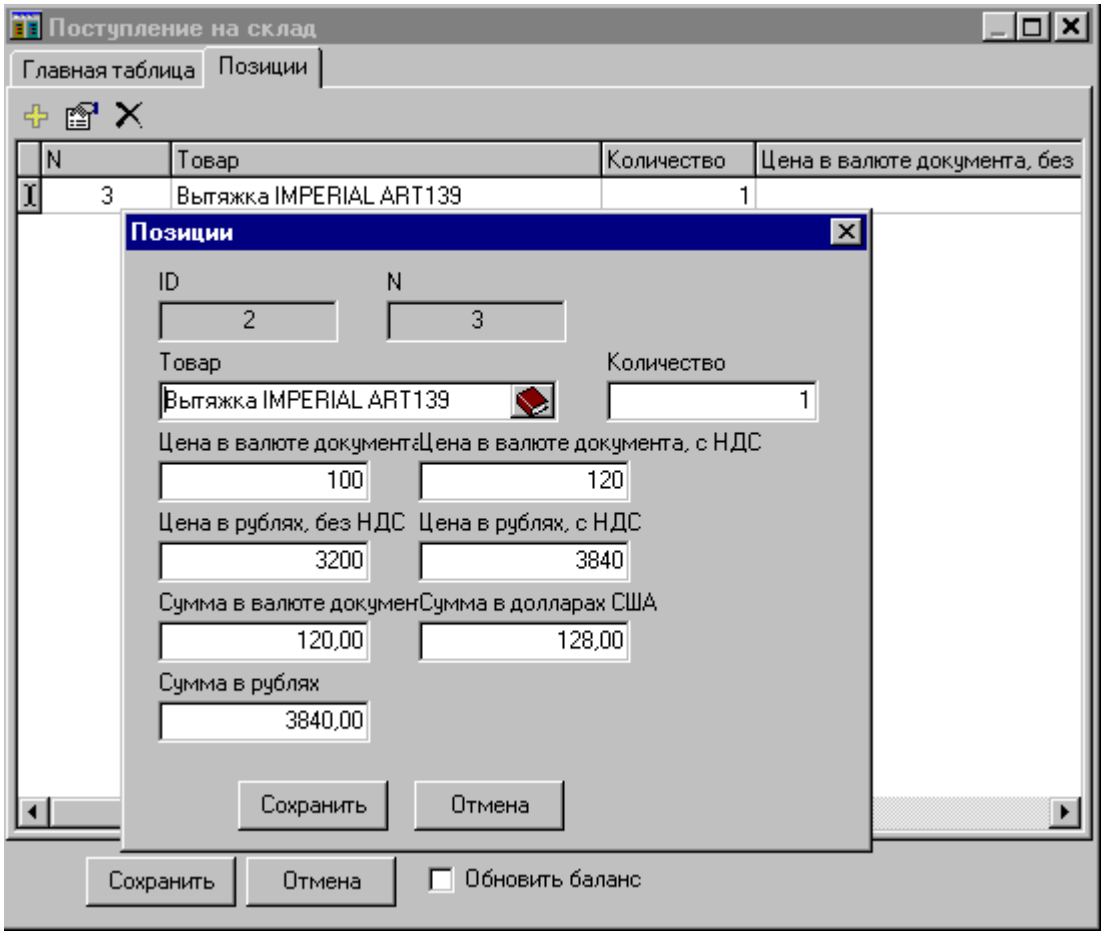
Выберем в окне «Метаданные» тип документа **Поступление на склад**, справа закладку «Данные» и нажмем кнопку **Добавить документ** (с плюсиком). Появится примитивный системный диалог ввода документа, который используется только для целей отладки:



На закладке «Главная таблица» введем:

Папка: **Все документы**
Склад: **Главный склад**
Дата поступления на склад: **сегодняшняя**
Вид документа: **0**
Корреспондирующий счет: **Поставщики**
Контрагент: **Фирма 1**
Валюта документа: **EURO**
Способ расчета сумм: **0**
Товар поступил: **птичка снята**
Дата документа: **сегодняшняя**
Ставка НДС: **20**
Курс валюты документа: **32**
Курс доллара США: **30**
Всего, в валюте документа: **120**
Всего, в долларах США: **128**
Всего в рублях: **3840**
Номер документа:**1**

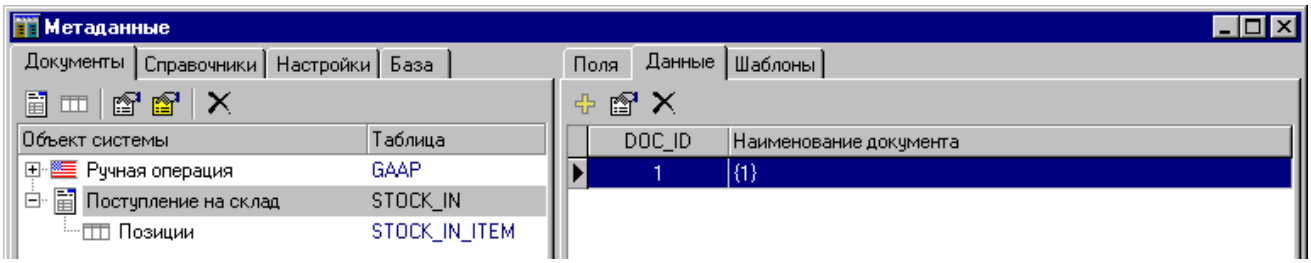
Перейдем здесь же на закладку «Позиции» и нажмем кнопку ***Добавить*** или клавишу ***Insert***.



Введем несколько позиций, заполняя поля:

Товар: **любой из справочника**
Кол-во: **1**
Цена в валюте документа без НДС: **100**
Цена в валюте документа с НДС: **120**
Цена в рублях без НДС: **3200**
Цена в рублях с НДС: **3840**
Сумма в валюте документа: **120**
Сумма в долларах США: **128**
Сумма в рублях: **3840**

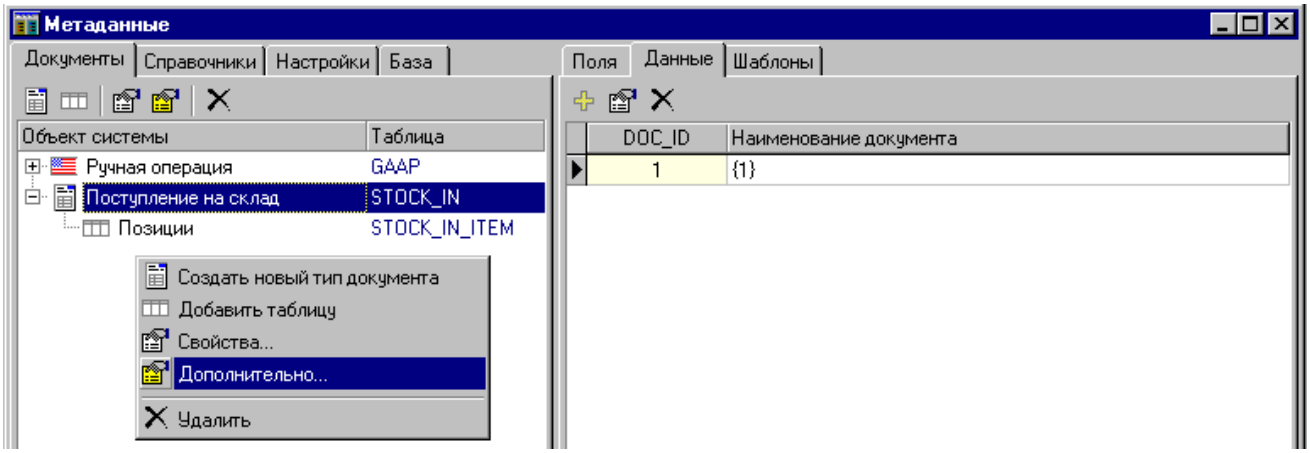
Сохраним весь документ, нажав кнопку ***Сохранить***. Мы видим, что в списке справа появился ID документа в фигурных скобках:



Для того чтобы документ отображался более приемлемым образом, нужно настроить форматирование наименований. Нечто подобное мы уже делали, когда создавали справочники.

Форматирование наименований и другие дополнительные свойства.

Вызовем диалог «Дополнительные свойства документа» с помощью контекстного меню:

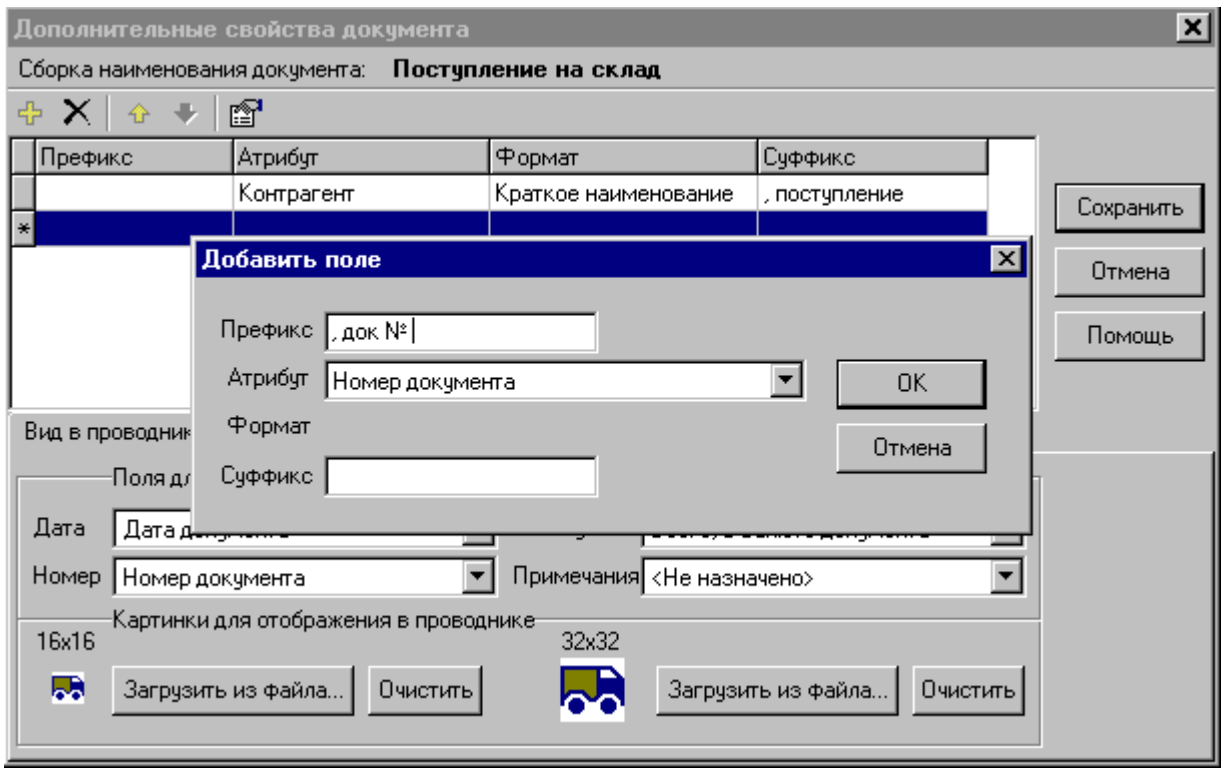


В появившемся диалоге в полях для поиска и отображения в проводнике укажем:

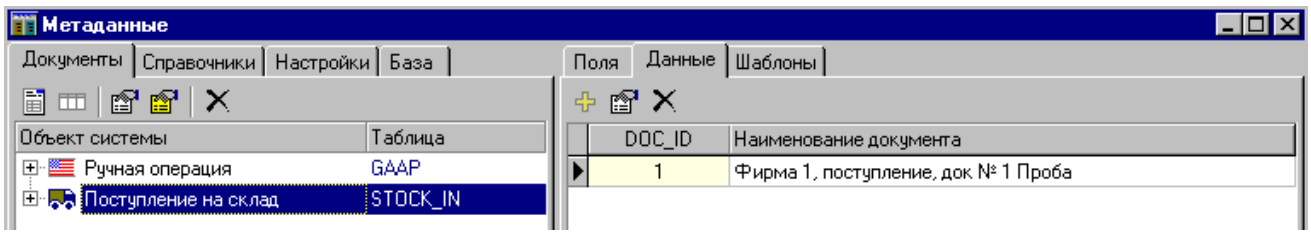
Дата: ***Дата документа*** Сумма: ***Всего, в валюте документа***
Номер: ***Номер документа*** Примечания: ***<не назначено>***

Неплохо бы еще назначить картинки для отображения документа в проводнике. Выберем их из файлов картинок формата BMP размером 16x16 пикселей (маленькая картинка) и 32x32 (большая).

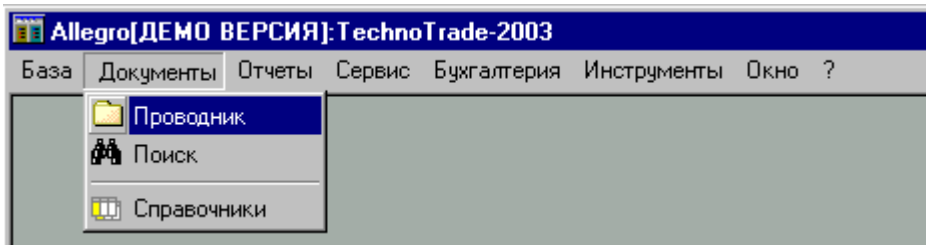
Удалим строку с ID в таблице «сборка наименования документа» и вставим 2 строки: ***Контрагент***, с суффиксом «***, поступление***» и ***Номер документа*** с префиксом «***, док.№*** » :



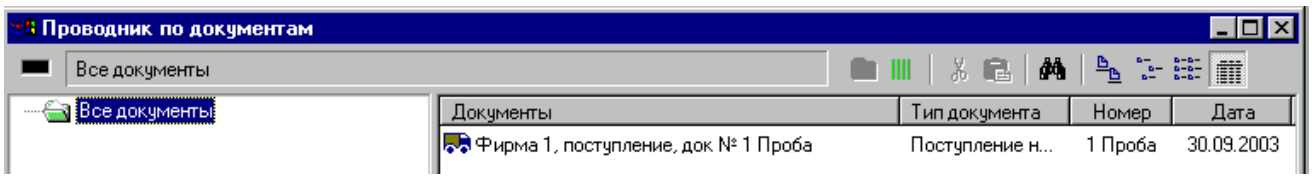
Сохраним все настройки, нажав кнопку «Сохранить». Обратим внимание, что теперь в списке «Данные» документ представлен своим отформатированным наименованием. Хранимая процедура в базе данных, создающая это наименование называется STOCK_IN_GET_NAMES.



Вызовем «Проводник по документам» через пункт **Документы/Проводник** Главного меню:



Созданный нами документ отображается в «Проводнике» в папке **Все документы\.** Причем здесь мы видим, кроме отформатированного наименования, еще такую дополнительную информацию, как Тип документа, Номер, Дату и Сумму.



Мы всегда можем изменить способ форматирования наименований документов. Эти наименования не хранятся в базе данных, а формируются «на ходу». Последнее, что нам осталось сделать в метаданных, это настроить шаблон автоматической бухгалтерской операции для всех документов типа «Поступление на склад».

Настраиваем шаблон бухгалтерской операции документа.

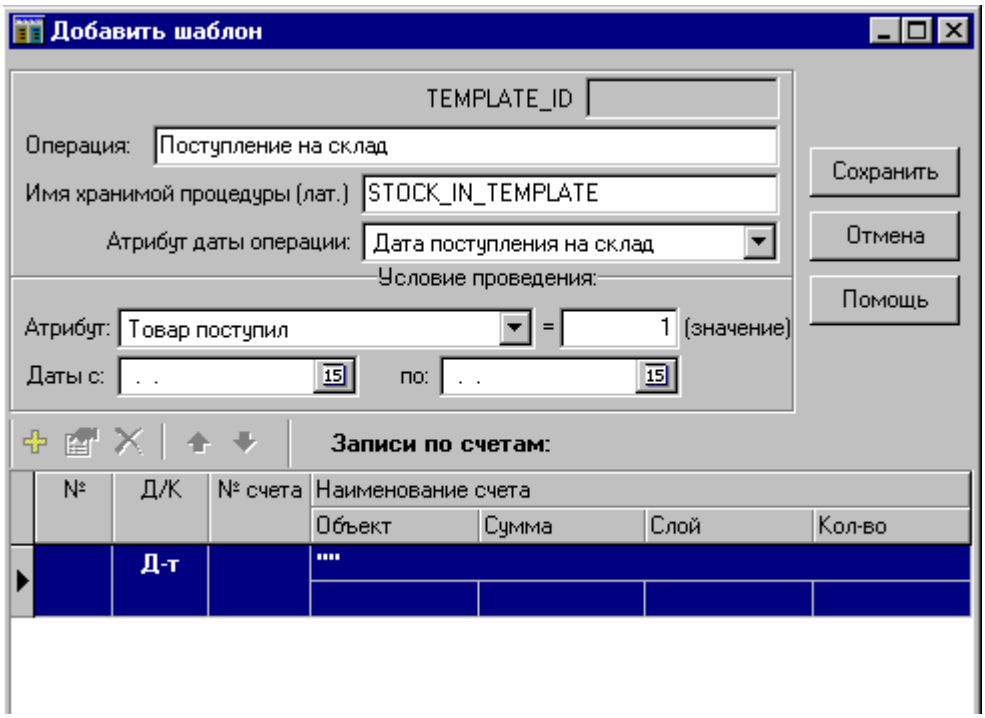
С самого начала мы решили, что тип документа «Поступление на склад» будет объединять операции 3 разных видов. Примем, что значение в поле **Вид документа** связано с операциями следующим образом:

0 – поставка с возникновением обязательств перед контрагентом
1 – простое приобретение, например, за наличные
2 – ввод начальных остатков.

Все эти виды операций мы осуществим по одной схеме.

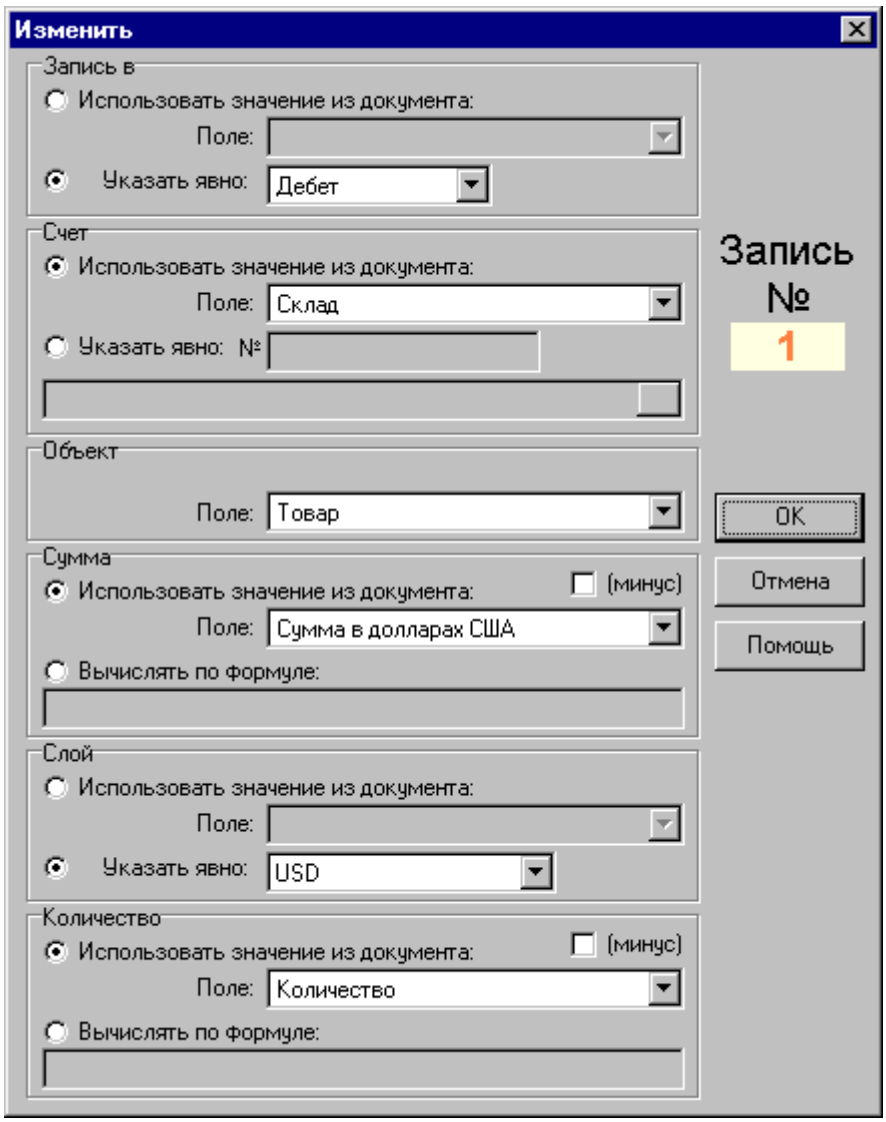
Вызовем окно «Метаданные» и выберем тип документа «Поступление на склад». Справа выберем закладку «Шаблоны» и используем пункт **Добавить шаблон** контекстного меню.

В появившемся диалоге введем:
Операция: **Поставка**
Имя хранимой процедуры: **STOCK_IN_TEMPLATE**
Атрибут даты операции: ***Дата поступления на склад***
Условие проведения:
  Атрибут: ***Товар поступил***
  Значение: **1**

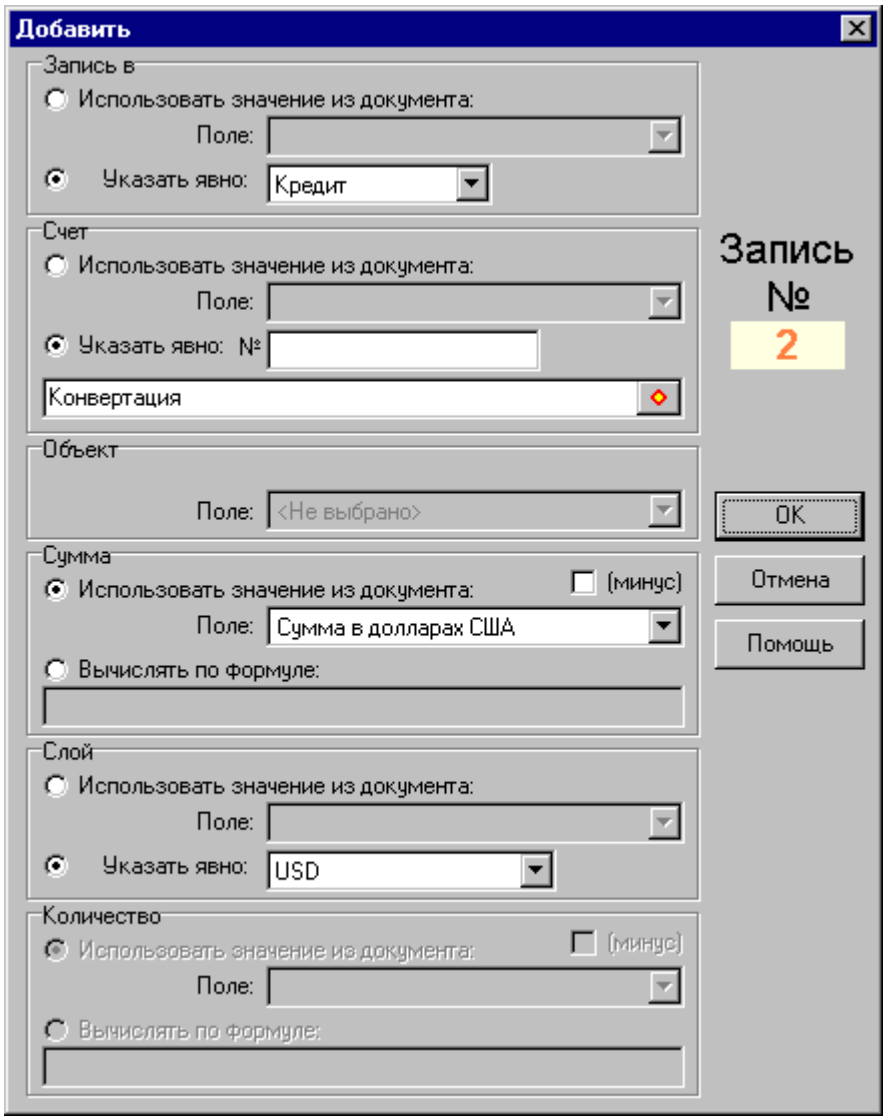


Теперь добавим в него записи по счетам. Каждая запись добавляется с помощью клавиши ***Insert*** или кнопки с плюсиком. Мы должны добавить 4 записи.

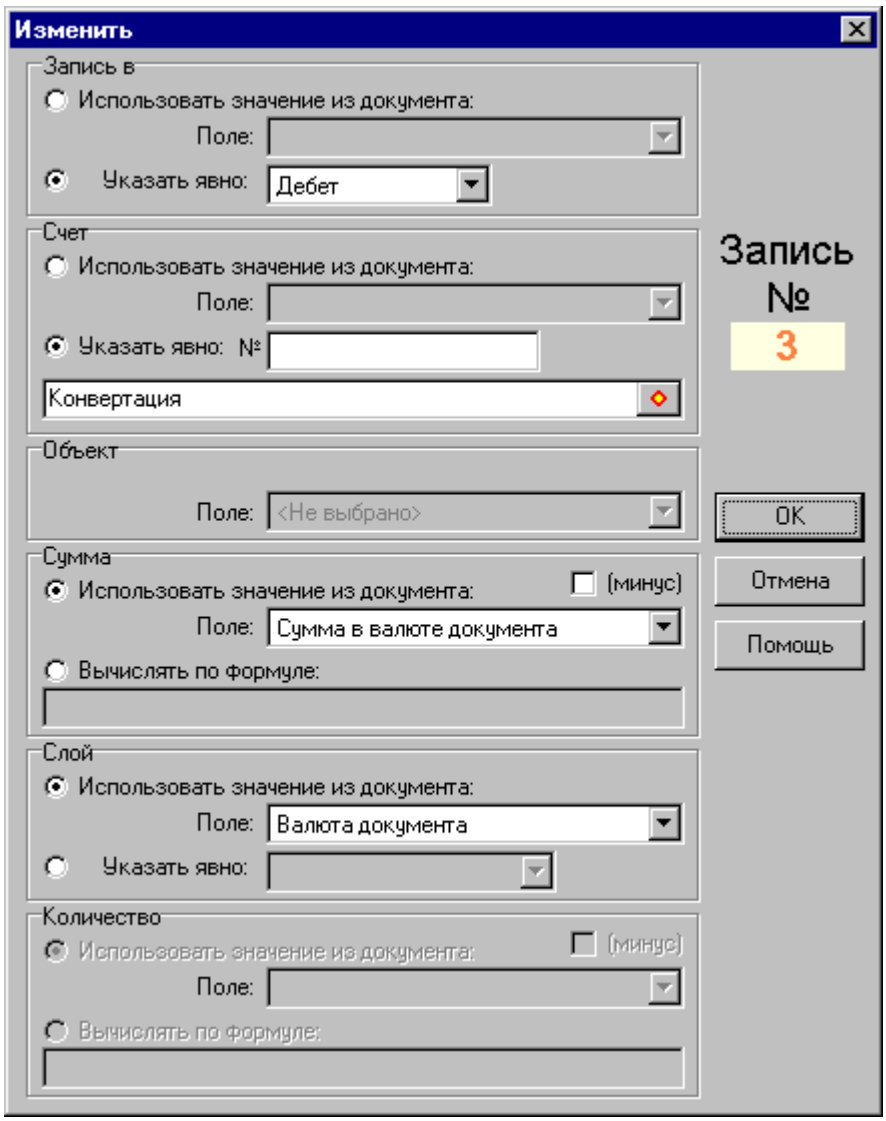
Добавим запись в ***Дебет***, указав в качестве счета поле документа ***Склад***, в качестве объекта поле документа ***Товар***, сумму возьмем из поля документа ***Сумма в долларах США***, слой укажем явно ***USD***, а количество возьмем из поля документа ***Количество***. Нажмем ***ОК***.



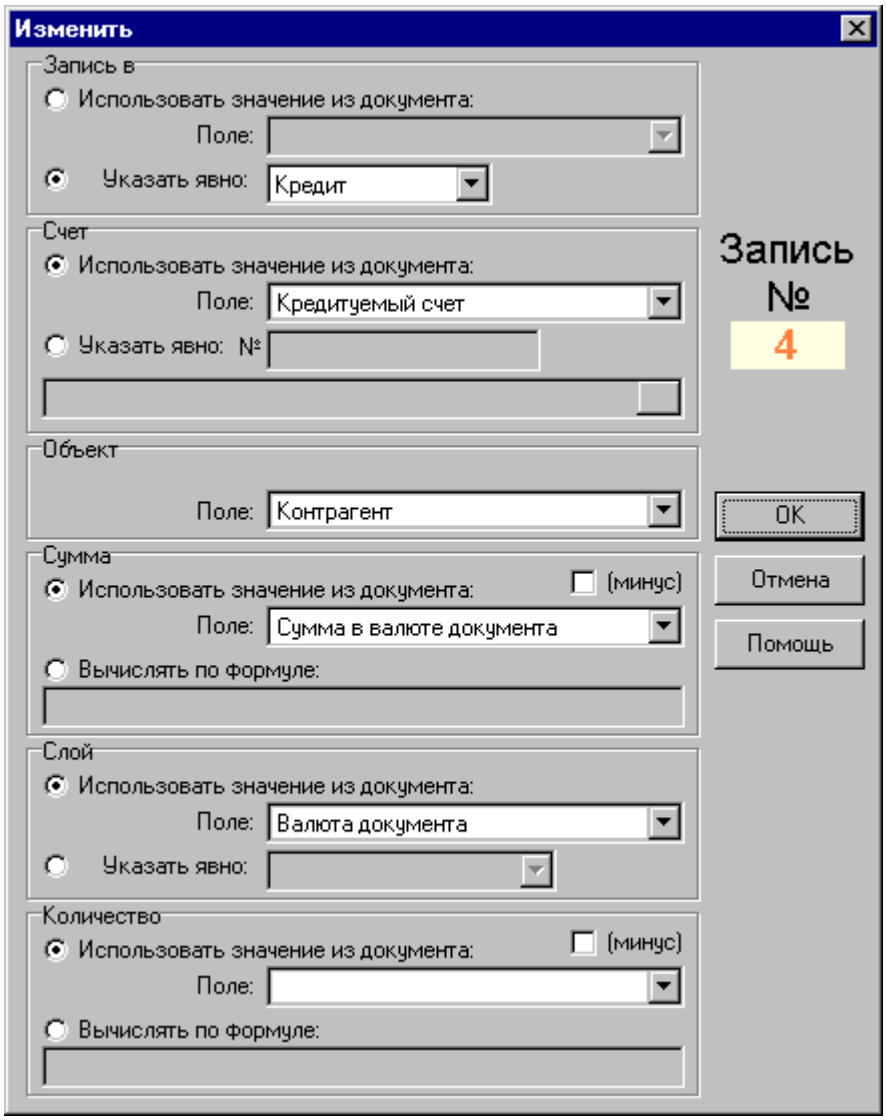
Добавим запись в ***Кредит***, указав явно в качестве счета явно ***Конвертацию*** (из регистра ***Обязательства и капитал***), а сумму возьмем из того же поля документа ***Сумма в долларах США***, слой укажем явно ***USD***. Нажмем ***OK.***



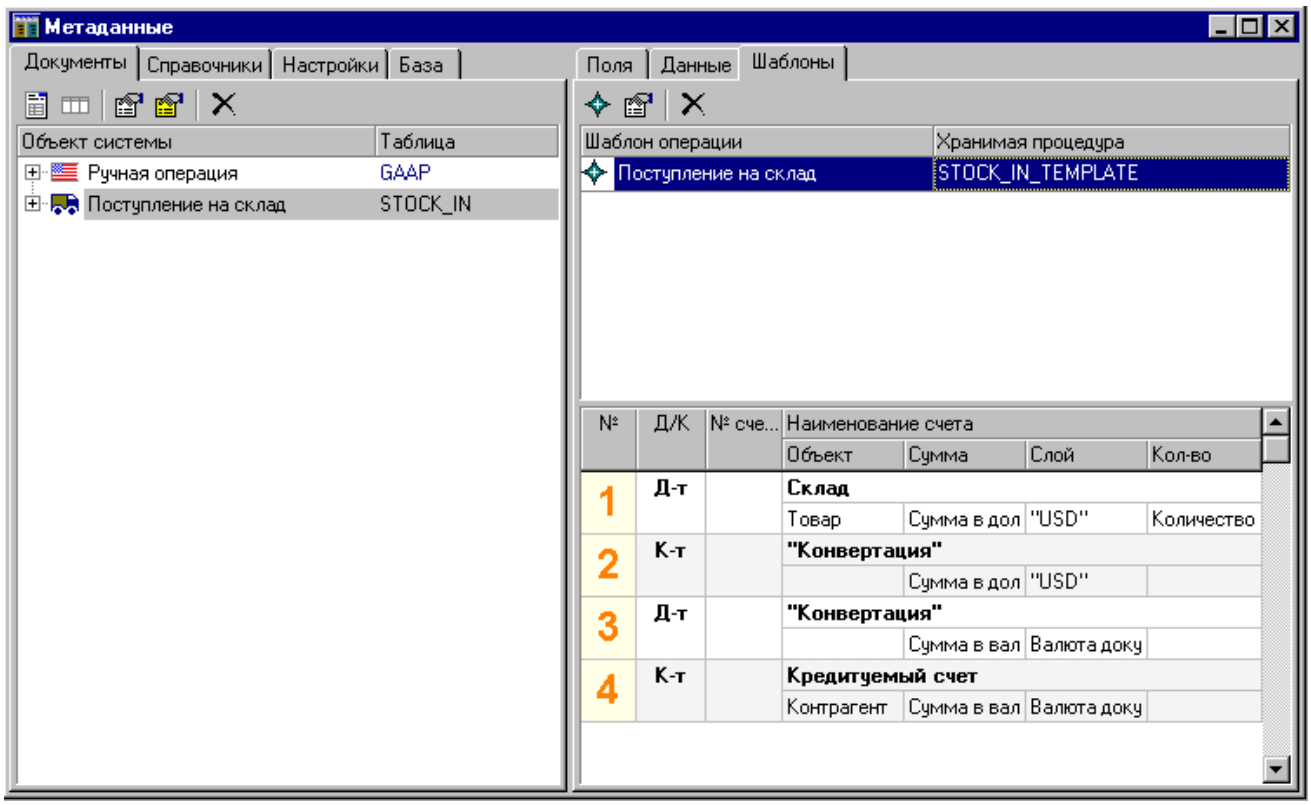
Добавим запись в ***Дебет***, указав явно в качестве счета ***Конвертацию***, сумму возьмем из поля документа ***Сумма в валюте документа***, а слой возьмем из поля ***Валюта документа***. Нажмем ***ОК***.



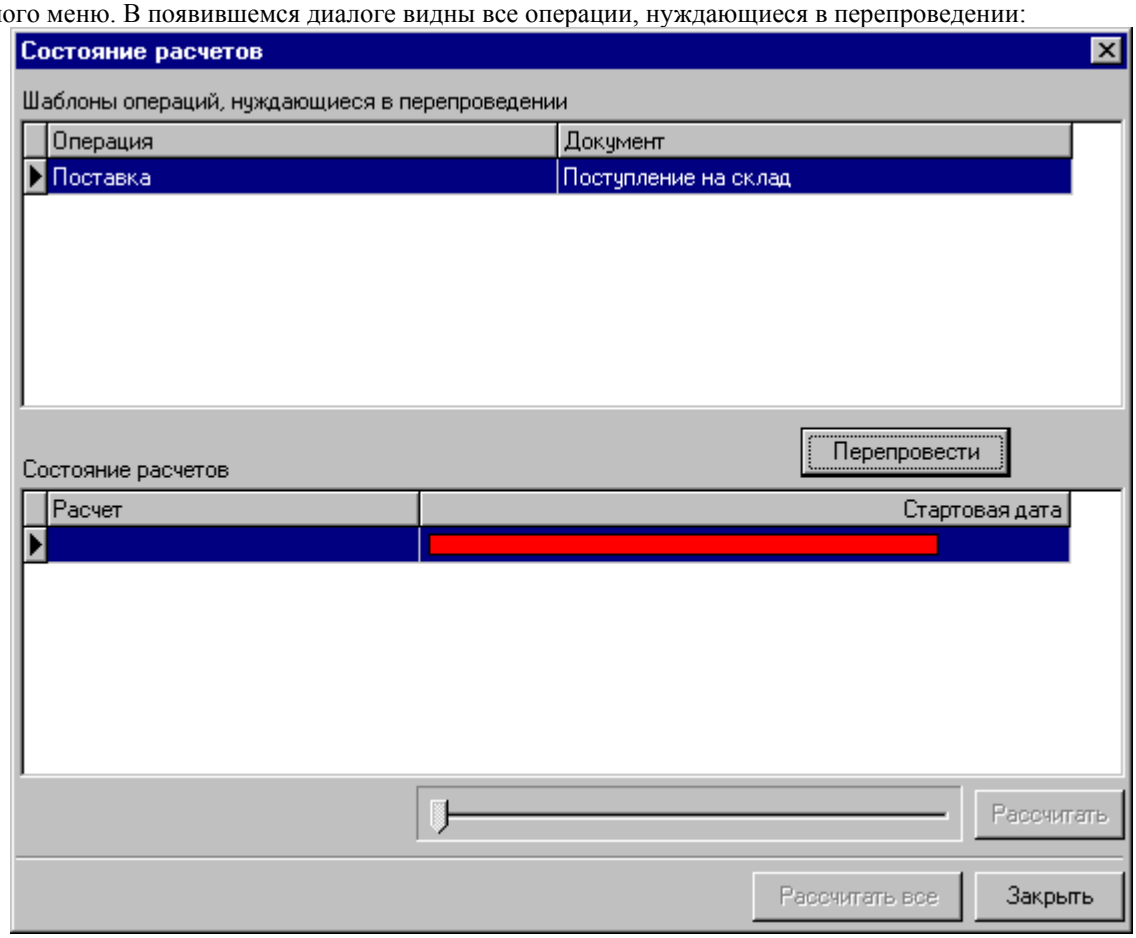
Добавим запись в ***Кредит***, используя поле документа ***Кредитуемый счет***, в качестве объекта поле документа ***Контрагент***, сумму возьмем из поля документа ***Сумма в валюте документа***, слой возьмем из поля ***Валюта документа***, а количество оставим пустым. Нажмем ***ОК***.



Сохраним шаблон. Теперь остается перепровести документы. После внесения изменений в шаблоны все операции, соответствующие этим шаблонам следует перепроводить. Перепроведение не занимает много времени, так как осуществляется в один прием для всех документов данного типа.



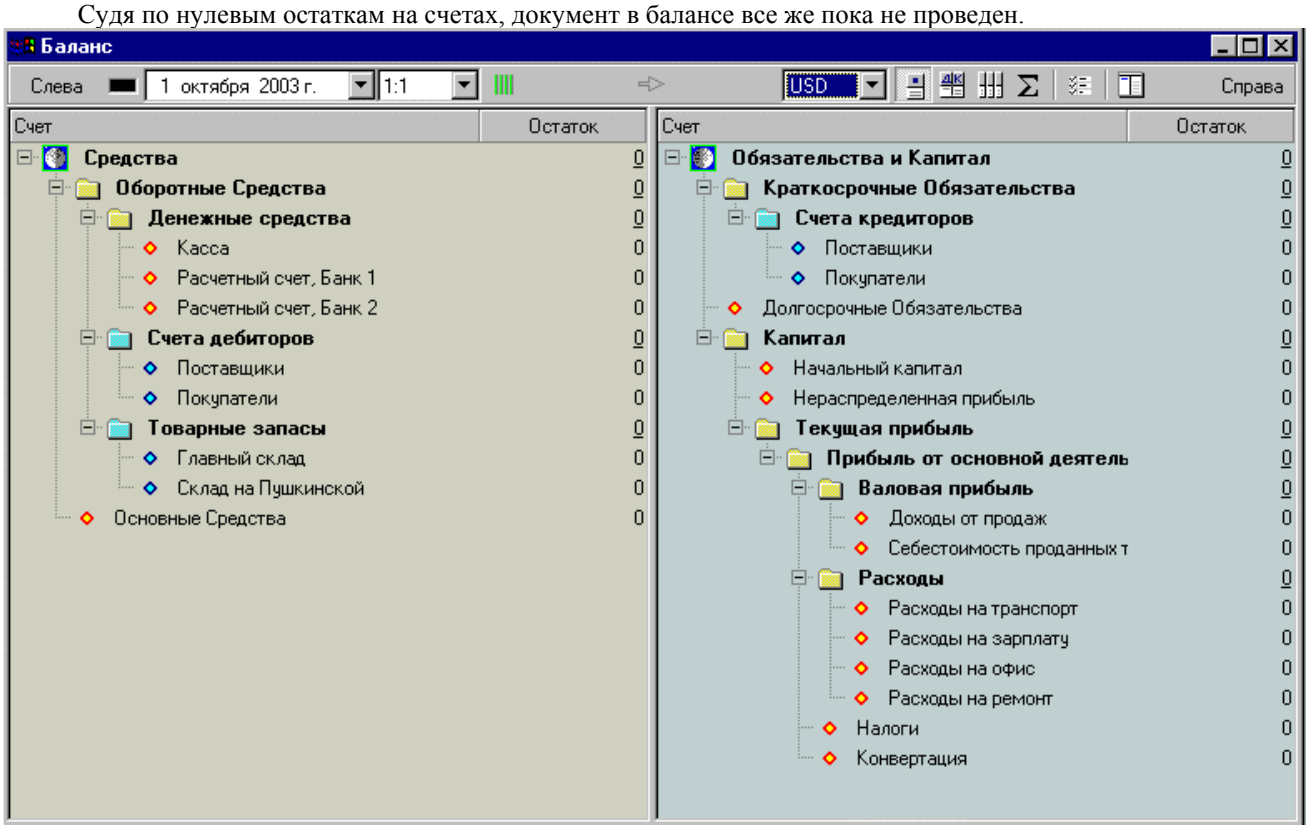
Для того чтобы перепровести документы, нужно использовать пункт **Бухгалтерия/Состояние расчетов** Главного меню. В появившемся диалоге видны все операции, нуждающиеся в перепроведении:



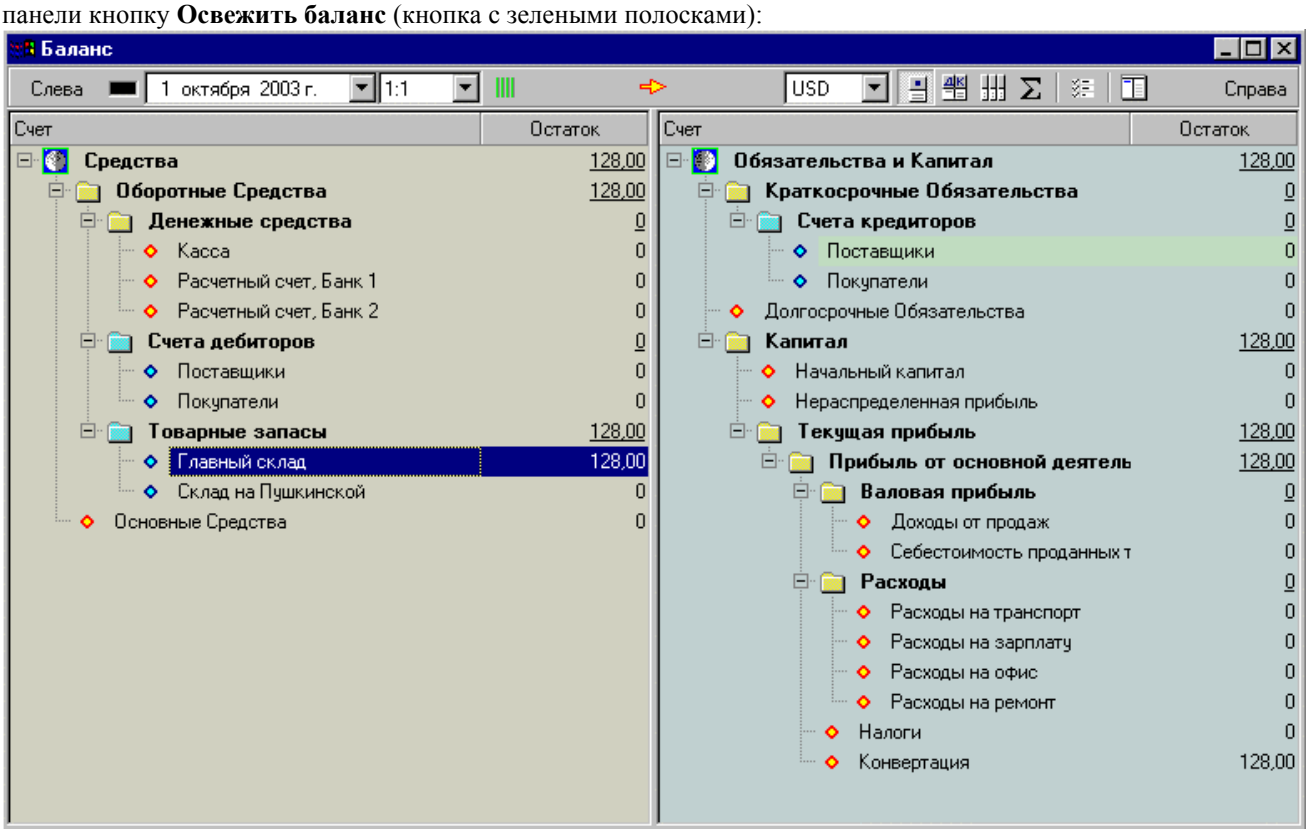
Нажмем кнопку **Перепровести**. Затем нажмем кнопку **Закрыть**.

Вызовем окно «Баланс» и выберем слой **USD**.

Судя по нулевым остаткам на счетах, документ в балансе все же пока не проведен.

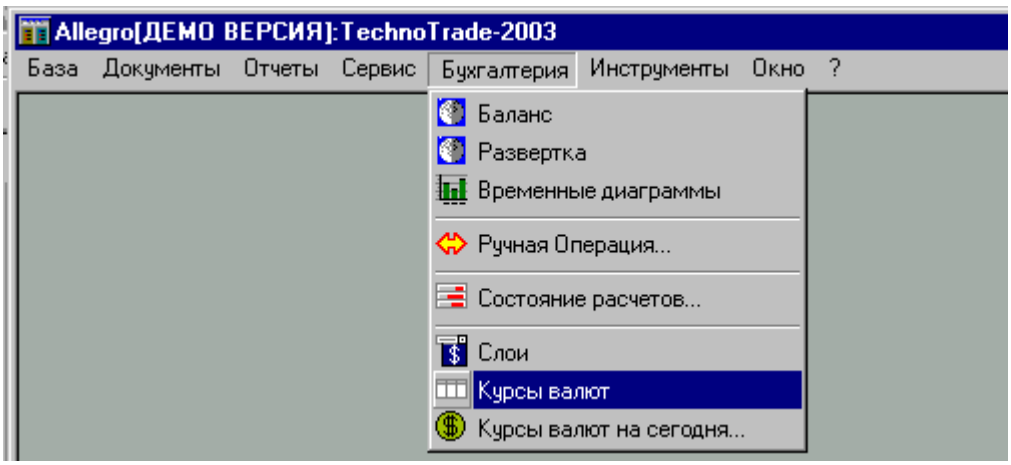


Вызовем наш документ на редактирование. Для этого в окне «Метаданные» выберем тип документов **Поступление на склад** и на закладке «Данные» дважды щелкнем на документе в списке. Появится примитивный редактор документа. Установим птичку в поле ***Товар поступил*** и сохраним документ. Документ автоматически перепровелся в системе. Теперь нужно освежить баланс, активизировав окно «Баланс» и нажав на верхней панели кнопку **Освежить баланс** (кнопка с зелеными полосками):

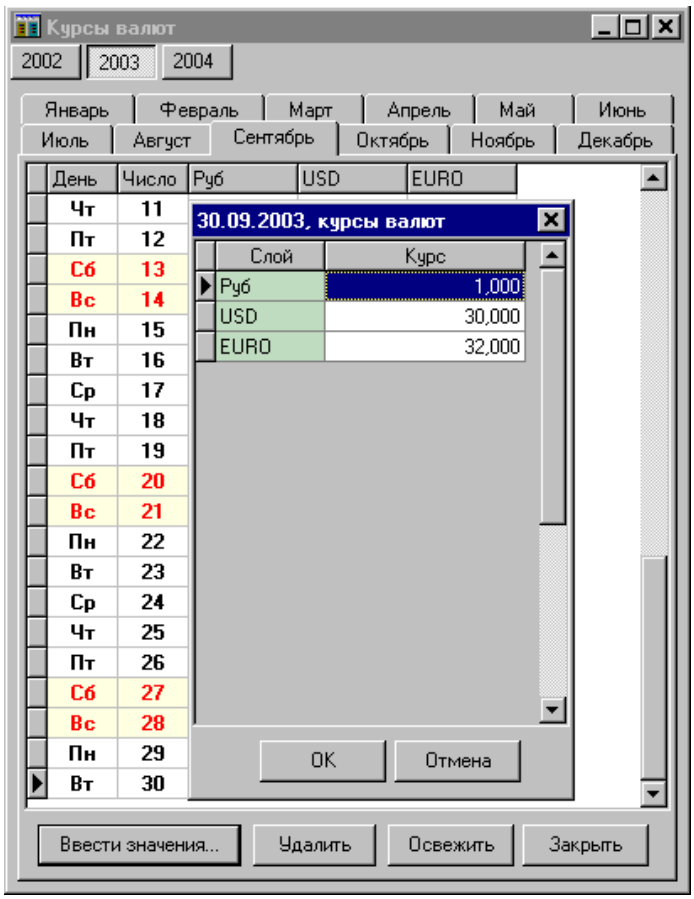


Как легко заметить, переключаясь в балансе между слоями USD и EURO, документ затронул суммы на счетах сразу в двух слоях. В слое EURO на счет ***Поставщики*** начислилось 120 EURO за счет ***Конвертации***, остаток на которой стал отрицательным. А в слое USD произошло начисление 128 USD на счет ***Главного склада*** за счет той же ***Конвертации***, которая в этом слое имеет положительный остаток.

Для того чтобы увидеть суммарный по всем слоям (консолидированный) баланс, нужно нажать кнопку со значком суммы на верхней панели окна «Баланс». Однако если нет системных курсов валют, то суммарный баланс не отображается и возникает соответствующее сообщение. Итак, сначала введем курсы валют. Для этого нужно использовать пункт ***Бухгалтерия/Курсы валют*** Главного меню:

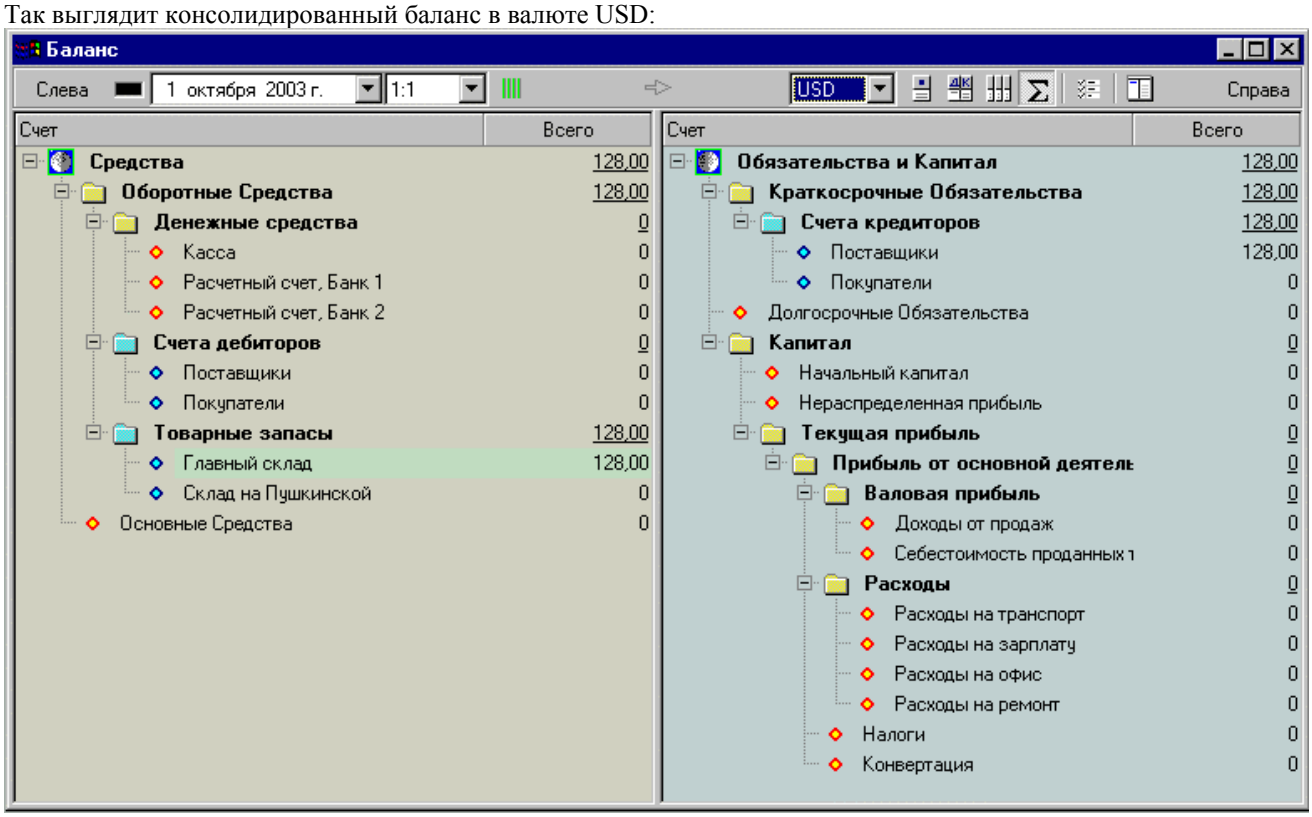


В появившемся окне выберем дату поступления на склад или более раннее и нажмем кнопку ***Ввести значения***. Введем курсы USD и EURO по отношению к рублю, для рублей введем курс, равный единице. Нажмем кнопку ***ОК***.

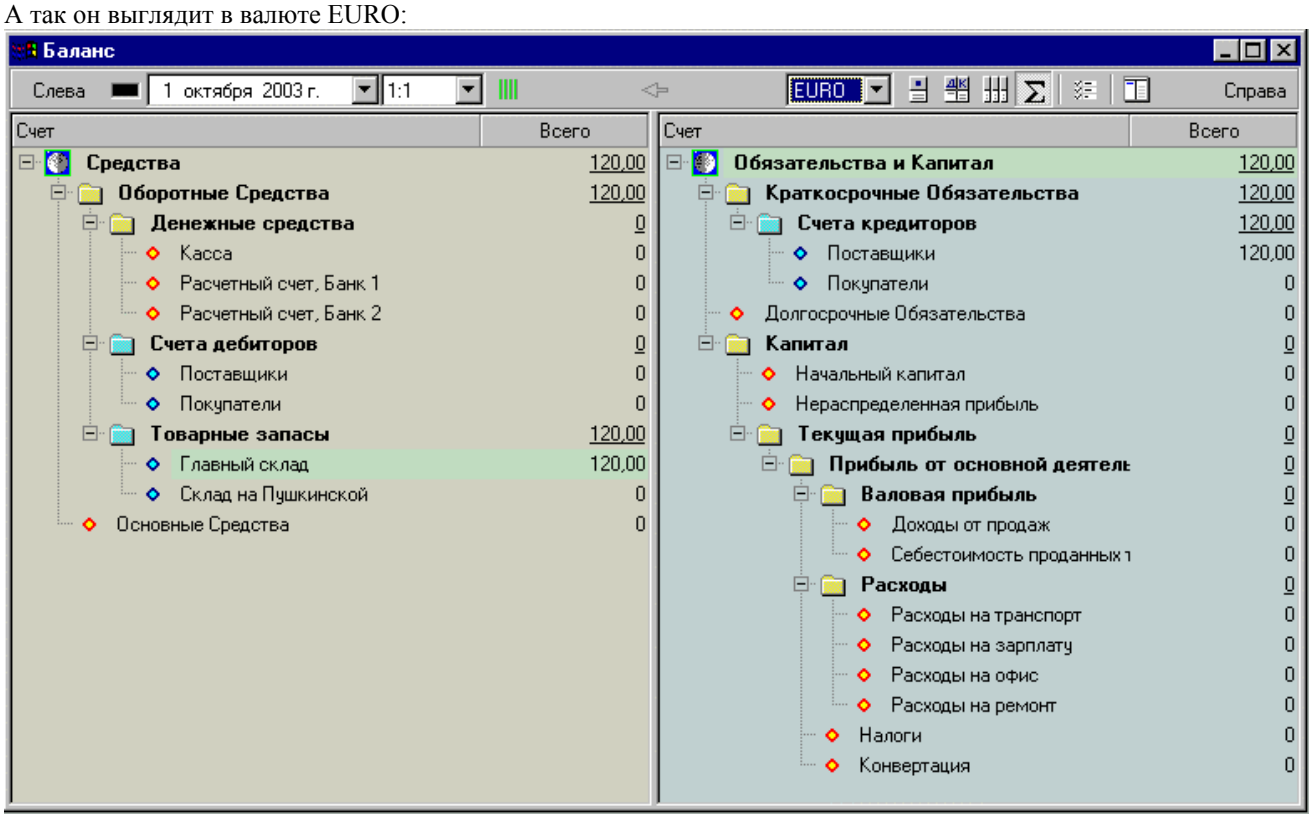


Закроем окно «Курсы валют» и вернемся к окну «Баланс».

Так выглядит консолидированный баланс в валюте USD:

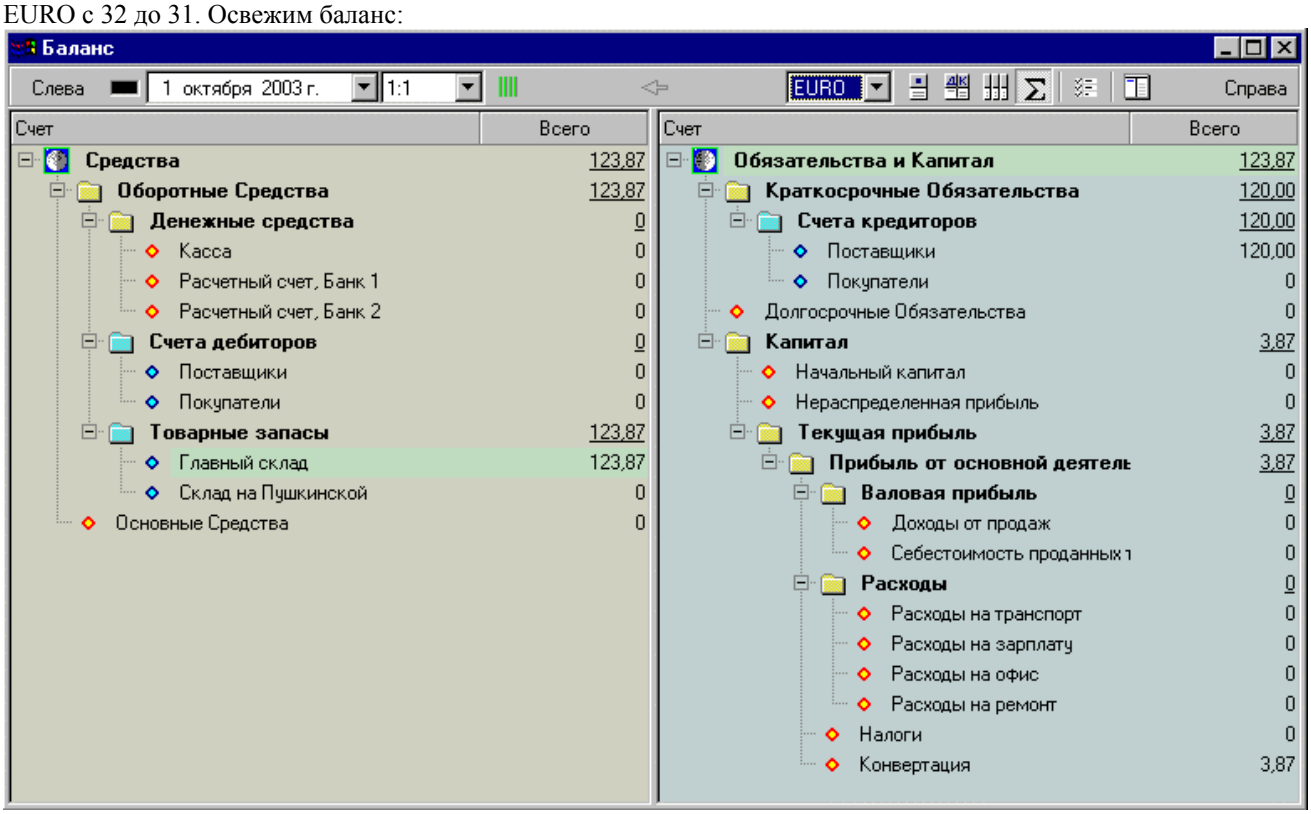


А так он выглядит в валюте EURO:



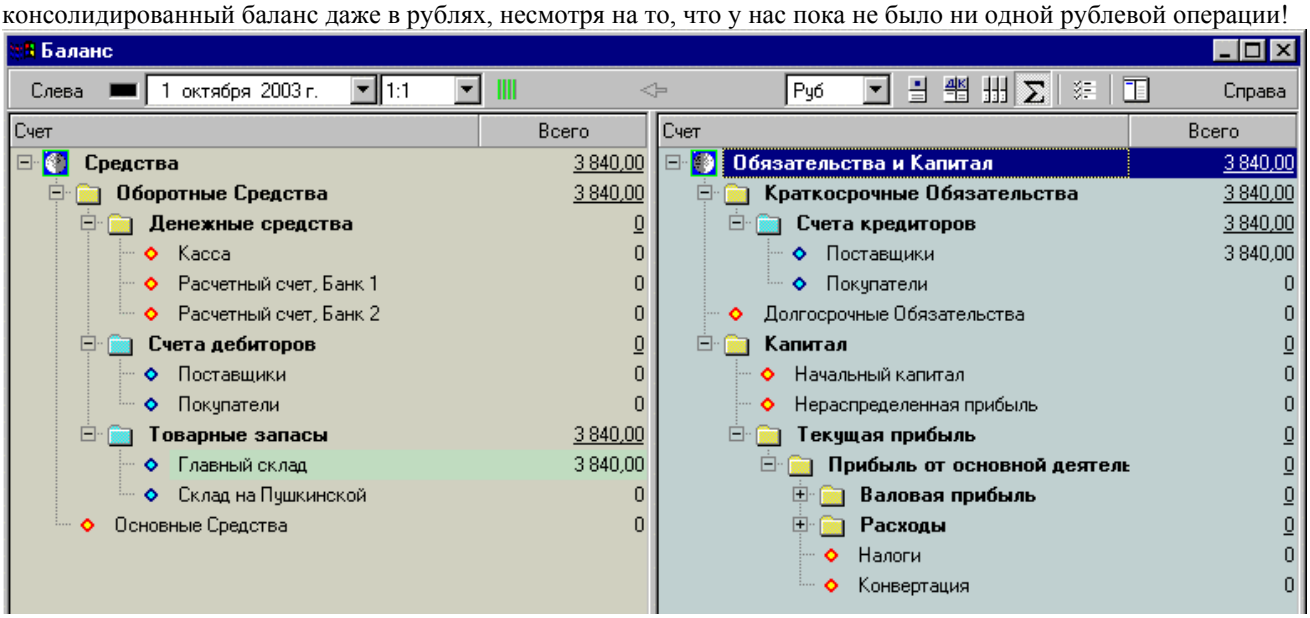
Как легко видеть, в консолидированном балансе записи в дебет и кредит счета ***Конвертация*** компенсируют друг друга. Видны средства на ***Главном складе*** и обязательства перед ***Поставщиками***.

Любопытно, а что произойдет, если изменится курс EURO, к примеру? Давайте изменим системный курс EURO с 32 до 31. Освежим баланс:



Мы видим, что падение курса EURO принесло компании прибыль 3,87 EURO, так как на счете ***Конвертация***, входящем в ***Текущую прибыль***, остаток стал ненулевым. Дело в том, что товары мы начислили в долларах США и их стоимость на ***Главном складе***, измеряемая в EURO возросла с падением курса EURO по отношению к доллару. А обязательства перед ***Поставщиками***, зафиксированные в EURO, остались прежними. Так как в силу балансового уравнения ***Капитал*** всегда равен разнице между ***Средствами*** и ***Обязательствами***, в результате падения курса EURO произошло увеличение ***Капитала***.

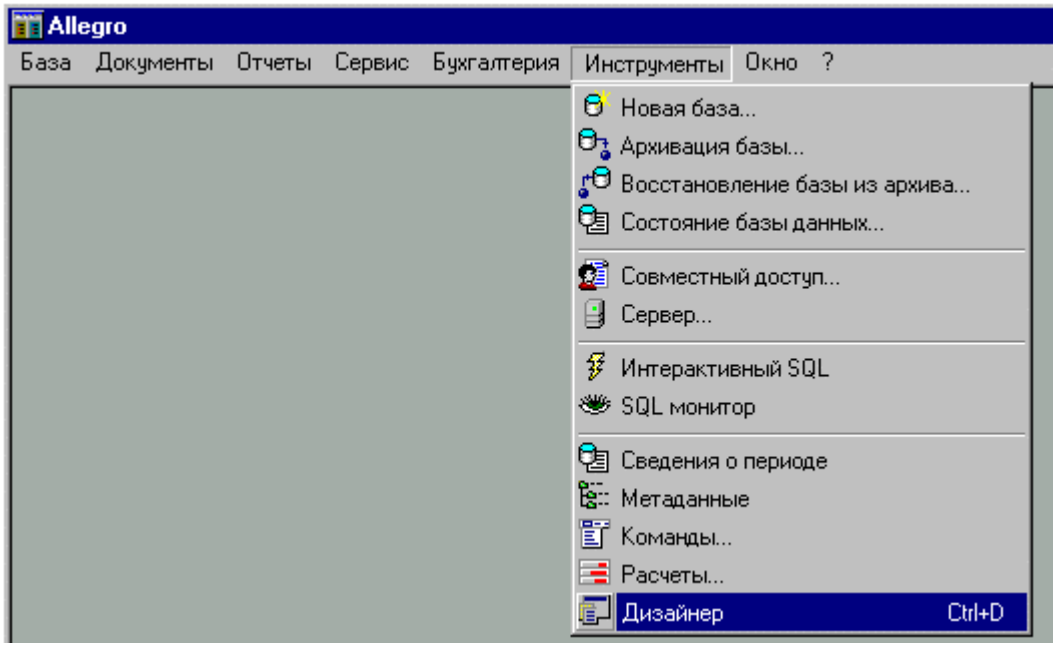
Разумеется, экономическая интерпретация такого результата может вызывать сомнения, так как прибылью называется изменение капитала компании в результате **ее эффективной деятельности**, чего в данном случае не было. Если Вы считаете так, то можно перенести счет ***Конвертация*** в корень каталога ***Обязательства и Капитал*** и переименовать его, скажем, в «Курсовую разницу». Важно в данном случае то, что многослойная структура баланса позволяет нам легко работать одновременно со всеми счетами, в какой угодно валюте и в любой из них отображать консолидированный баланс. При желании мы можем отобразить текущий консолидированный баланс даже в рублях, несмотря на то, что у нас пока не было ни одной рублевой операции!



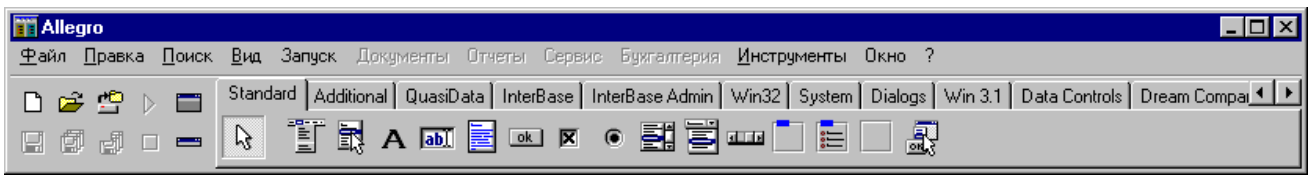
Глава 5. Создаем оконный интерфейс «Поступления на склад»

Создаем скриптовый проект

Переключимся в режим дизайнера, используя пункт ***Инструменты/Дизайнер*** Главного меню или комбинацию клавиш ***Ctrl+D***:

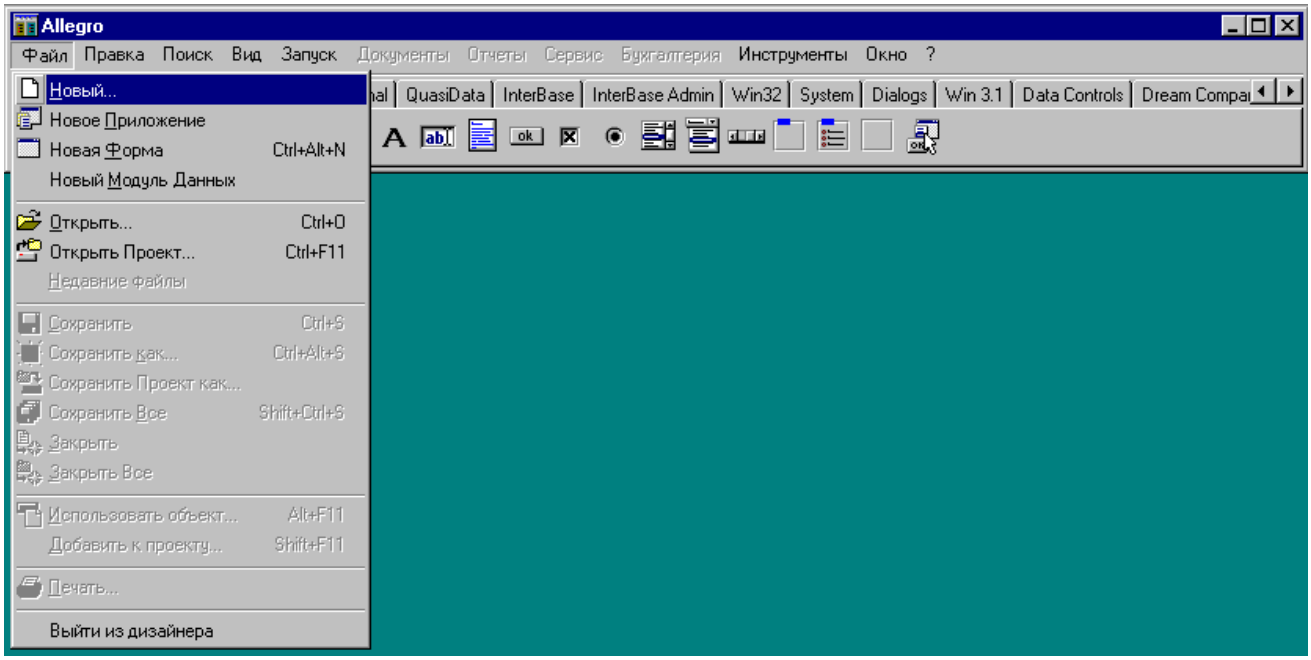


Главное окно программы «сожмется» и на нем возникнут органы управления, очень напоминающие интерактивную среду разработки программы Borland Delphi:



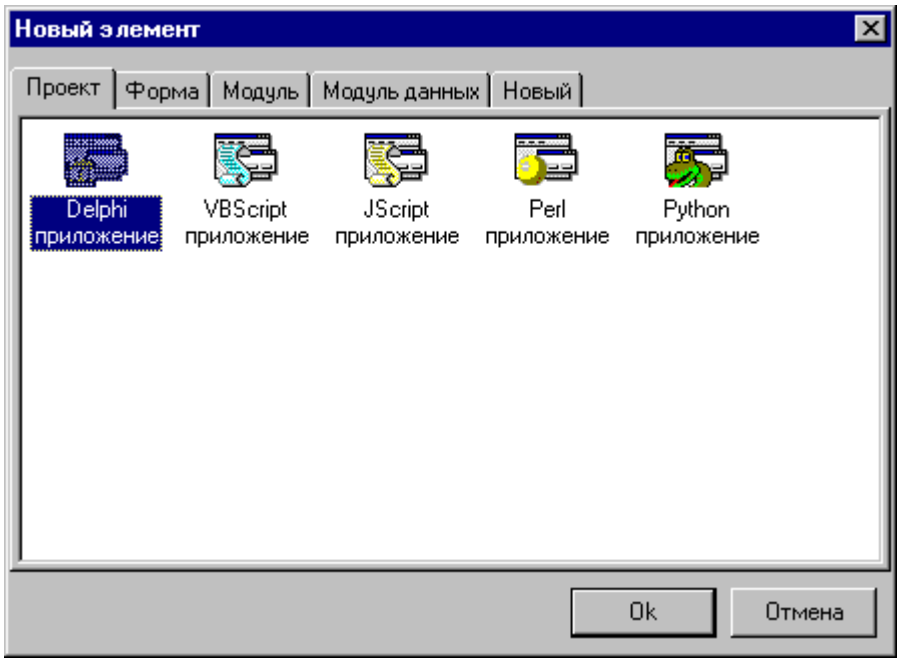
При создании оконных интерфейсов важно помнить правило: каждый **тип документа** привязывается к одному **проекту** оконного интерфейса. Каждый такой проект может содержать несколько окон, например, одно окно, служащее для редактирования позиций документа, а другое окно - для редактирования его шапки.

Итак, создадим новый проект. Для этого используем пункт меню **Файл/Новый**:



В появившемся диалоге видно, что программа позволяет создавать оконные интерфейсы на разных популярных скриптовых языках программирования.

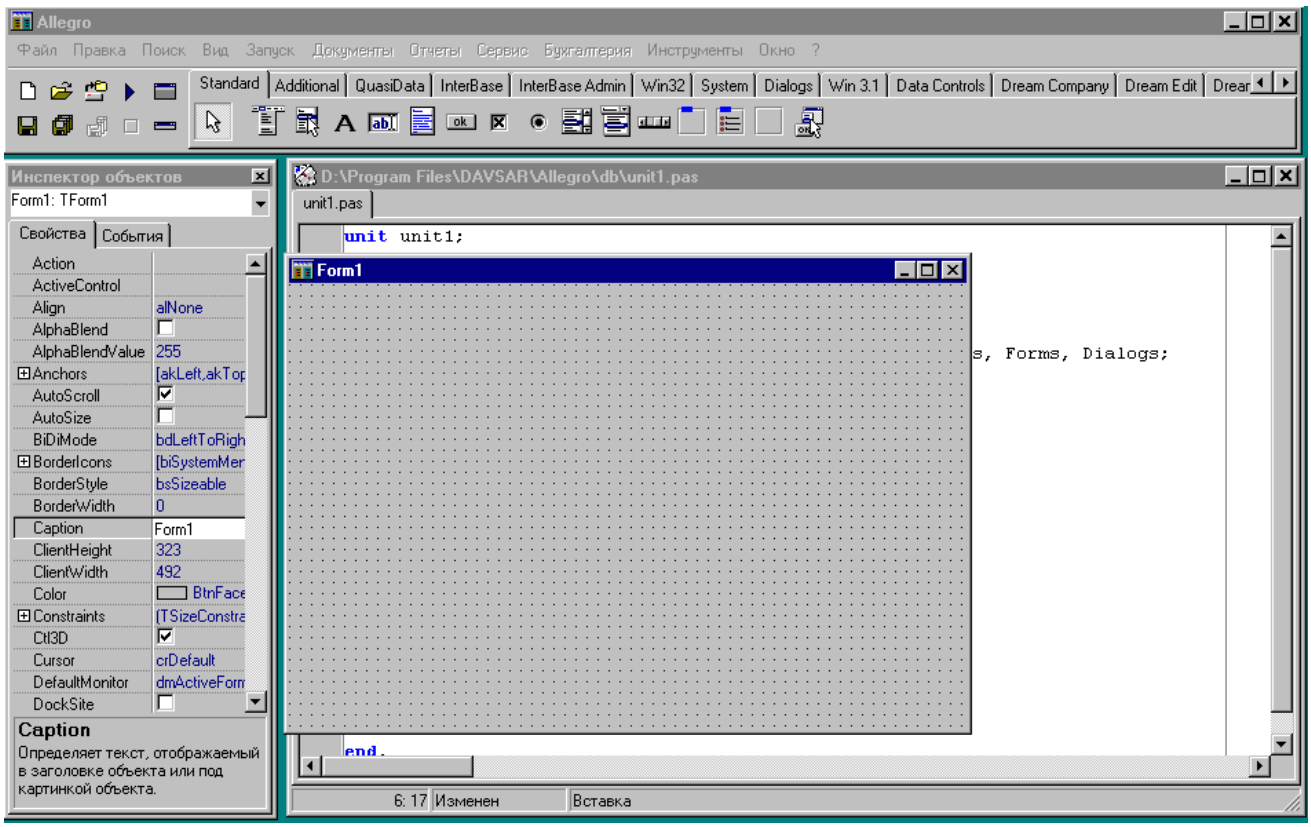
Мы выберем «Delphi приложение» и нажмем кнопку **OK**.



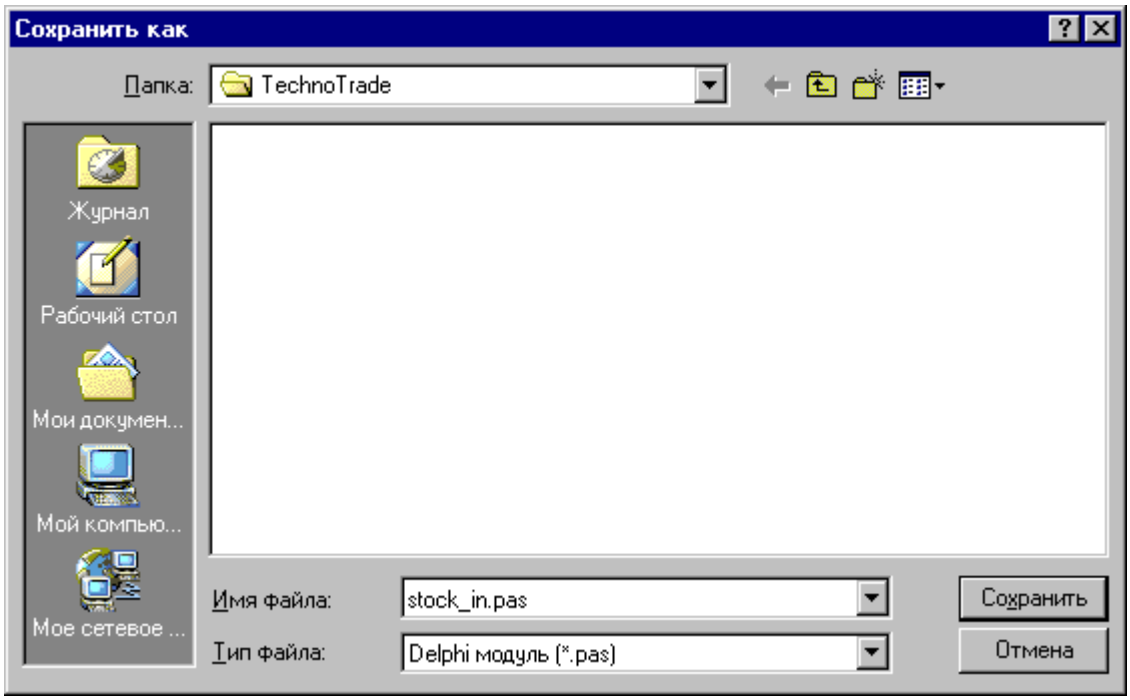
Перед нами появятся:

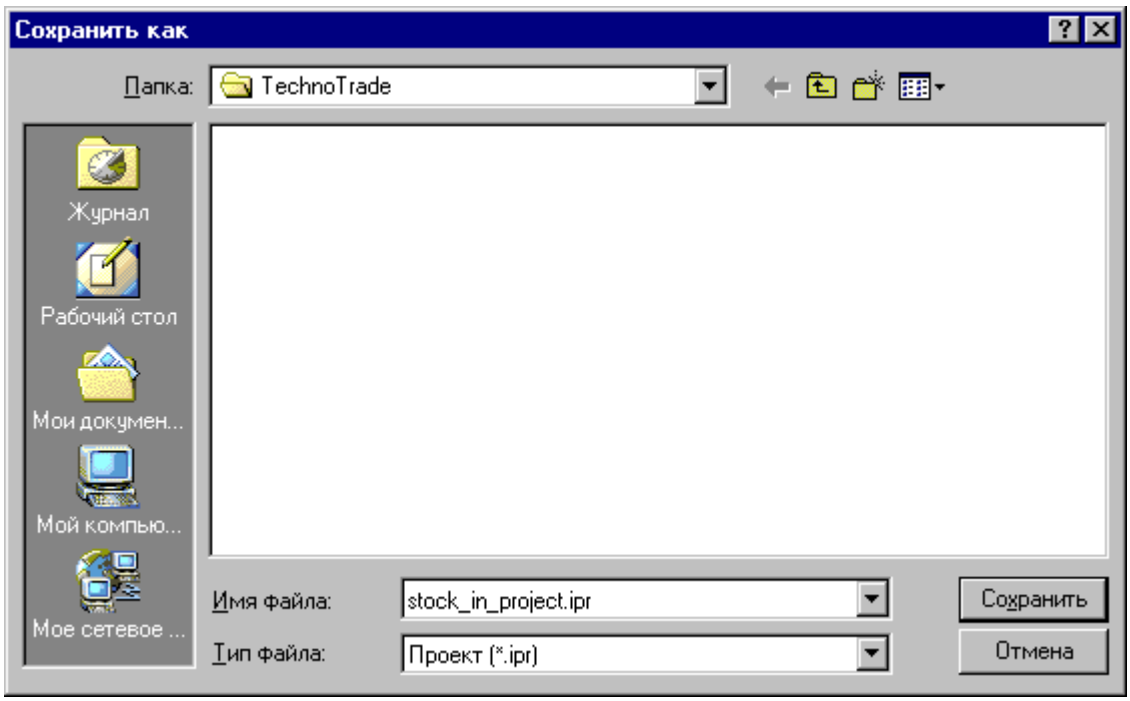
- Инспектор объектов
- Редактор текста с открытой закладкой модуля **Unit1**
- Пустая форма **Form1** с включенной координатной сеткой.

Переключаться между формой и редактором текста удобно клавишей ***F12***, а между формой и инспектором объектов – с помощью клавиши ***F11***.



Мы рекомендуем сразу сохранить проект, придумав названия для файла модуля и файла проекта. Здесь нужно придерживаться какой-то системы. Мы будем называть файл модуля так же, как мы называем тип документа, к которому будет привязан проект, то есть **stock_in.pas** (маленькими буквами). А файл проекта будем называть так же, как файл модуля, но с суффиксом **_project**, то есть в данном случае **stock_in_project.ipr**. Для сохранения проекта воспользуемся пунктом меню **Файл/Сохранить все** или кнопкой с изображением трех дискет. Программа попросит сначала ввести имя файла для модуля с расширением **pas**, а затем имя файла для проекта с расширением **ipr**. Сохранять все файлы мы будем в директоии проектов **\scripts\TechnoTrade**.





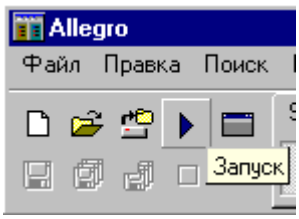
На диске создались три файла: **stock_in.pas, stock_in.dfm, stock_in_project.ipr**. В первом файле хранится текст скриптового модуля, во втором хранятся свойства формы, а в третьем – сведения о проекте.

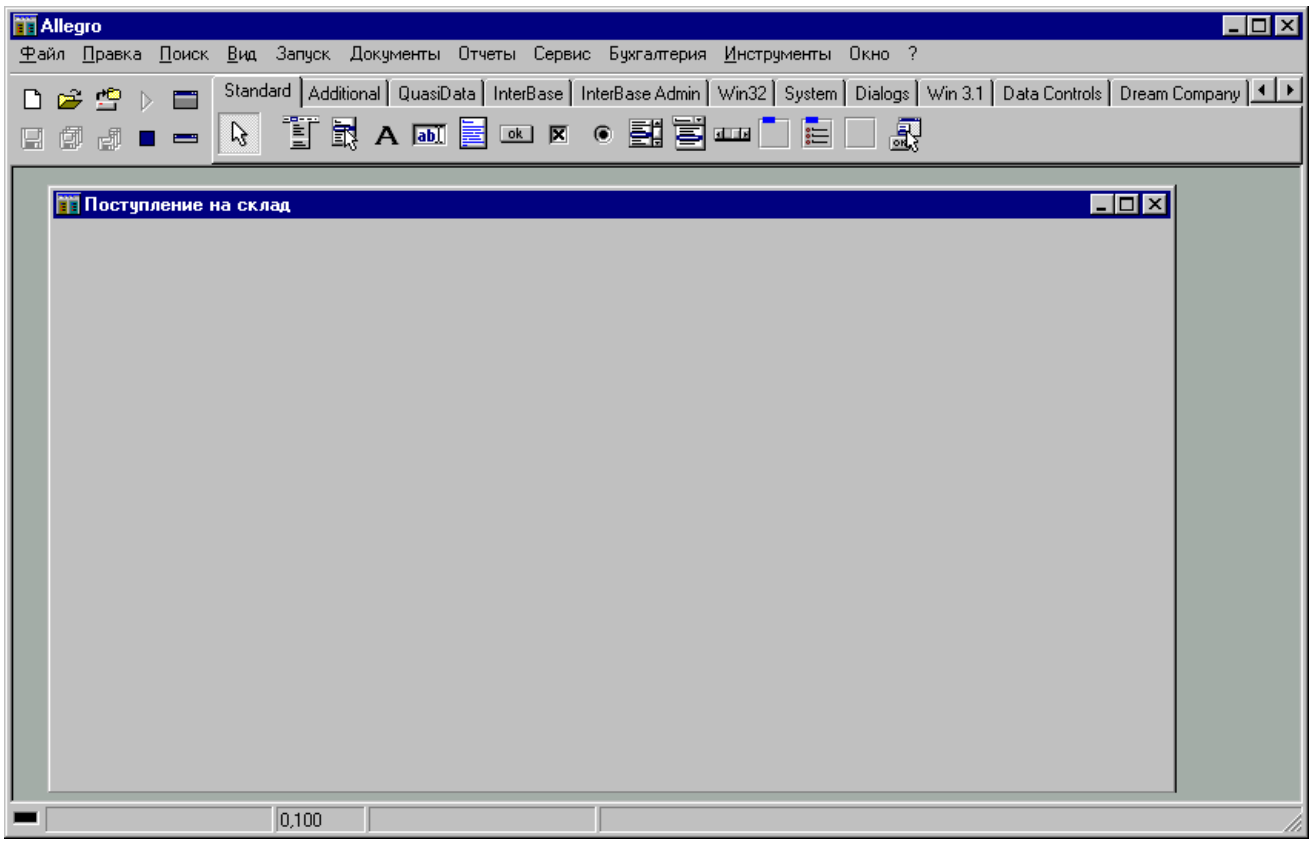
Теперь переименуем форму **Form1** и изменим некоторые его свойства. Для этого в «Инспекторе Объектов» выберем соответствующие свойства и введем значения:

Name	StockInForm
FormStyle	fsMDIChild
Caption	Поступление на склад

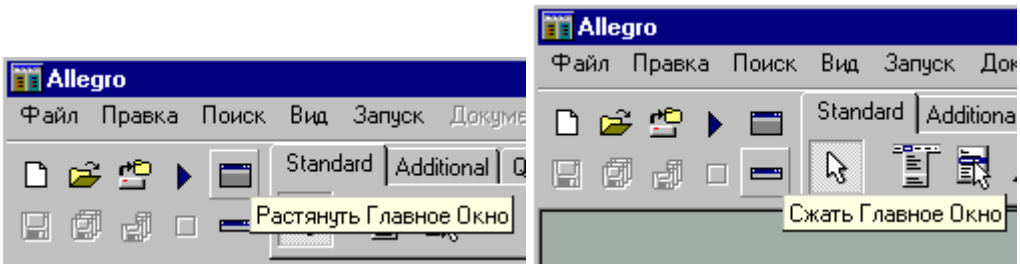
Мы установили стиль окна `FormStyle = fsMDIChild` для того, чтобы организовать многооконный интерфейс. В нем пользователь сможет работать одновременно с несколькими документами на одном экране.

Запустим проект, нажав клавишу ***F9*** или кнопку «Запуск»:





Теперь остановим проект, нажав Ctrl+F2 или кнопку «Остановка программы» под кнопкой «Запуск». Обратим внимание, что при запуске проекта главное окно Allegro «растянулось» и отобразило всю свою рабочую область. А при остановке проекта «сжалось» обратно. Мы в любой момент можем «растянуть» или «сжать» главное окно в режиме дизайна. Для этого можно воспользоваться кнопками «Растянуть Главное Окно» и «Сжать Главное Окно»:

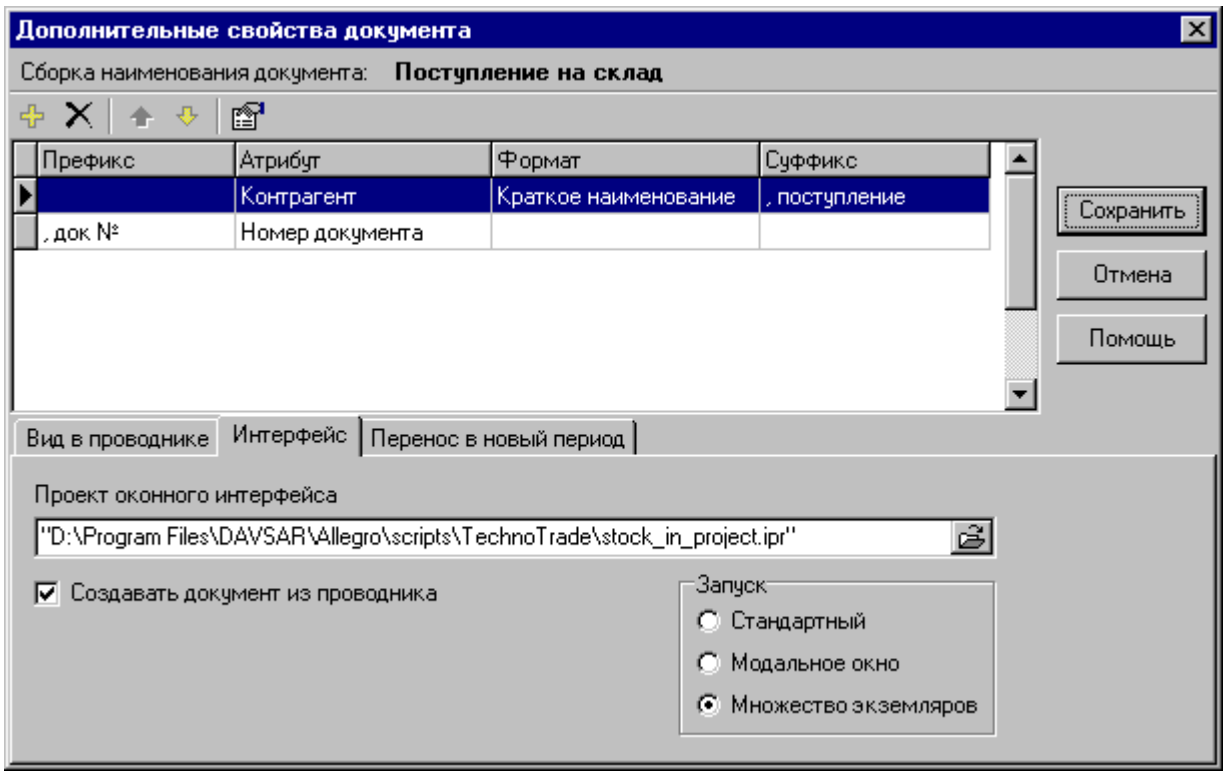


Выйдем из режима дизайна с помощью пункта ***Инструменты/Дизайнер*** Главного меню.

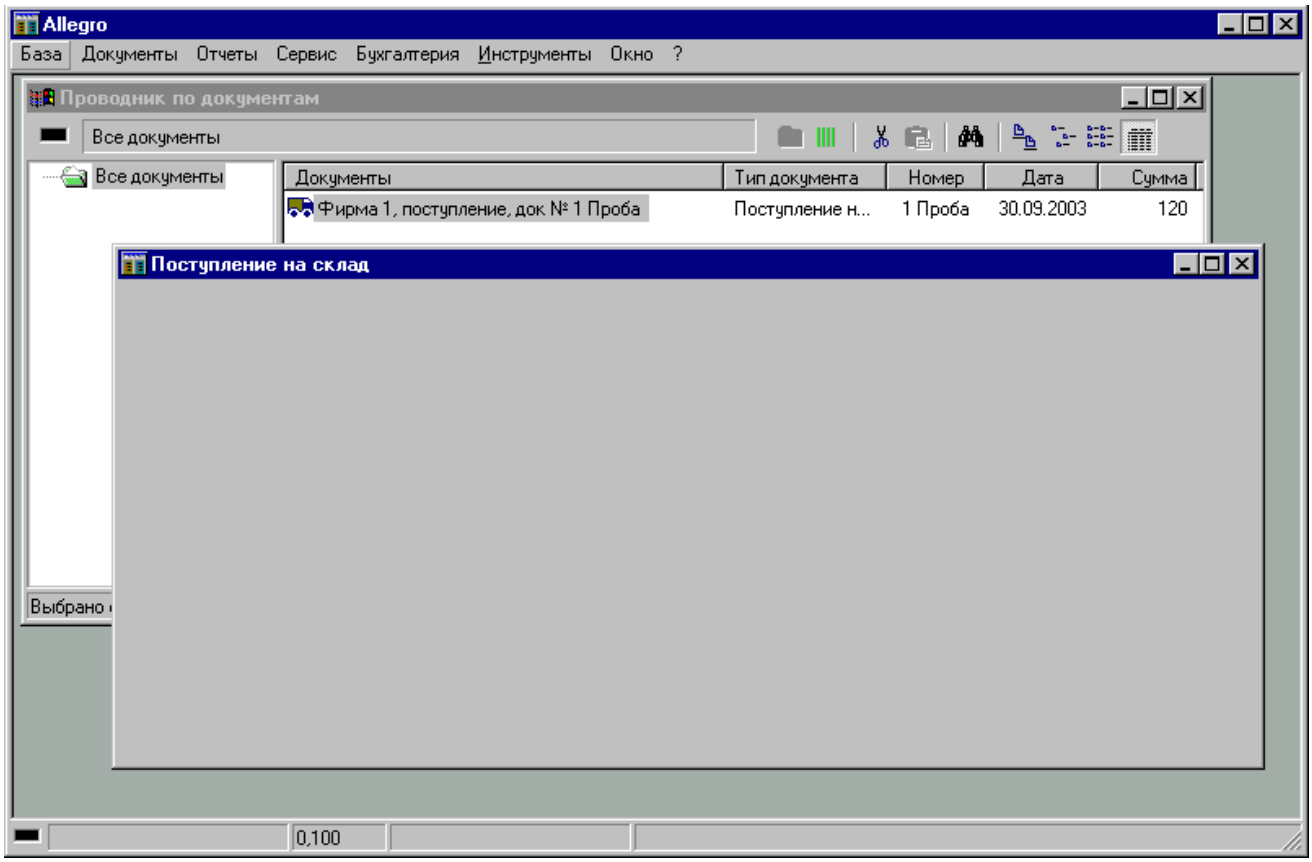
Привязываем проект интерфейса к типу документа

Привяжем проект оконного интерфейса к типу документа «Поступление на склад». Для этого вызовем окно «Метаданные», выберем тип документа «Поступление на склад» и вызовем его «Дополнительные свойства» через пункт контекстного меню ***Дополнительно***.

Выберем закладку «Интерфейс» и укажем имя файла проекта. Установим птичку «Создавать документ из проводника» и режим запуска «Множество экземпляров».



Нажмем кнопку ***Сохранить***. Теперь проект интерфейса привязан к своему типу документов. Откроем «Проводник по документам» и дважды щелкнем на имеющемся у нас документе. Появится окно только что созданного нами интерфейса:



Оконный интерфейс «Поступления на склад», добавляем компоненты.

Войдем в режим «Дизайнер» и откроем проект **stock_in_project.ipr** с помощью меню **Файл/Открыть проект**. Займемся добавлением компонентов на форму **StockInForm**. Компоненты расположены на палитрах с закладками **Standard**, **Additional**, и так далее:

Для того чтобы добавить компонент на форму, нужно сначала выбрать соответствующую палитру, щелкнув на ее закладке, выбрать компонент на палитре одиночным щелчком мыши, а затем щелкнуть однократно на форме в том месте, куда мы желаем поставить компонент. «Перетаскивать» компоненты с палитры на форму **не нужно**!

Для изменения значений свойств компонента, нужно выбрать соответствующий компонент на форме, щелкнув на нем мышью один раз. У компонента появляются квадратные точки по его периметру (с их помощью можно изменять его размеры), а свойства компонента отображаются в Инспекторе объектов.

Хорошим стилем программирования считается назначать свойство **Name** компонентов в соответствие с какой-то системой, а не оставлять их названия по умолчанию, которые присваиваются системой (Button1, Button2, Panel1 и т.д.). Вы можете разработать свой стиль называния компонентов. Мы с вами уже придерживались некоторой системы, дав название форме **StockInForm**, добавив имя класса **Form** в качестве суффикса. Мы будем использовать этот стиль для того, чтобы называть **формы** и **панели**. Для остальных компонентов мы будем использовать префикс из трех-четырех букв, говорящий нам о том, какого рода это компонент. Например, для кнопок будем использовать префикс **btn**, для запросов префикс **qry**, и так далее.

Расположим на форме следующие компоненты последовательно, один за другим, сразу назначая им свойства в Инспекторе объектов:

Панель Panel (палитра Standard)
Align = alTop
Height = 80
Caption = ‘’
Name = TopPanel
BevelOuter = bvNone

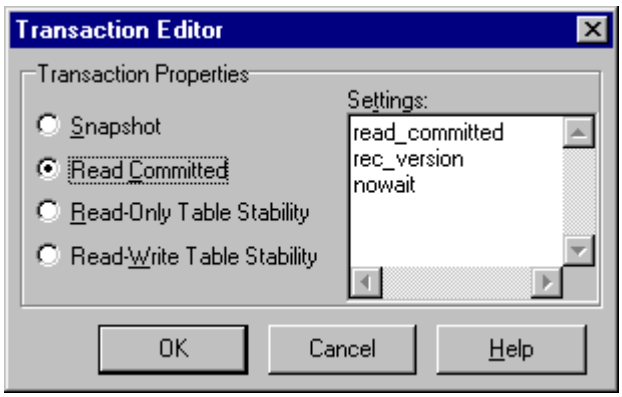
Сетка DBGridA (палитра Allegro)
Align = alTop
Height = 200
Name = dbgDetail

Сплиттер Splitter (палитра Additional)
Align = alTop
AutoSnap = False

Сетка DBGrid (палитра DataControls)
Align = alClient
Name = dbgGoods
Добавим компонент транзакции:

IBTransaction (палитра InterBase)
DefaultAction = taRollBack
DefaultDatabase = MainConnection.MainDatabase
Name = traCurrent

Дважды щелкнем на компоненте транзакции и в появившемся редакторе свойств компонента установим изоляцию транзакций Read Committed:



В режиме изоляции транзакций Read Committed пользователь сможет видеть все изменения, сделанные и подтвержденные в других транзакциях другими пользователями.

Добавим два компонента SQL-запросов. Один для главной таблицы документа, другой для подчиненной:

Компонент SQL-запроса IBDataSet (палитра InterBase)
Name = qryMaster
Transaction = traCurrent

Компонент SQL-запроса IBDataSet (палитра InterBase)
Name = qryDetail
Transaction = traCurrent

Добавим два компонента **DataSource** с палитры **InterBase**. Если там Вы не можете найти этот класс компонентов, то поищите его на палитре **DataAccess**:

Компонент DataSource (палитра InterBase)
Name = dsrMaster
DataSet = qryMaster

Компонент DataSource (палитра InterBase)
Name = dsrDetail
DataSet = qryDetail

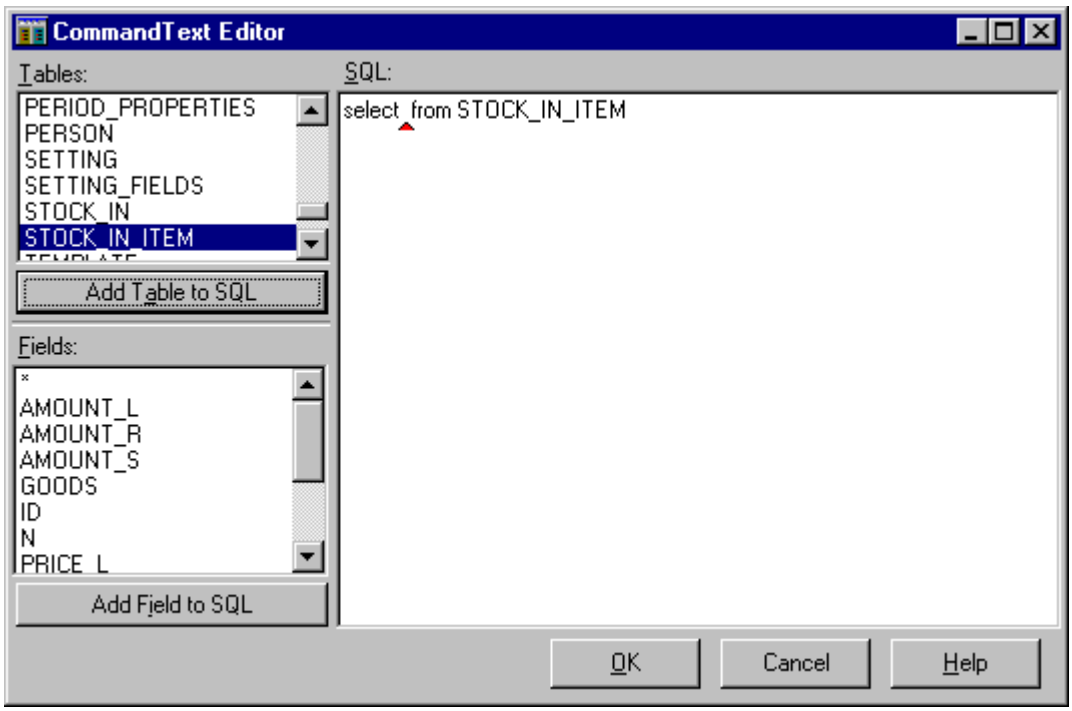
Дело в том, что экранные компоненты (сетки, и т.п.) не работают с компонентами запросов напрямую, а подключаются к ним через компоненты класса TDataSource. Это особенность реализации VCL-библиотеки Delphi.

Назначим у сетки **dbgDetail** свойство **`DataSource = dsrDetail`**.

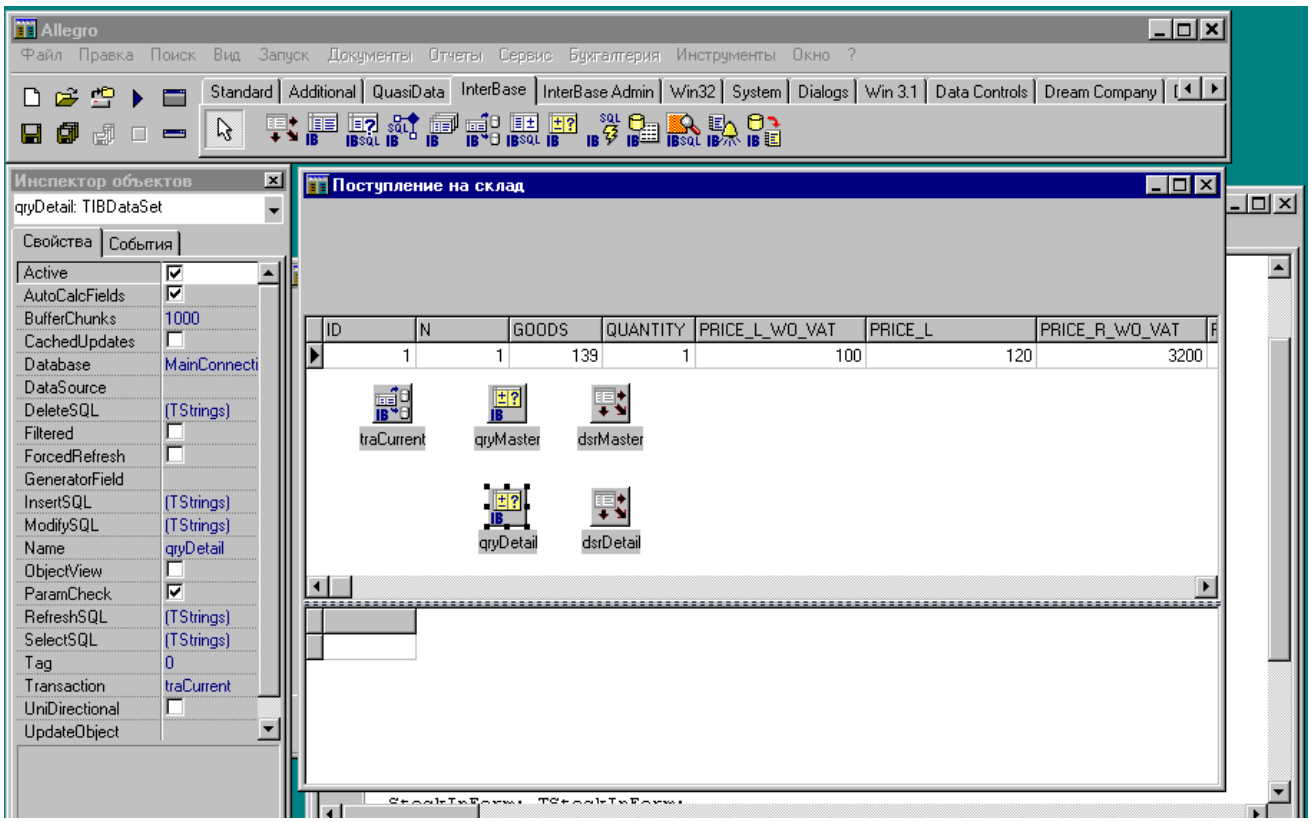
Компоненты запросов с палитры InterBase работают с сервером Firebird напрямую. Практически вся работа с базой данных осуществляется при помощи этих компонентов. Поэтому Вы должны хорошо изучить, как ими пользоваться. В большинстве случаев нам достаточно будет компонента IBDataSet. Этот компонент имеет 5 свойств, в которых хранятся тексты SQL-запросов (SelectSQL, InsertSQL, ModifySQL, DeleteSQL, RefreshSQL). В зависимости от состояния, в котором находится компонент, он посылает соответствующие SQL-запросы на сервер. С помощью запроса, записанного в SelectSQL, например, компонент получает набор данных с сервера.

Посмотрим, как это происходит.

Выберем компонент запроса **qryDetail** Дважды щелкнем на его свойстве **SelectSQL** в «Инспекторе объектов». Выберем в появившемся диалоге в верхнем списке таблицу **STOCK_IN_ITEM** и нажмем кнопку **Add Table to SQL**:



Вставим звездочку * между словами **select** и **from** и нажмем кнопку **OK**. Фактически мы вписали текст простейшего SQL-запроса, с помощью которого можно получить данные из подчиненной таблицы документа. В дальнейшем мы усложним этот текст, а пока давайте активизируем запрос, установив у компонента **qryDetail** в Инспекторе объектов свойство **Active = True**.



Мы видим, что в результате выполненного SQL-запроса, компонент получил с сервера набор данных и отобразил его в сетке. Сохраним все. Запустим проект с помощью ***F9***. Конечно, это еще мало похоже на готовый интерфейс ввода данных, но зато это уже работает! Для того чтобы довести это окно до «товарного вида» нам еще предстоит внести в него целый ряд усовершенствований…

Дорабатываем SQL-запросы позиций и шапки документа

Запрос, записанный нами в свойстве **SelectSQL** компонента **qryDetails** извлекает данные только из таблицы STOCK_IN_ITEM. В этой таблице вместо наименований товара хранятся их внутренние идентификаторы ID в поле GOODS Для того чтобы видеть не внутренние номера, а наименования товаров, нам нужен запрос, объединяющий таблицу STOCK_IN_ITEM с таблицей наименований объектов OBJECT_NAMES. К тому же мы хотим, чтобы записи отображались в определенном порядке. Поэтому изменим у компонента **qryDetails** свойство **SelectSQL**, вписав в него такой SQL-запрос:

```
select
  SI.ID,
  SI.N,
  SI.GOODS,
  O.SHORT_NAME ITEM_NAME,
  SI.QUANTITY,
  SI.PRICE_L,
  SI.PRICE_L_WO_VAT,
  SI.PRICE_R,
  SI.PRICE_R_WO_VAT,
  SI.AMOUNT_L,
  SI.AMOUNT_R,
  SI.AMOUNT_S
from
  STOCK_IN_ITEM SI,
  OBJECT_NAMES O
where
  SI.GOODS = O.OBJECT_ID
order by SI.N
```

В данном случае выборка происходит из 2 таблиц, каждой из них присвоено по псевдониму (SI и O), что сокращает и делает более читабельным тест запроса. Таблицы связаны условием:

```
SI.GOODS = O.OBJECT_ID
```

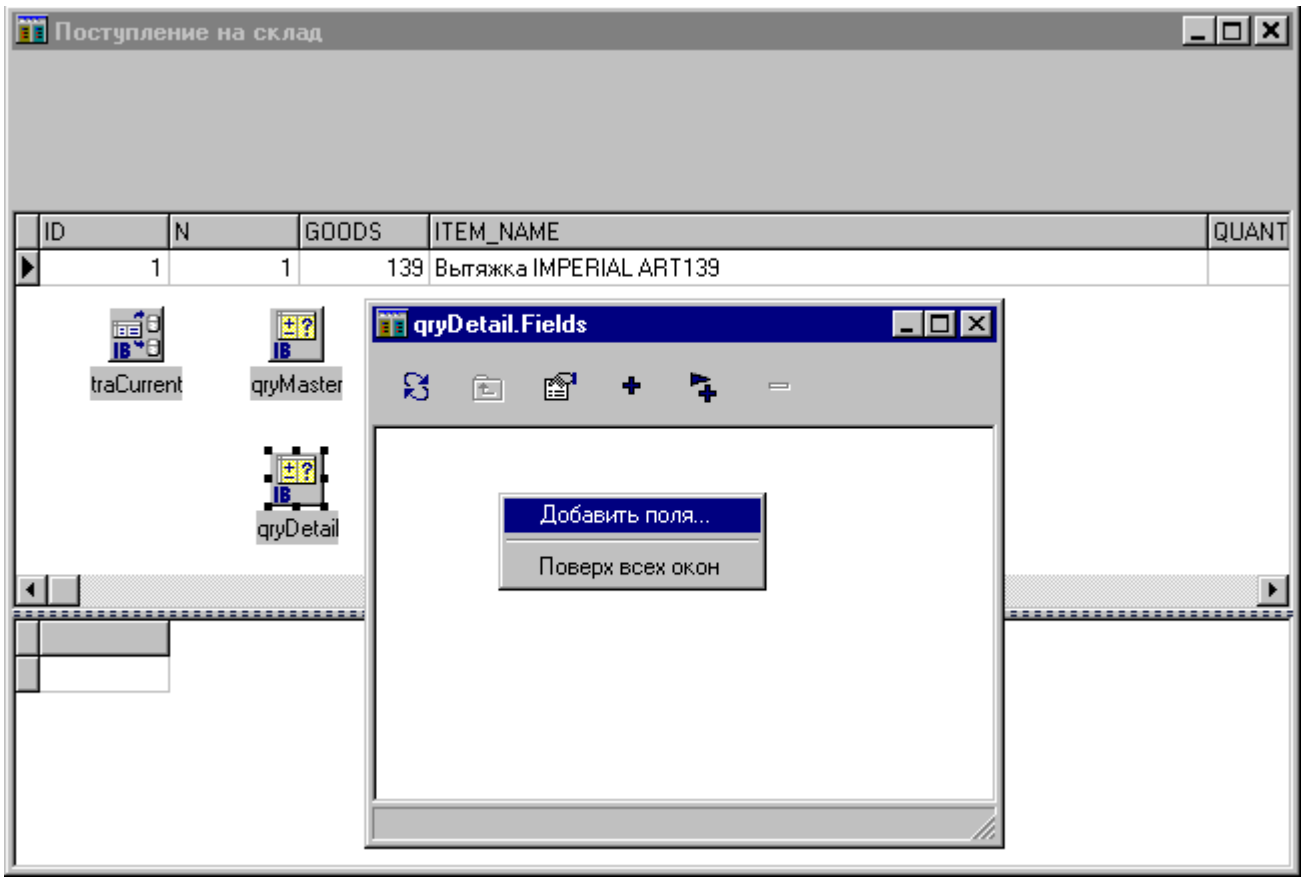
Результирующий набор упорядочен по полю N фразой:

```
order by SI.N
```

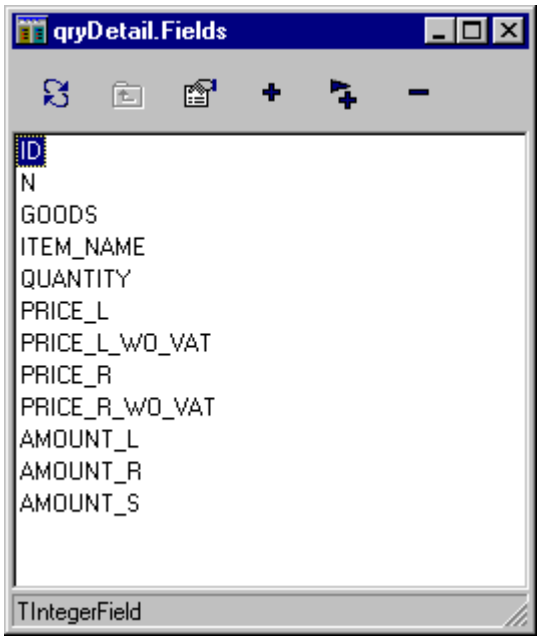
Активизируем запрос, установив свойство **Active = True**. Теперь мы видим наименование товара в наборе данных. Хорошо бы еще присвоить русские названия колонкам этого набора и скрыть некоторые внутренние и ненужные будущему пользователю колонки, например, ID и N. Компонент класса TIBDataSet работает с набором данных при помощи так называемых компонентов-полей, потомков класса TField библиотеки визуальных компонентов VCL Delphi. Можно создать «постоянные» (persistent) поля вручную на стадии разработки интерфейса и придать им нужные свойства. Если «постоянные» поля не были созданы, то компонент TIBDataSet создает вместо них «временные» поля сразу после того, как будет активизирован запрос SELECT. То, что мы видим в сетке сейчас - это «временные» поля, созданные автоматически. Их заголовки нас явно не устраивают.

Давайте создадим список полей вручную.

Для этого нужно дважды щелкнуть мышью на компоненте **qryDetail**. Появится редактор полей, пока пустой, так как «постоянных» полей у компонента еще нет. Вызовем правой кнопкой мыши контекстное меню и выберем пункт **Добавить поля…** Нам будет предложено выбрать какие-то поля из списка всех потенциальных полей, которые присуствуют в этом наборе данных, причем по умолчанию предлагается выбрать их все. Так и поступим, нажав **OK**.



«Постоянные» поля теперь созданы в списке и видны в редакторе полей:



Выбирая какие-то из них в редакторе полей, мы можем присвоить значения их свойствам в Инспекторе объектов. Параллельно мы можем следить за тем, как изменяется внешний вид нашей сетки.

У полей **ID**, **N, GOODS, PRICE_R, PRICE_R_WO_VAT** и **AMOUNT_R** установим значение свойства

```
Visible = False
```

Это отключит отображение соответствующих полей в сетке.

Изменим заголовки некоторых полей, присваивая свойству **DisplayLabel** новое значение:

ITEM_NAME	Наименование
QUANTITY	Кол-во
PRICE_L	Цена
PRICE_L_WO_VAT	Цена б/НДС
AMOUNT_L	Сумма
AMOUNT_S	Сумма, USD

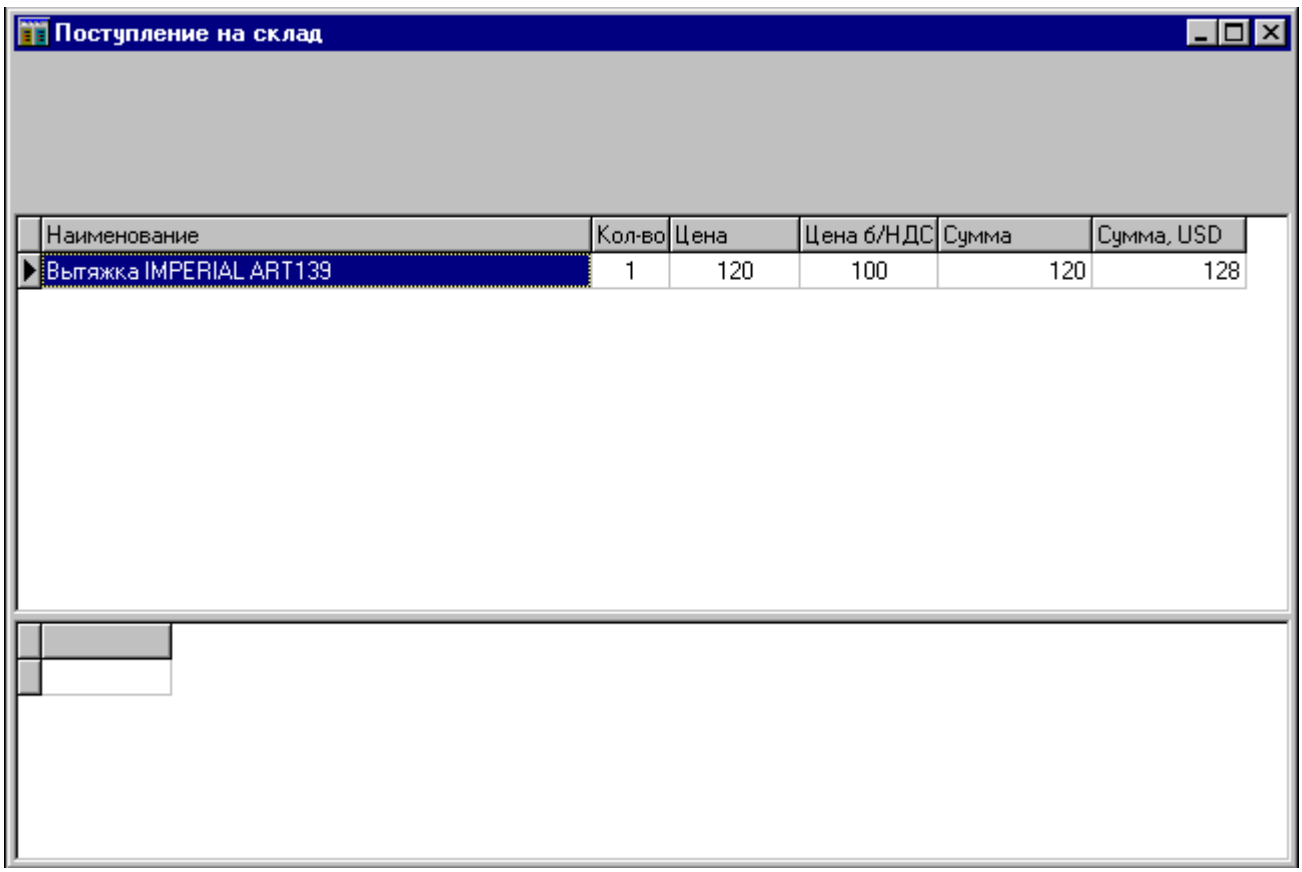
Изменим ширину отображения некоторых полей, изменяя значение свойства **DisplayWidth**:

ITEM_NAME	45
QUANTITY	5
PRICE_L	10
PRICE_L_WO_VAT	10
AMOUNT_L	12
AMOUNT_S	12

Изменим выравнивание текста в полях **QUANTITY, PRICE_L, PRICE_L_WO_VAT**, установив для них всех:

```
Alignment = taCenter
```

Закроем редактор полей. Сохраним и запустим проект (F9). Сейчас наш интерфейс выглядит так:



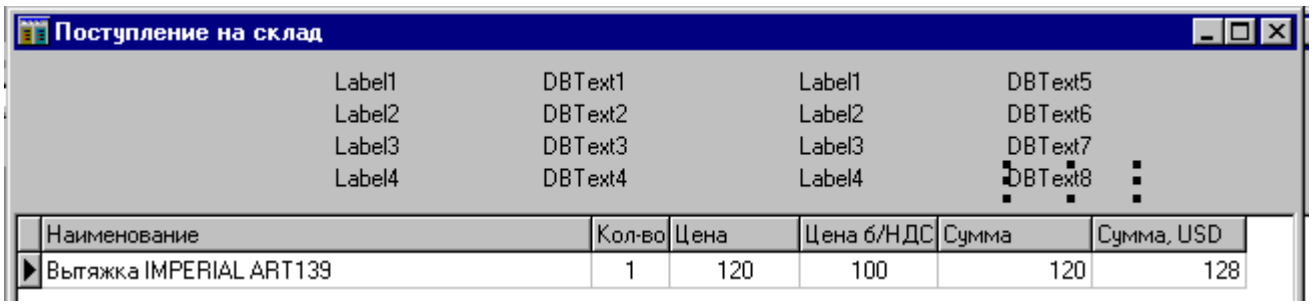
Пока что данные в сетке не редактируются, так как не созданы запросы UPDATE и DELETE.

Займемся пока текстом запроса шапки документа. У компонента **qryMaster** в свойство **SelectSQL** впишем такой текст:

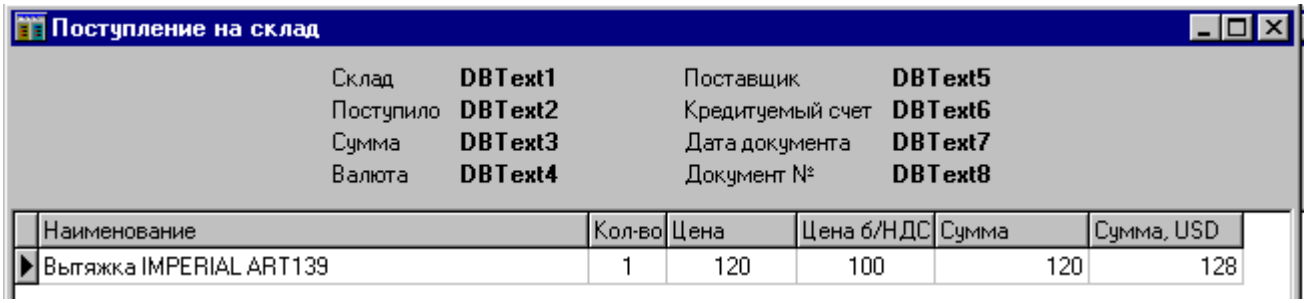
```
select
  SI.ID,
  SI.DIR_ID,
  SI.DOC_DATE,
  SI.DOC_KIND,
  SI.DOC_NO,
  SI.ENTRY_DATE,
  SI.HAS_ENTRY,
  SI.STOCK_ACC,
  A1.NAME STOCK_ACC_NAME,
  SI.LAYER_ID,
  L.SHORT_NAME LAYER_NAME,
  SI.VAT_RATE,
  SI.ACC_ID,
  A2.NAME CREDIT_ACC_NAME,
  SI.CONTRAGENT,
  O.SHORT_NAME CONTRAGENT_NAME,
  SI.CALC_MODE,
  SI.EXCH_RATE_L,
  SI.EXCH_RATE_S,
  SI.TOTAL_L,
  SI.TOTAL_R,
  SI.TOTAL_S
from
  STOCK_IN SI,
  OBJECT_NAMES O,
  LAYER L,
  ACC A1,
  ACC A2
where
  SI.CONTRAGENT = O.OBJECT_ID and
  SI.LAYER_ID = L.LAYER_ID and
  SI.STOCK_ACC = A1.ACC_ID and
  SI.ACC_ID = A2.ACC_ID
```

Здесь мы объединяем пять таблиц: главную таблицу документа «Поступление на склад» STOCK_IN, таблицу наименований объектов OBJECT_NAMES, таблицу слоев LAYER и дважды таблицу счетов ACC под разными псевдонимами (ACC1 и ACC2). Создадим «постоянные» поля дважды щелкнув на компоненте **qryMaster** и добавив все возможные поля в список.

Добавим на верхнюю панель формы надписи с названиями полей и компоненты для отображения данных шапки. Для этого используем компоненты **Label** с палитры **Standard** и **DBText** с палитры **DataControls**.



Впишем свойства **Caption** компонентов типа **TLabel** названия полей самых важных полей шапки документа:



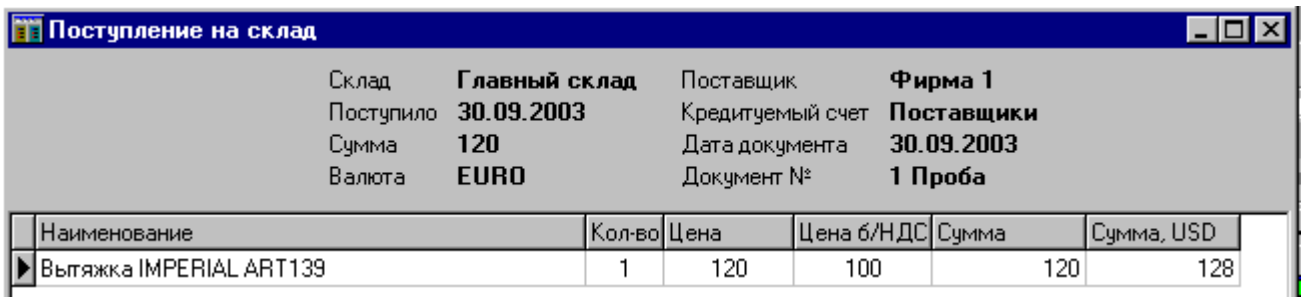
Выберем все компоненты DBText, держа нажатой клавишу Shift и щелкая по ним мышью. В Инспекторе объектов назначим им всем одновременно свойства:

```
AutoSize = True
DataSource = dsrMaster
Font.Style = [fsBold]
```

Свойства **Name** всех этих компонентов изменять не будем. А свойствам **DataField** назначим имена полей запроса:

DBText1	STOCK_ACC_NAME
DBText2	ENTRY_DATE
DBText3	TOTAL_L
DBText4	LAYER_NAME
DBText5	CONTRAGENT_NAME
DBText6	CREDIT_ACC_NAME
DBText7	DOC_DATE
DBText8	DOC_NO

Активизируем запрос **qryMaster**.



Итак, запросы SELECT для шапки и для позиций у нас почти готовы. Единственное, чего в них недостает, так это возможности запрашивать информацию а каком-то конкретном документе, с конкретным ID. Если бы документов было много, то мы сейчас видели бы их все позиции одновременно в нашей сетке **dbgDetail**. Мы не хотели с самого начала отягощать и без того сложную для многих читателей тему создания SQL-запросов всеми подробностями, и потому вносим все необходимые уточнения постепенно по ходу изложения.

Итак, если у нас имеется запрос вида:

```
select * from stock_in_item
```

То он возвратит нам все строки таблицы stock_in_item. Для того чтобы получить только те строки, которые относятся к документу с ID = 1050, мы могли бы написать такой запрос:

```
select * from stock_in_item
where ID = 1050
```

Однако такой подход требует каждый раз менять текст запроса, если мы хотим запросить тот или иной документ. Поэтому существует более гибкий механизм – параметризованные запросы. Для того чтобы написать параметризованный запрос в компонент типа TIBDataSet, мы должны включить в его текст переменную, предваряемую двоеточием, и называемую **параметром**, например:

```
select * from stock_in_item
where ID = :A
```

Прежде чем открыть запрос, мы должны будем сообщить компоненту значение параметра **A**. Делается это обычно во время выполнения программы (run-time) с помощью вызова метода **ParamByName**.

Добавим в каждый наш SQL-запрос параметр, с помощью которого будем передавать ID документа. Для этого в секцию **WHERE** к имеющимся условиям нужно добавить условие:

```
SI.ID = :ID
```

Это условие ограничит набор только строками, в которых значение поля ID равно значению параметра :ID.
Итак, окончательно текст запроса в свойстве **qryMaster.SelectSQL** выглядит так:

```
select
  SI.ID,
  SI.DIR_ID,
  SI.DOC_DATE,
  SI.DOC_KIND,
  SI.DOC_NO,
  SI.ENTRY_DATE,
  SI.HAS_ENTRY,
  SI.STOCK_ACC,
  A1.NAME STOCK_ACC_NAME,
  SI.LAYER_ID,
  L.SHORT_NAME LAYER_NAME,
  SI.VAT_RATE,
  SI.ACC_ID,
  A2.NAME CREDIT_ACC_NAME,
  SI.CONTRAGENT,
  O.SHORT_NAME CONTRAGENT_NAME,
  SI.CALC_MODE,
  SI.EXCH_RATE_L,
  SI.EXCH_RATE_S,
  SI.TOTAL_L,
  SI.TOTAL_R,
  SI.TOTAL_S
from
  STOCK_IN SI,
  OBJECT_NAMES O,
  LAYER L,
  ACC A1,
  ACC A2
where
  SI.CONTRAGENT = O.OBJECT_ID and
  SI.LAYER_ID = L.LAYER_ID and
  SI.STOCK_ACC = A1.ACC_ID and
  SI.ACC_ID = A2.ACC_ID and
  SI.ID = :ID
```

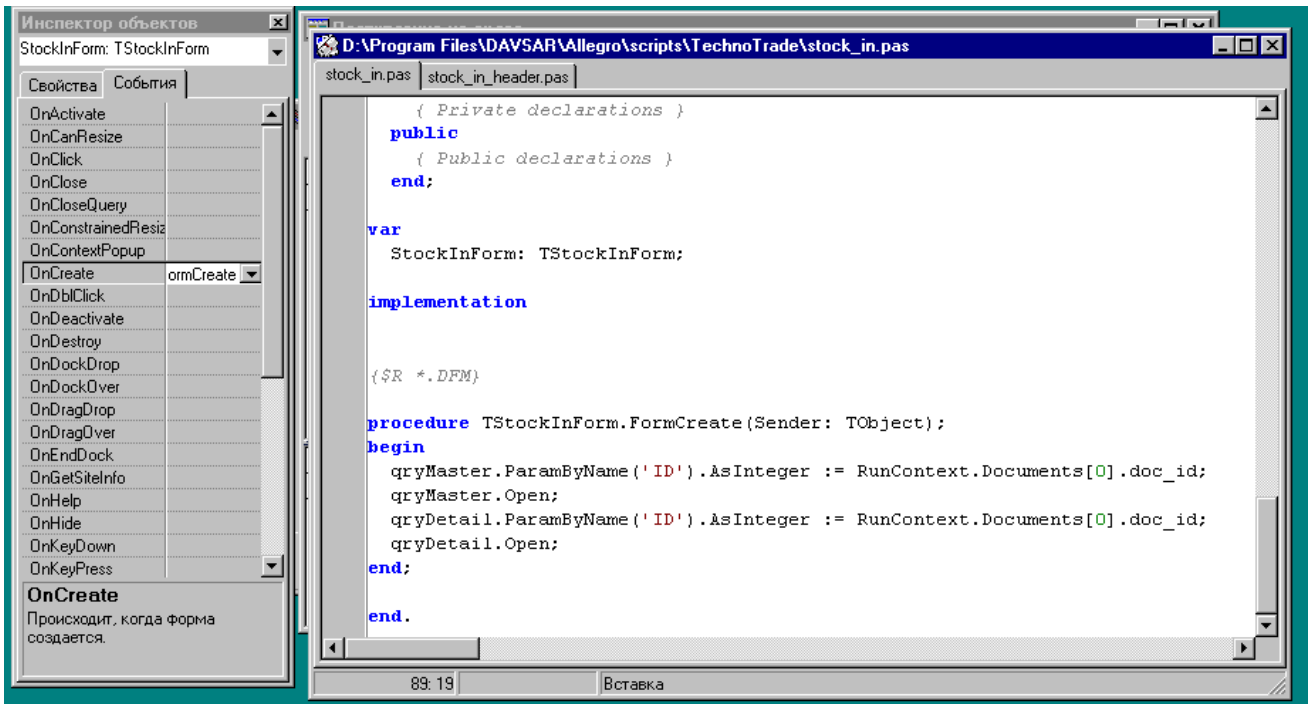
Аналогично добавим такое же условие в компонент запроса позиций. Окончательный текст в свойстве **qryDetail.SelectSQL** должен выглядеть так:

```
select
  SI.ID,
  SI.N,
  SI.GOODS,
  O.SHORT_NAME ITEM_NAME,
  SI.QUANTITY,
  SI.PRICE_L,
  SI.PRICE_L_WO_VAT,
  SI.PRICE_R,
  SI.PRICE_R_WO_VAT,
  SI.AMOUNT_L,
  SI.AMOUNT_R,
  SI.AMOUNT_S
from
  STOCK_IN_ITEM SI,
  OBJECT_NAMES O
where
  SI.GOODS = O.OBJECT_ID and
  SI.ID = :ID
order by SI.N
```

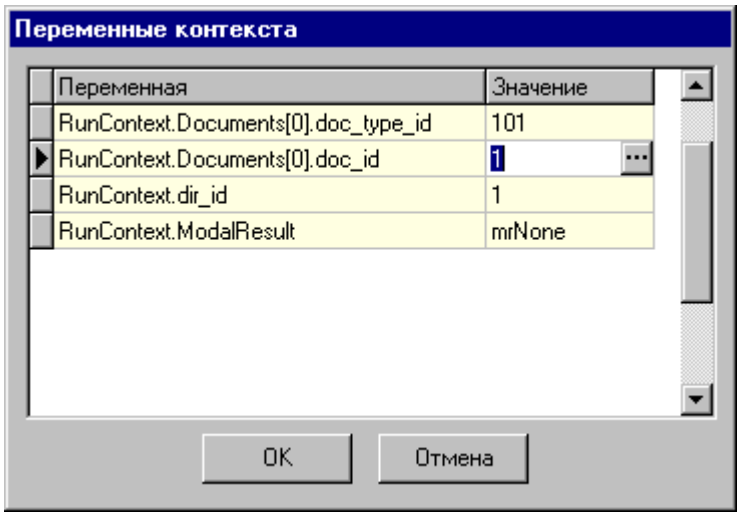
После внесения такого изменения в тексты запросов, если их теперь активизировать в режиме дизайна, то они будут отображать пустой набор, так как значение параметра ID не определено.

Теперь займемся передачей параметра ID. Выберем форму StockInForm, откроем закладку «События» в Инспекторе объектов и дважды щелкнем на событии **OnCreate**. В код программы вставится обработчик события. Впишем в него следующий текст:

```
procedure TStockInForm.FormCreate(Sender: TObject);
begin
  qryMaster.ParamByName('ID').AsInteger := RunContext.Documents[0].doc_id;
  qryMaster.Open;
  qryDetail.ParamByName('ID').AsInteger := RunContext.Documents[0].doc_id;
  qryDetail.Open; //открываем запрос позиций
end;
```



Смысл этого текста мы поясним позже, а пока назначим переменные контекста с помощью пункта меню **Запуск/Переменные контекста**. В появившемся диалоге установим значения первых двух величин, с помощью кнопочек с многоточием. Выберем в предложенных диалогах тип документа «Поступление на склад» и тот документ, что у нас имеется.



Установим свойство обоих запросов **Active = False**.

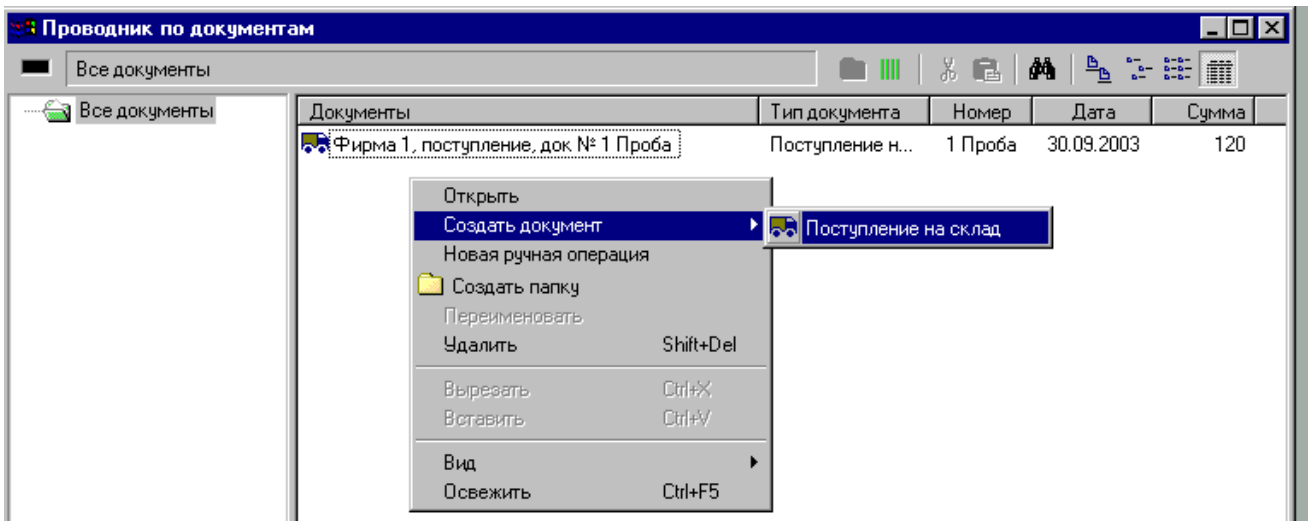
Запустим проект, нажав ***F9***.

Итак, у нас все работает. Выйдем из дизайнера и попробуем запустить проект из «Проводника по документам», щелкнув дважды на названии документа. А теперь, используя пункт контекстного меню «Создать» «Проводника», попробуем создать «Поступление на склад»:

Появится еще одно окно «Поступление на склад», но с пустым набором.

При вызове каждого экземпляра интерфейса документа на экран, программа создает специальный компонент **RunContext**, в котором помещает информацию о том, какой тип документа **doc_tye_id** вызван на экран и каков его **doc_id**. Собственно, мы этой информацией об **doc_id** и воспользовались, когда передавали значение параметра **`ID`** в запросы.

При создании нового документа программа Allegro присваивает **doc_id** отрицательное значение:

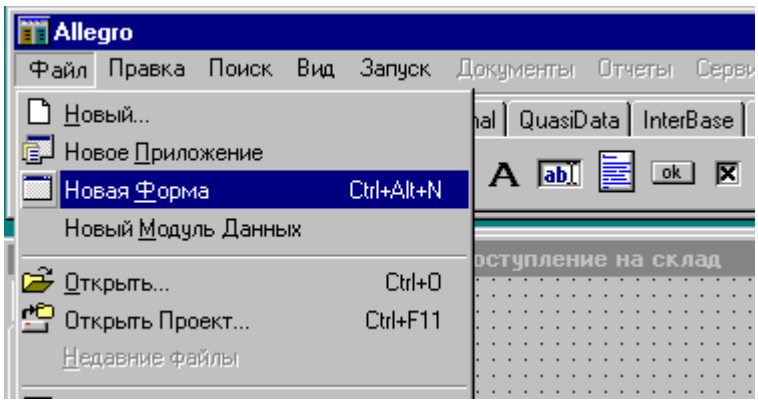


RunContext.Documents[0].doc_id = -1

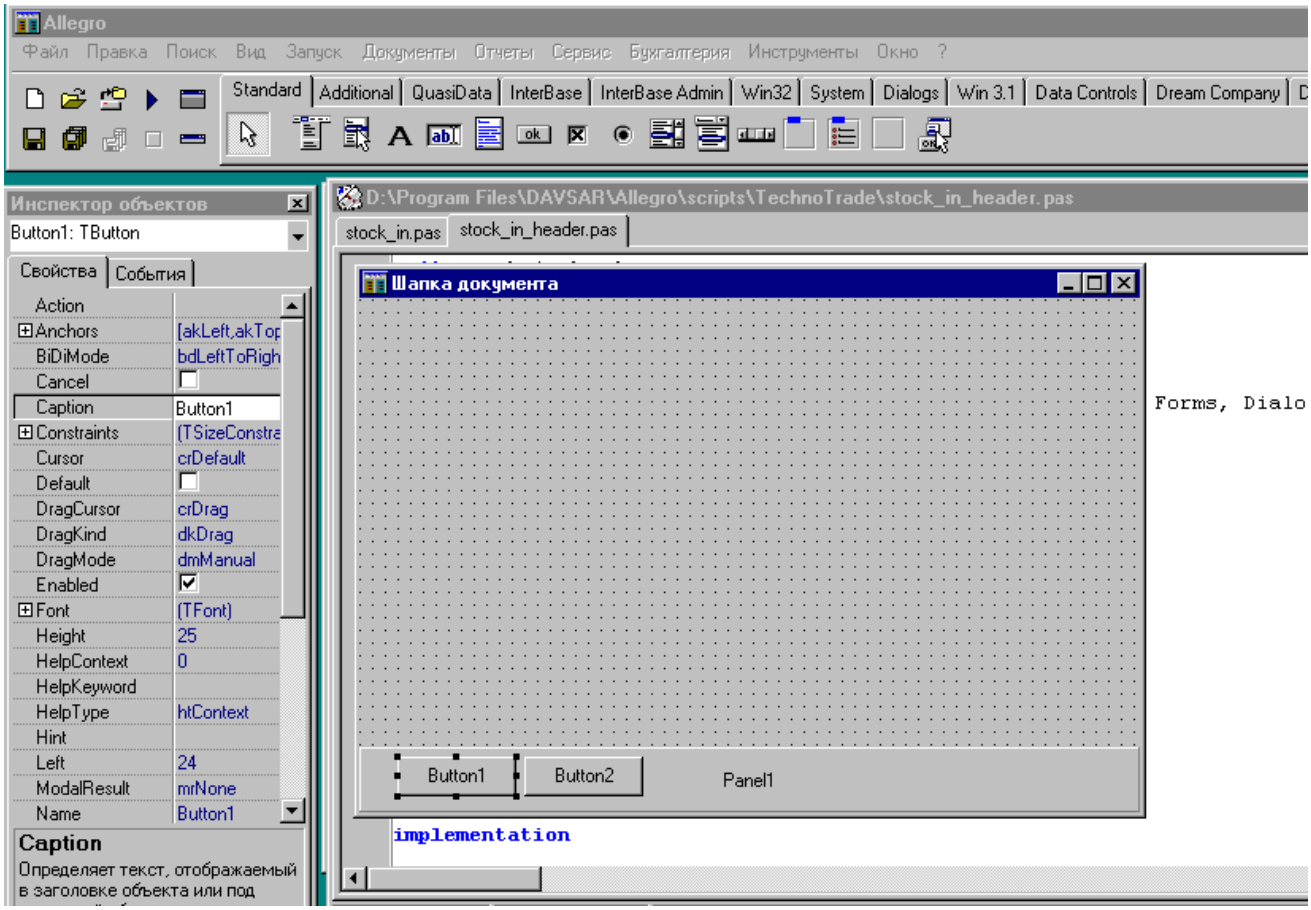
После того, как новый документ будет реально создан в базе данных, мы **обязаны** присвоить свойству **RunContext.Documents[0].doc_id** значение **ID созданного документа**. Это нужно для того, чтобы программа могла различать в многооконном интерфейсе открытые документы и не вызвала один и тот же документ дважды (в двух разных окнах). В принципе один проект оконного интерфейса может обслуживать одновременно несколько документов. Например, заказ и связанные с ним размещения и поставки. Для этого объект **RunContext** имеет свойство-массив **RunContext.Documents.** В простейшем случае используется один элемент массива с индексом 0. Подразумевается, что при вызове проекта оконного интерфейса всегда задействуется хотя бы один какой-то конкретный документ.

Добавляем окно редактирования шапки. SQL-запрос для выпадающего списка

Закроем все окна и перейдем в режим «Дизайнер».
Добавим в проект еще одну форму с помощью **Файл/Новая форма**.



Эта форма нам понадобится для редактирования шапки документа. Изменим свойство Name формы с **Form1** на **StockInHeaderForm**, а в свойство **Caption** впишем 'Шапка документа' . Сохраним модуль под именем **stock_in_header.pas**. Добавим компонент **Panel** с палитры **Standard**, установим в Инспекторе объектов его свойство **Align** в состояние **alBottom**. Панель прижмется к нижнему краю окна. Разместим на этой панели два компонента **Button** с палитры **Standard**.



Изменим свойства **Caption** кнопок **Button1** и **Button2** на **Сохранить** и **Отмена**, а в свойстве **Caption** панели сотрем текст, чтобы не отображать ее заголовок вообще. Присвоим значения еще некоторым свойствам:

Button1
Name = btnOK
Caption = OK

Button2
Name = btnCancel
Caption = Отмена
ModalResult = mrCancel
Cancel = True

Panel1
Name = ButtonPanel
BevelOuter = bvNone (уберем фаску)
Caption = ‘’

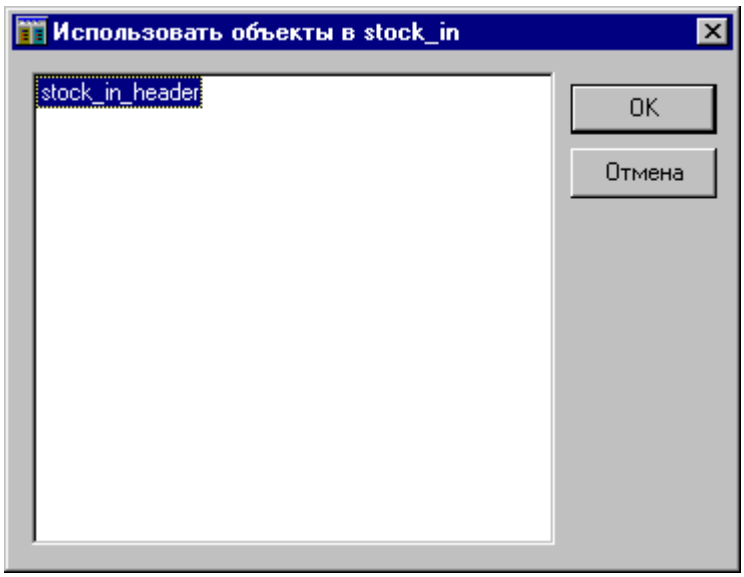
Значение **ModalResult** = **mrCancel** у второй кнопки означает, что при нажатии кнопки форма закроется, если она находилась в так называемом модальном режиме отображения. Свойство **Cancel** = **True** той же кнопки означает, что нажатие клавиши Escape на клавиатуре будет восприниматься так же, как если бы пользователь нажал эту кнопку.

Для того чтобы форма StockInFormHeader отображалась как диалог (с фиксированным размером) и в центре экрана, выберем ее и установим еще два ее свойства:

BorderStyle = bsDialog
Position = poScreenCenter

Для того чтобы программный модуль добавленной формы проекта «видел» переменные модуля главной формы, нужно, чтобы в секции **uses** раздела **implementation** было указано имя модуля главной формы. Это

объявление можно вписать вручную, а можно и с помощью меню **Файл/Использовать объект**. То же самое нужно сделать в модуле главной формы stock_in по отношению к модулю stock_in_header.



Теперь нужно организовать вызов формы для редактирования шапки из главной формы проекта. Вызов формы в модальном режиме (в режиме диалога) осуществляется с помощью метода **ShowModal** этой формы. Для того чтобы вызвать этот метод нам понадобится кнопка или какой-то другой орган управления, который пользователь мог бы использовать. Удобно для решения подобных задач использовать компонент **ActionList** с палитры **Standard**. Этот компонент позволяет составить список всех команд, которые могут понадобиться пользователю, а затем подключить к этим командам разные органы управления. Например, создав команду «Шапка…» можно подключить к ней кнопку, пункт контекстного меню и назначить горячую клавишу. Тогда пользователь сможет вызывать эту команду тем способом, который ему покажется более удобным. Централизованный список команд ActionList позволяет назначить каждой команде обработчик, заголовок, подсказку, пиктограмму и горячую клавишу.

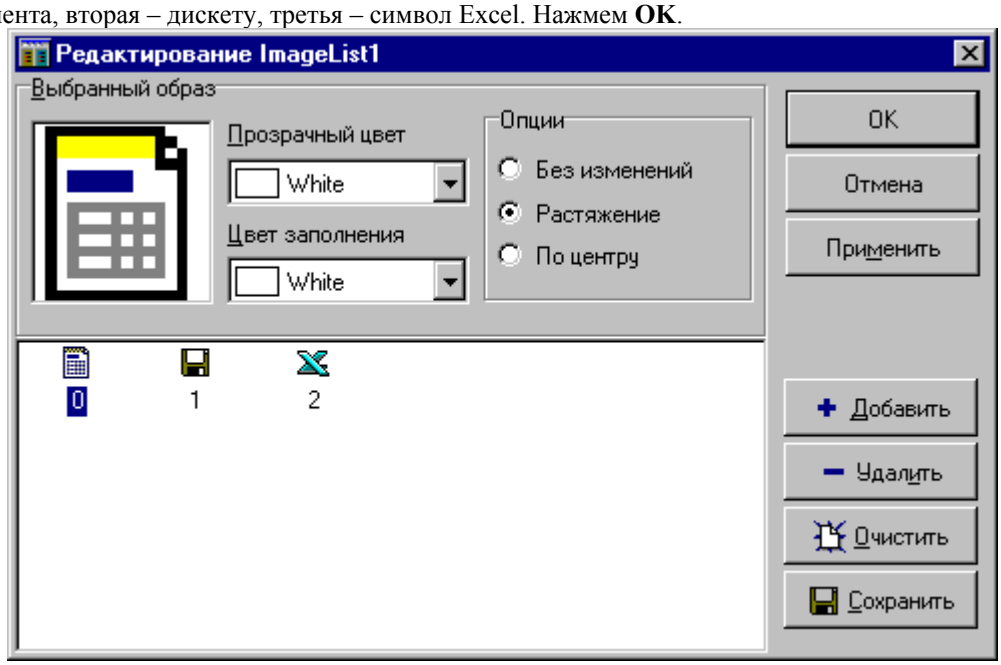
Выберем главную форму **StockInForm**. Добавим к ней компоненты:

ActionList (палитра **Standard**)

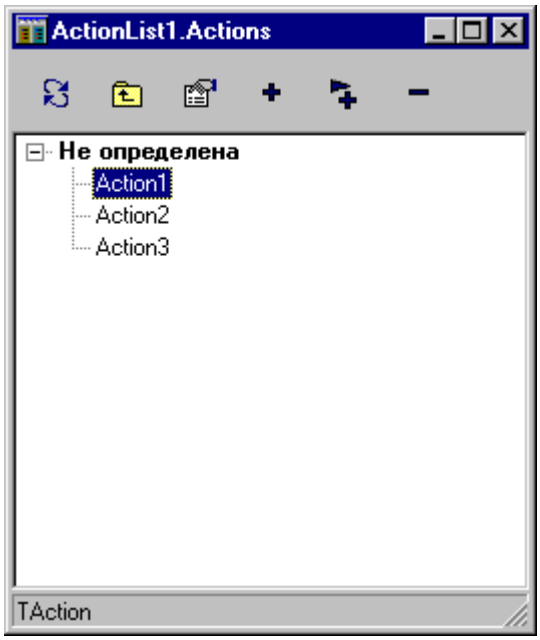
PopupMenu (палитра **Standard**)

ImageList (палитра **Win32**).

Дважды щелкнем на компоненте **ImageList1** мышью. В появившемся редакторе добавим в список три картинки размером 16x16 пиксель. Первая картинка (с индексом 0) будет изображать пиктограмму шапки документа, вторая – дискету, третья – символ Excel. Нажмем **OK**.



Компонент **ImageList** позволяет хранить список картинок одинакового размера и затем обращаться к ним по их индексу, который начинается с нуля. Размеры картинок определяются его свойствами **Width** и **Height**. Назначим компонентам **ActionList1** и **PopupMenu1** значение свойства **Images = ImageList1** в Инспекторе объектов. Дважды щелкнем на компоненте **ActionList1**. Появится редактор списка команд.



Добавим три команды (TAction). В Инспекторе объектов назначим им свойства:

Caption	Шапка…
ImageIndex	0
Hint	Шапка
Name	actHeader

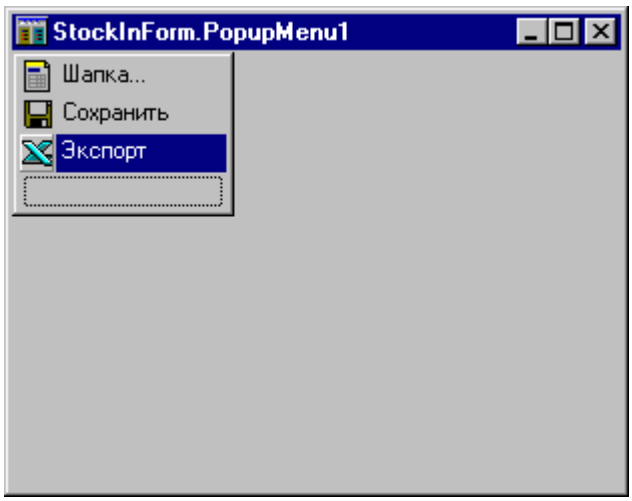
Caption	Сохранить
ImageIndex	1
Hint	Сохранить
Name	actSave
ShortCut	Ctrl+S

Caption	Отчет
ImageIndex	2
Hint	Отчет
Name	actReport

Закроем редактор списка команд.

Дважды щелкнем на компоненте **PopupMenu1** мышью. Появится редактор пунктов меню. Добавим в меню три пункта клавишей Insert. Последовательно выбирая пункты меню, присвоим их свойствам **Action** созданные только что нами команды:

actHeader
actSave
actReport



Закроем редактор меню.

Дважды щелкнем на компоненте **ActionList1**. В появившемся редакторе списка команд выберем команду «Шапка» (**actHeader**) и создадим ей обработчик события **OnExecute**.

Добавим в обработчике следующее выражение:

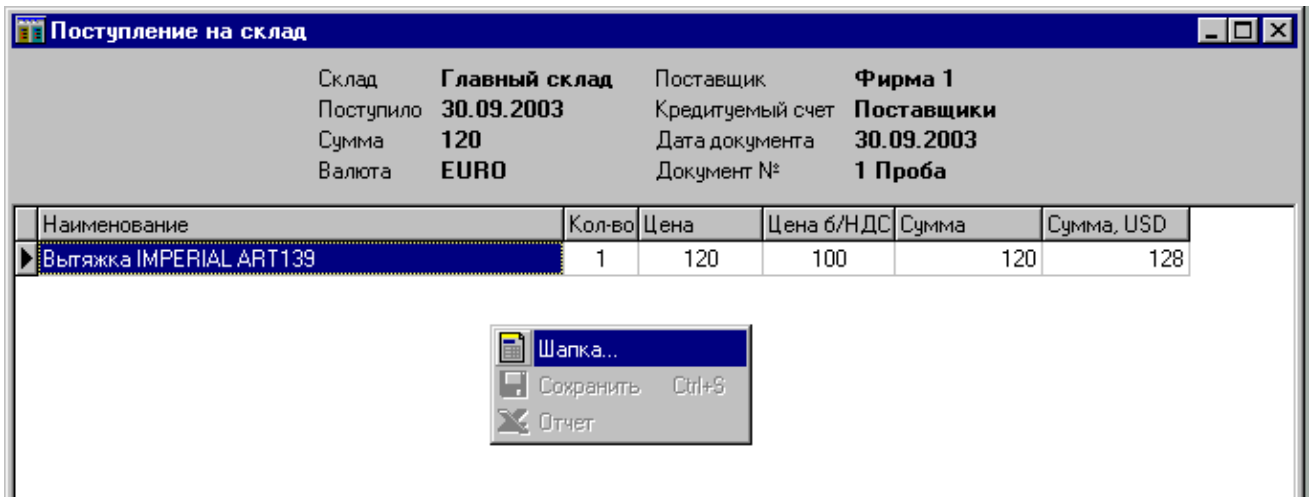
```
procedure TStockInForm.actHeaderExecute(Sender: TObject);
begin
  if StockInHeaderForm.ShowModal = mrOK then
    self.Caption := NameOfDocument('STOCK_IN', qryMaster.FieldByName('ID').AsInteger,
      traCurrent); //устанавливаем новый заголовок формы
end;
```

В обработчик **OnCreate** формы также добавим:

```
  self.Caption := NameOfDocument('STOCK_IN', qryMaster.FieldByName('ID').AsInteger,
    traCurrent); //устанавливаем новый заголовок формы
```

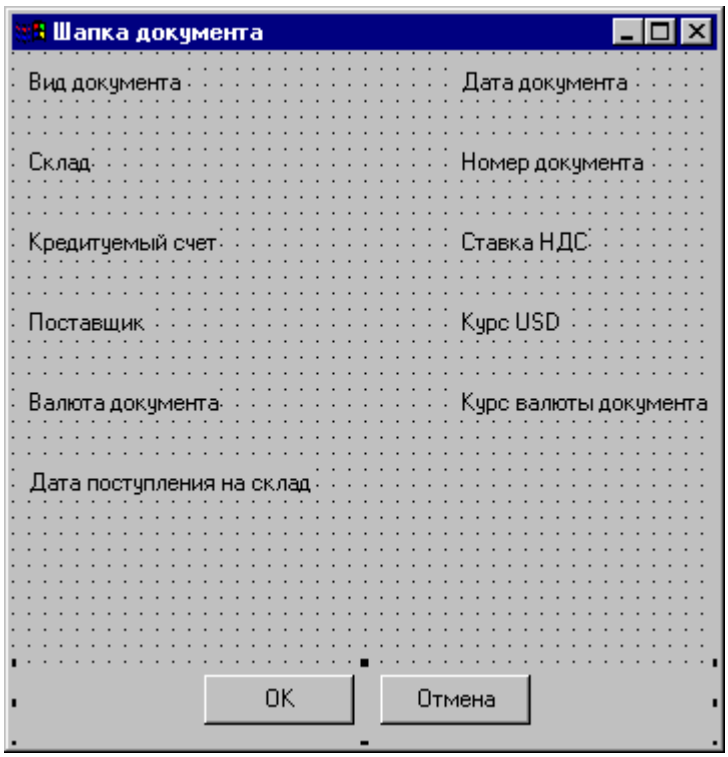
Здесь мы присваиваем заголовок форме. Функция **NameOfDocument** возвращает наименование документа в соответствии с теми правилами форматирования, которые для него заданы.

Переключимся на форму клавишей F12 и в Инспекторе объектов присвоим свойству **PopupMenu** формы **StockInForm** компонент **PopupMenu1**. Теперь у формы есть контекстное меню. Запустим проект (F9) и щелкнем правой кнопкой мыши на форме. Появится контекстное меню. Остается выбрать пункт «Шапка» и вызвать окно редактирования шапки. Окно вызвалось модально. Пока оно не закрыто, невозможно активизировать какое-либо другое окно программы. Закроем окно, нажав кнопку Cancel или клавишу Escape. Закроем окно документа и перейдем к добавлению компонентов на форму «шапки».

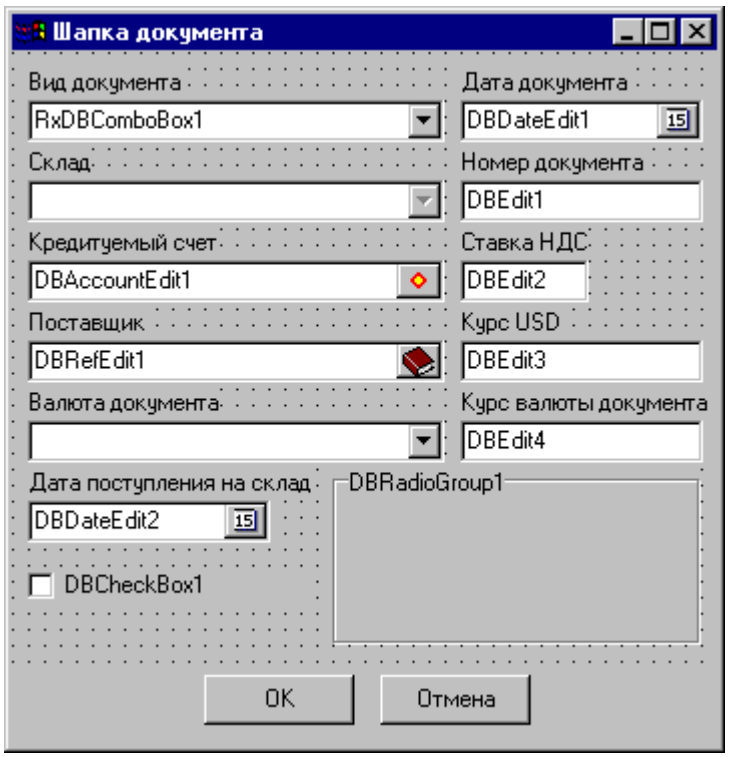


Для «массового» добавления компонента какого-то определенного класса нужно выбрать этот компонент на палитре, нажав клавишу Shift и щелкнув на компоненте мышью. Компонент выделится синим цветом на палитре. После этого достаточно щелкать мышью на форме и компонент будет добавляться многократно. Поступим так с компонентом Label с палитры Standard, нам нужно добавить его 8 раз на форму.

Для того чтобы отменить режим «массового» добавления компонента щелкнем на палитре на самом левом значке со стрелкой. Добавим 11 компонентов **Label**. Заменим свойство Caption добавленных нами компонентов на: **Вид документа, Дата документа, Склад, Номер документа, Кредитуемый счет, Ставка НДС, Поставщик, Курс USD, Валюта документа, Курс валюты документа, Дата поступления на склад.**



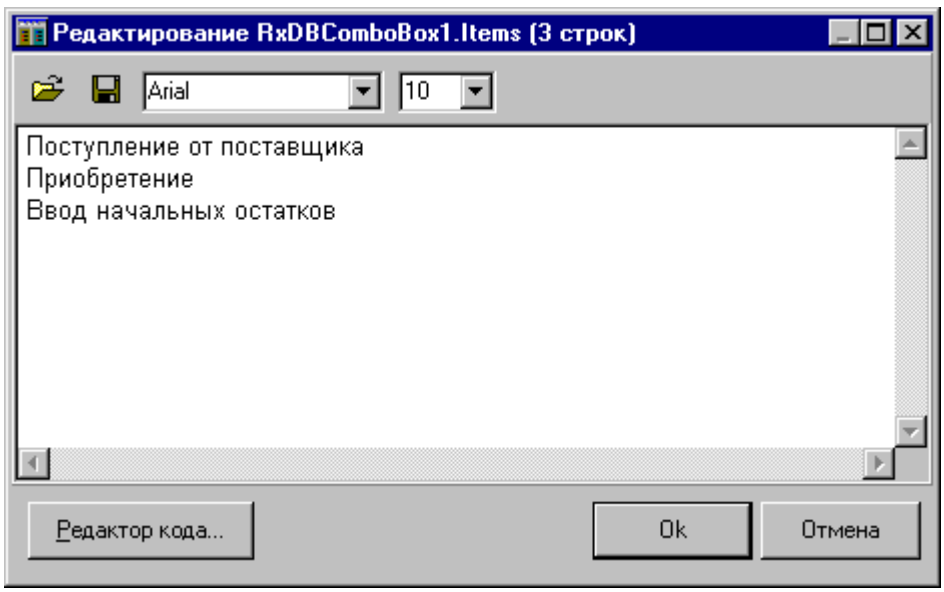
Добавим компоненты управления данными и назначим им свойства в Инспекторе объектов.



Поле «Вид документа»: компонент **RxDBComboBox** (палитра **RX DBAware**)

DataSource = StockInForm.dsrMaster
DataField = DOC_KIND
EnableValues = True
Name = cbxDOC_KIND
Style = csDropDownList
Items = Поступление от поставщика
Приобретение
Ввод начальных остатков
Values = 0
1
2

при заполнении свойств **Items** и **Values** в Инспекторе объектов вызывается редактор многостроковых текстовых данных типа TStrings:



Поле «Дата документа»: компонент **DBDateEdit** (палитра **RX DBAware**)
DataSource = StockInForm.dsrMaster
DataField = DOC_DATE
Name = edDOC_DATE

Поле «Склад»: компонент **RxDBLookupCombo** (палитра **RX DBAware**)
DataSource = StockInForm.dsrMaster
DataField = STOCK_ACC
Name = cbxSTOCK_ACC

Поле «Номер документа»: компонент **DBEdit** (палитра **Data Controls**)
DataSource = StockInForm.dsrMaster
DataField = DOC_NO
Name = edDOC_NO

Поле «Кредитуемый счет»: компонент **DBAccountEdit** (палитра **Allegro**)
DataSource = StockInForm.dsrMaster
DataField = ACC_ID
Name = aedACC_ID
Transaction = StockInForm.traCurrent

Поле «Ставка НДС»: компонент **DBEdit** (палитра **Data Controls**)
DataSource = StockInForm.dsrMaster
DataField = VAT_RATE
Name = edVAT_RATE

Поле «Поставщик»: компонент **DBRefEdit** (палитра **Allegro**)
ClassTableName = CONTRAGENT
DataSource = StockInForm.dsrMaster
DataField = CONTRAGENT
Name = refCONTRAGENT
Transaction = StockInForm.traCurrent

Поле «Курс USD»: компонент **DBEdit** (палитра **Data Controls**)
DataSource = StockInForm.dsrMaster
DataField = EXCH_RATE_S
Name = edEXCH_RATE_S

Поле «Валюта документа»: компонент **DBLayerComboBox** (палитра **Allegro**)
DataSource = StockInForm.dsrMaster
DataField = LAYER_ID
Name = cbxLAYER_ID
Transaction = StockInForm.traCurrent

Поле «Курс валюты документа»: компонент **DBEdit** (палитра **Data Controls**)
DataSource = StockInForm.dsrMaster
DataField = EXCH_RATE_L
Name = edEXCH_RATE_L

Поле «Дата поступления на склад»: компонент **DBDateEdit** (палитра **RX DBAware**)
DataSource = StockInForm.dsrMaster
DataField = ENTRY_DATE
Name = edENTRY_DATE

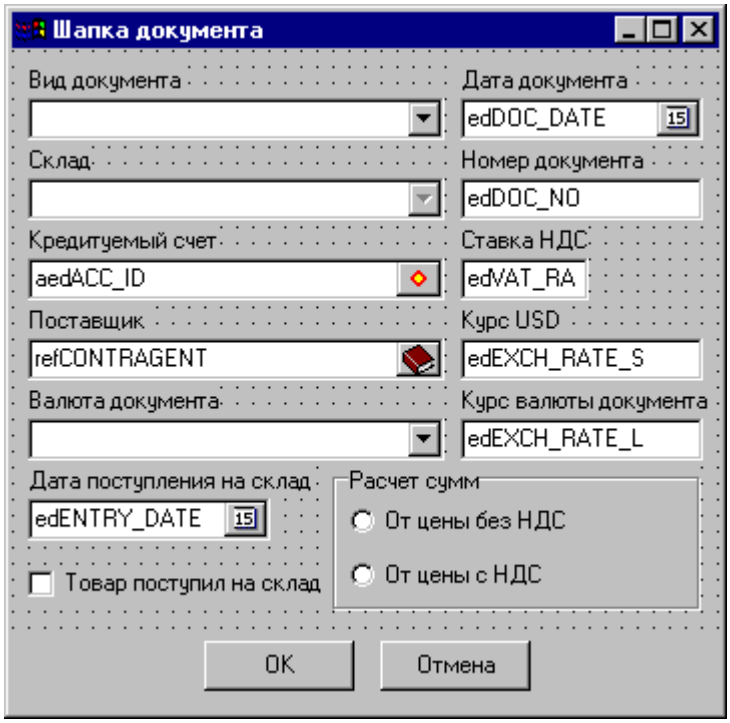
Добавим еще 2 компонента:

Поле «Товар поступил на склад»: компонент **DBCheckBox** (палитра **Data Controls**)

Caption = Товар поступил на склад
DataSource = StockInForm.dsrMaster
DataField = HAS_ENTRY
Name = cbHAS_ENTRY

Поле «Расчет сумм»: компонент **DBRadioGroup** (палитра **Data Controls**)

Caption = Расчет сумм
DataSource = StockInForm.dsrMaster
DataField = CALC_MODE
Name = rgCALC_MODE
Items = От цены без НДС
От цены с НДС
Values = 0
1



Сохраним все изменения.

Можно запустить проект и вызвать окно «шапки» с помощью контекстного меню, чтобы убедиться, что все поля документа (кроме поля «Склад») отображаются правильно. Не пытайтесь редактировать поля. У нас еще не введены необходимые для этого тексты SQL-запросов. Проверьте, как перемещается фокус ввода между компонентами окна «шапки» по нажатию клавиши Tab. Если Вам не нравится порядок перехода, то нужно вернуться в режим дизайна, выбрать форму шапки и вызвать пункт меню **Правка/Порядок перехода**. Появится диалог, в котором можно поменять порядок перехода между экранными объектами.

Остановим проект, если он запущен и вернемся в режим дизайна.

Для того чтобы организовать выпадающий список складов, добавим на форму 1 компонент **IBQuery** и 1 компонент **DataSource** (палитра **InterBase**) .

Установим свойства компонента **DataSource**:

DataSet = qryStocks
Name = dsrStocks

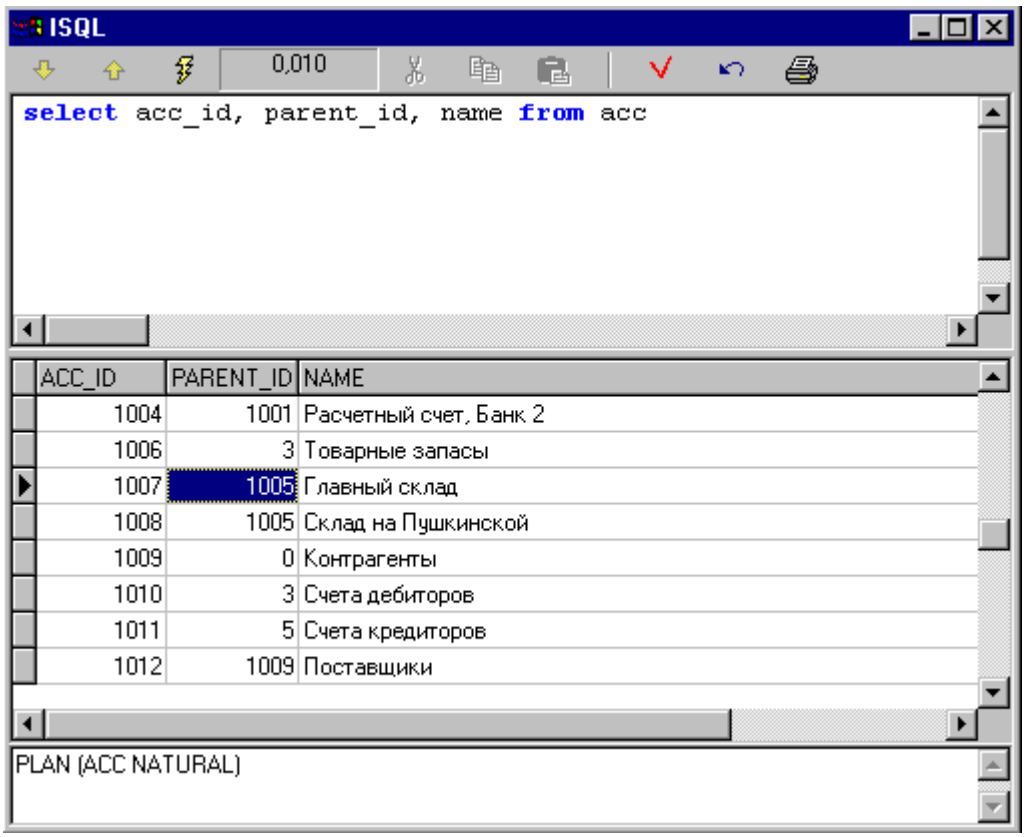
Установим свойства компонента **IBQuery**:

Transaction = StockInForm.traCurrent
Name = qryStocks

Теперь нам нужно вписать в свойство SQL компонента IBQuery текст SQL-запроса, который возвратил бы нам все имеющиеся в конфигурации склады. Для того, чтобы поэкспериментировать с запросами, «растянем» главное окно программы (кнопка **Растянуть Главное Окно**) и вызовем окно ISQL через меню Инструменты/Интерактивный SQL. Наберем такую команду:

```
select acc_id, parent_id, name
from acc
```

Нажмем Ctrl + Enter. Мы обратились к системной таблице счетов ACC и запросили внутренние ID счетов, внутренние ID их родителей и имена счетов. Найдем в полученном наборе счета складов. Мы видим, что они имеют PARENT_ID = 1005. Это ID регистра «Товары».



Изменим текст запроса так, чтобы он нам возвращал все счета, непосредственно входящие в регистр «Товары»:

```
select acc_id, name
from acc where parent_id = 1005
```

Нажмем Ctrl+Enter и убедимся, что это тот список складов, что нам нужен. После выполнения запроса текст запроса исчезает из верхнего окна. Поэтому вернемся на шаг назад (Ctrl+P) в окне ISQL, для того, чтобы скопировать текст запроса в буфер обмена Windows. Выделим текст запроса и скопируем его в буфер обмена (Ctrl+C). Сожмем Главное окно с помощью кнопки **Сжать Главное Окно**. Выберем компонент **qryStocks** и в Инспекторе объектов щелкнем дважды на его свойстве SQL.

Вставим текст запроса из буфера обмена (Ctrl+V) и нажмем **OK**.

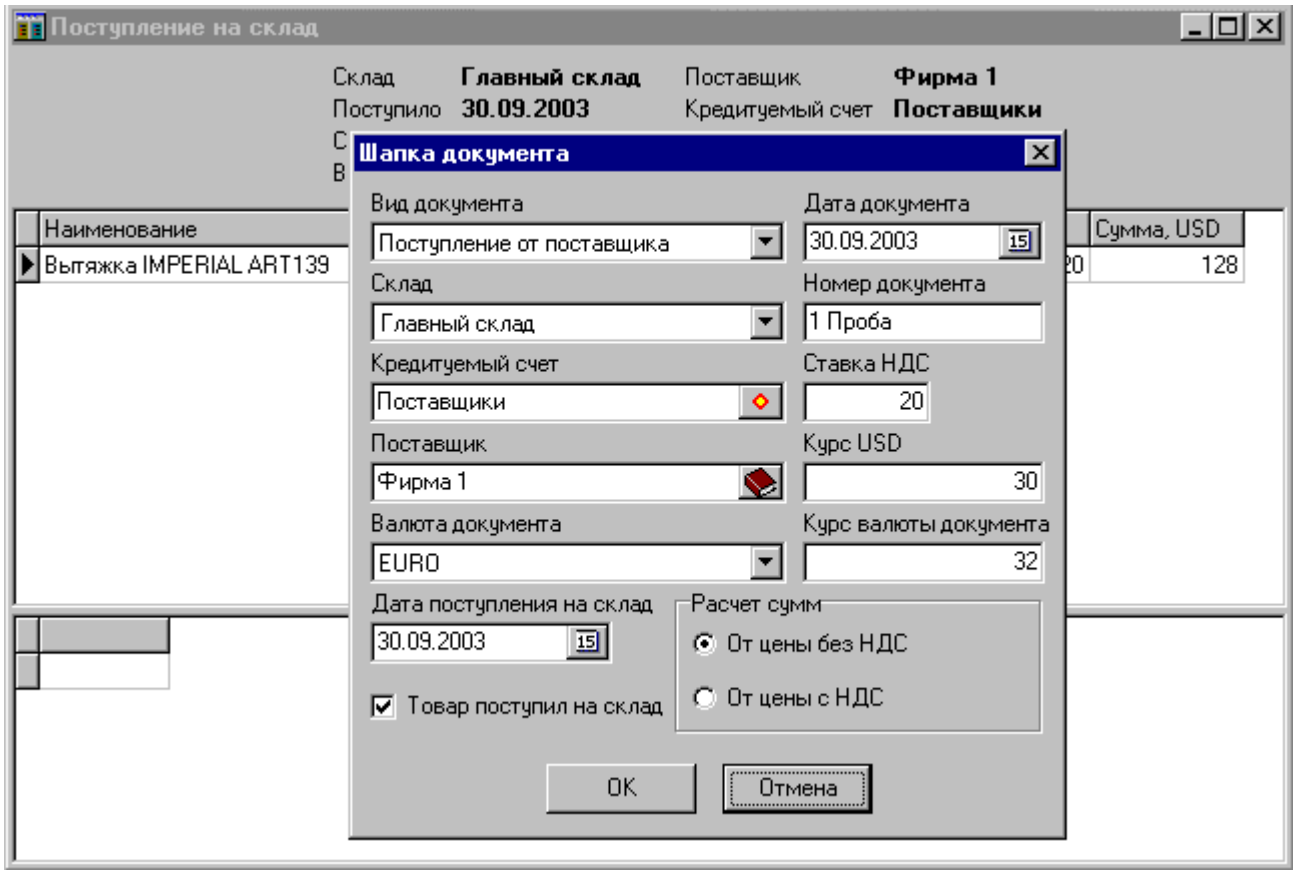
Нам остается привязать управляющий элемент поля «Склад» к запросу так, чтобы он получал из запроса значения для выпадающего списка. Сделаем это, установив свойства компонента **cbxSTOC_ACC** именно в той последовательности, как здесь перечислено:

```
LookupSource = dsrStocks
LookupField = ACC_ID
LookupDisplay = NAME
```

Для того чтобы запрос открылся при показе формы на экране, создадим обработчик события **OnShow** формы и впишем в него такую команду:

```
qryStocks.Open;
```

Теперь запустим проект (F9).

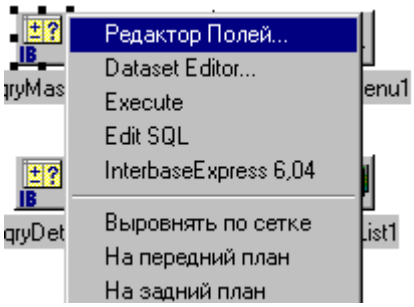


Закроем все окна и выйдем из режима «Дизайнер».

Создаем SQL-запросы Update и Insert для «шапки» документа.

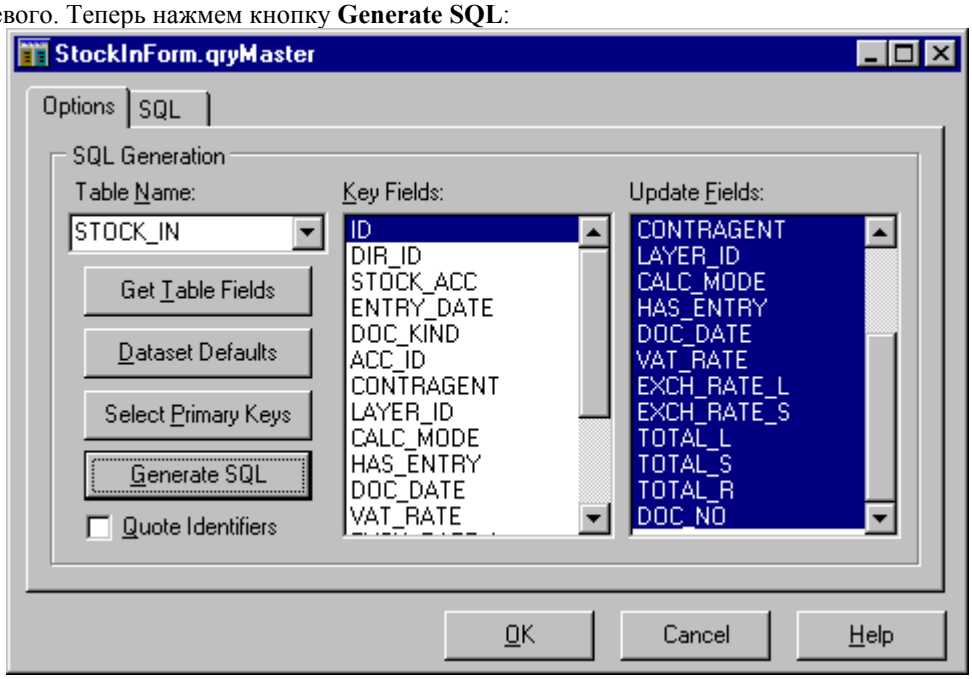
Включим режим «Дизайнер» и откроем проект **stock_in_project.ipr**.

Выберем на главной форме проекта компонент **qryMaster** и щелкнем на нем правой кнопкой мыши. Появится контекстное меню компонента. Используем пункт меню **DataSet Editor**. Перед нами появится редактор SQL-запросов вставки, модификации и удаления.

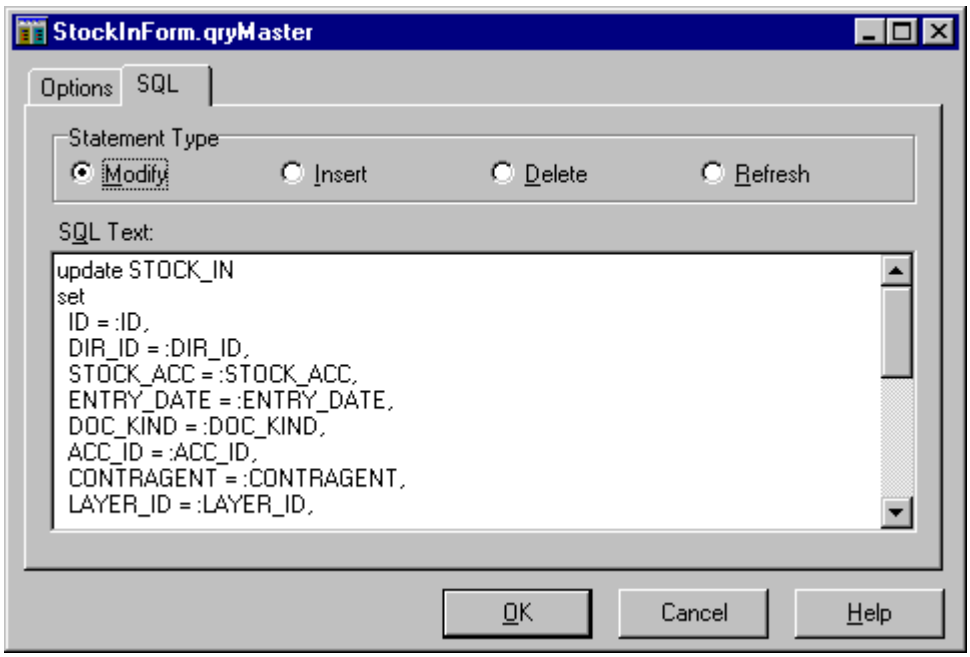


Этот редактор весьма полезен при создании SQL-запросов, так как помогает нам на основе запроса, записанного в свойстве SelectSQL, сформировать все остальные запросы автоматически. Редактор имеет две закладки: Options и SQL. На первой закладке требуется выделить в списке **Key Fields** ключевые поля, а в правом списке **Upadate Fields** – поля, которые необходимо модифицировать. Нажмем кнопку **Get Table Fields**, чтобы

выбрать все поля таблицы STOCK_IN, а затем кнопку **Select Primary Keys** , чтобы выбрать поле ID в качестве ключевого. Теперь нажмем кнопку **Generate SQL**:



Редактор сам сформирует все тексты запросов и покажет их на закладке «SQL»:



Переключая радиокнопки Statement Type, мы можем посмотреть тексты всех запросов. Обратим внимание, что запросы Insert, Update Delete работают только с одной таблицей, хотя в запросе Select мы использовали объединение нескольких таблиц. Но это правильно, так как нам нужно изменять данные только в таблице документа, а остальные таблицы мы используем лишь в целях получения дополнительной информации при отображении данных. К сожалению, запрос Refresh, сформированный редактором, тоже запрашивает поля одной таблицы, поэтому мы не сможем использовать его в таком виде. Очистим текст запроса Refresh. Очистим также текст запроса Delete, так как удаление документа мы не будем использовать. Нажмем **OK**.

Дважды щелкнем на свойстве SelectSQL компонента **qryMaster** в Инспекторе объектов, выделим весь текст запроса и скопируем (Ctrl+C) его в буфер обмена Windows. Закроем диалог редактора свойства SelectSQL и дважды щелкнем на свойстве RefreshSQL. В появившемся редакторе вставим (Ctrl+V) текст запроса из буфера обмена. Таким образом, в запросах SelectSQL и RefreshSQL для «шапки» документа мы будем использовать один

и тот же текст. Запрос RefreshSQL используется после вставки или изменения данных для того, чтобы освежить текущую строку набора.

Теперь набор данных в компоненте **qryMaster**, стал редактируемым.

Взглянем на переменные контекста, вызвав соответствующее окно через меню **Запуск/Переменные контекста**. Нужно убедиться, что в переменных контекста сейчас указан конкретный документ. Если это не так, установим тип документа и текущий документ (первые две строки). Выйдем из окна «Переменные контекста».

Запустим проект (F9).

Вызовем окно шапки и убедимся, что управляющие элементы теперь работают. Однако внесенные изменения пока не сохраняются.

Остановим проект и внесем некоторые команды в тексты модулей.

Добавим объявление переменной после **implementation**:

```
var
  Modified: boolean;
```

В этой переменной мы будем хранить признак того, что документ редактировался.

В обработчик **OnCreate** формы **StockInForm**, перед другими командами, добавим команды:

```
traCurrent.StartTransaction; //стартовать транзакцию
Modified := False;
```

Старт транзакции должен выполняться при запуске проекта, до того, как будут открыты SQL-запросы. Нам необходимо явно стартовать транзакцию, чтобы потом мы могли подтвердить ее или откатить. Все сделанные нами изменения в базе данных после подтверждения транзакции запомнятся окончательно, а в случае отката транзакции, наоборот, все изменения в базе данных, сделанные после старта транзакции, будут отменены.

Дважды щелкнем на компоненте **ActionList1**, выберем в редакторе списка команду «Сохранить» и создадим к ней обработчик события **OnExecute**.

Впишем в обработчик следующий тест:

```
traCurrent.CommitRetaining; //подтвердить транзакцию
Modified := False;
```

Мы вызвали метод подтверждения транзакции. У компонента класса TIBTransaction существует еще один метод подтверждения транзакции, который называется просто Commit. Разница между ними в том, что метод CommitRetaining оставляет транзакцию активной, и SQL-запросы остаются открытыми. Так как нам в данном случае хочется иметь возможность делать иногда сохранения в процессе редактирование документа, мы выбираем этот метод. Иначе нам пришлось бы каждый раз после сохранения документа переоткрывать все запросы.

В свойстве **DefaultAction** компонента транзакции **traCurrent** мы указали TARollback. Это означает, что при закрытии окна транзакция откатится автоматически. Разумеется, мы должны выдать предупреждение пользователю, если он попытается закрыть окно, что изменения могут быть утеряны. Лучше всего предложить ему сделать выбор: сохранить изменения, не сохранять их или отказаться от закрытия окна и продолжить редактирование.

Для того чтобы это реализовать, создадим для формы **StockInForm** обработчик события **OnCloseQuery**. Этот обработчик вызывается всякий раз, когда пользователь пытается закрыть форму. Обработчик имеет параметр **CanClose**, который можно изменить на False внутрь процедуры обработки и тогда форма не закроется. Поэтому впишем в обработчик такой текст:

```
if Modified then
  case MessageDlg('Сохранить изменения в документе?', mtConfirmation,
            MkSet(mbYes, mbNo, mbCancel), 0) of
  mrYes: miSave.Click; //вызов обработчика OnClick пункта меню miSave
  mrCancel: Canclose := False;
  end;
```

Создадим у компонента **qryMaster** обработчик события **AfterPost** и впишем в него:

```
Modified := True;
```

Мы должны еще позаботиться о том, чтобы вызвать метод **Post** компонента **qryMaster**, так как только при вызове этого метода SQL-запросы вставки (insert) и модификации данных (update) посылаются на сервер. Перейдем к форме шапки. Создадим обработчик нажатия кнопки **btnOK**, дважды щелкнув на этой кнопке. Впишем в него такой текст:

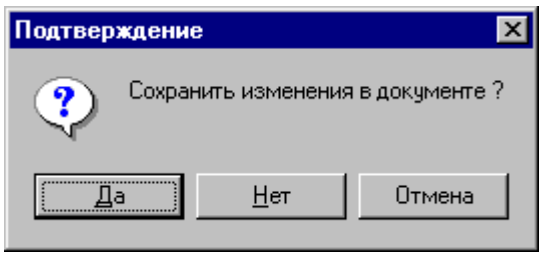
```
with StockInForm.qryMaster do
if InSet(State, MkSet(dsEdit, dsInsert)) then
begin
  Post;
  RunContext.Documents[0].doc_id := FieldByName('ID').AsInteger;
end;
self.ModalResult := mrOK;
```

Прежде чем применять метод **Post**, нужно убедиться, что компонент находится в режиме редактирования или вставки. Поэтому мы предварительно проверяем, входит ли его состояние **State** во множество состояний [dsEdit, dsInsert]. Если метод **Post** отработал без ошибок, то свойству **ModalResult** формы будет присвоено значение mrOK, что автоматически приведет к закрытию формы.

Если данные в каких-то полях были изменены, но мы не хотим посылать эти изменения на сервер, нужно вызвать у метод **Cancel** компонента **qryMaster**. Сделаем так, чтобы независимо от того, каким способом пользователь закрывает окно редактирования шапки, если компонент находился при этом в состоянии редактирования или вставки, то будет вызван метод **Cancel**. Для этого создадим у формы **StockInHeaderForm** обработчик формы **OnClose** и впишем в него такой текст:

```
with StockInForm.qryMaster do
if InSet(State, MkSet(dsEdit, dsInsert)) then
   Cancel;
```

Запустим проект и вызовем окно шапки документа с помощью контекстного меню. Изменим данные в полях шапки, например, поменяем «Склад». Нажмем **OK**. А теперь попытаемся закрыть окно документа. Мы должны увидеть предложение сохранить документ:



Попробуем сохранить документ. Несколько раз запустим проект, в каких-то случаях откажемся от сохранения, убедимся, что все правильно работает. Итак, редактирование шапки документа мы реализовали. Теперь реализуем создание нового документа.

Прежде, чем добавлять позиции в новый документ «Поступление на склад», пользователь всегда заполняет его шапку. Документы в Allegro создаются, как правило, из «Проводника по документам», который просто вызывает наш проект оконного интерфейса. Проект должен проверить значение переменной контекста **RunContext.Documents[0].doc_id** и если оно не положительно, сразу принять решение о том, что это новый документ и вызвать форму для редактирования «шапки» с помощью метода **ShowModal**. Нам нужно сделать так, чтобы пользователь мог отменить создание документа на этом этапе, нажав кнопку «Отмена» в окне редактирования «шапки». Вызовем окно «шапки» в событии **OnCreate** (при создании) главной формы проекта. Если пользователь откажется от создания нового документа, нажав «Отмена», то метод **ShowModal** вернет значение, отличное от **mrOK**. Это и будет сигналом к ому, чтобы завершить работу проекта. Для того чтобы завершить работу проекта нужно закрыть его главную форму **StockInForm**. Так как в обработчике события **OnCreate** формы закрыть эту форму нельзя, используем компонент таймера. Запустим таймер в обработчике **OnCreate** формы, если требуется ее закрыть. Когда произойдет событие таймера, то обработчик этого события закроет главную форму.

Добавим на форму StockInForm компонент **Timer** (палитра **System**) и установим его свойства:

Enabled = False

Interval = 100

А в обработчике **OnCreate** формы, после выражения

```
qryMaster.Open;
```

добавим такой текст:

```
 if RunContext.Documents[0].doc_id <=0 then
 begin
  qryMaster.Insert;
  {присваиваем начальные значения полям}
  qryMaster.FieldByName('DIR_ID').AsInteger := RunContext.dir_id;
  qryMaster.FieldByName('DOC_KIND').AsInteger := 0;
  qryMaster.FieldByName('CALC_MODE').AsInteger := 1;
  qryMaster.FieldByName('DOC_DATE').AsDateTime := Date;
  qryMaster.FieldByName('HAS_ENTRY').AsInteger := 1;
  qryMaster.FieldByName('ENTRY_DATE').AsDateTime := Date;
  qryMaster.FieldByName('VAT_RATE').AsCurrency := 20;
  qryMaster.FieldByName('TOTAL_L').AsCurrency := 0;
  qryMaster.FieldByName('TOTAL_S').AsCurrency := 0;
  qryMaster.FieldByName('TOTAL_R').AsCurrency := 0;
  qryMaster.FieldByName('CONTRAGENT').AsInteger := 0;
  qryMaster.FieldByName('STOCK_ACC').AsInteger := 1007; //Главный склад
  qryMaster.FieldByName('ACC_ID').AsInteger := 1012; //Поставщики

  if StockInHeaderForm.ShowModal <> mrOK then
    Timer1.Enabled := True;
 end;
 if Timer1.Enabled then
  exit;
```

Выберем компонент **`Timer`**, создадим обработчик его события **OnTimer** и впишем в него:

```
Timer1.Enabled := False;
Modified := False;
Self.Close;
```

Метод **Self.Close** закроет главную форму, когда произойдет событие таймера. А оно произойдет только в том случае, если пользователь отказался от создания нового документа, нажав кнопку **Отмена** в окне редактирования шапки.

При создании любой новый документ **должен** получить новое уникальное значение для поля ID с помощью специального генератора **DOC_ID_GEN**, существующего в базе данных. Займемся подключением генератора **DOC_ID_GEN** к компоненту **qryMaster**. Для этого выберем компонент **qryMaster** в Инспекторе объектов и дважды щелкнем на его свойстве **GeneratorField**. Появится редактор свойства:

Установим свойства:
Generator = DOC_ID_GEN
Field = ID
Increment By = 1
Apply Event = On Post

Эти установки означают, что для получения новых значений поля ID, если оно еще пустое, будет использоваться генератор **DOC_ID_GEN**, каждый раз увеличивая свое значение на единицу. Запуск генератора будет производиться в момент вызова метода **Post** компонента **qryMaster**.

Нам нужно еще разобраться с обязательными и необязательными полями. Дважды щелкнем на компоненте **qryMaster** и в Инспекторе объектов установим свойство **Required** для следующих полей:

Поле	Свойство Required
STOCK_ACC_NAME	False
LAYER_NAME	False
CREDIT_ACC_NAME	False
CONTRAGENT_NAME	False
DOC_DATE	True
DOC_NO	True
VAT_RATE	True
EXCH_RATE_S	True
EXCH_RATE_L	True
ENTRY_DATE	True

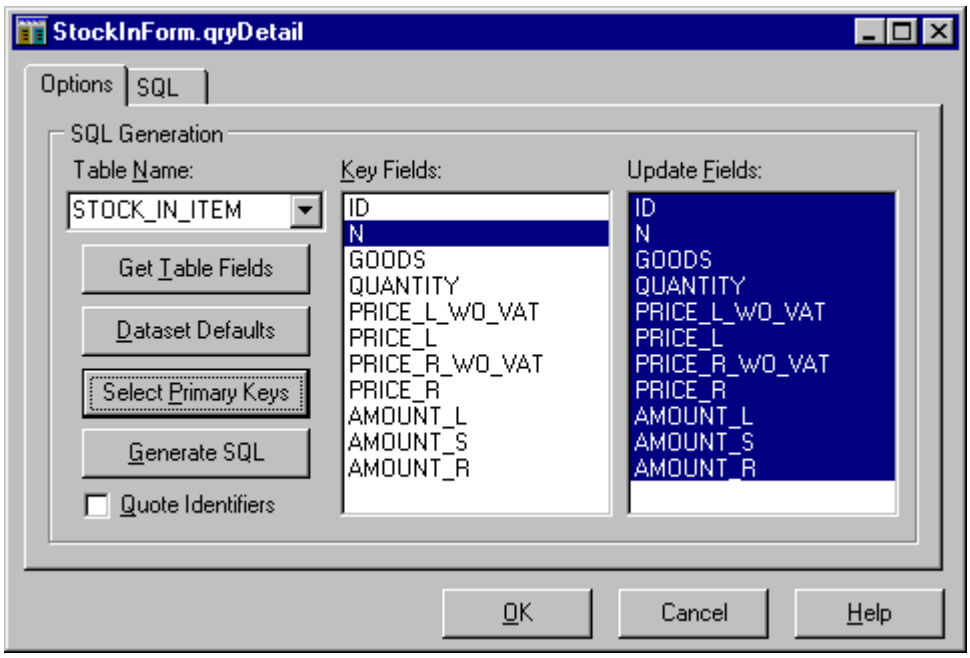
Сохраним проект. Растянем Главное окно Allegro, вызовем «Проводник по документам» и попытаемся создать из него с помощью через контекстное меню новое «Поступление на склад». Попробуем отменить, нажав **Отмена**. Попробуем создание еще раз, заполним все поля шапки и нажмем **OK**. Закроем окно документа, на вопрос о сохранении ответим **Да**. Документ создан, хоть он пока и не виден в «Проводнике». Для того чтобы его увидеть, нужно освежить проводник (Ctrl+F5).

Для того чтобы после каждого сохранения документа проводник освежался автоматически, добавим в конец текста обработчика **OnExecute** команды меню **actSave** вызов следующей процедуры:

```
RefreshExplorer; //пересветить «Проводник по документам»
```

Создаем SQL-запросы Insert, Update и Delete для «позиций» документа

Вызовем ***DataSet Editor*** через контекстное меню компонента ***qryDetails***.



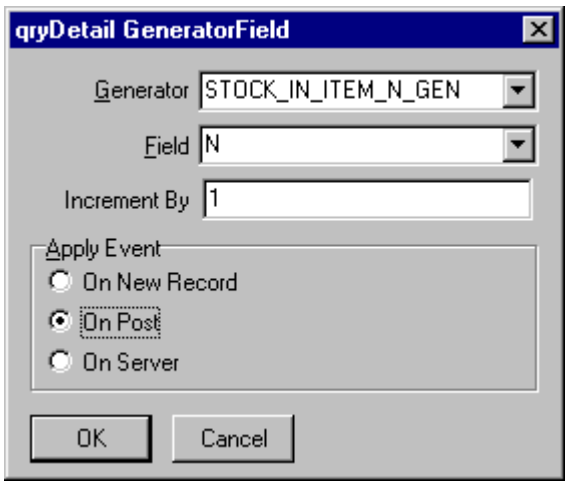
Нажмем кнопку **Get Table Fields** для того, чтобы выделить все поля. Затем нажмем кнопку **Select Primary Keys** для того, чтобы выделить ключевое поле **N**. Теперь нажмем кнопку **Generate SQL**. Нажмем кнопку **OK**.

Скопируем текст запроса из свойства **SelectSQL** в **RefreshSQL**, удалив из него секцию **order by** и условие **SI.ID =:ID** и добавив в условие **SI.N = :N**.

Текст запроса RefreshSQL должен выглядеть так:

```
select
  SI.ID,
  SI.N,
  SI.GOODS,
  O.SHORT_NAME ITEM_NAME,
  SI.QUANTITY,
  SI.PRICE_L,
  SI.PRICE_L_WO_VAT,
  SI.PRICE_R,
  SI.PRICE_R_WO_VAT,
  SI.AMOUNT_L,
  SI.AMOUNT_R,
  SI.AMOUNT_S
from
  STOCK_IN_ITEM SI,
  OBJECT_NAMES O
where
  SI.GOODS = O.OBJECT_ID and
  SI.N = :N
```

Установим у компонента **qryDetails** также значение свойства **GeneratorField**. В качестве генератора будем использовать генератор **STOCK_IN_ITEM_N_GEN**.



Затем, откроем в Инспекторе объектов закладку «События» и назначим компоненту **qryDetail** на события **AfterPost** имеющийся уже обработчик **qryMasterAfterPost**, выбрав его из выпадающего списка. Это мы сделали для того, чтобы после любого изменения в позициях вызывался один и тот же обработчик, устанавливающий нашу переменную **Modified** в значение **True**.

Сохраним проект. Запустим его, предварительно установив в переменных контекста наш первый документ, у которого имелась позиция. Пока не будем добавлять новые позиции. Изменим количество или цену у той, что есть. С помощью клавиши «стрелка вниз» уйдем с имеющейся строки, чтобы подтвердить редактирование. Мы видим, что теперь позиции редактируются!

Нам осталось разобраться с добавлением новых позиций и расчетом всех цен и сумм.

Но прежде доработаем немного окно **StockInForm**, добавив в него еще несколько управляющих элементов.

Увеличим высоту верхней панели до 112 пикселей. Для этого выберем компонент TopPanel и в Инспекторе объектов установим свойство **Height = 112**.

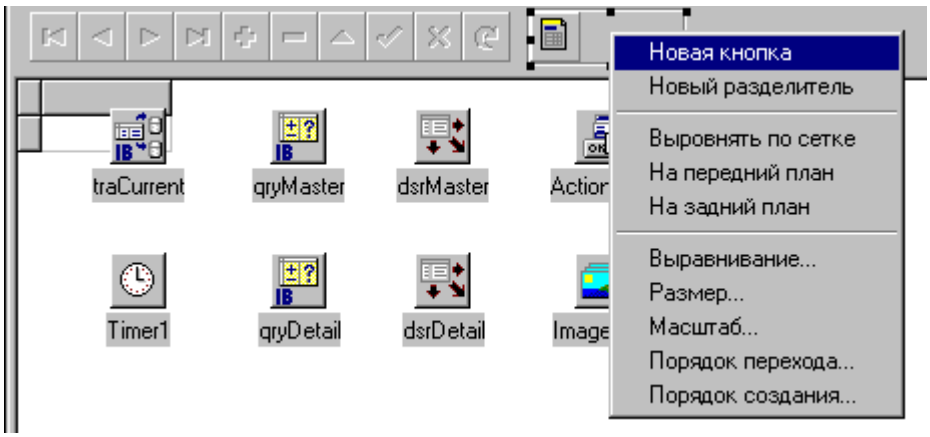
Разместим на ней компонент **DBNavigator** (палитра **DataControls**) и назначим ему свойства:

DataSource = dsrDetail
Flat = True
Left = 8
Top = 80
ShowHint = True

Навигатор поможет управлять набором данных в подчиненной таблице.
Заодно добавим компонент **ToolBar** (палитра **Win32**) и назначим ему свойства:

Align = alNone
ButtonHeight = 25
ButtonWidth = 25
EdgeBorders = [ebLeft, ebBottom, ebRight, ebTop]
Images = ImageList1
Flat = True
Left = 256
Top = 78
Width = 80
ShowHint = True

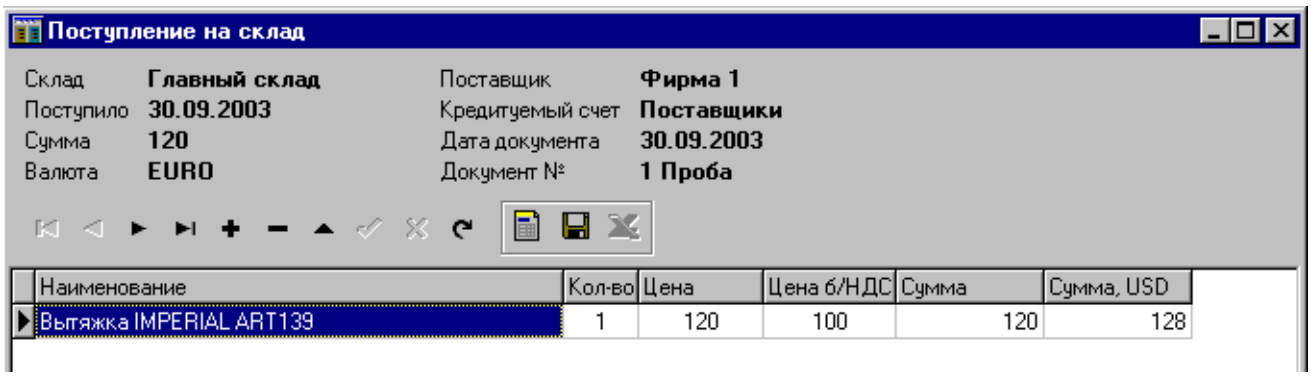
Этот компонент представляет собой панель инструментов. Создадим на ней три кнопки. Для этого нужно использовать пункт **Новая кнопка** контекстного меню компонента **ToolBar1**:



В инспекторе объектов назначим каждой кнопке свою команду из имеющегося у нас списка:

actHeader
actSave
actReport

Переместим все надписи на панели влево, запустим проект (F9):



Оформление верхней панели теперь не должно вызывать нареканий. Пользователь имеет все необходимые для работы кнопки.

Попробуем добавить запись в набор, нажав кнопку с плюсиком или клавишу ***Insert***. Отменим добавление клавишей ***Escape***. Мы видим, что новая запись вставляется **перед** уже имеющейся. Так работает метод **Insert** компонента типа TIBDataSet. Это явно собьет с толку пользователя, привыкшего набирать приходные документы сверху вниз, позицию за позицией. Для того чтобы преодолеть это неприятное явление в документе «Поступление на склад» мы отменим добавление **Insert** и вызовем метод **Append**, который всегда добавляет запись в **конец** набора, в отличие от метода **Insert**.

Для этого создадим для компонента **qryDetail** обработчик **AfterInsert** и впишем в него такой текст:

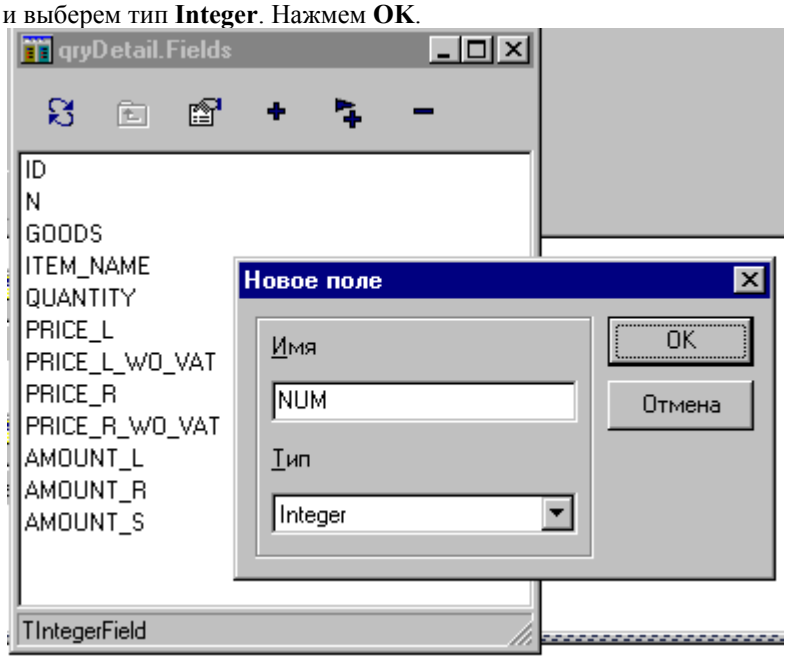
```
var
  lock_insert: boolean;

procedure TStockInForm.qryDetailAfterInsert(DataSet: TDataSet);
begin
  if not lock_insert then
  begin
    lock_insert := True;
    qryDetail.Cancel; //отменяем режим вставки
    qryDetail.Append; //вставляем в конец набора
  end;
  lock_insert := False;
end;
```

Логическая переменная lock_insert используется нами для того, чтобы заблокировать рекурсивное повторное исполнение кода, так как после вызова метода **Append** снова произойдет событие **AfterInsert** , которое вновь вызвало бы **Append** и так до бесконечности…

Запустим проект (F9). Попробуем добавить запись. Теперь запись добавляется в конец набора.

Хорошо бы еще иметь нумерацию строк набора, так как пользователь наверняка проверяет правильность ввода по количеству введенных строк…

Для того чтобы это реализовать, используем вычисляемое поле. Дважды щелкнем на компоненте **qryDetail**. Появится редактор полей. Нажмем кнопку с плюсиком, чтобы добавить новое поле. Дадим ему имя **NUM** и выберем тип **Integer**. Нажмем **OK**.



Теперь в инспекторе объектов придадим ему свойства:

FieldKind = fkCalculated
Alignment = taCenter
Index = 2
DisplayLabel = №
DisplayWidth = 5

Мы создали вычисляемое поле. Для того чтобы вычисляемые поля получали значения, необходимо присваивать им их в событии **OnCalcFields**t. Создадим обработчик этого события у компонента **qryDetail** и впишем в него такой код:

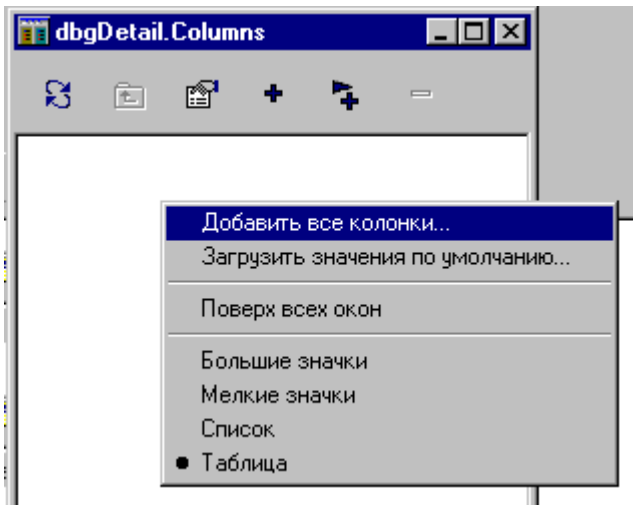
```
procedure TStockInForm.qryDetailCalcFields(DataSet: TDataSet);
begin
  with DataSet do
    FieldByName('NUM').AsInteger := RecNo;
end;
```

Для того чтобы после удаления не возникало «дырки» в нумерации, переоткроем запрос после удаления позиции. Для этого создадим у компонента **qryDetail** обработчик события **AfterDelete** и впишем в него такой код:

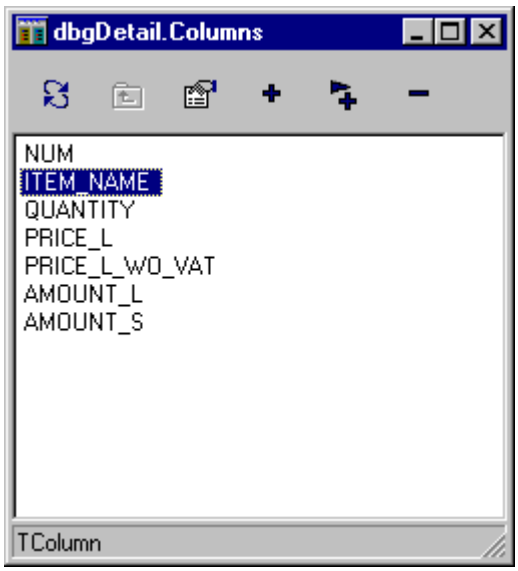
```
procedure TStockInForm.qryDetailAfterDelete(DataSet: TDataSet);
begin
  Modified := True;
  DataSet.Close;
  DataSet.Open;
end;
```

Теперь создадим «постоянные» (Persistent) колонки у сетки **dbgDetail**. Если не созданы «постоянные» колонки у сетки, то она автоматически создает «временные», как только в источнике данных оказывается открытый набор, что пока у нас и происходит. Существует ряд плюсов в том, чтобы создать «постоянные» колонки. Например, тогда легко можно настроить их некоторые свойства в Инспекторе объектов.

Для того чтобы создать «постоянные» колонки у сетки **dbgDetail**, нужно выбрать ее и в Инспекторе объектов дважы щелкнуть на свойстве **Columns**. Появится редактор колонок, в котором следует добавить колонки и привязать их к полям в источнике данных. Проще всего воспользоваться пунктом контекстного меню редактора **Добавить все колонки**, а затем удалить ненужные:



Удалим все колонки кроме NUM, ITEM_NAME, QUANTITY, PRICE_L, PRICE_L_WO_VAT, AMOUNT_L, AMOUNT_S:



Выберем в списке колонку **NUM** и в Инспекторе объектов дважды щелкнем на его свойства **Color**. Установим цвет для этой колонки. Теперь выберем в списке колонку **ITEM_NAME** и установим свойство :

```
ButtonStyle = cbsEllipsis
```

Закроем редактор колонок. Раскроем в Инспекторе объектов подпункты свойства **Options** сетки. Установим птичку **dgAlwaysShowEditor** (для того, чтобы по ячейкам сетки можно было перемещаться клавишами со стрелками и при этом сразу оказываться в режиме редактирования ячейки). Снимем птичку с **dgCancelOnExit**, чтобы при потере сеткой фокуса ввода набор данных оставался в режиме редактирования.

Теперь запустим проект (F9). Заметим, что первая колонка окрашена, а во второй колонке, при попадании экранного курсора в нее, высвечивается кнопка с многоточием. Это и есть стиль кнопки **cbsEllipsis**. Для того чтобы по нажатию кнопки что-то происходило, нужно для сетки создать обработчик события **OnEditButtonClick**. Вернемся в режим дизайна и добавим на форму компонент диалога справочника **RefDialog** с палитры **Allegro** и установим его свойства:

```
Transaction  = traCurrent
ClassTableName = GOODS
Title = Выберите товар
Options = [doValueRequired]
```

Выберем компонент сетки **dbgDetail ,** создадим обработчик события **OnEditButtonClick,** и впишем в него такой код:

```
procedure TStockInForm.dbgDetailEditButtonClick(Sender: TObject);
begin
  with RefDialog1 do
  if Execute then //вызываем диалог на экран и проверяем,
                  //был ли выбран товар
  begin
    qryDetail.Edit; //переводим набор данных в режим редактирования
    {присваиваем значение ID товара полю GOODS набора}
    qryDetail.FieldByName('GOODS').AsInteger := Object_ID;
    {присваиваем значение краткое наименование товара полю ITEM_NAME набора}
    qryDetail.FieldByName('ITEM_NAME').AsString:=
                        NameOfRefObject(Object_ID, 'SHORT_NAME');
    {перемещаем фокус ввода в сетке в поле "Количество"}
    dbgDetail.SelectedField := qryDetail.FieldByName('QUANTITY');
  end;
end;
```

У компонента **RefDialog1** метод **Execute** вызывет диалог на экран. Если пользователь нажал в диалоге кнопку **Выбрать**, то метод **Exceute** возвращает **True** и в свойстве **Object_ID** компонента оказывается значение, равное **ID** объекта справочника. Функция **NameOfRefObject** по **ID** запрашивает наименование объекта определенного типа, в данном случае, **краткое наименование**.

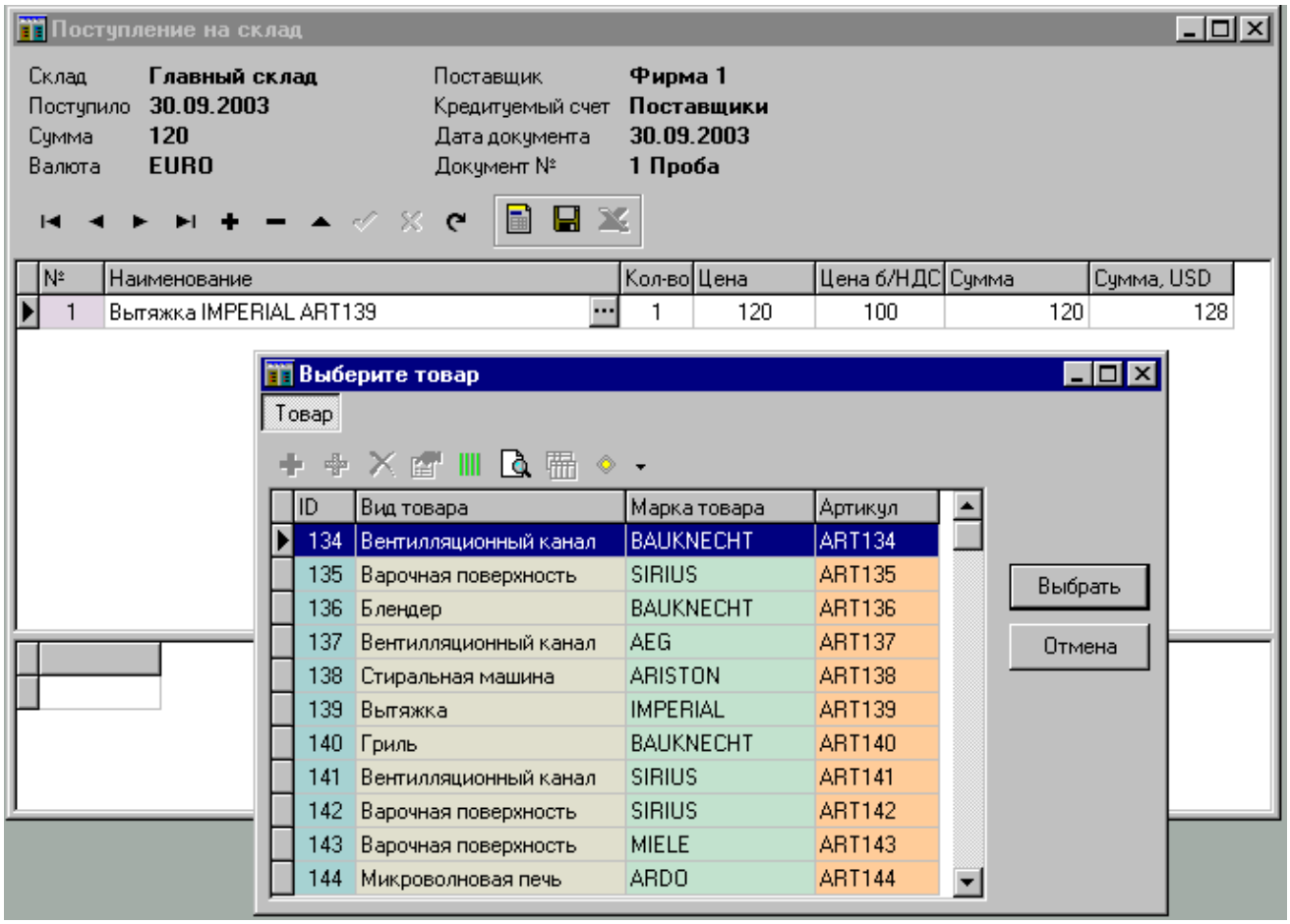
Ели пользователь закрыл диалог справочника кнопкой **Отмена**, метод **Execute** возвращает **False**.

Запустим проект еще раз. Попробуем нажать на кнопку с многоточием или дважды щелкнуть мышью в колонке «Наименование». Перед нами появится диалог справочника, из которого можно выбрать товар. Мы имеем возможность, если нужно добавить в справочник новый товар, упорядочить справочник по любой колонке, щелкнув на ее заголовке, включить фильтрацию, одним словом, нам доступны все функции справочной системы.

Выберем какой-нибудь товар в сетке справочника и нажмем кнопку **Выбрать**.

Мы видим, что наименование выбранного нами товара появилось в текущей строке документа.

Вернемся в режим дизайна.



Нам осталось реализовать вычисление полей сумм и еще некоторые мелочи.

Выберем компонент запроса **qryDetail** и дважды щелкнем на нем. В редакторе полей выберем поле **ID** и в Инспекторе объектов уберем птичку со свойства **Required**, чтобы оно не требовало ввода значений. У полей **ITEM_NAME, QUANTITY, PRICE, PRICE_L_WO_VAT**, наоборот, установим птичку к **Required**. Эти поля будут требовать ввода значений. Закроем редактор полей.

Выберем компонент **dsrDetail**, связывающий компонент запроса и сетку. Создадим для него обработчик события **OnDataChange** и впишем в него такой код:

```
procedure TStockInForm.dsrDetailDataChange(Sender: TObject; Field: TField);
begin
  if (Field <> nil) and
     ((Field.FieldName = 'PRICE_L') or
      (Field.FieldName = 'PRICE_L_WO_VAT') or
      (Field.FieldName = 'QUANTITY'))then
  with qryDetail do
```

```
  begin
    {Если режим расчета сумм на основе цен с НДС
     и изменено поле "Цена с НДС"  или "Количество"}
    if ((Field.FieldName = 'PRICE_L')  or (Field.FieldName = 'QUANTITY'))
      and (qryMaster.FieldByName('CALC_MODE').AsInteger = 1) then
    begin

      FieldByName('AMOUNT_L').AsCurrency :=
        FieldByName('PRICE_L').AsCurrency *
        FieldByName('QUANTITY').AsInteger;
      FieldByName('PRICE_L_WO_VAT').AsCurrency :=
          round(FieldByName('PRICE_L').AsCurrency * 1000 * 100/
          (qryMaster.FieldByName('VAT_RATE').AsCurrency + 100))/1000;

      FieldByName('PRICE_R').AsCurrency :=
          FieldByName('PRICE_L').AsCurrency *
          qryMaster.FieldByName('EXCH_RATE_L').AsCurrency;
      FieldByName('PRICE_R_WO_VAT').AsCurrency :=
          round(FieldByName('PRICE_R').AsCurrency * 1000 * 100/
          (qryMaster.FieldByName('VAT_RATE').AsCurrency + 100))/1000;
      FieldByName('AMOUNT_R').AsCurrency :=
          round(FieldByName('PRICE_R').AsCurrency *
          FieldByName('QUANTITY').AsInteger);
    end
    else
    {Если режим расчета сумм на основе цен без НДС и
    изменено поле "Цена без НДС" или "Количество"}
    if ((Field.FieldName = 'PRICE_L_WO_VAT')   or (Field.FieldName = 'QUANTITY'))
       and (qryMaster.FieldByName('CALC_MODE').AsInteger = 0) then
    begin

      FieldByName('AMOUNT_L').AsCurrency :=
        round(FieldByName('PRICE_L_WO_VAT').AsCurrency * 10 *
          FieldByName('QUANTITY').AsInteger *
          (qryMaster.FieldByName('VAT_RATE').AsCurrency + 100))/1000;
      FieldByName('PRICE_L').AsCurrency :=
        round(FieldByName('PRICE_L_WO_VAT').AsCurrency * 10 *
          (qryMaster.FieldByName('VAT_RATE').AsCurrency + 100))/1000;

      FieldByName('PRICE_R_WO_VAT').AsCurrency :=
          FieldByName('PRICE_L_WO_VAT').AsCurrency *
          qryMaster.FieldByName('EXCH_RATE_L').AsCurrency;;
      FieldByName('PRICE_R').AsCurrency :=
          round(FieldByName('PRICE_R_WO_VAT').AsCurrency * 10 *
          (qryMaster.FieldByName('VAT_RATE').AsCurrency + 100))/1000;
      FieldByName('AMOUNT_R').AsCurrency :=
          round(FieldByName('PRICE_R_WO_VAT').AsCurrency * 10 *
          FieldByName('QUANTITY').AsInteger *
          (qryMaster.FieldByName('VAT_RATE').AsCurrency + 100))/1000;
    end;
    {Расчет суммы в долларах}
    FieldByName('AMOUNT_S').AsCurrency :=
            FieldByName('AMOUNT_L').AsCurrency *
            qryMaster.FieldByName('EXCH_RATE_L').AsCurrency/
            qryMaster.FieldByName('EXCH_RATE_S').Ascurrency;
  end;
end;
```

Событие **OnDataChange** происходит после редактирования любого поля. При этом само поле передается в качестве параметра **Field** в обработчик события и мы можем проанализировать, какое именно поле было изменено. В зависимости от того, какой режим расчетов установлен в шапке документа (от цен с НДС или от цен без НДС) и того, какое поле было изменено («цена с НДС» или «Цена без НДС») происходят разные расчеты с присвоением новых значений остальным полям. С целью округления до 3 десятичных знаков мы умножаем некоторые величины на 1000, округляем их до целого функцией **round** и затем делим на 1000.

Для того чтобы поле ID не было пустым, создадим еще у компонента **qryDetail** обработчик **BeforePost**:

```
procedure TStockInForm.qryDetailBeforePost(DataSet: TDataSet);
begin
  qryDetail.FieldByName('ID').AsInteger := qryMaster.FieldByName('ID').AsInteger;
end;
```

Событие **BeforePost** происходит непосредственно перед тем, как данные будут посланы на сервер SQL-командой INSERT или UPDATE. Поле ID является связующим между Главной таблицей документа и подчиненной, то есть в пределах одного документа значение ID одинаково и в «шапке» и в «позициях».

Запустим проект. Установим в шапке значение «Расчет сумм от цены с НДС» и попробуем поменять «Цену» или «Количество». Мы видим, что вторая цена и все суммы рассчитываются автоматически. Мы можем также добавлять новые позиции в документ. Добавим несколько позиций. Для каждой из них необходимо, как минимум, выбрать товар из справочника и указать цену и количество.

Теперь сделаем так, чтобы, в зависимости от выбранного режима (расчет от цен с НДС или цен без НДС), вторая цена была недоступна для ручного редактирования. Лучше вообще сделать ее невидимой. Заодно запретим ручное редактирование полей сумм и отметим их иным цветом.

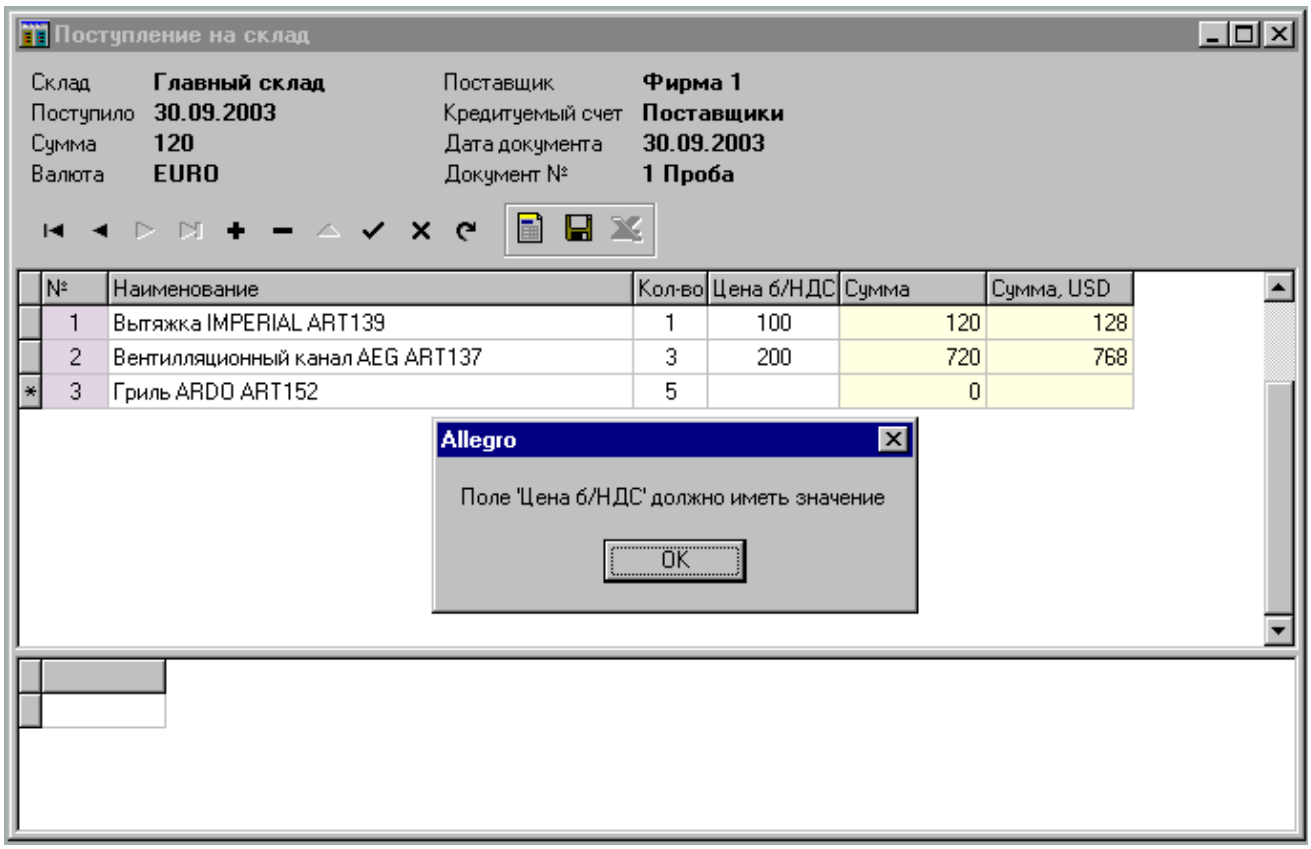
Для того чтобы запретить редактирование полей сумм в сетке, вызовем редактор колонок сетки (свойство **Columns** в Инспекторе объектов), выберем колонки **AMOUNT_L** и **AMOUNT_R**, окрасим их в другой цвет (свойство **Color**) и установим свойство **ReadOnly** = **True**.

Выберем компонент **dsrMaster** и создадим для него обработчик события **OnDataChange**:

```
procedure TStockInForm.dsrMasterDataChange(Sender: TObject; Field: TField);
begin
  {устанавливаем видимость колонок цены, в зависимости от режима расчета сумм}
  dbgDetail.Columns[3].Visible := qryMaster.FieldByName('CALC_MODE').AsInteger=1;
  dbgDetail.Columns[4].Visible := not dbgDetail.Columns[3].Visible;
end;
```

Теперь, в зависимости от режима расчета сумм, в сетке будет отображаться только нужная цена.

Запустим проект и добавим несколько позиций товара. Проверим, как все работает. Например, попробуем сохранить строку, в которой какое-то из обязательных полей оставлено пустым. Программа должна сообщить нам о том, что требуется значение в этом поле.



Собственно, интерфейс ввода данных для документа «Поступление на склад» можно было бы считать готовым. Но мы хотим еще показать, как реализовывается «поиск по нажатию клавиш», значительно ускоряющий поиск товаров и, как результат - ввод данных в документы.

Организуем поиск товаров «по нажатию клавиш»

Из бесед с заказчиком нам известно, что справочник товаров не так часто пополняется новыми позициями. Во всяком случае, непосредственно при вводе поступления на склад новые товары в справочнике, как правило, не создаются. Поэтому мы организуем интерфейс ввода, ориентированный не столько на добавление новых, сколько на максимально быстрый поиск уже имеющихся товаров. Известно, что заказчик использует локальную сеть 100 Mbit. Это позволяет нам организовать очень быстрый поиск, работающий практически мгновенно после нажатия каждой клавиши. Например, если пользователь нажмет букву «с», то сразу появляется список всех товаров, содержащих букву «с» в наименованиях. Если он после этого нажмет букву «т», то остаются все товары, содержащие в наименованиях подстроку «ст». Если он введет следующую комбинацию букв после пробела, то наш поиск отберет наименования по одновременному вхождению в них обеих комбинаций: до и после пробела. Если пользователь хочет быстро отыскать все стиральные машины BOSCH, то ему достаточно будет набрать «ст bos» и, скорее всего, в списке останутся только стиральные машины BOSCH. Практика показывает, что этот способ поиска наименований чрезвычайно эффективен и позволяет с огромной скоростью набирать документы. К тому же способ поиска «по вхождению» в наименования очень гибок: если пользователю известен артикул товара, то достаточно набрать артикул, Если же он хочет увидеть все товары какого-то вида, например, «вытяжки» достаточно набрать «выт» или доже просто букву «ж», так как в других наименованиях эта буква не встречается. С таким же успехом можно найти все товары с маркой AEG, набрав «aeg». SQL-Запрос, осуществляющий поиск по вхождению, использует конструкцию

```
SELECT * FROM <таблица>
WHERE          (UPPERCASE(<поле>) LIKE '%<ПРИЗНАК1>%') AND
(UPPERCASE(<поле>) LIKE '%<ПРИЗНАК2>%') AND
и так далее…
```

Для реализации «поиска по вхождению» имеется специальный компонент-сетка **DBGridA** с палитры **Allegro**, который мы использовали в качестве сетки для позиций документа. В отличие от простой сетки (**DBGrid**

с палитры **Data Controls**) сетка **DBGridA** имеет специальное событие **OnSetEditText**, которое происходит после нажатии каждой клавиши в режиме редактирования и в строковом параметре **Value** передает текущий в ячейке текст в обработчик. Вызвав в этом обработчике SQL-запрос, осуществляющий поиск по вхождению, мы сможем отобразить результаты поиска во второй сетке (нижней) и дать возможность пользователю вставить найденный товар в основную сетку.

Итак, добавим на форму компонент **IBQuery** с палитры **InterBase**. Назовем его **qryGoods** , установим свойство:

Transaction = traCurrent

а в свойстве SQL напишем запрос:

```
SELECT O1.OBJECT_ID, O1.SHORT_NAME
FROM GOODS G, OBJECT_NAMES O1
WHERE G.ID = O1.OBJECT_ID
```

Этот SQL-запрос запрашивает все наименования товаров из таблицы **OBJECT_NAMES**, он нужен нам лишь для того, чтобы создать «постоянные» поля у компонента запроса и настроить колонки нижней сетки. Щелкнем дважды на компоненте **qryGoods** и в редакторе полей добавим все поля с помощью соответствующего пункта контекстного меню. Для поля SHORT_NAME установим свойство:

DisplayLabel = Наименование

Добавим компонент **DataSource** с палитры **InterBase** (или **Data Access**) , назовем его **dsrGoods** и установим его свойство:

DataSet = qryGoods

У нижней сетки **dbgGoods** установим

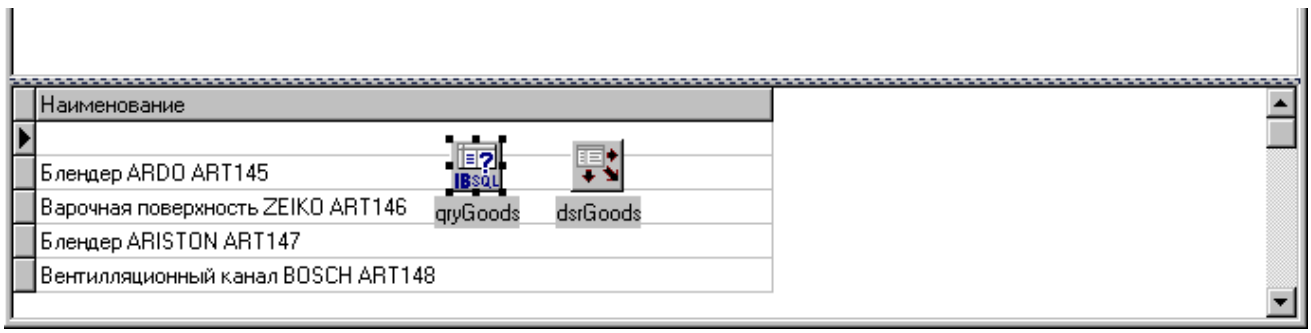
DataSource = dsrGoods

а в свойстве **Options** установим птичку на **dgRowSelect**.

Теперь создадим еще «постоянную» колонку у нижней сетки **dgbGoods**. Для этого вызовем редактор свойства **Columns**, дважды щелкнув на нем в Инспекторе объектов, добавим в список одну колонку и установим в Инспекторе объектов ее свойство:

FieldName = SHORT_NAME

Откроем запрос, установив свойство **Active** = **True**. Убедимся, что список всех наименований товаров появился в нижней сетке:



Закроем запрос, установив

Active = False

Теперь для сетки позиций документа **dbgDetail** создадим обработчик события **OnSetEditText**:

```
{Поиск по нажатию любой клавиши в сетке позиций}
procedure TStockInForm.dbgDetailSetEditText(Sender: TObject; ACol, ARow: Integer;
    const Value: String);
var
  s: string;
begin

  if Value = LastValue then
    exit; //защита от повторного поиска по тем же признакам
  if dbgDetail.SelectedField <> qryDetail.FieldByName('ITEM_NAME') then
    exit; //ограничение поиска нажатием клавиш в поле "Наименование"

  LastValue := Value;

  {Конструирование запроса из строки признаков, разделенных пробелами}
  s := 'SELECT O1.OBJECT_ID, O1.SHORT_NAME'#13+
      'FROM GOODS G, OBJECT_NAMES O1'#13+
      'WHERE G.ID = O1.OBJECT_ID AND O1.OBJECT_ID <> 0';
  if Value <> '' then
    s:=Format('%s AND'#13'%s',[s, StrToVarcharFilter('O1.SHORT_NAME','',Value)]);
  ResetCurrentTime; //сброс счетчика времени

  {включение светодиода в статусной строке Allegro}
  StatusBarDisplay(clLime,0,0,0,'','','');
  with qryGoods do
  begin
    Close;
    SQL.Text := s;
    Open; //открытие запроса
  end;

  {отображение в статусной строке времени, затраченного на поиск}
  StatusBarDisplay(clBlack,0,0,0,GetCurrentTimeStr,'','');
end;
```

В секцию **var** в разделе **implementation** модуля **StockIn** добавим объявление переменной:

```
  LastValue: string;
```

Добавим также в обработчик события **AfterInsert** компонента **qryDetail** строки, выделенные здесь жирным шрифтом:

```
procedure TStockInForm.qryDetailAfterInsert(DataSet: TDataSet);
begin
  if not lock_insert then
  begin
    lock_insert := True;
    qryDetail.Cancel; //отменяем режим вставки
    qryDetail.Append; //вставляем в конец набора

    LastValue := '';
    dbgDetail.SelectedField := qryDetail.FieldByName('ITEM_NAME');
                //перемещаем фокус ввода в поле наименования
  end;
  lock_insert := False;
end;
```

Создадим у сетки позиций **dbgDetail** обработчик события **OnKeyPress**, для того, чтобы при нажатии пользователем клавиши ***Enter*** фокус ввода из сетки позиций перемещался бы в сетку найденных наименований товаров:

```
{Нажатие клавиш Enter в списке позиций}
procedure TStockInForm.dbgDetailKeyPress(Sender: TObject; var Key: Char);
begin
  if (Key = 13) and not qryGoods.IsEmpty then
    dbgGoods.SetFocus; //Перемещение фокуса ввода в нижний список
end;
```

После того, как пользователь выберет в нижней сетке нужный товар, он двойным щелчком мыши на нем или нажатием клавиши ***Enter*** скопирует его в сетку позиций. При этом фокус ввода переместится обратно в сетку позиций. Реализуем это в виде двух обработчиков для нижней сетки **dgbGoods**. Сначала создадим обработчик для двойного щелчка мыши **OnDblClick**:

```
{Копирование позиции из нижнего списка товаров двойным щелчком мыши}
procedure TStockInForm.dbgGoodsDblClick(Sender: TObject);
begin
  qryDetail.Edit; {Присвоение значений полям позиции}
  qryDetail.FieldByName('GOODS').AsInteger :=
     qryGoods.FieldByName('OBJECT_ID').AsInteger;
  qryDetail.FieldByName('ITEM_NAME').AsString :=
     qryGoods.FieldByName('SHORT_NAME').AsString;
  LastValue := qryDetail.FieldByName('ITEM_NAME').AsString;
  dbgDetail.SelectedField := qryDetail.FieldByName('QUANTITY');
  dbgDetail.SetFocus; //Возвращение фокуса ввода в сетку позиций
end;
```

А теперь создадим обработчик **OnKeyPress:**

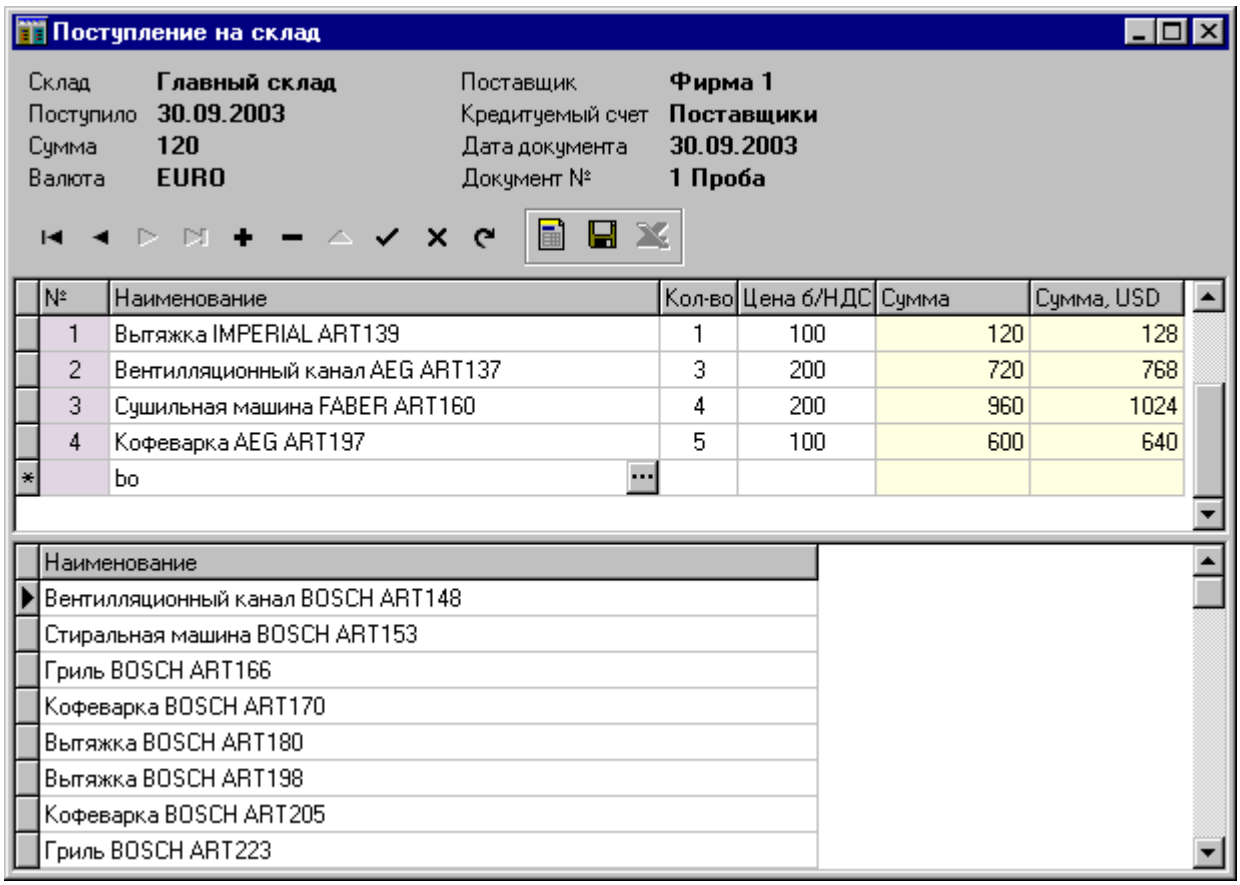
```
procedure TStockInForm.dbgGoodsKeyPress(Sender: TObject; var Key: Char);
begin
  if Key = 13 then
    dbgGoodsDblClick(nil) // Выбор и добавление товара в сетку позиций на Enter
  else if Key = 27 then
    dbgDetail.SetFocus; //Простое возвращение фокуса ввода в сетку позиций на Esc
end;
```

Для отображения подсказки при наведении курсора мыши на нижнюю сетку **dbgGoods**, установим у нее свойства:

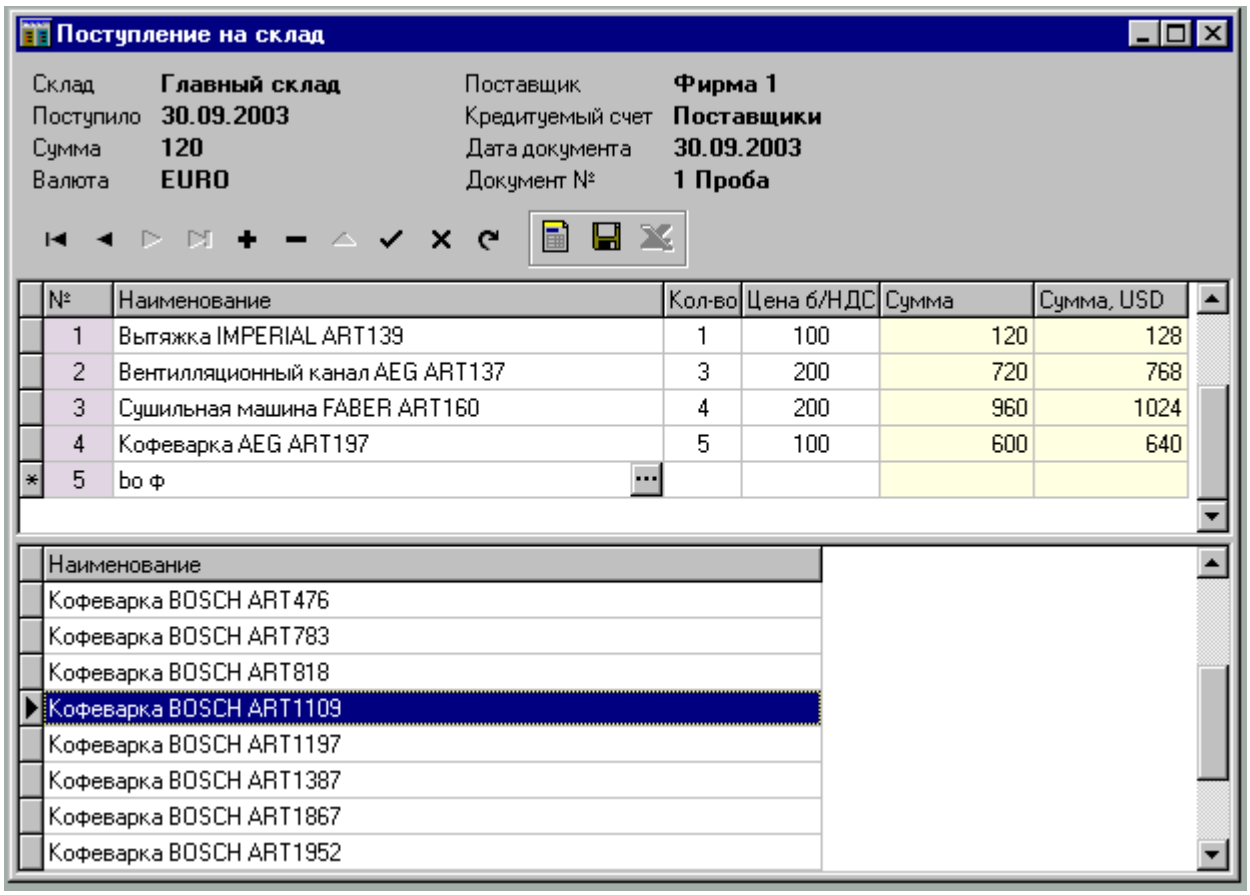
Hint = Для перемещения в этот список нажмите Enter
Для выбора из этого списка также нажмите Enter
ShowHint = True

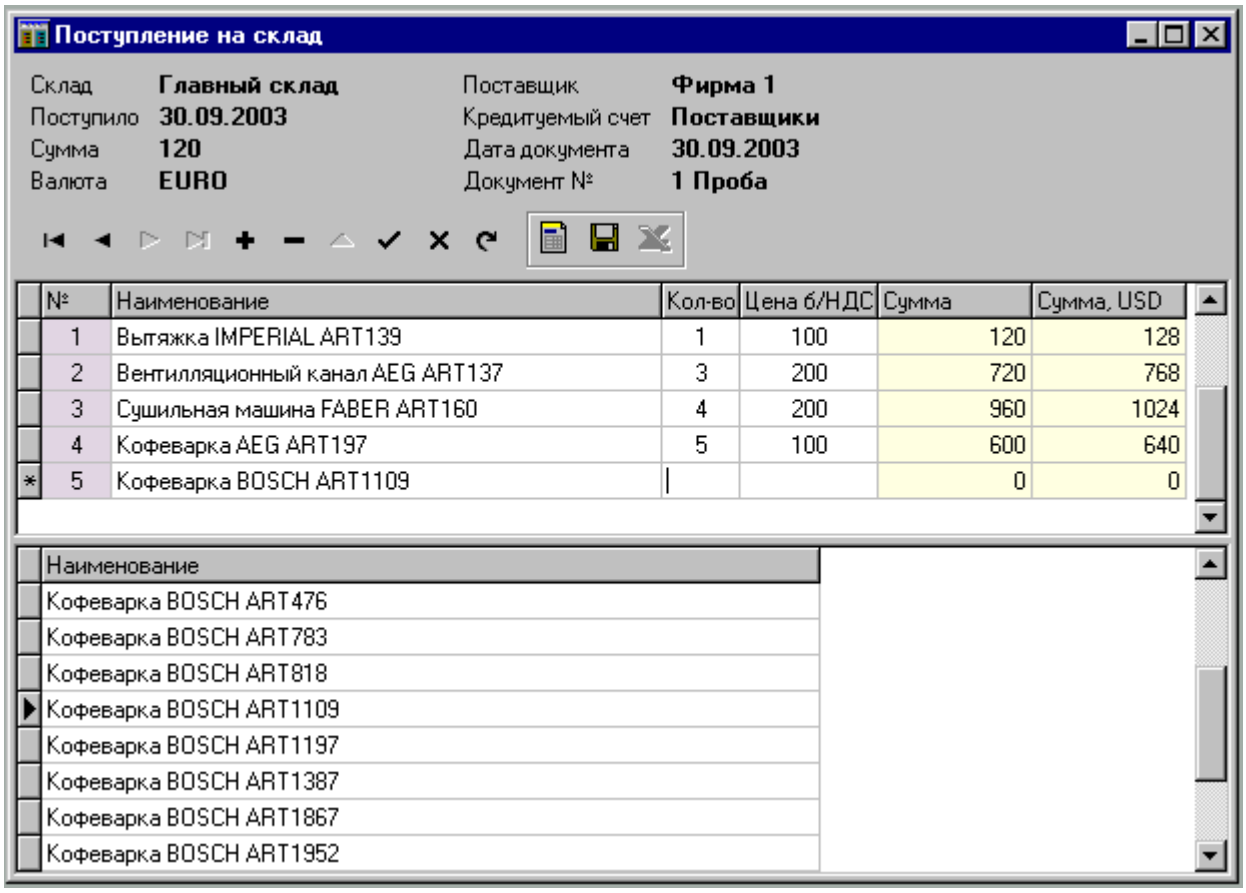
Теперь запустим проект и последовательно выполним следующие действия:

- Выберем нижнюю позицию в документе
- Нажмем на клавиатуре «стрелку вниз». Должна добавиться пустая строка, а фокус ввода должен оказаться в поле «Наименование»
- Наберем с клавиатуры две буквы: **bo**. В сетке наименований должны остаться только товары марки BOSCH:



- Нажмем клавишу «пробел» и после него нажмем русскую букву **ф**. В сетке наименований должны остаться только кофеварки BOSCH. Нажмем клавишу ***Enter***. Фокус ввода должен переместиться в сетку наименований. Выберем нужную кофеварку из списка, перемещаясь по списку клавишами «стрелка вверх» и «стрелка вниз»
- Нажмем еще раз клавишу ***Enter***. Выбранный из списка товар вставится в текущую позицию документа, а фокус ввода вернется в сетку позиций и переместится в ячейку «Количество», ожидая ввода пользователем количества выбранного товара.
- Вводим количество и нажимаем клавишу «стрелка вправо» для того, чтобы перейти в ячейку, в которой вводится цена.
- Введем цену и нажмем клавишу «стрелка вниз». Нажатие этой клавиши приведет к посылке всей набранной позиции на сервер, а в сетку добавится новая пустая позиция.
- Для отмены добавления/редактирования очередной позиции нужно нажать клавишу ***Escape*** 2 раза. Первое нажатие отменяет редактирование ячейки, второе – редактирование всей строки.
- Сохранение документа производится нажатием кнопки с дискетой или комбинации горячих клавиш ***Ctrl+S***.


Поступление на склад
Склад Главный склад
Поступило 30.09.2003
Сумма 120
Валюта EURO
Поставщик Фирма 1
Кредитуемый счет Поставщики
Дата документа 30.09.2003
Документ № 1 Проба
№ Наименование Кол-во Цена б/НДС Сумма Сумма, USD
1 Вытяжка IMPERIAL ART139 1 100 120 128
2 Вентилляционный канал AEG ART137 3 200 720 768
3 Сушильная машина FABER ART160 4 200 960 1024
4 Кофеварка AEG ART197 5 100 600 640
5 bo ф
Наименование
Кофеварка BOSCH ART476
Кофеварка BOSCH ART783
Кофеварка BOSCH ART818
Кофеварка BOSCH ART1109
Кофеварка BOSCH ART1197
Кофеварка BOSCH ART1387
Кофеварка BOSCH ART1867
Кофеварка BOSCH ART1952


Поступление на склад
Склад Главный склад
Поступило 30.09.2003
Сумма 120
Валюта EURO
Поставщик Фирма 1
Кредитуемый счет Поставщики
Дата документа 30.09.2003
Документ № 1 Проба
№ Наименование Кол-во Цена б/НДС Сумма Сумма, USD
1 Вытяжка IMPERIAL ART139 1 100 120 128
2 Вентилляционный канал AEG ART137 3 200 720 768
3 Сушильная машина FABER ART160 4 200 960 1024
4 Кофеварка AEG ART197 5 100 600 640
5 Кофеварка BOSCH ART1109 0 0
Наименование
Кофеварка BOSCH ART476
Кофеварка BOSCH ART783
Кофеварка BOSCH ART818
Кофеварка BOSCH ART1109
Кофеварка BOSCH ART1197
Кофеварка BOSCH ART1387
Кофеварка BOSCH ART1867
Кофеварка BOSCH ART1952

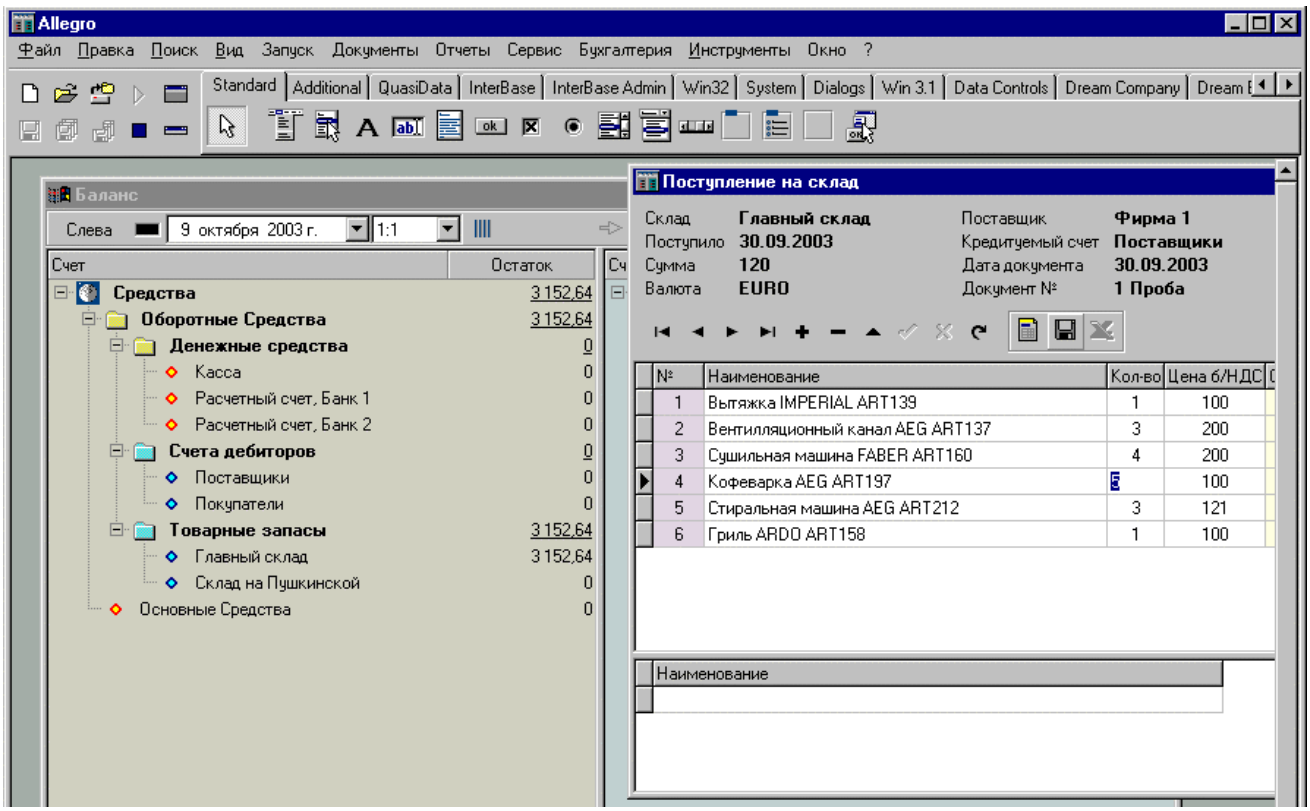
Реализуем перепроведение документа «Поступление на склад».

Для того чтобы документ перепровелся, достаточно вызвать процедуру **MakeEntries**. В эту процедуру нужно передать всего два параметра: название главной таблицы документа и указатель на компонент транзакции. После перепроведения документа и подтверждения транзакции желательно всегда еще вызывать процедуру **ReafreshBalance**, которая освежит баланс компании, если окно баланса присутствует на экране.

Добавим вызов этих двух процедур (они выделены жирным шрифтом) в обработчик **OnExecute** команды **actSave**:

```
procedure TStockInForm.actSaveExecute(Sender: TObject);
begin
  MakeEntries('STOCK_IN', qryMaster.FieldByName('ID').AsInteger, traCurrent);
  traCurrent.CommitRetaining; //подтвердить транзакцию
  Modified := False;
  RefreshExplorer; //пересветить "Проводник по документам"
  RefreshBalance;  //пересветить "Баланс"
end;
```

Запустим проект, вызовем окно «Баланс» через меню **Бухгалтерия /Баланс**, выберем в балансе слой **USD**. Вызовем на передний план окно документа через меню **Окно/Поступление на склад** и отодвинем его так, чтобы были видны суммы в окне «Баланс» на счетах **Товарных запасов**. Теперь изменим что-то в цифрах документа и сохраним. Сумма на счете товарных запасов в балансе должна измениться:



Нам осталось реализовать перерасчет сумм в главной таблице документа STOCK_IN. Это суммы в полях TOTAL_L, TOTAL_S и TOTAL_R. В принципе без этих полей можно было бы обойтись, но иногда удобно иметь поля сумм в главной таблице, например, в данном случае мы используем одну из этих сумм для отображения «сумм» документов «Поступление на склад» в «Проводнике по документам». Также удобно иметь сумму документа при поиске по сумме и для ускорения некоторых отчетов.

Мы сделаем расчет сумм в момент сохранения документа.

Добавим к форме **StockInForm** еще один компонент **IBQuery** с палитры **InterBase**.

Установим его свойство:

```
Transaction = traCurrent
DataSource = dsrMaster
Name = qryTotals
```

Дважды щелкнем на свойстве SQL и впишем такой запрос:

```
SELECT SUM(AMOUNT_L), SUM(AMOUNT_R), SUM(AMOUNT_S)
FROM  STOCK_IN_ITEM
WHERE  ID = :ID
```

А в обработчик события **OnExecute** команды **actSave** добавим ряд строк, которые выделены здесь жирным шрифтом:

```
procedure TStockInForm.actSaveExecute(Sender: TObject);
begin
  qryTotals.Open;
  qryMaster.Edit;
  qryMaster.FieldByName('TOTAL_L').AsCurrency := qryTotals.Fields[0].AsCurrency;
  qryMaster.FieldByName('TOTAL_R').AsCurrency := qryTotals.Fields[1].AsCurrency;
  qryMaster.FieldByName('TOTAL_S').AsCurrency := qryTotals.Fields[2].AsCurrency;
  qryMaster.Post;
  qryTotals.Close;
  MakeEntries('STOCK_IN', qryMaster.FieldByName('ID').AsInteger, traCurrent);
  traCurrent.CommitRetaining; //подтвердить транзакцию
  Modified := False;
  RefreshExplorer; //пересветить "Проводник по документам"
  RefreshBalance;  //пересветить "Баланс"
end;
```

Здесь мы обращаемся к полям компонента запроса **qryTotals** по их индексам. А к полям компонента **qryMaster** – по именам. Оба этих способа можно использовать. Способ обращения по имени предпочтительнее, когда мы хотим, чтобы у программы был более читабельный текст.

Дабавим еще маленький штрих – ограничение на минимальный размер формы на экране. Для этого в Инспекторе объектов для формы **StockInForm** в свойстве **Constraints** зададим:

```
MinHeight = 500
MinWidth = 650
```

Запустим проект, изменим какую-нибудь цену и нажмем кнопку сохранения. Обратим внимание, что сумма документа, отображаемая на верхней панели, изменилась. Теперь она всегда становится равна сумме позиций после сохранения документа. Попробуем уменьшить размер окна. Мы видим, что наши ограничения на минимальный размер окна тоже работают.

Еще один штрих, которого не хватает – возможности видеть, был ли изменен документ и нуждается ли он в сохранении. Наиболее элегантный способ этого отображения – управлять свойством **Enabled** кнопки «сохранить».

Найдем с помощью поиска ***Ctrl+F*** все места в тесте модуля **stock_in**, в которых происходит **присвоение** переменной **Modified** значения **True** или **False**.

Вставим после каждого такого присвоения строку:

```
actSave.Enabled := Modified;
```

В принципе, можно вообще отказаться от использования переменной **Modified** и использовать вместо нее свойство **Enabled**, но мы так не поступим: это может сделать текст нашей программы менее понятным. Запустим проект и убедимся, что кнопка с дискетой неактивна. Внесем какое-нибудь изменение и убедимся в том, что кнопка стала активной.

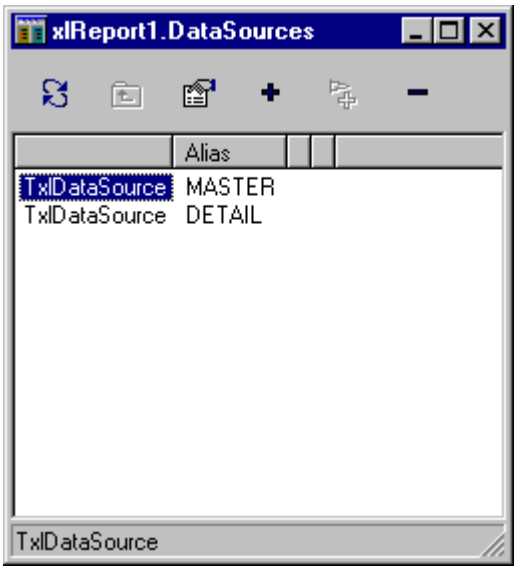
Построение отчета

Программа Allegro не имеет встроенных средств печати. Вместо этого используется экспорт в Microsoft Excel, который должен быть установлен на компьютере. Экспорт в Excel осуществляется при помощи встроенного в программу компонента **XLReport** фирмы AfalinaSoft.

Откроем проект документа «Поступление на склад» в режиме дизайна.

Добавим на форму компонент **XLReport** с палитры **Print**.

Вызовем в Инспекторе объектов редактор свойства **DataSources** и добавим в список 2 источника данных:

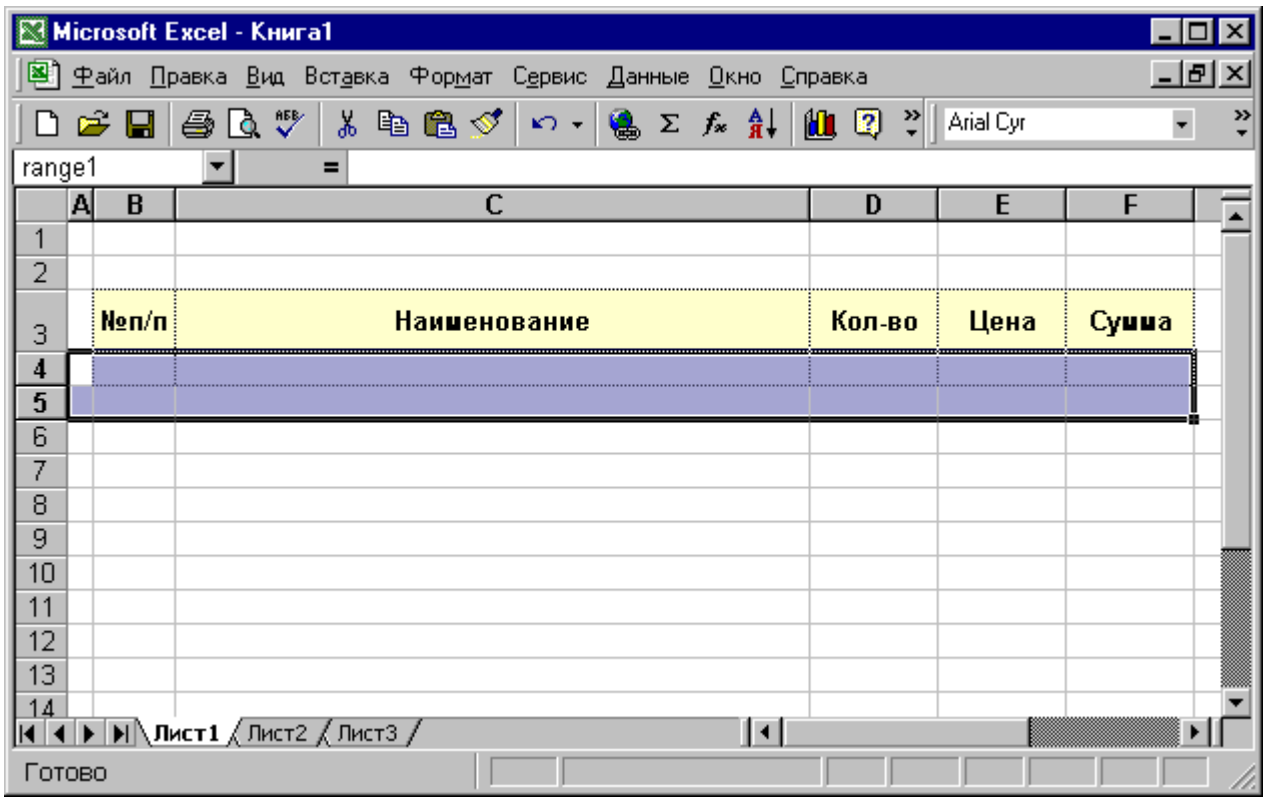


В инспекторе объектов подключим первый источник к главной таблице документа:
Dataset = qryMaster
Alias = MASTER

Второй подключим к подчиненной таблице документа:
Dataset = qryDetail
Alias = DETAIL
Range = range1

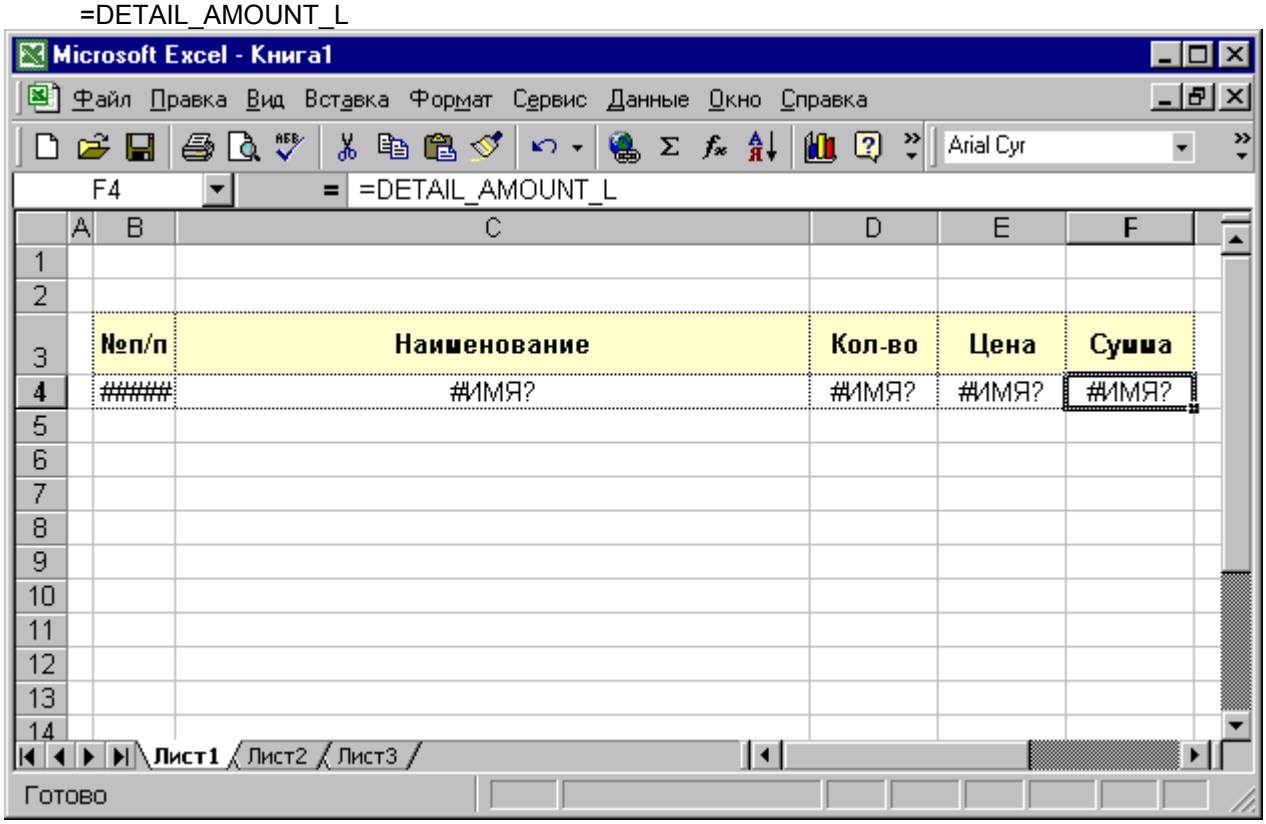
Свойство **Range** содержит имя региона ячеек range1 шаблона Excel, который нам еще предстоит создать.

Теперь создадим шаблон. Для этого дважды щелкнем на компоненте **XLReport1**. Вызовется Microsoft Excel. Распишем заголовки будущих колонок документа, оставив слева одну пустую колонку. Выделим ячейки будущей строки документа, захватив эту пустую колонку и одну пустую добавочную строку снизу, как показано на рисунке. Назовем выделенный регион ячеек range1, набрав это слово в выпадающем списке слева:

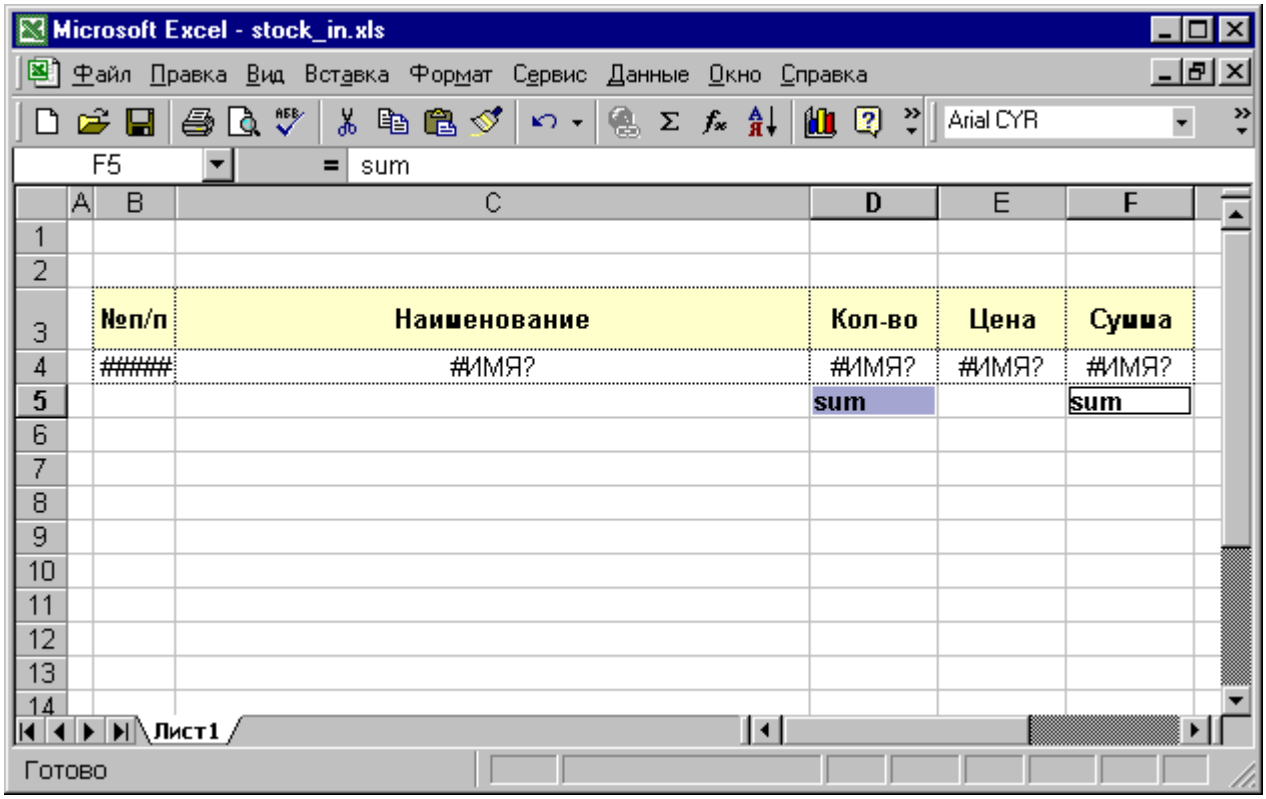


Теперь выберем по одной каждую ячейку будущей строки документа и впишем в нее знак равенства, затем название (ALIAS) источника данных и имя поля, которое следует отобразить в этой ячейке, разделенные знаком подчеркивания:

```
=DETAIL_NUM
=DETAIL_ITEM_NAME
=DETAIL_QUANTITY
=DETAIL_PRICE_L
=DETAIL_AMOUNT_L
```



Удалим лишние листы (Лист2 и Лист3) из книги, сохраним Книгу в файле под названием stock_in.xls в той же директории, где мы храним модули нашего проекта. В дополнительной строке региона, под ячейкой, в которой должно выводиться количество, и под ячейкой в которой должна выводиться сумма напишем слово **sum** и придадим этим ячейкам жирный шрифт:



Сохраним изменения в шаблоне, закроем Excel и вернемся к нашему проекту.

В Инспекторе объектов у компонента **XLReport1** в свойстве **XLSTemplate** запишем имя файла шаблона **stock_in.xls**.

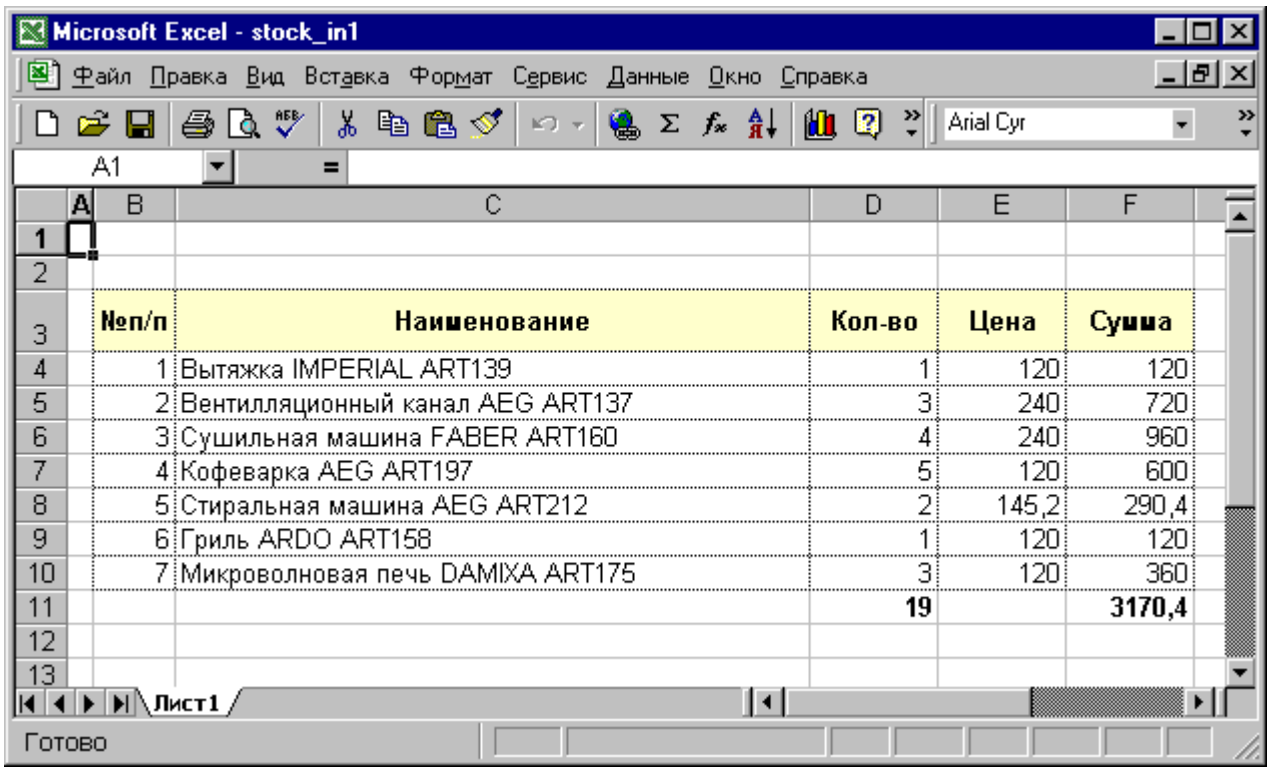
Дважды щелкнем на компоненте **ActionList1**, выберем в списке команд **Отчет** и в Инспекторе объектов назначим этой команде обработчик события **OnExecute**:

```
procedure TStockInForm.actReportExecute(Sender: TObject);
begin
  XLReport1.Report;
end;
```

Выберем в проекте компонент **XLReport1** и создадим ему обработчик события **OnProgress**:

```
{Отображение прогресса в статусной строке}
procedure TStockInForm.xlReport1Progress(Report: TxlReport; const Position, Max: Integer);
begin
  if Position = 0 then
    StatusBarDisplay(clBlack, Position, 0, Max, '', '', '')
  else
    StatusBarDisplay(clLime, Position, 0, Max, '', 'Экспорт в Excel...', '');
end;
```

Запустим проект и нажмем кнопку «Отчет». Произойдет экспорт в Excel и отображение отчета на экране.



№п/п	Наименование	Кол-во	Цена	Сумма
1	Вытяжка IMPERIAL ART139	1	120	120
2	Вентилляционный канал AEG ART137	3	240	720
3	Сушильная машина FABER ART160	4	240	960
4	Кофеварка AEG ART197	5	120	600
5	Стиральная машина AEG ART212	2	145,2	290,4
6	Гриль ARDO ART158	1	120	120
7	Микроволновая печь DAMIXA ART175	3	120	360
		19		**3170,4**

Закроем Excel. Остановим работу проекта. Дважды щелкнем на компоненте **XLReport1** для того, чтобы вызвать шаблон.

Отформатируем ячейки: поля нумерации количества и цены сделаем выравнивание по центру, сумму отформатируем как числа с 2 десятичными разрядами.

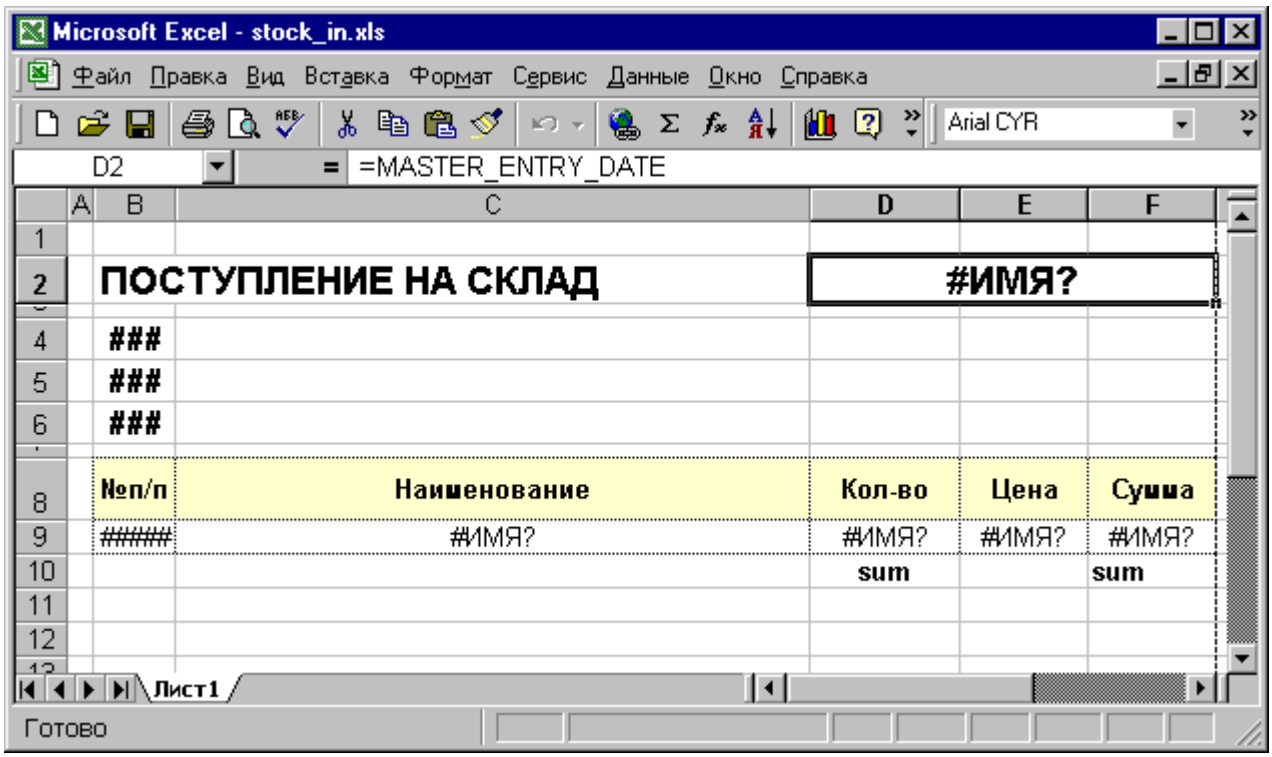
Добавим 5 рядов ячеек вверху документа для размещения полей шапки.

Добавим крупным шрифтом надпись «Поступление на склад»

Справа от нее объединим три ячейки и впишем в них =MASTER_ENTRY_DATE, зададим в этих ячейках принудительное форматирование в виде даты.

Ниже на трех строках разместим жирным шрифтом:

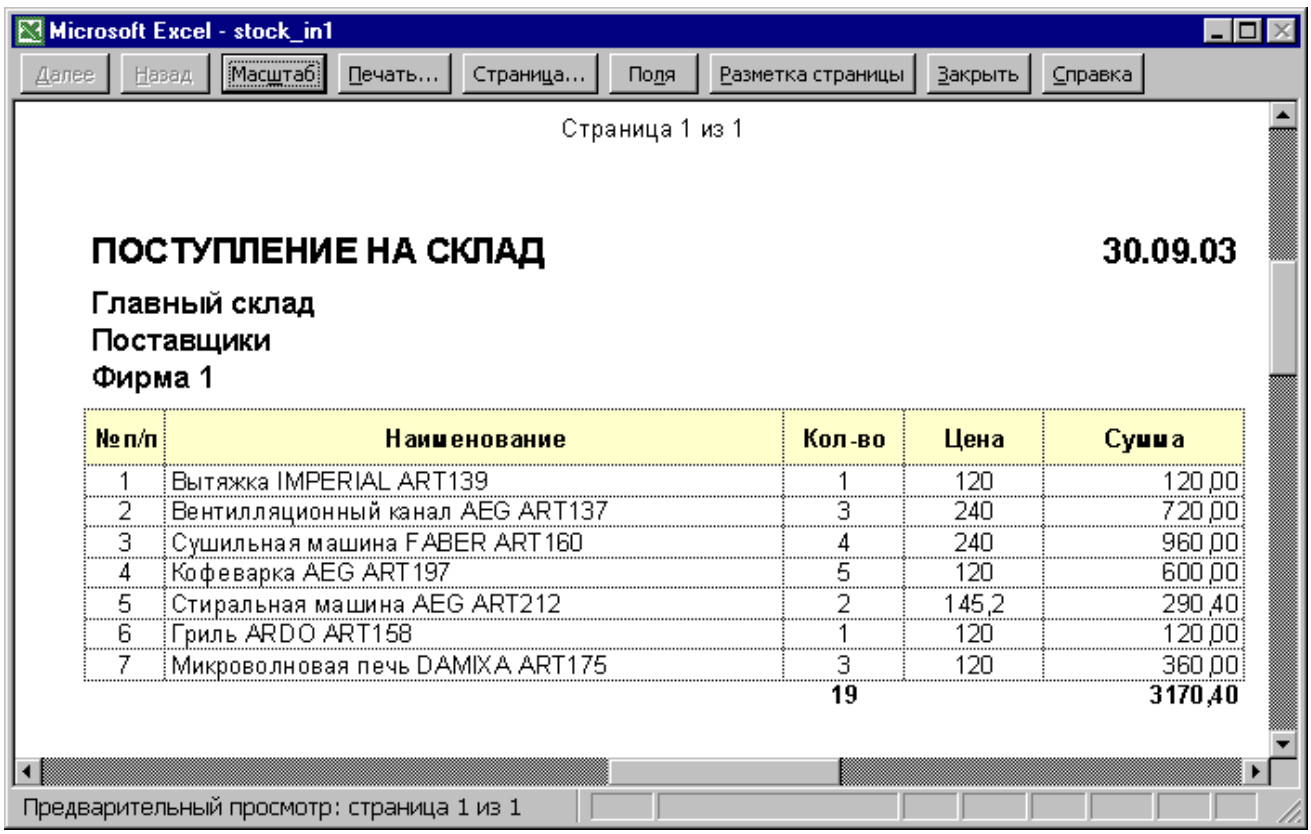
```
=MASTER_STOCK_ACC_NAME
=MASTER_CREDIT_ACC_NAME
=MASTER_CONTRAGENT_NAME
```



Для отображения нумерации страниц, в «Параметрах страницы» на закладке «Колонтитулы» создадим верхний колонтитул вида «Страница 1 из ?». Еще объявим в «Параметрах страницы» сквозные строки $8:$8 на закладке «Лист».

Сохраним шаблон и закроем Excel.

Запустим проект и сделаем экспорт. Вызовем в Excel «Предварительный просмотр»:



Итак, мы полностью реализовали интерфейс документа «Поступление на склад».

Полный листинг проекта «Поступление на склад»

```
unit stock_in;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  ExtCtrls, ComCtrls, Menus,DCMenuEditor,DBGridA, DBGrids, IBCustomDataSet,
  IBDatabase, DB,StdCtrls, DBCtrls,ActnList,AlgCtrls,IBQuery,xlReport;

type
  TStockInForm = class(TForm)
    TopPanel: TPanel;
    dbgDetail: TDBGridA;
    Splitter1: TSplitter;
    dbgGoods: TDBGrid;
    qryMaster: TIBDataSet;
    qryDetail: TIBDataSet;
    traCurrent: TIBTransaction;
    dsrMaster: TDataSource;
    dsrDetail: TDataSource;
    qryDetailID1: TIntegerField;
    qryDetailN1: TIntegerField;
    qryDetailGOODS1: TIntegerField;
    qryDetailITEM_NAME1: TIBStringField;
    qryDetailQUANTITY1: TIntegerField;
    qryDetailPRICE_L1: TIBBCDField;
    qryDetailPRICE_L_WO_VAT1: TIBBCDField;
    qryDetailPRICE_R1: TIBBCDField;
    qryDetailPRICE_R_WO_VAT1: TIBBCDField;
    qryDetailAMOUNT_L1: TIBBCDField;
    qryDetailAMOUNT_R1: TIBBCDField;
    qryDetailAMOUNT_S1: TIBBCDField;
    qryMasterID1: TIntegerField;
    qryMasterDIR_ID1: TIntegerField;
    qryMasterDOC_DATE1: TDateField;
    qryMasterDOC_KIND1: TIntegerField;
    qryMasterDOC_NO1: TIBStringField;
    qryMasterENTRY_DATE1: TDateField;
    qryMasterHAS_ENTRY1: TSmallintField;
    qryMasterSTOCK_ACC1: TIntegerField;
    qryMasterLAYER_ID1: TIntegerField;
    qryMasterLAYER_NAME1: TIBStringField;
    qryMasterVAT_RATE1: TIBBCDField;
    qryMasterACC_ID1: TIntegerField;
    qryMasterCONTRAGENT1: TIntegerField;
    qryMasterCONTRAGENT_NAME1: TIBStringField;
    qryMasterCALC_MODE1: TIntegerField;
    qryMasterTOTAL_L1: TIBBCDField;
    qryMasterTOTAL_R1: TIBBCDField;
    qryMasterTOTAL_S1: TIBBCDField;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    Label4: TLabel;
    DBText1: TDBText;
    DBText2: TDBText;
    DBText3: TDBText;
    DBText4: TDBText;
```

```
  Label5: TLabel;
  Label6: TLabel;
  Label7: TLabel;
  Label8: TLabel;
  DBText5: TDBText;
  DBText6: TDBText;
  DBText7: TDBText;
  DBText8: TDBText;
  qryMasterSTOCK_ACC_NAME1: TIBStringField;
  qryMasterCREDIT_ACC_NAME1: TIBStringField;
  PopupMenu1: TPopupMenu;
  ImageList1: TImageList;
  qryMasterEXCH_RATE_L1: TFloatField;
  qryMasterEXCH_RATE_S1: TFloatField;
  Timer1: TTimer;
  ActionList1: TActionList;
  actHeader: TAction;
  actSave: TAction;
  actReport: TAction;
  N1: TMenuItem;
  N2: TMenuItem;
  N3: TMenuItem;
  DBNavigator1: TDBNavigator;
  ToolBar1: TToolBar;
  ToolButton1: TToolButton;
  ToolButton2: TToolButton;
  ToolButton3: TToolButton;
  qryDetailNUM1: TIntegerField;
  RefDialog1: TRefDialog;
  qryGoods: TIBQuery;
  qryGoodsOBJECT_ID1: TIntegerField;
  qryGoodsSHORT_NAME1: TIBStringField;
  dsrGoods: TDataSource;
  qryTotals: TIBQuery;
  xlReport1: TxlReport;
  procedure FormCreate(Sender: TObject);
  procedure actHeaderExecute(Sender: TObject);
  procedure actSaveExecute(Sender: TObject);
  procedure FormCloseQuery(Sender: TObject; var CanClose: Boolean);
  procedure qryMasterAfterPost(DataSet: TDataSet);
  procedure Timer1Timer(Sender: TObject);
  procedure qryDetailAfterInsert(DataSet: TDataSet);
  procedure qryDetailCalcFields(DataSet: TDataSet);
  procedure qryDetailAfterDelete(DataSet: TDataSet);
  procedure dbgDetailEditButtonClick(Sender: TObject);
  procedure dsrDetailDataChange(Sender: TObject; Field: TField);
  procedure qryDetailBeforePost(DataSet: TDataSet);
  procedure dsrMasterDataChange(Sender: TObject; Field: TField);
  procedure dbgDetailSetEditText(Sender: TObject; ACol, ARow: Integer; const Value: String);
  procedure dbgDetailKeyPress(Sender: TObject; var Key: Char);
  procedure dbgGoodsDblClick(Sender: TObject);
  procedure dbgGoodsKeyPress(Sender: TObject; var Key: Char);
  procedure actReportExecute(Sender: TObject);
  procedure xlReport1Progress(Report: TxlReport; const Position, Max: Integer);
private
  { Private declarations }
public
  { Public declarations }
end;
```

```
var
  StockInForm: TStockInForm;

implementation
uses
  stock_in_header;


{$R *.DFM}

var
  Modified: boolean;
  LastValue: string;

procedure TStockInForm.FormCreate(Sender: TObject);
begin
  traCurrent.StartTransaction; //стартовать транзакцию
  Modified := False;
  actSave.Enabled := Modified;
  qryMaster.ParamByName('ID').AsInteger := RunContext.Documents[0].doc_id;
  qryMaster.Open;

  if RunContext.Documents[0].doc_id <=0 then
  begin
    qryMaster.Insert; {Начальные значения полей нового документа}
    qryMaster.FieldByName('DIR_ID').AsInteger := RunContext.dir_id;
    qryMaster.FieldByName('DOC_KIND').AsInteger := 0;
    qryMaster.FieldByName('CALC_MODE').AsInteger := 1;
    qryMaster.FieldByName('DOC_DATE').AsDateTime := Date;
    qryMaster.FieldByName('HAS_ENTRY').AsInteger := 1;
    qryMaster.FieldByName('ENTRY_DATE').AsDateTime := Date;
    qryMaster.FieldByName('VAT_RATE').AsCurrency := 20;
    qryMaster.FieldByName('TOTAL_L').AsCurrency := 0;
    qryMaster.FieldByName('TOTAL_S').AsCurrency := 0;
    qryMaster.FieldByName('TOTAL_R').AsCurrency := 0;
    qryMaster.FieldByName('CONTRAGENT').AsInteger := 0;
    qryMaster.FieldByName('STOCK_ACC').AsInteger := 1007; //Главный склад
    qryMaster.FieldByName('ACC_ID').AsInteger := 1012; //Поставщики
    if StockInHeaderForm.ShowModal <> mrOK then
      Timer1.Enabled := True;
  end;

  if Timer1.Enabled then
    exit;

  self.Caption := NameOfDocument('STOCK_IN', qryMaster.FieldByName('ID').AsInteger,
      traCurrent); //устанавливаем новый заголовок формы

qryDetail.ParamByName('ID').AsInteger := RunContext.Documents[0].doc_id;
  qryDetail.Open;
end;

procedure TStockInForm.actHeaderExecute(Sender: TObject);
begin
   if StockInHeaderForm.ShowModal = mrOK then
      self.Caption := NameOfDocument('STOCK_IN', qryMaster.FieldByName('ID').AsInteger,
        traCurrent); //устанавливаем новый заголовок формы

end;
```

```
procedure TStockInForm.actSaveExecute(Sender: TObject);
begin
  qryTotals.Open;
  qryMaster.Edit;
  qryMaster.FieldByName('TOTAL_L').AsCurrency := qryTotals.Fields[0].AsCurrency;
  qryMaster.FieldByName('TOTAL_R').AsCurrency := qryTotals.Fields[1].AsCurrency;
  qryMaster.FieldByName('TOTAL_S').AsCurrency := qryTotals.Fields[2].AsCurrency;
  qryMaster.Post;
  qryTotals.Close;
  MakeEntries('STOCK_IN', qryMaster.FieldByName('ID').AsInteger, traCurrent);
  traCurrent.CommitRetaining; //подтвердить транзакцию
  Modified := False;
  actSave.Enabled := Modified;
  RefreshExplorer; //пересветить "Проводник по документам"
  RefreshBalance;  //пересветить "Баланс"
end;

procedure TStockInForm.FormCloseQuery(Sender: TObject; var CanClose: Boolean);
begin
  if Modified then
  case MessageDlg(
     'Сохранить изменения в документе ?', mtConfirmation,
     mkSet(mbYes, mbNo, mbCancel), 0) of
  mrYes: actSaveExecute(nil); //вызов обработчика OnExecute команды actSave
  mrCancel: Canclose := False;
  end;
end;

procedure TStockInForm.qryMasterAfterPost(DataSet: TDataSet);
begin
  Modified := True;
  actSave.Enabled := Modified;
end;

procedure TStockInForm.Timer1Timer(Sender: TObject);
begin
  Timer1.Enabled := False;
  Modified := False;
  actSave.Enabled := Modified;
  Self.Close;
end;

var
  lock_insert: boolean;

procedure TStockInForm.qryDetailAfterInsert(DataSet: TDataSet);
begin
  if not lock_insert then
  begin
    lock_insert := True;
    qryDetail.Cancel; //отменяем режим вставки
    qryDetail.Append; //вставляем в конец набора
    LastValue := '';
    dbgDetail.SelectedField := qryDetail.FieldByName('ITEM_NAME');
                  //перемещаем фокус ввода в поле наименования
  end;
  lock_insert := False;
end;
```

```
procedure TStockInForm.qryDetailCalcFields(DataSet: TDataSet);
begin
  with DataSet do
    FieldByName('NUM').AsInteger := RecNo;
end;


procedure TStockInForm.qryDetailAfterDelete(DataSet: TDataSet);
begin
  Modified := True;
  DataSet.Close;
  DataSet.Open;
  actSave.Enabled := Modified;
end;

procedure TStockInForm.dbgDetailEditButtonClick(Sender: TObject);
begin
  with RefDialog1 do
  if Execute then //вызываем диалог на экран и проверяем, был ли выбран товар
  begin
    qryDetail.Edit; //переводим набор данных в режим редактирования

    {присваиваем значение ID товара полю GOODS набора}
    qryDetail.FieldByName('GOODS').AsInteger := Object_ID;

    {присваиваем значение краткое наименование товара полю ITEM_NAME набора}
    qryDetail.FieldByName('ITEM_NAME').AsString :=
                NameOfRefObject(Object_ID, 'SHORT_NAME');

    {перемещаем фокус ввода в сетке в поле "Количество"}
    dbgDetail.SelectedField := qryDetail.FieldByName('QUANTITY');
  end;
end;

procedure TStockInForm.dsrDetailDataChange(Sender: TObject; Field: TField);
begin
  if (Field <> nil) and
     ((Field.FieldName = 'PRICE_L') or
      (Field.FieldName = 'PRICE_L_WO_VAT') or
      (Field.FieldName = 'QUANTITY'))then
  with qryDetail do
  begin
    {Если режим расчета сумм на основе цен с НДС
    и изменено поле "Цена с НДС"  или "Количество"}
    if ((Field.FieldName = 'PRICE_L')  or (Field.FieldName = 'QUANTITY')) and
       (qryMaster.FieldByName('CALC_MODE').AsInteger = 1) then
    begin

      FieldByName('AMOUNT_L').AsCurrency :=
        FieldByName('PRICE_L').AsCurrency *
        FieldByName('QUANTITY').AsInteger;
      FieldByName('PRICE_L_WO_VAT').AsCurrency :=
          round(FieldByName('PRICE_L').AsCurrency * 1000 * 100/
          (qryMaster.FieldByName('VAT_RATE').AsCurrency + 100))/1000;

      FieldByName('PRICE_R').AsCurrency :=
          FieldByName('PRICE_L').AsCurrency *
          qryMaster.FieldByName('EXCH_RATE_L').AsCurrency;
      FieldByName('PRICE_R_WO_VAT').AsCurrency :=
```

```
        round(FieldByName('PRICE_R').AsCurrency * 1000 * 100/
        (qryMaster.FieldByName('VAT_RATE').AsCurrency + 100))/1000;
      FieldByName('AMOUNT_R').AsCurrency :=
        round(FieldByName('PRICE_R').AsCurrency *
        FieldByName('QUANTITY').AsInteger);
    end
    else
    {Если режим расчета сумм на основе цен без НДС и
    изменено поле "Цена без НДС" или "Количество"}
    if ((Field.FieldName = 'PRICE_L_WO_VAT')   or (Field.FieldName = 'QUANTITY')) and
      (qryMaster.FieldByName('CALC_MODE').AsInteger = 0) then
    begin

      FieldByName('AMOUNT_L').AsCurrency :=
       round(FieldByName('PRICE_L_WO_VAT').AsCurrency * 10 *
        FieldByName('QUANTITY').AsInteger *
        (qryMaster.FieldByName('VAT_RATE').AsCurrency + 100))/1000;
      FieldByName('PRICE_L').AsCurrency :=
       round(FieldByName('PRICE_L_WO_VAT').AsCurrency * 10 *
        (qryMaster.FieldByName('VAT_RATE').AsCurrency + 100))/1000;

      FieldByName('PRICE_R_WO_VAT').AsCurrency :=
        FieldByName('PRICE_L_WO_VAT').AsCurrency *
        qryMaster.FieldByName('EXCH_RATE_L').AsCurrency;;
      FieldByName('PRICE_R').AsCurrency :=
        round(FieldByName('PRICE_R_WO_VAT').AsCurrency * 10 *
        (qryMaster.FieldByName('VAT_RATE').AsCurrency + 100))/1000;
      FieldByName('AMOUNT_R').AsCurrency :=
        round(FieldByName('PRICE_R_WO_VAT').AsCurrency * 10 *
        FieldByName('QUANTITY').AsInteger *
        (qryMaster.FieldByName('VAT_RATE').AsCurrency + 100))/1000;
    end;
    {Расчет суммы в долларах}
    FieldByName('AMOUNT_S').AsCurrency :=
          FieldByName('AMOUNT_L').AsCurrency *
          qryMaster.FieldByName('EXCH_RATE_L').AsCurrency/
          qryMaster.FieldByName('EXCH_RATE_S').Ascurrency;
  end;
end;

procedure TStockInForm.qryDetailBeforePost(DataSet: TDataSet);
begin
  qryDetail.FieldByName('ID').AsInteger := qryMaster.FieldByName('ID').AsInteger;
end;

procedure TStockInForm.dsrMasterDataChange(Sender: TObject; Field: TField);
begin
  {устанавливаем видимость колонок цены, в зависимости от режима расчета сумм}
  dbgDetail.Columns[3].Visible := qryMaster.FieldByName('CALC_MODE').AsInteger=1;
  dbgDetail.Columns[4].Visible := not dbgDetail.Columns[3].Visible;
end;

{Поиск по нажатию любой клавиши в сетке позиций}
procedure TStockInForm.dbgDetailSetEditText(Sender: TObject; ACol, ARow: Integer;
    const Value: String);
var
  s: string;
begin
  if Value = LastValue then
    exit; //защита от повторного поиска по тем же признакам
```

```
  if dbgDetail.SelectedField <> qryDetail.FieldByName('ITEM_NAME') then
    exit; //ограничение поиска нажатием клавиш в поле "Наименование"

  LastValue := Value;

  {Конструирование запроса из строки признаков, разделенных пробелами}
  s := 'SELECT O1.OBJECT_ID, O1.SHORT_NAME'#13+
       'FROM GOODS G, OBJECT_NAMES O1'#13+
       'WHERE G.ID = O1.OBJECT_ID AND O1.OBJECT_ID <> 0';
  if Value <> '' then
    s:=Format('%s AND'#13'%s',[s, StrToVarcharFilter('O1.SHORT_NAME','',Value)]);
  ResetCurrentTime; //сброс счетчика времени

  {включение светодиода в статусной строке Allegro}
  StatusBarDisplay(clLime,0,0,0,'','','');
  with qryGoods do
  begin
    Close;
    SQL.Text := s;
    Open; //открытие запроса
  end;

  {отображение в статусной строке времени, затраченного на поиск}
  StatusBarDisplay(clBlack,0,0,0,GetCurrentTimeStr,'','');
end;

{Нажатие клавиш Enter в списке позиций}
procedure TStockInForm.dbgDetailKeyPress(Sender: TObject; var Key: Char);
begin
  if (Key = 13) and not qryGoods.IsEmpty then
    dbgGoods.SetFocus; //Перемещение фокуса ввода в нижний список
end;

{Копирование позиции из нижнего списка товаров двойным щелчком мыши}
procedure TStockInForm.dbgGoodsDblClick(Sender: TObject);
begin
  qryDetail.Edit; {Присвоение значений полям позиции}
  qryDetail.FieldByName('GOODS').AsInteger :=
     qryGoods.FieldByName('OBJECT_ID').AsInteger;
  qryDetail.FieldByName('ITEM_NAME').AsString :=
     qryGoods.FieldByName('SHORT_NAME').AsString;
  LastValue := qryDetail.FieldByName('ITEM_NAME').AsString;
  dbgDetail.SelectedField := qryDetail.FieldByName('QUANTITY');
  dbgDetail.SetFocus; //Возвращение фокуса ввода в сетку позиций
end;

procedure TStockInForm.dbgGoodsKeyPress(Sender: TObject; var Key: Char);
begin
  if Key = 13 then
    dbgGoodsDblClick(nil) // Выбор и добавление товара в сетку позиций на Enter
  else if Key = 27 then
    dbgDetail.SetFocus; //Простое возвращение фокуса ввода в сетку позиций на Esc
end;

procedure TStockInForm.actReportExecute(Sender: TObject);
begin
  XLReport1.Report;
end;
```

```
{Отображение прогресса в статусной строке}
procedure TStockInForm.xlReport1Progress(Report: TxlReport; const Position, Max: Integer);
begin
  if Position = 0 then
    StatusBarDisplay(clBlack, Position, 0, Max, '', '', '')
  else
    StatusBarDisplay(clLime, Position, 0, Max, '', 'Экспорт в Excel...', '');
end;

end.
```

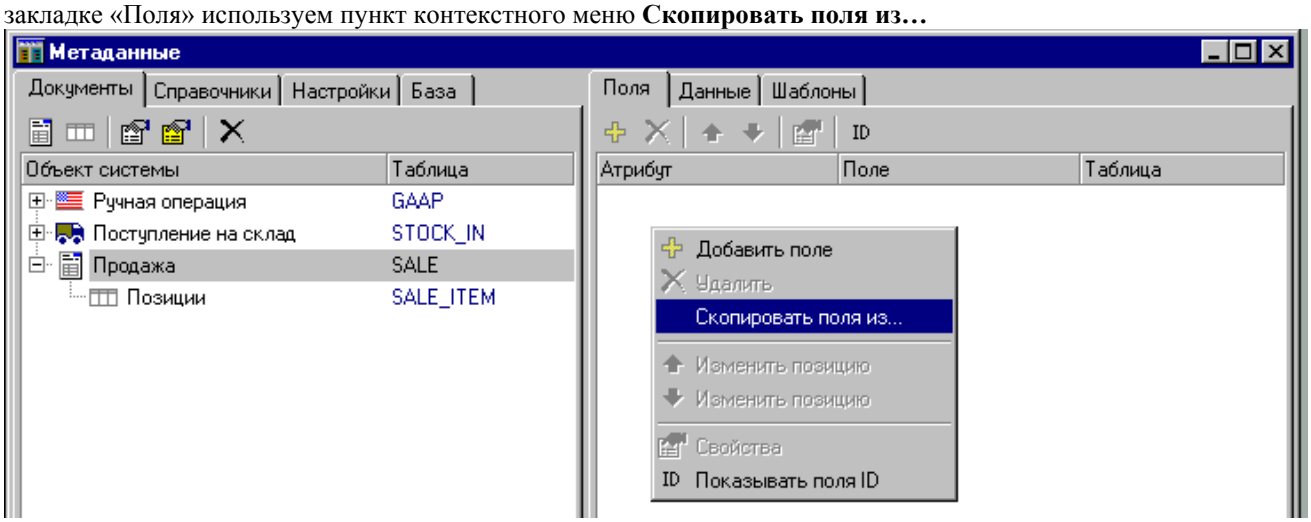
```
unit stock_in_header;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  ExtCtrls, StdCtrls,RxDBComb, AlgCtrls, RXDBCtrl, DBCtrls,RxLookup,IBQuery, DB;

type
  TStockInHeaderForm = class(TForm)
    ButtonPanel: TPanel;
    btnOK: TButton;
    btnCancel: TButton;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    Label4: TLabel;
    Label5: TLabel;
    Label6: TLabel;
    Label7: TLabel;
    Label8: TLabel;
    Label9: TLabel;
    Label10: TLabel;
    Label11: TLabel;
    cbxDOC_KIND: TRxDBComboBox;
    edDOC_DATE: TDBDateEdit;
    cbxSTOC_ACC: TRxDBLookupCombo;
    edDOC_NO: TDBEdit;
    aedACC_ID: TDBAccountEdit;
    edVAT_RATE: TDBEdit;
    refCONTRAGENT: TDBRefEdit;
    edEXCH_RATE_S: TDBEdit;
    cbxLAYER_ID: TDBLayerComboBox;
    edEXCH_RATE_L: TDBEdit;
    edENTRY_DATE: TDBDateEdit;
    cbHAS_ENTRY: TDBCheckBox;
    rgCALC_MODE: TDBRadioGroup;
    qryStocks: TIBQuery;
    dsrStocks: TDataSource;
    procedure FormShow(Sender: TObject);
    procedure btnOKClick(Sender: TObject);
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
```

```
var
  StockInHeaderForm: TStockInHeaderForm;

implementation
uses
  stock_in;

{$R *.DFM}


procedure TStockInHeaderForm.FormShow(Sender: TObject);
begin
  qryStocks.Open;
end;

procedure TStockInHeaderForm.btnOKClick(Sender: TObject);
begin
  with StockInForm.qryMaster do
  if InSet(State, MkSet(dsEdit, dsInsert)) then
  begin
    Post;
    RunContext.Documents[0].doc_id := FieldByName('ID').AsInteger;
  end;
  self.ModalResult := mrOK;
end;

procedure TStockInHeaderForm.FormClose(Sender: TObject; var Action: TCloseAction);
begin
  with StockInForm.qryMaster do
  if InSet(State, MkSet(dsEdit, dsInsert)) then
    Cancel;
end;

end.
```

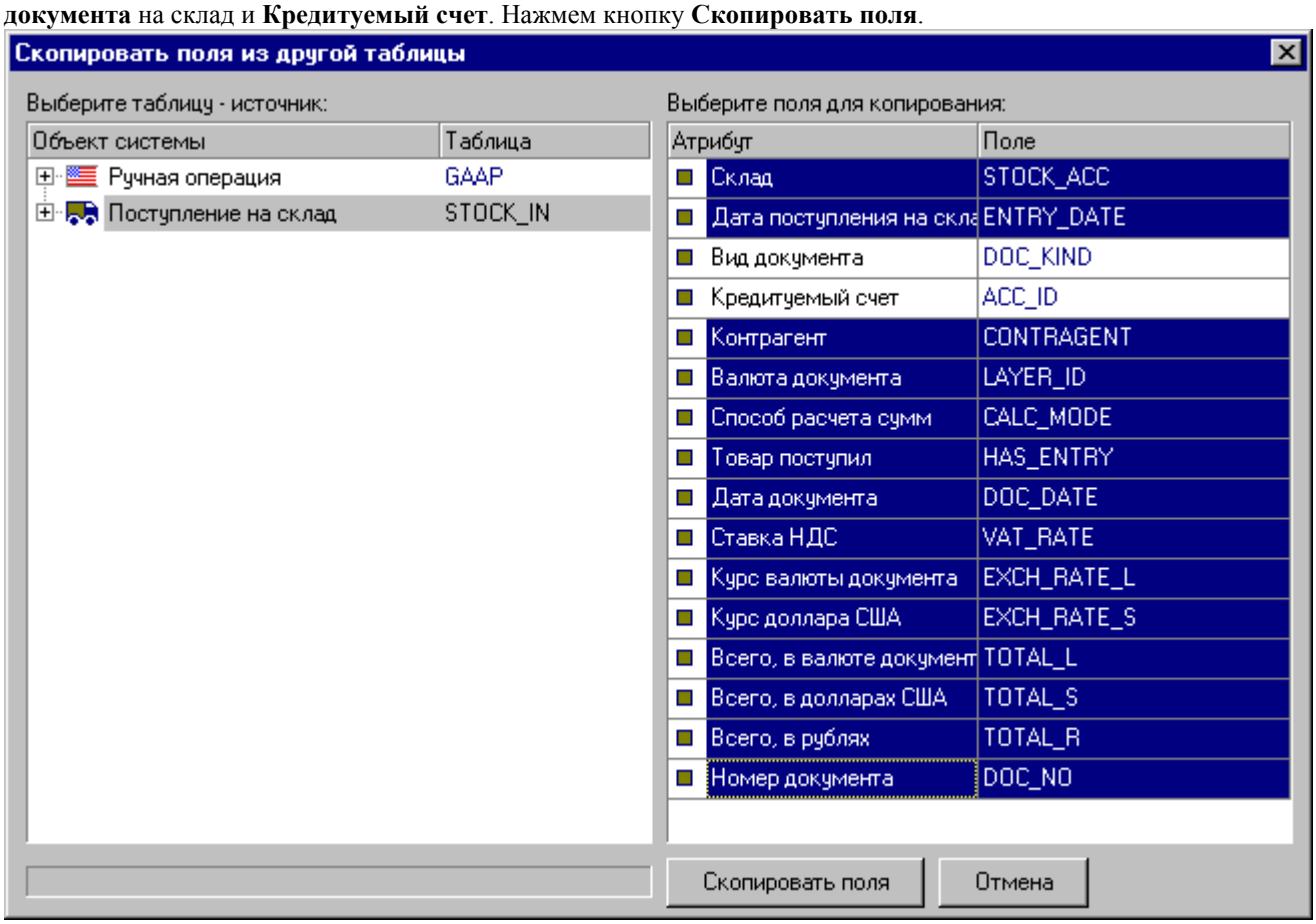
Глава 6. Создание метаданных документа «Продажа»

Создаем тип документа «Продажа»

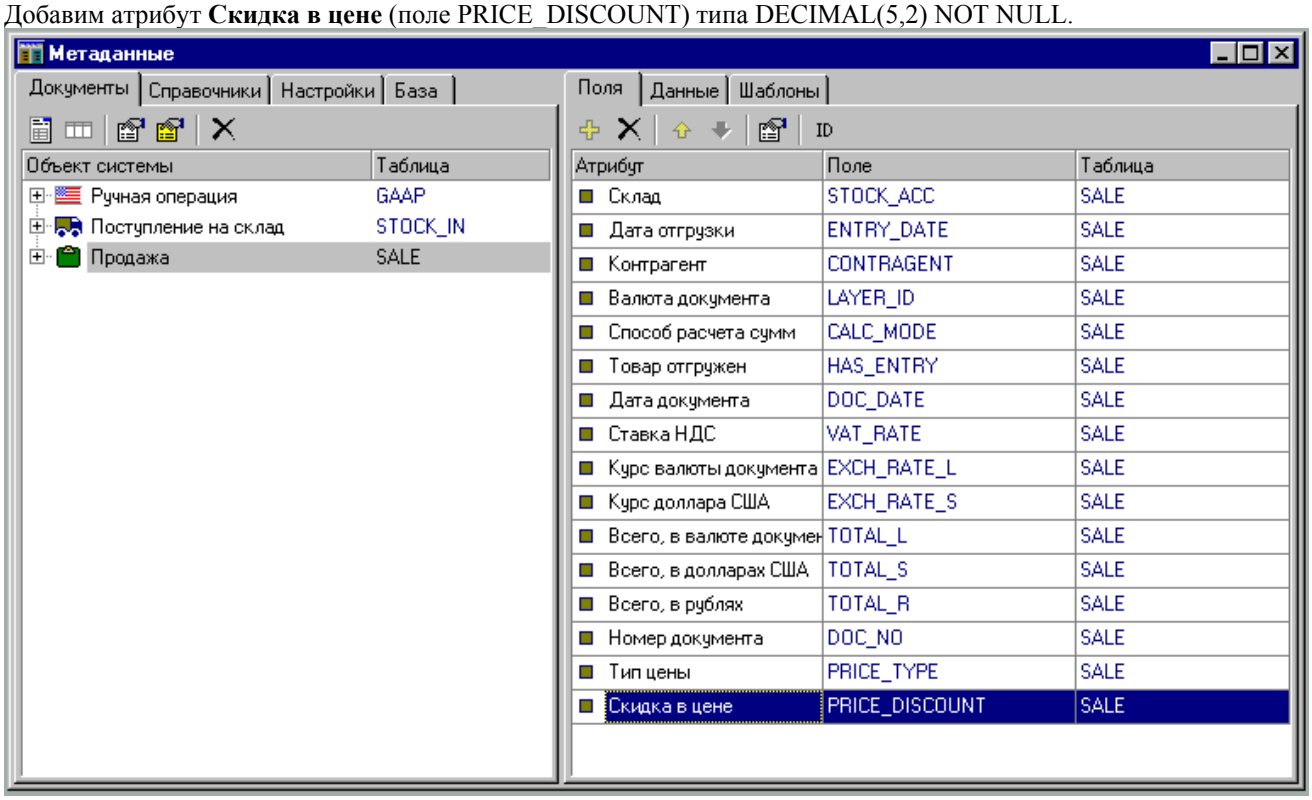
Вызовем окно «Метаданные». На закладке «Документы», используя контекстное меню, создадим новый тип документа **Продажа** (таблица SALE). Создадим подчиненную таблицу к нему, используя пункт контекстного меню **Добавить таблицу**. Назовем подчиненную таблицу **Позиции** (таблица SALE_ITEM). Большинство полей документа «Продажа» повторяет поля документа «Поступление на склад», поэтому мы просто скопируем эти поля из одного типа документа в другой. Выберем слева главную таблицу документа (SALE), а справа, на закладке «Поля» используем пункт контекстного меню **Скопировать поля из…**



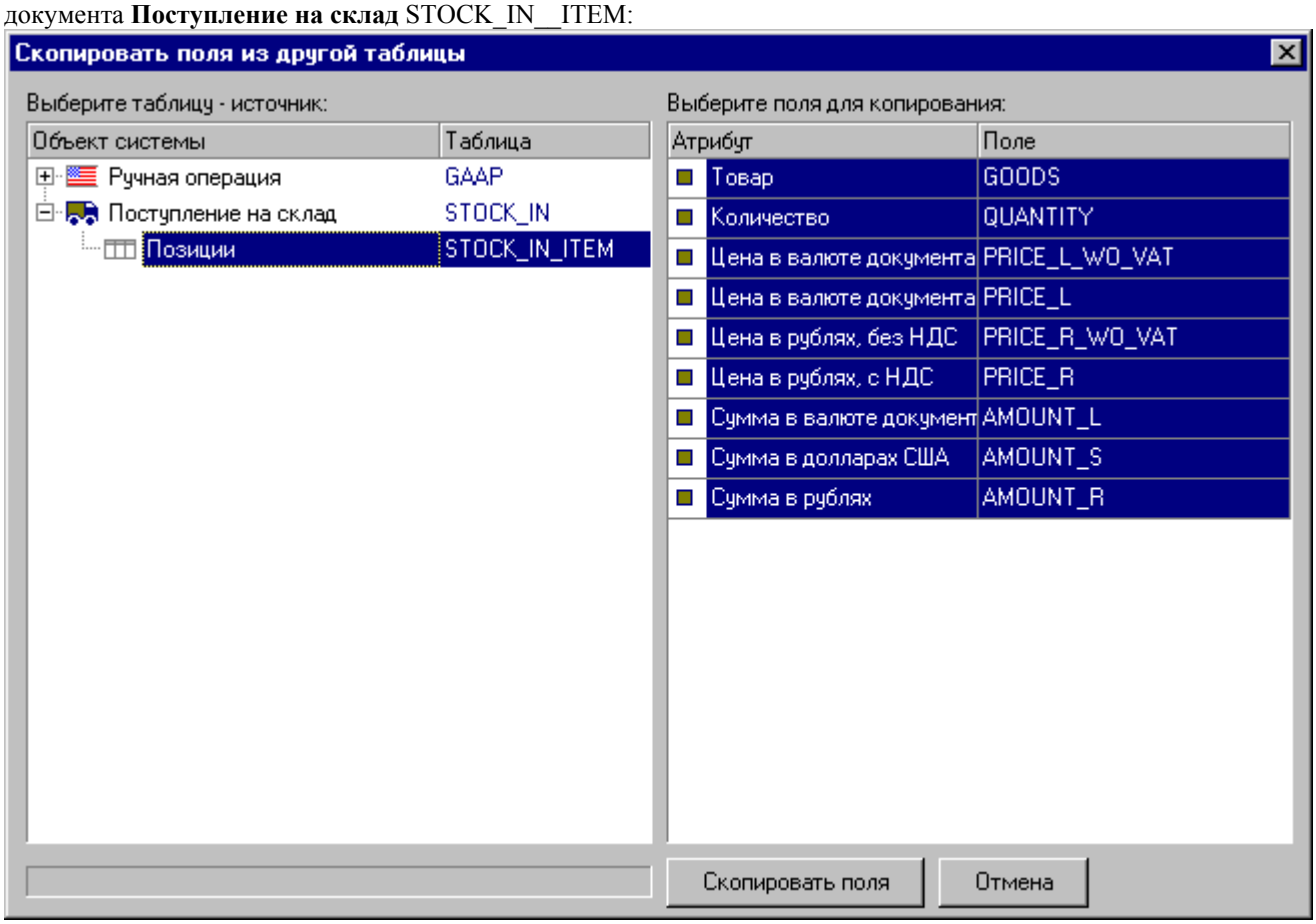
В появившемся диалоге выберем слева таблицу STOCK_IN документа **Поступление на склад**. По умолчанию все поля в правом списке выделены. Снять выделение или выделить поле можно, наведя курсор мыши на это поле и нажимая комбинацию клавиши ***Ctrl*** и левой кнопки мыши. Снимем выделение с атрибутов **Вид документа** на склад и **Кредитуемый счет**. Нажмем кнопку **Скопировать поля**.



Переименуем атрибут **Дата поступления на склад** в **Дата отгрузки** Переименуем атрибут **Товар поступил** в **Товар отгружен.** Добавим атрибут **Тип цены** (поле PRICE_TYPE) типа INTEGER NOT NULL. Добавим атрибут **Скидка в цене** (поле PRICE_DISCOUNT) типа DECIMAL(5,2) NOT NULL.



Выберем слева подчиненную таблицу SALE_ITEM и скопируем для нее все поля из подчиненной таблицы документа **Поступление на склад** STOCK_IN__ITEM:



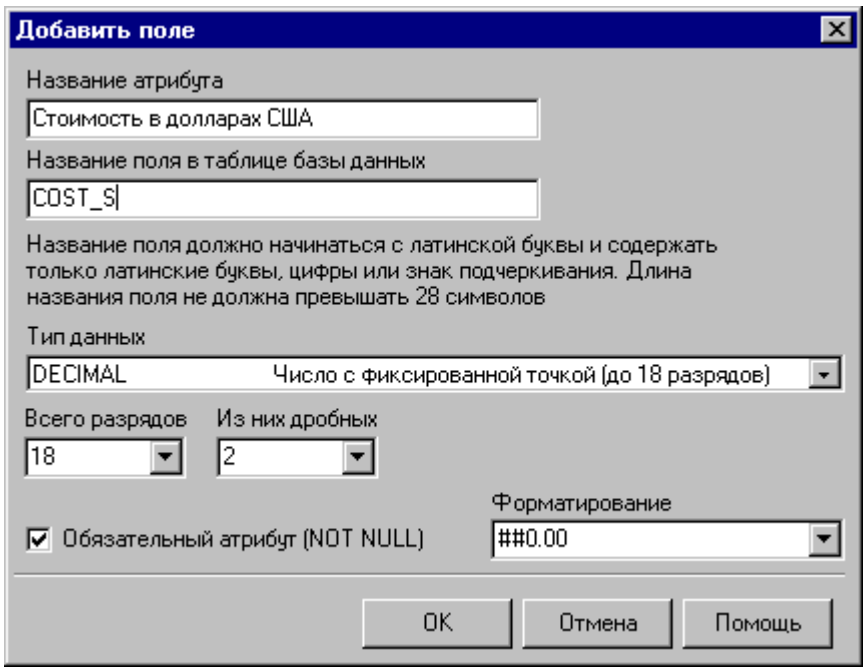
Создание полей путем копирования значительно экономит время разработчика и позволяет стандартизировать названия и типы одинаковых по смыслу полей в разных документах.

Теперь мы должны решить вопрос списания проданных товаров. Здесь возможны разные подходы. В переговорах с заказчиком мы договорились списывать товары со склада по методу **средней себестоимости**. Средняя себестоимость единицы товара рассчитывается как разница между **суммарными затратами на приобретение** товара и **уже списанными суммами**, деленная на **текущее количество**. Если вычислять среднюю стоимость единицы товара, затем ее округлять и умножать на количество, то могут возникнуть ненулевые остатки стоимости при нулевых количествах товара. Для того чтобы избежать этого неприятного явления, мы будем списывать всю суммарную среднюю стоимость того количества товара, которое отгружается. Например, если остаточная средняя стоимость всей партии из трех стиральных машин равна $1000, то при продаже трех машин мы спишем сразу ровно $1000 и не будем вычислять стоимость каждой единицы, так как она может оказаться дробной. Если же из этих трех машин отгружается две машины, то мы спишем ровно $666.66, а при отгрузке последней спишем $333.34. То есть округлять до центов мы будем не себестоимость **единицы** товара, а себестоимость всей отгружаемой **партии**. Рассчитывать себестоимость мы будем в момент отгрузки и запоминать ее в документе, отдельно для каждой позиции.

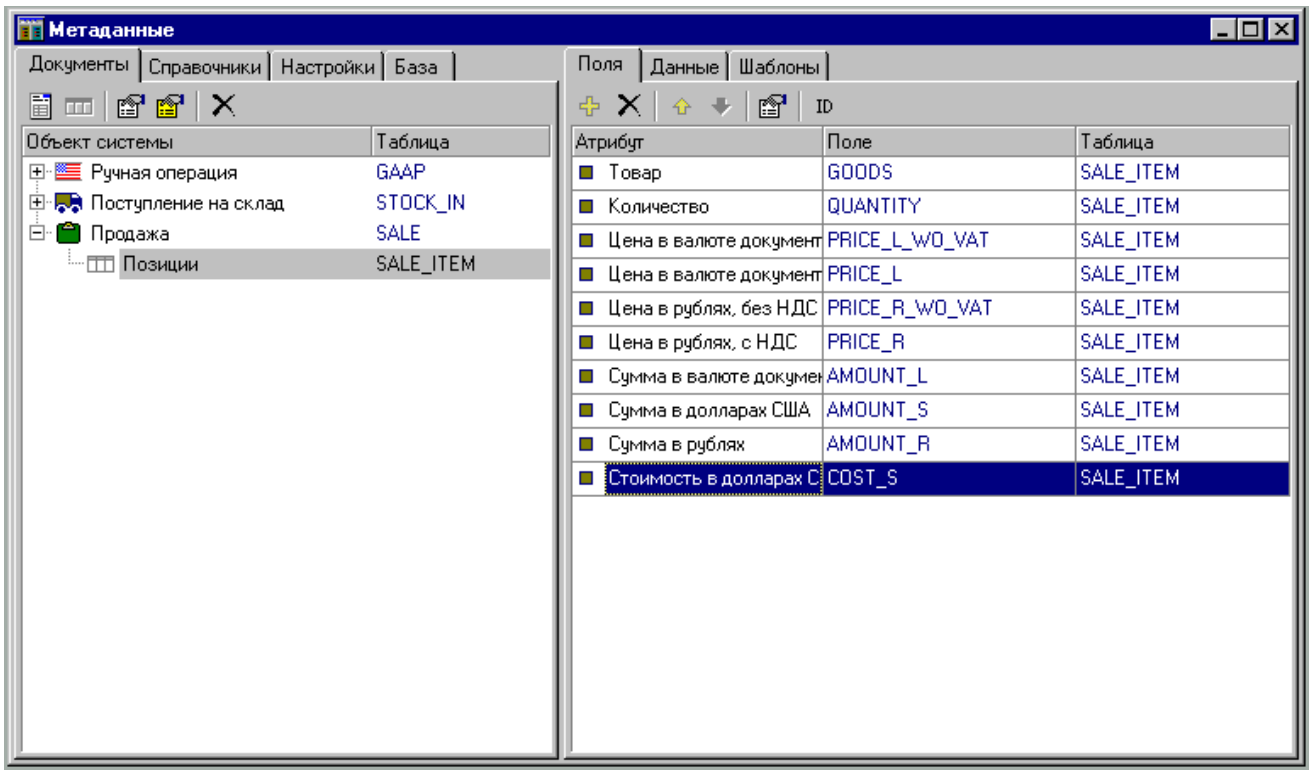
Несмотря на уверения заказчика о том, что документ «Продажа» формируется всегда при наличии всего отгружаемого товара на складе, мы предусмотрим ситуацию, когда это не так. Наш опыт подсказывает нам, что рано или поздно возникает ситуация, в которой менеджер хочет распечатать документ отгрузки лишь для того, чтобы выставить счет покупателю, а сама отгрузка происходит позже – когда все необходимые позиции действительно будут приобретены и окажутся в наличии на складе. Поэтому, как и в документе «Поступление на склад», документ «Продажа» будет иметь птичку «Отгружено». В момент сохранения документа с установленной птичкой «Отгружено» мы и будем окончательно проверять, достаточно ли товара на складе и вычислять среднюю стоимость каждой позиции. Проводить бухгалтерскую операцию мы тоже будем только при условии, что товар отгружен (птичка установлена).

Для того чтобы это реализовать, нам понадобиться дополнительное поле в подчиненной таблице.

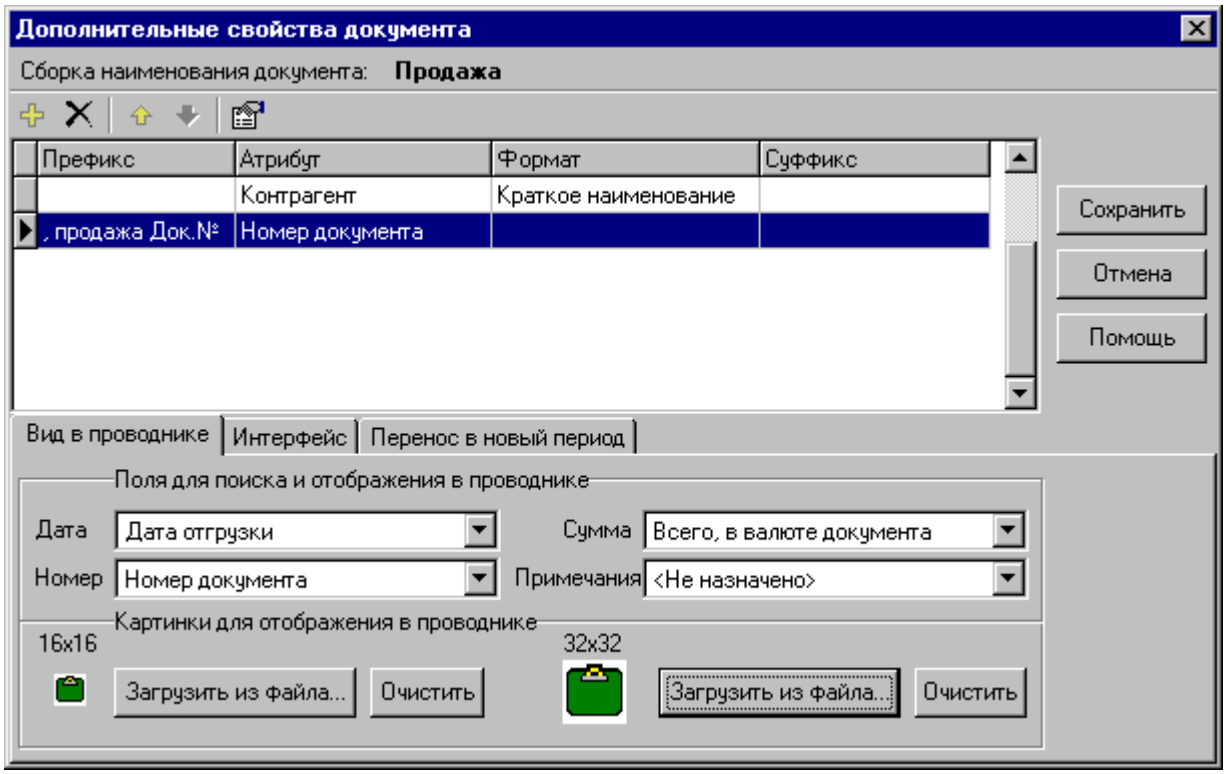
Создадим это поле. Назовем атрибут **Стоимость в долларах США**, поле **COST_S** типа DECIMAL(18,2):



Фактически разница AMOUNT_S – COST_S создает валовую прибыль от продаж в долларах США, что позволит нам в дальнейшем построить красивые отчеты о продажах, основанные на прямых SQL-запросах к таблицам документов.



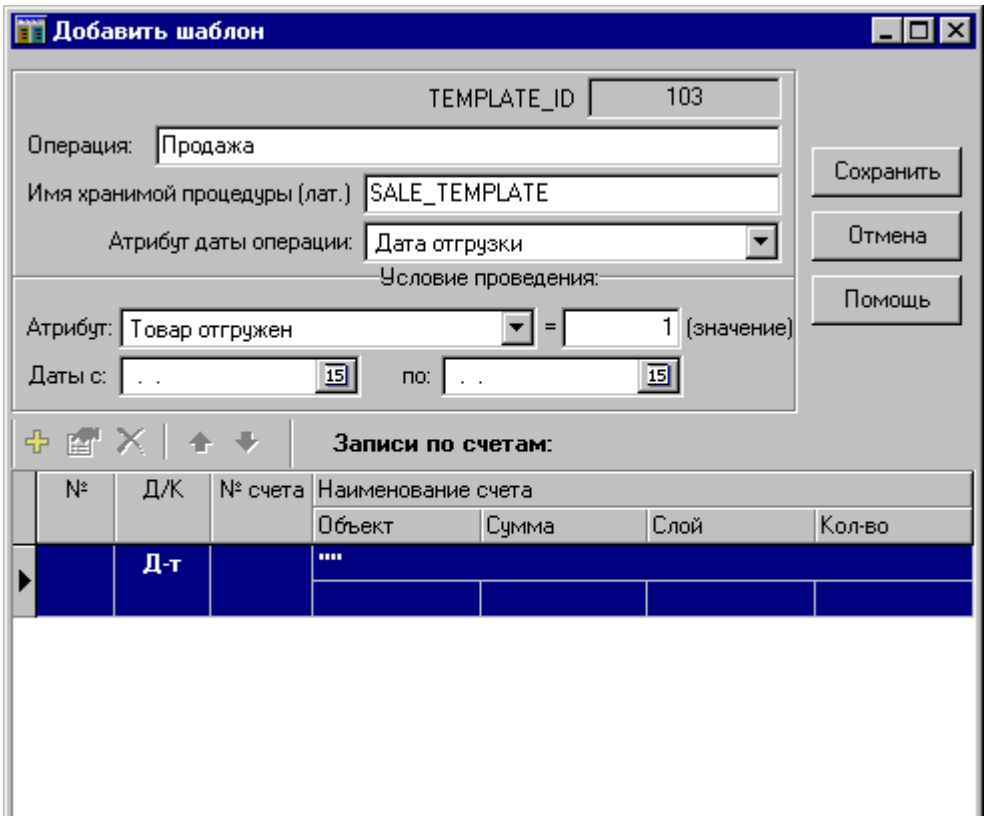
Вызовем **Дополнительные свойства документа** и создадим форматирование наименований из имени контрагента и номера документа с префиксом «**, продажа, Док№** ». Установим также поля **Даты**, **Номера** и **Суммы** для отображения в проводнике:



Настраиваем шаблон бухгалтерской операции документа «Продажа»

Создадим архивную копию базы данных через **Инструменты/Архивация базы**.

Создадим к типу документов **Продажа** шаблон операции, под названием **Продажа**. Имя хранимой процедуре назначим SALE_TEMPLATE, в качестве даты операции будет использоваться **Дата отгрузки**., а в качестве условия проведения значение атрибута **Товар отгружен**, равное 1:

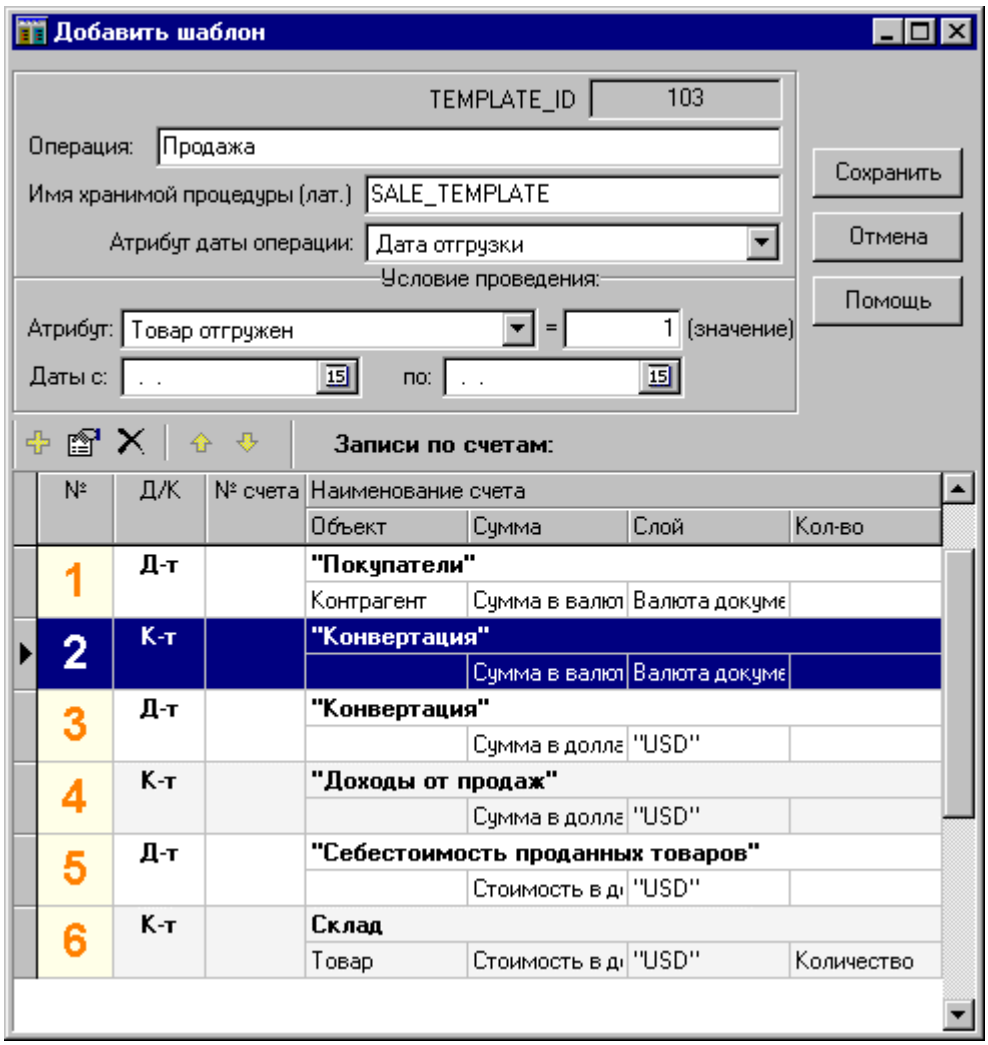


Подумаем, как нам провести продажу. Очевидно, что начислять покупателю на счет сумму следует в валюте сделки. Эта валюта может зависеть от того, как стороны договорились вести свои счета. Очевидно также, что списание себестоимости товара должно производиться в долларах США, так как стоимость товарных запасов мы решили учитывать в долларах США. Начислять списанную стоимость мы будем на счет ***Себестоимости проданных товаров*** (в каталоге ***Валовой прибыли*** правой стороны баланса). Начислять эту стоимость, видимо, также имеет смысл в долларах США. Некоторые варианты возможны с начислением суммы на счет ***Доходов от продаж***. Мы можем начислять доходы от продаж в валюте сделки. Тогда можно не использовать счет ***Конвертации***. Если же мы хотим начислять доходы в долларах США или российских рублях, то нам придется использовать этот счет. Есть некоторое преимущество в том, чтобы все же использовать счет ***Конвертации***. При изменении курсов валют изменение показателя на счете ***Доходов от продаж*** не так заметно в консолидированном балансе, как ненулевой остаток на счете ***Конвертации***. Так как программа Allegro позволяет легко создавать новые счета и изменять шаблоны проводок, мы можем отложить окончательное решение этого вопроса до очередной встречи с нашим заказчиком. Пока же будем исходить из того, что раз уж товарные запасы измеряются в долларах США, то и валовую прибыль логично вычислять в той же валюте. Так как расходный счет валовой прибыли (себестоимость проданных товаров) долларовый, то пускай будет долларовым и доходный счет валовой прибыли. А это означает, что мы будем использовать конвертацию и наша бухгалтерская операция будет содержать 6 записей по счетам:

Запись в	**Счет**	**Объект**	**Сумма (поле)**	**Слой**	**Количество**
Дебет	«Покупатели»	Поле: Контрагент	Сумма в валюте документа	Поле: Валюта документа	
Кредит	«Конвертация»		Сумма в валюте документа	Поле: Валюта документа	
Дебет	«Конвертация»		Сумма в долларах США	USD	

Кредит	«Доходы от продаж»		Сумма в долларах США	USD	
Дебет	«Себестоимость проданных товаров»		Стоимость в долларах США	USD	
Кредит	Поле: Склад	Поле: Товар	Стоимость в долларах США	USD	Поле: Количество

Введем записи по счетам на основании этой таблицы. И сохраним шаблон операции:



Вызовем через меню **Бухгалтерия/Состояние расчетов** окно «Состояние расчетов» и нажмем кнопку **Перепровести**.

Метаданные документа «Продажа» готовы.

Глава 7. Генерация документов

Генерация 100 документов поступлений и 300 документов продаж

Для отладки конфигурации нам понадобятся данные, которых пока нет. Вводить вручную сотни документов мы не будем, вместо этого мы создадим специальный проект для генерации документов. Цены и количества мы будем создавать с помощью генератора случайных чисел, а товары и фирмы выбирать случайным образом из справочников.

Вызовем проводник по документам и создадим в нем две папки. Одну назовем «Поступления», а другую – «Продажи». Вызовем окно интерактивного SQL и сделаем запрос:

```
select * from doc_dir
```

Запомним или запишем на бумаге ID имеющихся у нас папок. Допустим, эти ID равны 1001 и 1002.

Теперь займемся созданием генератора документов.

Включим режим «Дизайнер» и создадим новое приложение. Сохраним модуль формы в файле **monte_carlo.pas**, а проект в файле **monte_carlo_project.ipr**. Введем заголовок формы:

```
Caption = Розыгрыш поступлений и продаж
```

Для того чтобы форма была дочерним окном главной формы, установим ее свойство:

```
FormStyle = fsMDIChild
```

Для того чтобы форма имела фиксированный размер, установим свойство:

```
BorderStyle = bsDialog
```

Добавим на форму с палитры **InterBase** компоненты, назначая им имена (свойство **Name**):

Компонент	**Name**
IBTransaction	traCurrent
IBQuery	qryGoods
IBQuery	qryContragent
IBQuery	qry
IBDataSet	qryStock_In
IBDataSet	qryStock_In_Item
IBDataSet	qrySale
IBDataSet	qrySale_Item

Установим у компонента **traCurrent** свойство:

```
DefaultDatabase = MainConnection.MainDatabase
```

Дважды щелкнем на компоненте **traCurrent** и установим изоляцию транзакций **ReadCommitted**.
Выделим все остальные компоненты и назначим им свойство:

```
Transaction = traCurrent
```

Дважды щелкая на свойстве SQL компонентов в редакторе SQL-команды запишем:

Компонент	**Текст запроса**
qryGoods	select ID from GOODS where ID <> 0
qryContragent	select ID from CONTRAGENT where ID <> 0

Теперь создадим у формы обработчик **OnCreate:**

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  traCurrent.StartTransaction;
  qryContragent.Open;
  qryContragent.FetchAll;
  qryGoods.Open;
  qryGoods.FetchAll;
end;
```

Добавим компонента **ProgressBar** с палитры **Win32.**
Добавим 2 компонента **Button** с палитры **Standard** и присвоим им имена и заголовки:

Компонент	**Name**	**Caption**
Button1	btnDelete	Удалить документы
Button2	btnGenerate	Создать документы

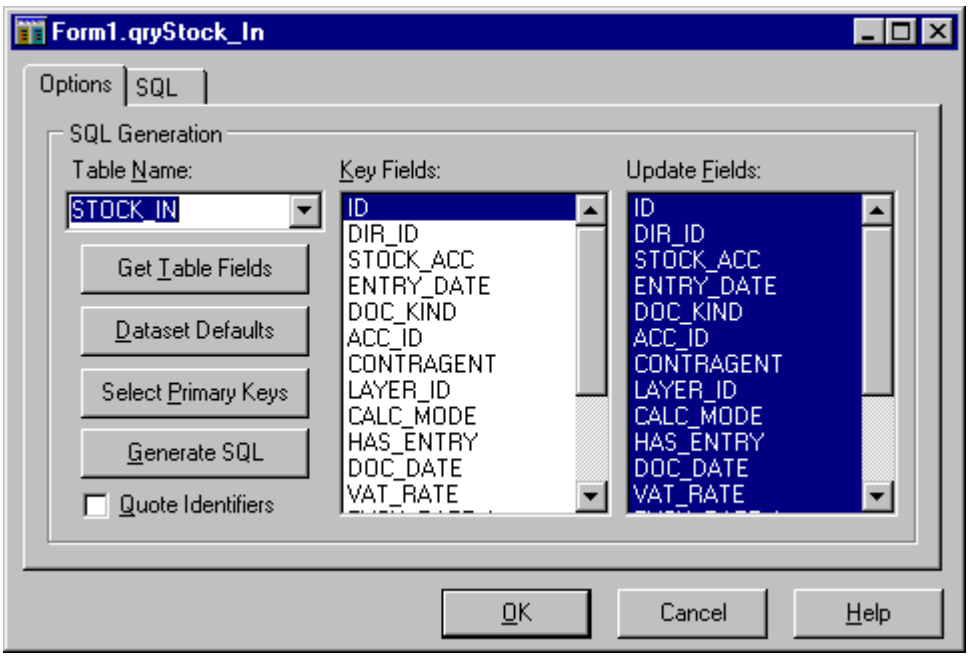
Расположим все компоненты так, как показано на рисунке:

Мы должны создать SQL-запросы в компонентах, которые будут работать с документами. Запишем в свойствах SelectSQL следующие запросы:

Компонент	**Текст запроса**
qryStock_In	select * from STOCK_IN
QrySale	select * from SALE order by ENTRY_DATE
QryStock_In_Item	select * from STOCK_IN_ITEM where ID = -1
QrySale_Item	select * from SALE_ITEM where ID = -1

В запросе документов продаж мы упорядочили набор по датам отгрузки. А в запросы позиций мы ввели условие ID = -1 для того, чтобы получать пустые наборы при SELECT.

Вызывая через контекстное меню компонент DataSet Editor, создадим для всех этих компонентов тексты запросов InsertSQL, ModifySQL, DeleteSQL, RefreshSQL. Для этого в редакторе последовательно нажимаем кнопки: **Get Table Fields**, **Select Primary Keys**, **Generate SQL**, **OK**.



Растянем главное Allegro окно с помощью кнопки **Растянуть Главное Окно**, вызовем окно ISQL с помощью меню **Инструменты/Интерактивный SQL** и создадим в базе данных хранимую процедуру, которая будет обновлять поля сумм TOTAL_L, TOTAL_S, TOTAL_R каждого документа «Поступление на склад» в таблице STOCK_IN на основе сумм его позиций AMOUNT_L, AMOUNT_S, AMOUNT_R в таблице STOCK_IN_ITEM:

```
create procedure update_stock_in_totals
as
declare variable id integer;
declare variable t1 decimal(18,2);
declare variable t2 decimal(18,2);
declare variable t3 decimal(18,2);
begin
  for select id, sum(amount_l),sum(amount_s),sum(amount_r)
  from stock_in_item
  group by id
  into id, :t1, :t2, :t3
  do update stock_in
  set total_l = :t1,
     total_s = :t2,
     total_r = :t3
  where id = :id;
  update template set is_changed = 1 where template_id = 102;
end
```

Создадим еще такую же хранимую процедуру для документов типа «Продажа»:

```
create procedure update_sale_totals
as
declare variable id integer;
declare variable t1 decimal(18,2);
declare variable t2 decimal(18,2);
declare variable t3 decimal(18,2);
begin
  for select id, sum(amount_l),sum(amount_s),sum(amount_r)
  from sale_item
  group by id
  into id, :t1, :t2, :t3
```

```
  do update sale
  set total_l = :t1,
     total_s = :t2,
     total_r = :t3
  where id = :id;
  update template set is_changed = 1 where template_id = 103;
end
```

Мы будем использовать следующий алгоритм генерации документов:

1. Добавляем 100 записей в компонент **qryStock_In**. Каждая запись соответствует одному документу поступления. В процессе добавления документы распределяем равномерно по датам на 300 дней, с интервалом 3 дня, начиная с 1 января 2003 г. Затем добавляем 350 записей в компонент **qrySale**. Каждая запись соответствует одному документу продаж. В процессе добавления распределяем их по одному документу на каждую дату, начиная с 1 января 2003г. В каждом документе выбираем случайного контрагента, присваивая случайное целое число свойству **RecNo** (номер записи) компонента **qryContragent** и копируя значение поля **ID** из этого компонента в документ. В результате мы получим 450 документов (пока без позиций) в таблицах базы.
2. В цикле создаем 1000 позиций «Поступлений на склад», используя компонент **qryStock_In_Item**. Документ для добавления позиции выбираем каждый раз, присваивая случайное целое число свойству **RecNo** компонента **qryStock_In**. После посылки компонентом **qryStock_In_Item** каждой новой записи на сервер, переоткрываем его запрос **SelectSQL**. Благодаря тому, что в этом запросе имеется условие ID = -1, после переоткрытия запрос возвращает пустой набор. Это будет защищать нас от замедления процесса генерации позиций, вызванного накоплением позиций в компоненте **qryStock_In_Item**. После создания позиции в документе поступления, распределяем эту партию случайным образом по нескольким документам продаж, датированным позднее данного поступления. Это делаем во внутреннем цикле путем вставки записей в компонент **qrySale_Item**. После посылки компонентом **qrySale_Item** каждой новой записи на сервер, его запрос **SelectSQL** переоткрываем.
3. Вызываем хранимые процедуры **update_stock_in_totals** и **update_sale_totals** для приведения в соответствие сумм в шапках документов суммам в их позициях.
4. Подтверждаем транзакцию, перепроводим все документы вызывая процедуру **UpadateTurmovers**.
5. Вызываем хранимую процедуру (она описана далее) для расчета себестоимости проданных товаров и обновления этой себестоимости в позициях документов продаж.
6. Подтверждаем транзакцию, перепроводим все документы вызывая процедуру **UpadateTurmovers**.

Создадим в базе данных с помощью окна ISQL хранимую процедуру, которая занимается вычислением средней стоимости проданных позиций на основе проводок в таблице ACC_TURN и записывает полученные значения в позиции документов продаж в поле **cost_s**. Эта процедура обходит таблицу проводок в определенном порядке: **id** товара, дата проводки, **id** шаблона операции. Встречая при сканировании операцию «продажа» (**template_id = 103**), рассчитанная средняя стоимость записывается в позицию соответствующего документа **doc_id**. Обнуление остаточной стоимости **amount** и количества **quantity** происходит в момент прохождения «границы» между разными товарами. Далее записи в дебет увеличивают эти показатели, а записи в кредит – уменьшают их. Средневзвешенная цена вычисляется как отношение остаточной стоимости к количеству **amount/quantity** на момент отгрузки.

Текст процедуры:

```
create procedure sale_item_update_cost(acc_id integer)
as
  /*переменные для хранения полей*/
  declare variable op_date date;
  declare variable layer_id integer;
  declare variable object_id integer;
  declare variable template_id integer;
  declare variable doc_id integer;
  declare variable debit decimal(18,2);
  declare variable credit decimal(18,2);
  declare variable quantity_debit decimal(18,3);
  declare variable quantity_credit decimal(18,3);
  /*переменные для хранения текущей суммы и количества в партии товара*/
  declare variable amount decimal(18,2);
```

```
  declare variable quantity decimal(18,3);
  declare variable avg_cost double precision;
  /*переменная для хранения текущего id товара*/
  declare variable goods integer;
begin
  goods = -1;

  for select
    op_date, layer_id, object_id, template_id, doc_id,
    debit, credit, quantity_debit, quantity_credit
  from
    acc_turn
  where
    acc_id = :acc_id and layer_id = 2
  order by
    object_id, op_date, template_id
  into
    :op_date, :layer_id, :object_id, :template_id, :doc_id,
    :debit, :credit, :quantity_debit, :quantity_credit
  do
  begin
    if (goods <> object_id) then /*начинаем очередной товар*/
    begin
      amount = 0;
      quantity = 0;
      goods = object_id;
    end
    if (template_id = 103) then /*103 - Продажа*/
    begin
      if (quantity <> 0) then
        avg_cost = amount / quantity;
      else
        avg_cost = 0;
      credit = quantity_credit * avg_cost;
      /*изменяем стоимость в позициях документа "Продажи"*/
      update sale_item set cost_s = quantity * :avg_cost
      where id = :doc_id and goods = :object_id;
    end
    amount = amount + debit - credit;
    quantity = quantity + quantity_debit - quantity_credit;
  end
  update template set is_changed = 1 where template_id = 103;
end;
```

Осталось создать обработчики для кнопок «Удалить документы» и «Создать документы». Удаление документов может нам понадобиться, если нас не удовлетворят результаты генерации, или если наш проект будет работать неполностью или неправильно, из-за каких-то программных ошибок. При нажатии кнопки «Удалить документы» программа должна удалить все документы и все проводки, связанные с ними.

Создадим обработчики **OnClick** для кнопок «Удалить документы» и «Создать документы». Их обработчики содержат достаточно много программного текста. Поэтому приводим полный листинг модуля:

```
unit monte_carlo;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  IBQuery, IBDatabase, ComCtrls, StdCtrls,IBCustomDataSet;
```

```
type
  TForm1 = class(TForm)
    qryContragent: TIBQuery;
    traCurrent: TIBTransaction;
    qryGoods: TIBQuery;
    ProgressBar1: TProgressBar;
    qryStock_In: TIBDataSet;
    qrySale: TIBDataSet;
    btnGenerate: TButton;
    btnDelete: TButton;
    qry: TIBQuery;
    qryStock_In_Item: TIBDataSet;
    qrySale_Item: TIBDataSet;
    ProgressBar2: TProgressBar;
    procedure FormCreate(Sender: TObject);
    procedure btnDeleteClick(Sender: TObject);
    procedure btnGenerateClick(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.DFM}


procedure TForm1.FormCreate(Sender: TObject);
begin
  traCurrent.StartTransaction;
  qryContragent.Open;
  qryContragent.FetchAll;
  qryGoods.Open;
  qryGoods.FetchAll;
end;

procedure TForm1.btnDeleteClick(Sender: TObject);
begin
  Screen.Cursor := crHourGlass;
  try
    with qry do
    begin
      SQL.Text := 'DELETE FROM STOCK_IN';
      ExecSQL;
      SQL.Text := 'DELETE FROM SALE';
      ExecSQL;
      SQL.Text := 'update template set is_changed = 1'#13+
                  'where template_id in (102, 103)';
      ExecSQL;
    end;
    qryStock_in.Close;
    qrySale.Close;

    traCurrent.CommitRetaining;    //подтверждение транзакции
    UpdateTurnovers(True);   //перепроведение документов
```

```
    btnGenerate.Enabled := True;
  finally
    Screen.Cursor := crDefault;
  end;
  ShowMessage('Удаление документов завершено');
end;

procedure TForm1.btnGenerateClick(Sender: TObject);
const
  stock_in_count = 100;
  stock_in_item_count = 1000;
  sale_count = 350;
var
  i, contragent,
  goods, quantity, price_USD,
  sales_from, sale_quantity: integer;
begin
  btnGenerate.Enabled := False;
  qryStock_in.Open;
  qrySale.Open;
  ProgressBar1.Max := stock_in_count + sale_count;
  {Генерация поступлений на склад (только шапки)}
  for i := 1 to stock_in_count do
  begin
    {выбираем случайного контрагента}
    with qryContragent do
    begin
      RecNo := random(RecordCount) + 1;
      contragent := FieldByName('ID').AsInteger;
    end;
    {создаем "Поступление на склад"}
    with qryStock_In do
    begin
      Insert; //вставка записи в набор
      FieldByName('DIR_ID').AsInteger := 1001; //Папка "Поступления"
      FieldByName('STOCK_ACC').AsInteger := 1007; //Счет "Главный склад"
      {Поступления происходят раз в 3 дня, начиная с января 2003 г}
      FieldByName('ENTRY_DATE').AsDateTime := EncodeDate(2003,1,1) + 3*i;
      FieldByName('DOC_KIND').AsInteger := 0; //Поступление от поставщика
      FieldByName('ACC_ID').AsInteger := 1012; //Счет "Поставщики"
      FieldByName('CONTRAGENT').AsInteger := contragent;
      {1 + остаток от деления ID контрагента на 3 в качестве валюты документа}
      FieldByName('LAYER_ID').AsInteger := 1 + contragent mod 3;
      FieldByName('CALC_MODE').AsInteger := 1; //Расчет от цены с НДС
      FieldByName('HAS_ENTRY').AsInteger := 1; //Товар поступил на склад
      FieldByName('DOC_DATE').AsDateTime := FieldByName('ENTRY_DATE').AsDateTime;
      FieldByName('VAT_RATE').AsInteger := 20; //Ставка НДС
      FieldByName('EXCH_RATE_S').AsCurrency := 30+random(100)/100; //курс доллара
      {курс валюты документа зависит от валюты документа}
      case FieldByName('LAYER_ID').AsInteger of
      1: FieldByName('EXCH_RATE_L').AsCurrency := 1; //курс рубля
      2: FieldByName('EXCH_RATE_L').AsCurrency :=
         FieldByName('EXCH_RATE_S').AsCurrency; //курс доллара
      3: FieldByName('EXCH_RATE_L').AsCurrency := 32+random(100)/100; //курс ЕВРО
      end;
      {Всем суммам "всего" присваиваем 0}
      FieldByName('TOTAL_L').AsCurrency := 0;
      FieldByName('TOTAL_S').AsCurrency := 0;
      FieldByName('TOTAL_R').AsCurrency := 0;
      {Документы просто нумеруем подряд}
```

```
    FieldByName('DOC_NO').AsString := IntToStr(i);
    Post; //посылка на сервер
   end;
   ProgressBar1.Position := i; //отображаем прогресс
  end;

  {Генерация продаж (только шапки)}
  for i := 1 to sale_count do
  begin
   {выбираем случайного контрагента}
   with qryContragent do
   begin
    RecNo := random(RecordCount) + 1;
    contragent := FieldByName('ID').AsInteger;
   end;
   {создаем "Продажу"}
   with qrySale do
   begin
    Insert; //вставка записи в набор
    FieldByName('DIR_ID').AsInteger := 1002; //Папка "Продажи"
    FieldByName('STOCK_ACC').AsInteger := 1007; //Счет "Главный склад"
    {Продажи происходят каждый день, начиная с 1 января 2003 г}
    FieldByName('ENTRY_DATE').AsDateTime := EncodeDate(2003,1,1) + i;
    FieldByName('CONTRAGENT').AsInteger := contragent;
    {1 + остаток от деления ID контрагента на 3 в качестве валюты документа}
    FieldByName('LAYER_ID').AsInteger := 1 + contragent mod 3;
    FieldByName('CALC_MODE').AsInteger := 1; //Расчет от цены с НДС
    FieldByName('HAS_ENTRY').AsInteger := 1; //Товар отгружен со склада
    FieldByName('DOC_DATE').AsDateTime := FieldByName('ENTRY_DATE').AsDateTime;
    FieldByName('VAT_RATE').AsInteger := 20; //Ставка НДС
    FieldByName('EXCH_RATE_S').AsCurrency := 30+random(100)/100; //курс доллара
    {курс валюты документа зависит от валюты документа}
    case FieldByName('LAYER_ID').AsInteger of
    1: FieldByName('EXCH_RATE_L').AsCurrency := 1; //курс рубля
    2: FieldByName('EXCH_RATE_L').AsCurrency :=
       FieldByName('EXCH_RATE_S').AsCurrency; //курс доллара
    3: FieldByName('EXCH_RATE_L').AsCurrency := 32+random(100)/100; //курс ЕВРО
    end;
    {Всем суммам "всего" присваиваем 0}
    FieldByName('TOTAL_L').AsCurrency := 0;
    FieldByName('TOTAL_S').AsCurrency := 0;
    FieldByName('TOTAL_R').AsCurrency := 0;
    {Документы просто нумеруем подряд}
    FieldByName('DOC_NO').AsString := IntToStr(i);
    FieldByName('PRICE_TYPE').AsInteger := 0;
    FieldByName('PRICE_DISCOUNT').AsCurrency := 0;
    Post; //посылка на сервер
   end;
   ProgressBar1.Position := stock_in_count + i; //отображаем прогресс
  end;
//  ShowMessage('Генерация шапок документов завершена');

  {Генерация позиций в обоих типах документов}
  ProgressBar2.Max := stock_in_item_count;
  for i := 1 to stock_in_item_count do
  begin
   {выбираем случайный товар}
   with qryGoods do
   begin
    RecNo := random(RecordCount) + 1;
```

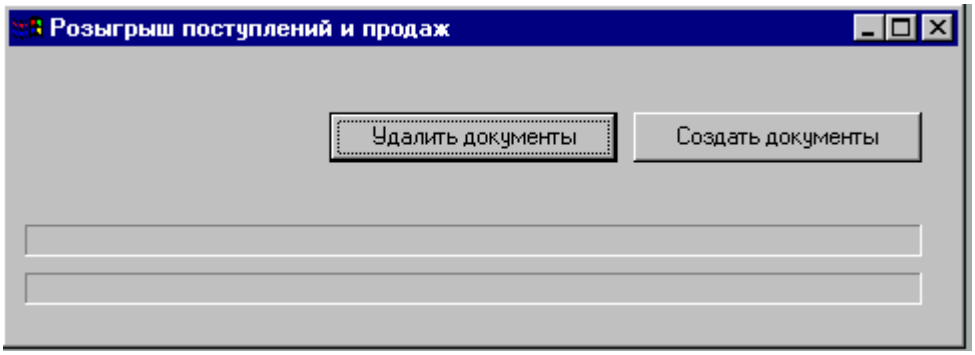
```
  goods := FieldByName('ID').AsInteger;
end;
{выбираем случайное поступление на склад}
with qryStock_In do
  RecNo := random(RecordCount) + 1;
{разыгрываем величины}
quantity := random(5) + 1; //поступившее количество (партия)
price_USD := 100 + random(200); //закупочная цена USD
{создаем строку в поступлении на склад}
with qryStock_In_Item do
begin
  Open;
  Insert;
  FieldByName('ID').AsInteger := qryStock_In.FieldByName('ID').AsInteger;
  FieldByName('GOODS').AsInteger := goods;
  FieldByName('QUANTITY').AsInteger := quantity;
  {пересчитываем цену в валюту документа
  по кросс-курсу и округляем до центов}
  FieldByName('PRICE_L').AsCurrency := round(price_USD *
    qryStock_In.FieldByName('EXCH_RATE_S').AsCurrency /
    qryStock_In.FieldByName('EXCH_RATE_L').AsCurrency * 100)/100;
  {Вычисляем цену без НДС в валюте документа}
  FieldByName('PRICE_L_WO_VAT').AsCurrency :=
    FieldByName('PRICE_L').AsCurrency/
    (1 + qryStock_In.FieldByName('VAT_RATE').AsCurrency/100);
  {пересчитываем цену в рубли и округляем до копеек}
  FieldByName('PRICE_R').AsCurrency := round(price_USD *
    qryStock_In.FieldByName('EXCH_RATE_S').AsCurrency * 100)/100;
  {Вычисляем цену без НДС в рублях}
  FieldByName('PRICE_R_WO_VAT').AsCurrency :=
    FieldByName('PRICE_R').AsCurrency/
    (1 + qryStock_In.FieldByName('VAT_RATE').AsCurrency/100);
  {Вычисляем суммы}
  FieldByName('AMOUNT_L').AsCurrency :=
    FieldByName('PRICE_L').AsCurrency * quantity;
  FieldByName('AMOUNT_S').AsCurrency :=  price_USD * quantity;
  FieldByName('AMOUNT_R').AsCurrency :=
    FieldByName('PRICE_R').AsCurrency * quantity;

  {находим документ продажи, совпадающий по дате с этим поступлением}
  qrySale.Locate('ENTRY_DATE',
      qryStock_In.FieldByName('ENTRY_DATE').AsDateTime,
      MkSet());
  sales_from := qrySale.RecNo; //запоминаем № строки набора
  Post;
  Close;
end;

{делаем наценку 10%}
price_USD := price_USD * 1.1;
{повторяем идущий далее цикл, пока не распродадим всю эту партию}
repeat
  sale_quantity := random(quantity) + 1; //продаваемое количество
  {выбираем случайную продажу}
  with qrySale do
    RecNo := sales_from + random(RecordCount - sales_from) + 1;
  {создаем позицию в продажах}
  with qrySale_Item do
  begin
    Open;
```

```
      Insert;
      FieldByName('ID').AsInteger := qrySale.FieldByName('ID').AsInteger;
      FieldByName('GOODS').AsInteger := goods;
      FieldByName('QUANTITY').AsInteger := sale_quantity;
      {пересчитываем цену в валюту документа
      по кросс-курсу и округляем до центов}
      FieldByName('PRICE_L').AsCurrency := round(price_USD *
        qrySale.FieldByName('EXCH_RATE_S').AsCurrency /
        qrySale.FieldByName('EXCH_RATE_L').AsCurrency * 100)/100;
      {Вычисляем цену без НДС в валюте документа}
      FieldByName('PRICE_L_WO_VAT').AsCurrency :=
        FieldByName('PRICE_L').AsCurrency/
        (1 + qrySale.FieldByName('VAT_RATE').AsCurrency/100);
      {пересчитываем цену в рубли и округляем до копеек}
      FieldByName('PRICE_R').AsCurrency := round(price_USD *
        qrySale.FieldByName('EXCH_RATE_S').AsCurrency * 100)/100;
      {Вычисляем цену без НДС в рублях}
      FieldByName('PRICE_R_WO_VAT').AsCurrency :=
        FieldByName('PRICE_R').AsCurrency/
        (1 + qrySale.FieldByName('VAT_RATE').AsCurrency/100);
      {Вычисляем суммы}
      FieldByName('AMOUNT_L').AsCurrency :=
        FieldByName('PRICE_L').AsCurrency * sale_quantity;
      FieldByName('AMOUNT_S').AsCurrency :=  price_USD * sale_quantity;
      FieldByName('AMOUNT_R').AsCurrency :=
        FieldByName('PRICE_R').AsCurrency * sale_quantity;
      {списываем пока стоимость 0}
      FieldByName('COST_S').AsCurrency := 0;
      Post;
      Close;
    end;
    quantity := quantity - sale_quantity; //уменьшаем количество в партии
  until quantity <= 0;
  ProgressBar2.Position := i; //отображаем прогресс
end;

{Расчет сумм в шапках документов}
with qry do
begin
  SQL.Text := 'EXECUTE PROCEDURE update_stock_in_totals';
  ExecSQL;
  SQL.Text := 'EXECUTE PROCEDURE update_sale_totals';
  ExecSQL;
  {Удаляем документы продаж, которые не получили позиций вообще}
  SQL.Text := 'DELETE FROM SALE WHERE TOTAL_S = 0';
  ExecSQL;
end;

traCurrent.CommitRetaining;    //подтверждение транзакции
UpdateTurnovers(True);   //перепроведение документов
//  ShowMessage('Генерация позиций документов завершена');

{Расчет средней себестоимости проданных товаров}
with qry do
begin
  SQL.Text := 'EXECUTE PROCEDURE sale_item_update_cost(1007)';
  ExecSQL;
end;

traCurrent.CommitRetaining;    //подтверждение транзакции
```

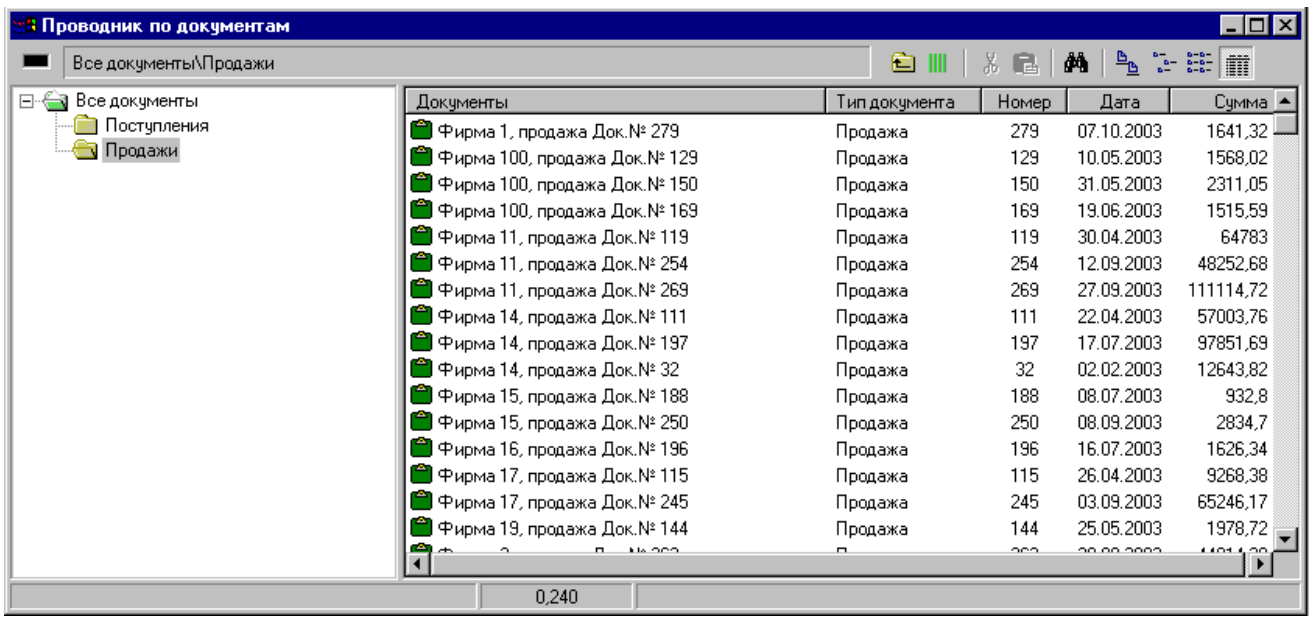
```
  UpdateTurnovers(True);   //перепроведение документов

  ProgressBar1.Position := 0;
  ProgressBar2.Position := 0;
  ShowMessage('Создание документов завершено');
end;

end.
```

Мы не будем встраивать этот проект в конфигурацию, так как он понадобится нам всего один раз. Мы просто запустим его из режима дизайнера и создадим все документы. После запуска проекта:

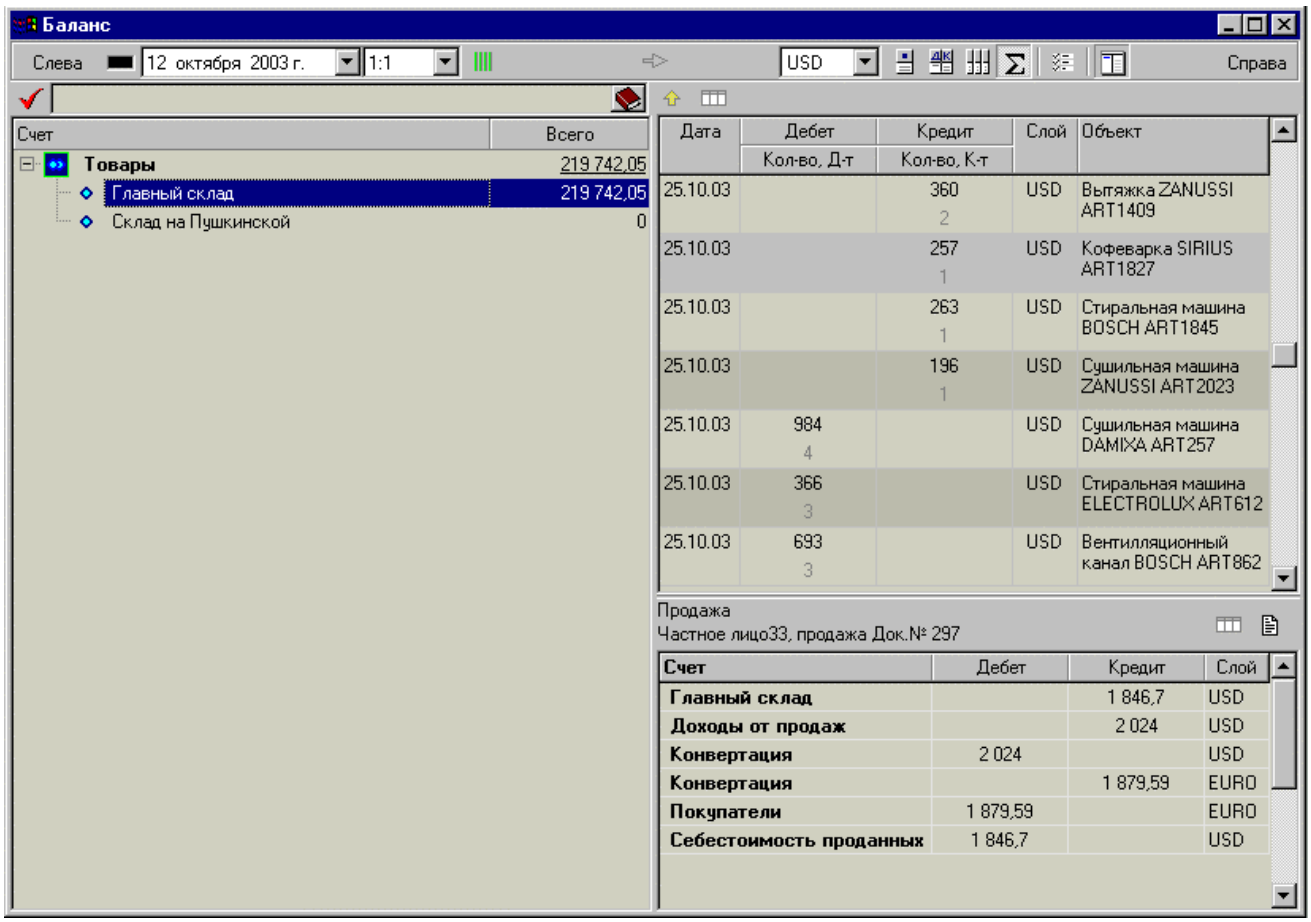
- Удаляем документы
- Создаем документы



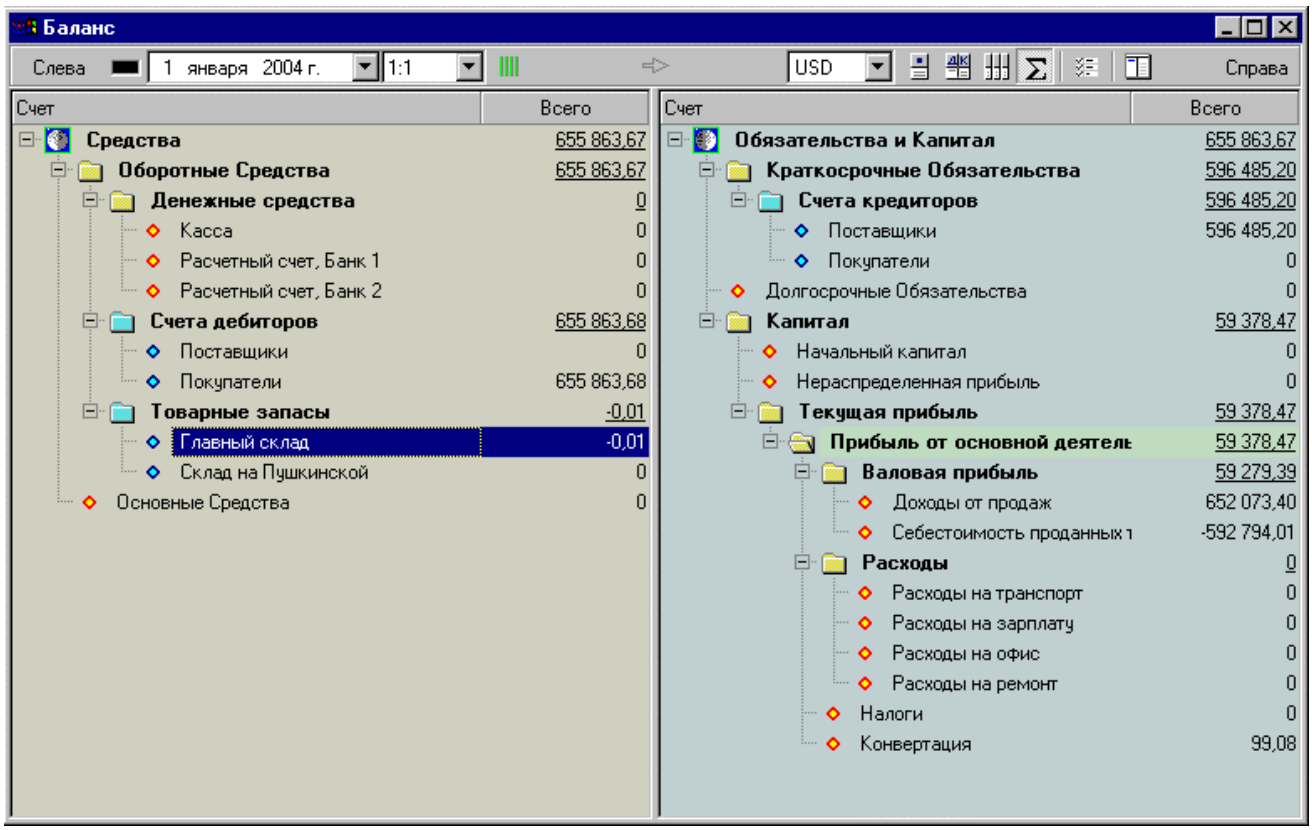
Выйдем из режима дизайнера. Откроем «Проводник про документам». Теперь у нас имеется 100 документов поступлений на склад и около 300 документов продаж:



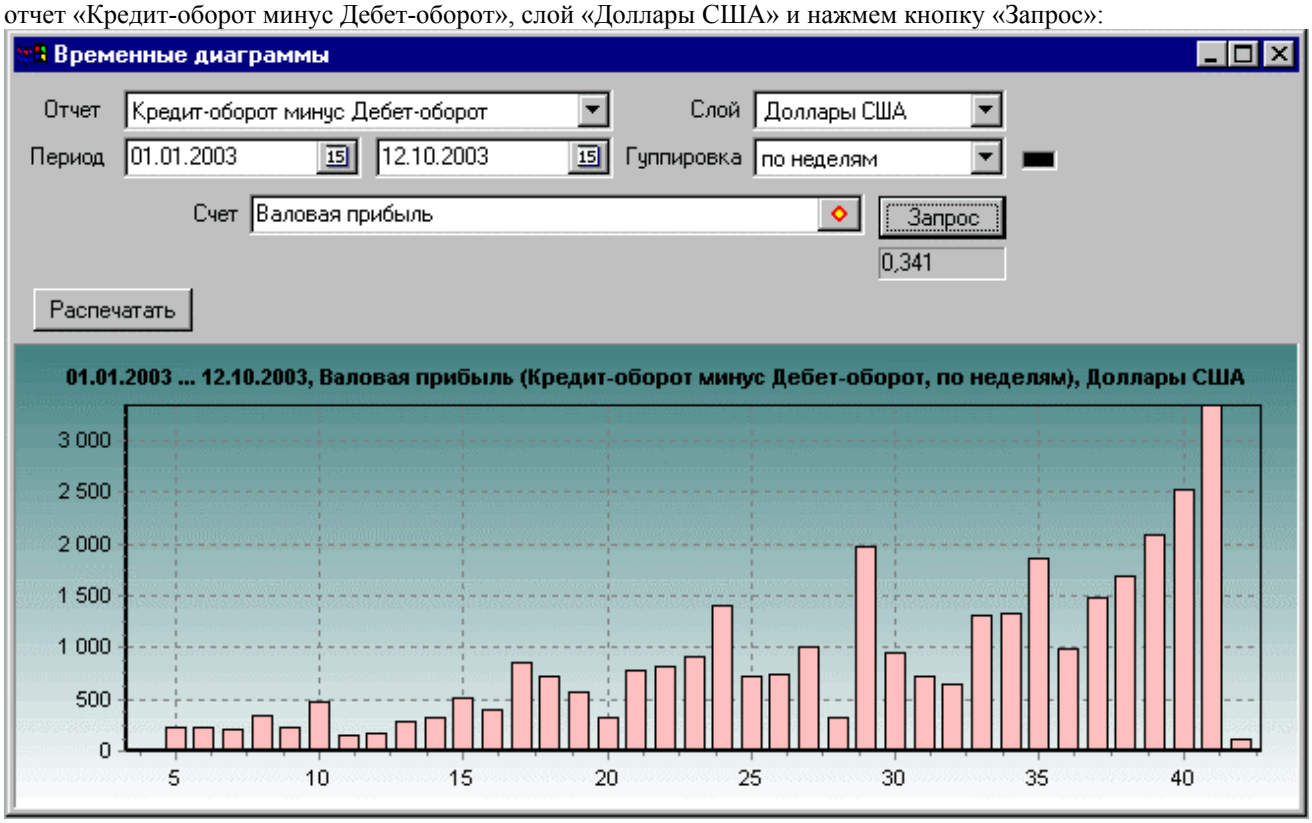
Вызовем окно «Баланс» и откроем в нем регистр «Товары», слой «USD». Выберем счет «Главный склад» и вызовем Т-счет (F3):



В Т-счете мы видим движение товаров через главный склад. Записи в дебет соответствуют поступлениям на склад, записи в кредит – продажам. Для того чтобы проверить, правильно ли мы рассчитали среднюю стоимость товаров, нужно взглянуть на баланс счета «Главный склад» на дату более позднюю, чем дата самого последнего документа. Алгоритм генерации документов устроен был так, что все, что приходило на склад должно было отгрузиться. Поэтому, скажем, на 1 января 2004 г. на «Главном складе» товаров быть не должно и сумма этого счета должна быть нулевой. Выберем «запредельную» дату в селекторе дат окна «Баланс» и взглянем на баланс всей компании. Для этого слева откроем «Средства», а справа – «Обязательства и Капитал». Включим режим «консолидированно» и выберем валюту USD:



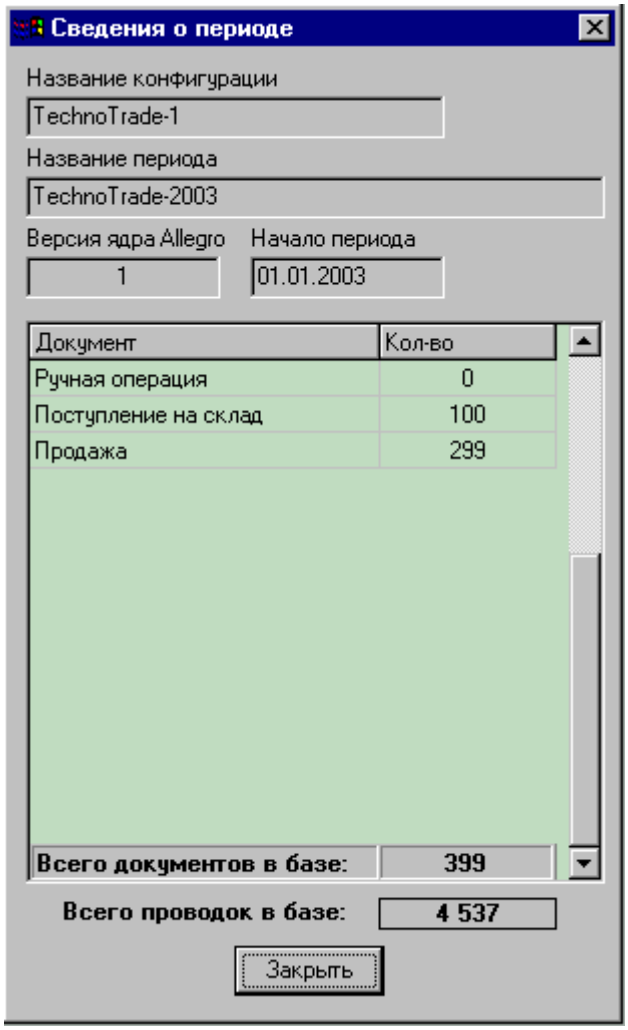
Мы видим, что стоимость товарного остатка на Главном складе близка к нулю. Значит мы все сделали правильно. Остаточная цифра -1 цент связана с ошибками округления при определении средней стоимости в документах, содержащих несколько раз один и тот же товар (у заказчика такие документы не предвидятся). Сумма средств компании около $655 000. Валовая годовая прибыль около $60 000.Все средства $655 000 находятся в счетах дебиторов у «Покупателей». Все обязательства $595 000 сосредоточены на счетах кредиторов у «Поставщиков». Прибыль неравномерно росла в течение года, что можно увидеть на диаграмме. Для получения диаграммы воспользуемся пунктом меню **Бухгалтерия/Временные диаграммы** и в появившемся окне выберем отчет «Кредит-оборот минус Дебет-оборот», слой «Доллары США» и нажмем кнопку «Запрос»:



Создадим архивную копию базы данных.

На этом мы закончили розыгрыш поступлений и продаж.

Мы можем увидеть общее число документов и бухгалтерских проводок в базе данных. Для этого нужно использовать пункт меню **Инструменты/Сведения о периоде**. Появится окно:



Итого мы создали около 400 документов. И получили около 4500 проводок.

Для выяснения количества сгенерированных позиций можно сделать в окне ISQL запрос такого рода:

```
select count(*) from sale_item
union
select count(*) from stock_in_item
```

Мы получим что-то порядка 1000 одних и 1730 других.

Если поделить число проводок на общее число проведенных позиций, то мы получим:

4500 / 2730 = 1.65

Таким образом, наши шаблоны проводок создают в среднем менее 2 проводок на каждую позицию документа. Это при том, что шаблон «поступления на склад» содержит записи по 4 счетам, а шаблон отгрузки содержит записи по 6-и счетам. Казалось бы мы должны были получить 4x1000 + 6x1730 = 14380 проводок, а мы имеем всего 4530. Экономия связана с группировкой записей перед вставкой их в таблицу проводок. Экономия важна для времени запроса баланса. Заметим, что при имеющихся 400 документах мы получили баланс за 0.1 сек. Это одно из преимуществ смешанных проводок – значительная экономия места и выигрыш в скорости.

При создании любой конфигурации имеет смысл осуществлять генерацию документов наподобие той, что мы с вами произвели. Это позволит экспериментально оценить время, которое затрачивается на SQL-запросы и убедиться в том, что запросы работают быстро и правильно. Мы рекомендуем тестировать конфигурацию до того, как пользователи начнут вносить данные. А отлаживать и тестировать ее на пустых таблицах или на ничтожном количестве данных весьма неудобно и рискованно для разработчика.

Решаем проблему хранения справочных цен

Вопрос о способе хранения текущих цен на товары является одним из существенных вопросов, которые приходится решать разработчику конфигурации. Имеется всего три возможности для хранения цен а Allegro:

- В справочнике товаров
- В таблицах настроек
- В документах

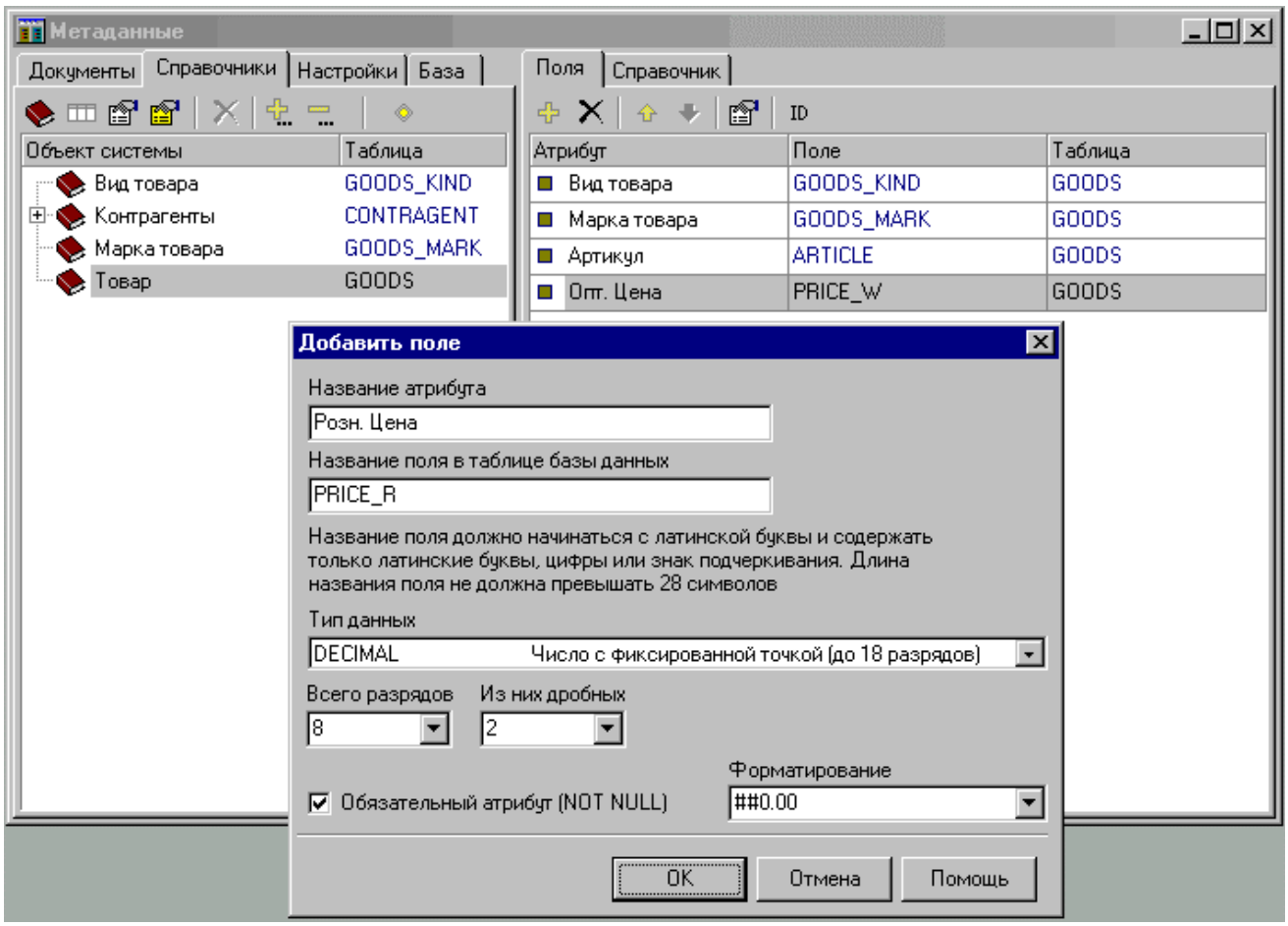
Хранение цен в справочнике товаров является наиболее простым, но не всегда верным решением. Недостатки такого способа хранения: цены всегда доступны для редактирования, невозможно создать какие-то особые прайсы для «эксклюзивных» покупателей, отсутствует возможность хранить историю цен. Преимущество хранения цен в справочнике – это простота решения для программиста и для пользователя.

Хранение цен в таблицах настроек позволяет иметь единые прайс-листы, переходящие из периода в период. Преимущество в том, что доступом к редактированию цен при таком способе хранения можно управлять.

Хранение цен в документах – наиболее корректный и гибкий способ решения этой задачи, особенно если заказчик конфигурации имеет развитую систему ценообразования в своей компании. Например, возможно создать тип документа «Прайс-лист», в шапке которого имеется название группы товаров и пять полей количеств товара, при которых действуют пять разных цен. В подчиненной таблице завести колонку товара и пять колонок цен. Затем можно создать несколько таких документов и весь имеющиеся наименования товаров распределить между этими документами, в зависимости от того, в какую ценовую группу они попадают. Далее с помощью SQL-запросов легко можно определить цену любого товара, зависящую от отгружаемого количества. Хранение цен в документах-прайсах позволяет гибче разделять товары на группы, создать, при необходимости, историю цен, легче решаются вопросы валюты, в которой хранятся цены.

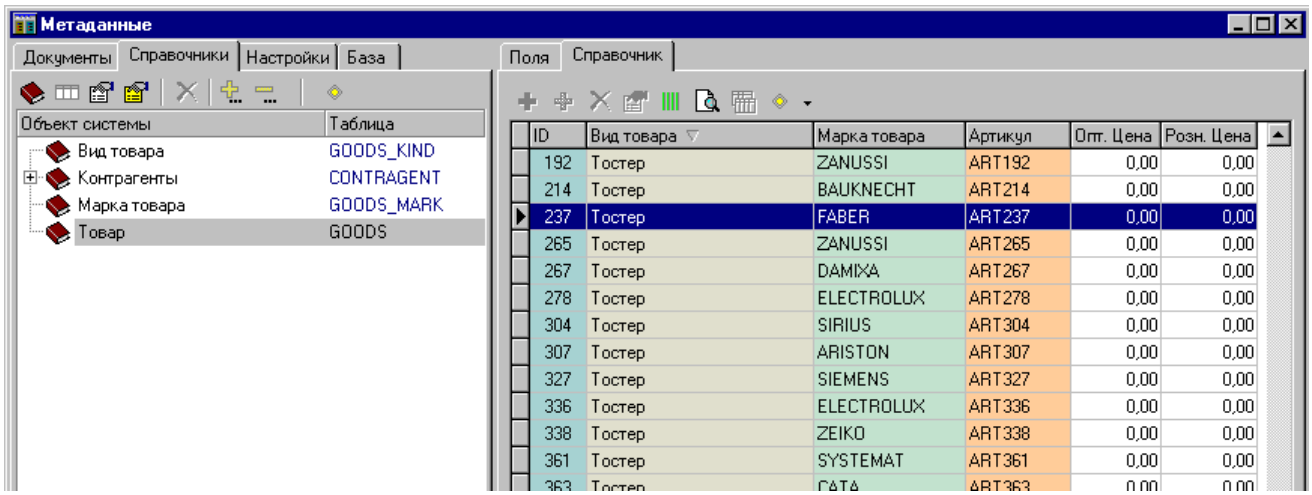
Обсуждая с нашим гипотетическим заказчиком (компанией TechnoTrade) проблему цен, мы пришли к решению, что в данном случае вполне возможно использовать простейший способ хранения цен – в справочнике товаров. Менеджеры, занимающиеся товаром, достаточно компетентны в номенклатуре, каждый отвечает за определенную группу товаров, и менеджеру удобно при создании нового товара в справочнике одновременно вводить его текущую цену. Открытый доступ к текущим ценам в данной задаче создает скорее дополнительные удобства, чем проблемы. Заказчик полагает, что система постоянных скидок, закрепленных за покупателями в справочнике контрагентов, достаточна для решения большинства вопросов, связанных с ценами. Заказчик желает иметь **две цены с НДС** в долларах США для каждого товара: оптовую и розничную. Заказчик также хочет иметь интерфейс (или отчет), в котором можно было бы сравнивать прайсовые цены товаров с их текущей себестоимостью, если эти товары имеются на складе и видеть «процент наценки».

Вызовем окно «Метаданные» и добавим в справочник «Товар» два поля:



Нам вполне достаточно 8 десятичных знаков в поле, так как не предполагается наличие товаров с ценой свыше миллиона долларов. Поле оптовой цены мы назвали PRICE_W, а розничной – PRICE_R.

Откроем закладку «Справочник» и отрегулируем ширину колонок так, чтобы все поля хорошо помещались в сетке. Сохраним настройку сетки (Ctrl+S):



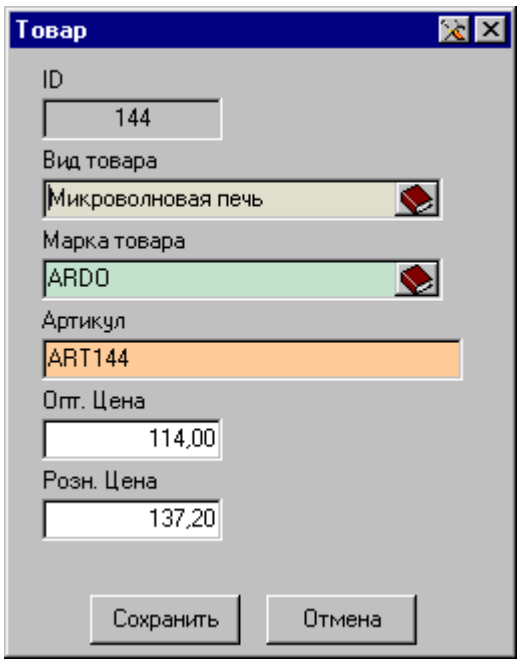
Как мы смогли убедиться, добавление новых полей в справочник не составляет проблемы. Даже если этот справочник уже используется. В данном случае товары, имеющиеся в справочнике, используются документами поступлений и продаж. Заполним цены имеющихся товаров произвольными значениями от $100 до $300, используя значение поля ID, которое в справочнике товаров имеет диапазон значений около 100..2000. Розничные цены назначим в 1.2 раза выше оптовых. Для этого вызовем окно интерактивного SQL и выполним команду:

```
update goods
set price_w = (1000 + id)/10,
    price_r = 1.2*(1000 + id)/10
where id <> 0
```

Подтвердим транзакцию.

Примечание: условие ID <> 0 необходимо всегда использовать в командах update, которые изменяют значения каких-либо полей всего справочника, иначе они не выполнятся, так как элемент любого справочника с ID=0 защищен от модификации системой. При попытке модификации элемента с ID=0 возникает исключение (exception) с сообщением: exception3 cannot change default (empty) value.

Окно редактирования элемента справочника выглядит теперь так:



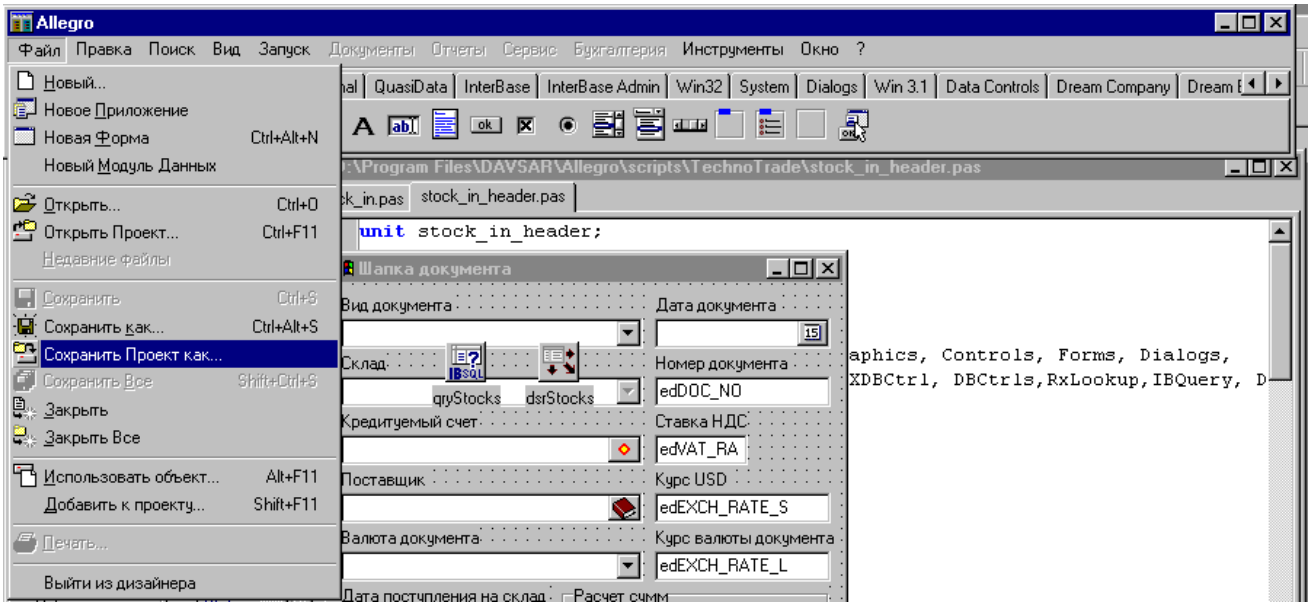
Глава 8. Создаем оконный интерфейс документа «Продажа»

Создаем оконный интерфейс документа «Продажа» путем копирования

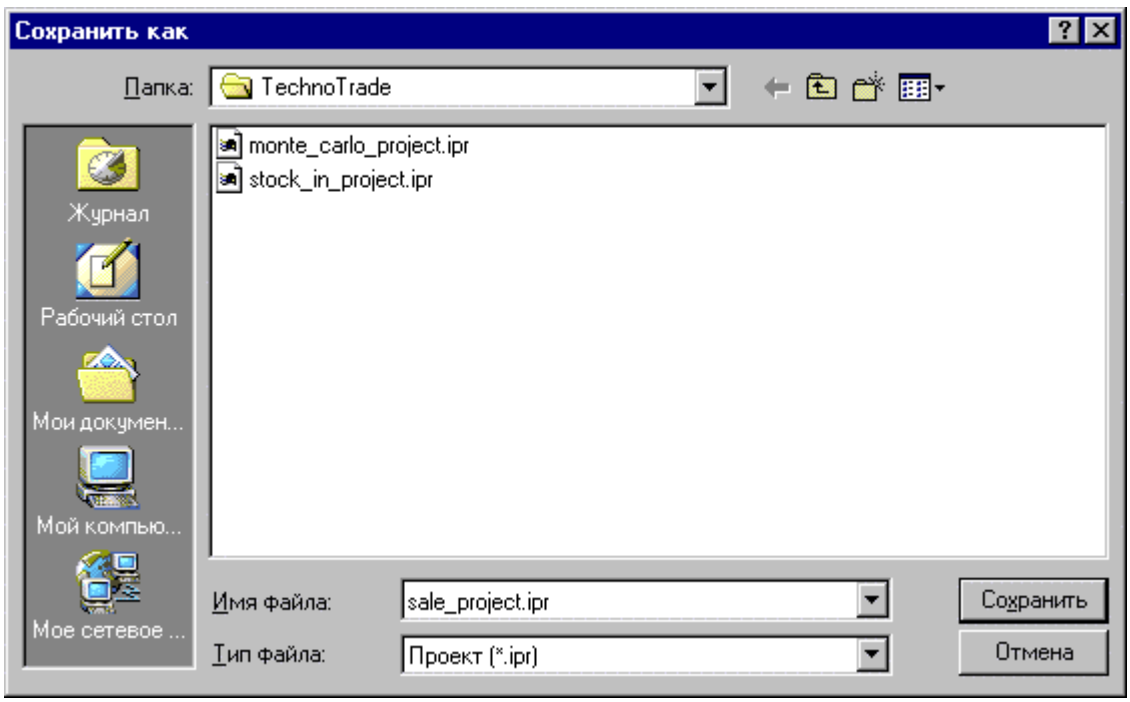
Мы создадим проект оконного интерфейса, взяв за основу имеющийся у нас проект «Поступления на склад».

Включим режим «Дизайнер».

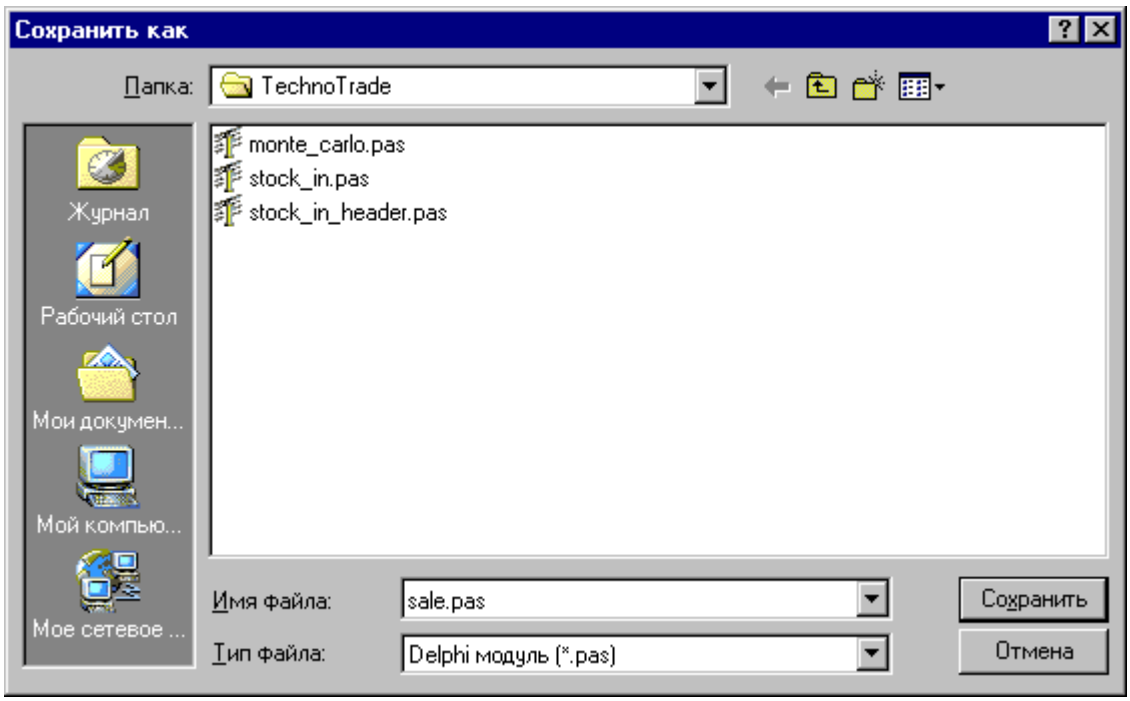
Откроем проект **stock_in_project.ipr** и сохраним проект под новым именем, используя пункт меню **Файл/Сохранить Проект как…**

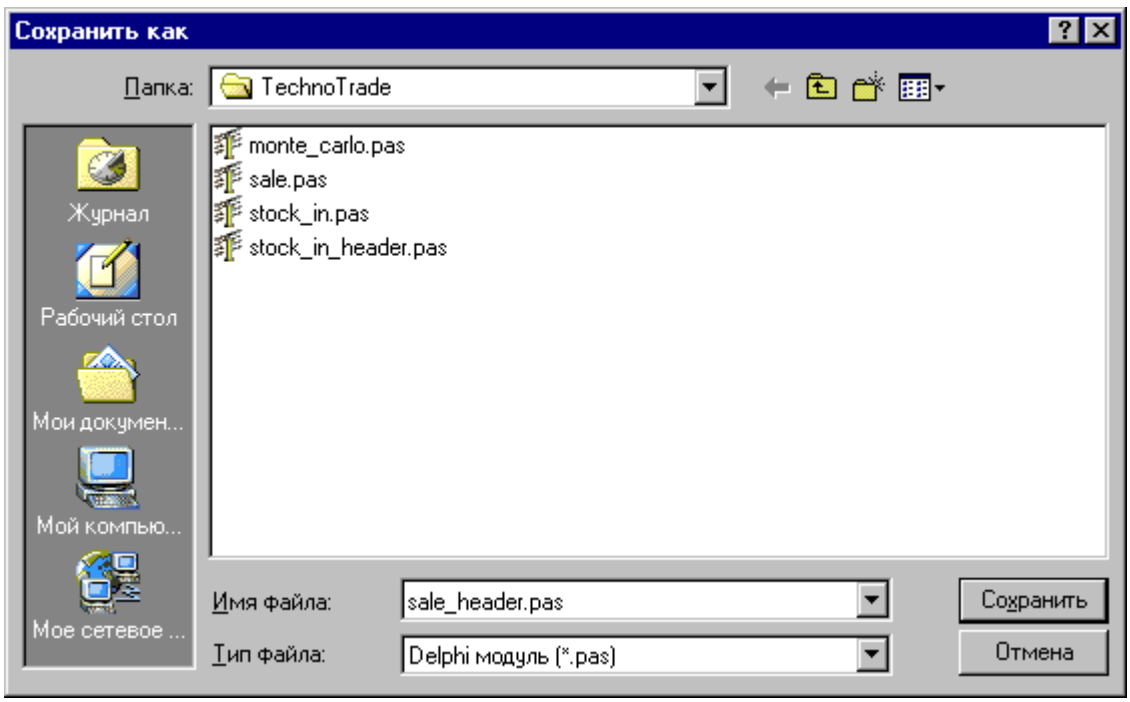


Назовем файл проекта **sale_project.ipr**



Теперь сохраним входящие в проект модули **stock_in.pas** и **stock_in_header.pas** под новыми именами: **sale.pas** и **sale_header.pas**. Для этого выберем по очереди соответствующие закладки в редакторе текста и вызовем пункт меню **Файл/Сохранить как…**





Внимание! Переименовывать модули следует именно так, из под среды дизайнера. При этом переименовывается не только файл, но и имя модуля (unit) в его тексте. Эти два имени должны совпадать. Поэтому не следует пытаться переименовывать модули при помощи обычных средств работы с файлами в операционной системе Windows.

Итак, теперь у нас имеется новый проект, полученный путем «клонирования» старого. Но нам необходимо внести еще ряд изменений, прежде чем пробовать его запустить. Прежде всего следует заменить ссылки в секции **uses** раздела **implementation.** В модуле **sale** в этой секции нужно заменить

```
uses stock_in_header;
```

на

```
uses sale_header;
```

Аналогично в модуле **sale_header** в этой секции нужно заменить

```
uses stock_in;
```

на

```
uses sale;
```

Везде в модуле заменим упроминание главной таблицы STOCK_IN на SALE.

Переименуем в Инспекторе объектов форму **StockInForm** в **SaleForm** и изменим ее заголовок **Caption** на «Продажа», а форму **StockInHeaderForm** в **SaleHeaderForm**. При помощи поиска ***Ctrl+R*** заменим все упоминания формы **StockInHeaderForm** в тексте модуля **sale** на **SaleHeaderForm** (таких упоминаний должно быть 2). Аналогично заменим все упоминания формы **StockInForm** во втором модуле заменим на **SaleForm.**

На форме **SaleForm** удалим компонент **DBText6** и **Label6** с надписью «Кредитуемый счет». В компоненте Label5 в свойстве Caption заменим значение «Поставщик» на «Покупатель», а в компоненте Label2 значение «Поступило» на «Отгружено».

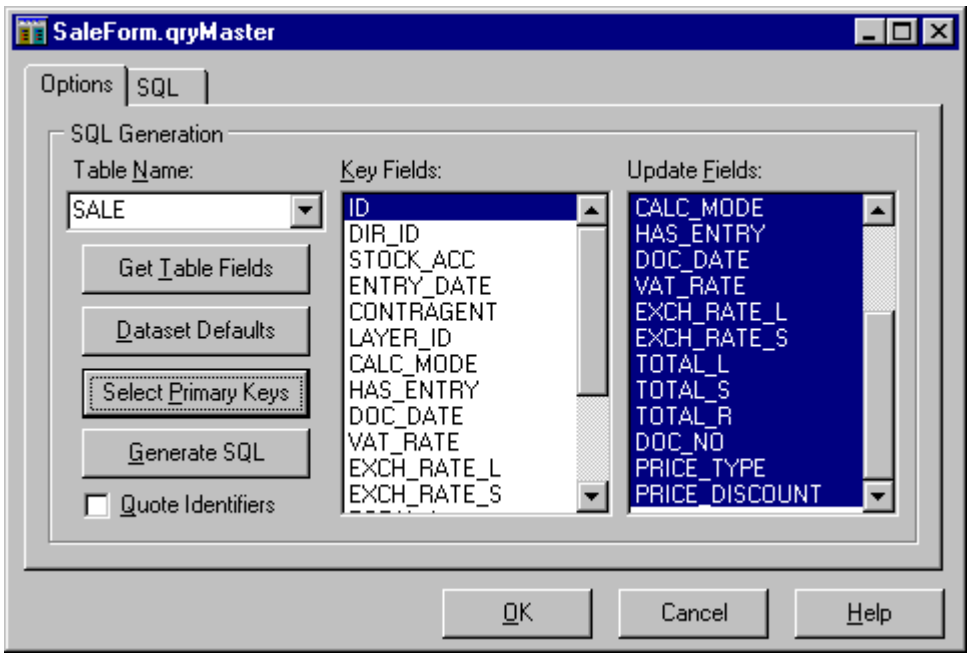
Сохраним все изменения. Теперь нам необходимо заменить тексты SQL-запросов, чтобы компоненты **qryMaster** и **qryDetail** обращались не к таблицам документа «Поступление на склад», а к таблицам документа «Продажа». Для этого в свойстве **SelectSQL** компонента **qryMaster** вместо старого текста введем новый:

```
select
  S.ID,
  S.DIR_ID,
  S.DOC_DATE,
  S.DOC_NO,
  S.ENTRY_DATE,
  S.HAS_ENTRY,
  S.STOCK_ACC,
  A1.NAME STOCK_ACC_NAME,
  S.LAYER_ID,
  L.SHORT_NAME LAYER_NAME,
  S.VAT_RATE,
  S.CONTRAGENT,
  O.SHORT_NAME CONTRAGENT_NAME,
  S.CALC_MODE,
  S.EXCH_RATE_L,
  S.EXCH_RATE_S,
  S.TOTAL_L,
  S.TOTAL_R,
  S.TOTAL_S
  S.PRICE_TYPE,
  S.PRICE_DISCOUNT
from
  SALE S,
  OBJECT_NAMES O,
  LAYER L,
  ACC A1
where
  S.CONTRAGENT = O.OBJECT_ID and
  S.LAYER_ID = L.LAYER_ID and
  S.STOCK_ACC = A1.ACC_ID and
  S.ID = :ID
```

Удалим также тексты запросов из свойств **InsertSQL, ModifySLQ, RefreshSQL**.

Дважды щелкнем на компоненте **qryMaster** и в редакторе полей удалим поля DOC_KIND, ACC_ID и CREDIT_ACC_NAME.

С помощью контекстного меню компонента **qryMaster** вызовем DataSet Editor. Выберем все поля таблицы SALE с помощью кнопки **Get Table Fields**. В списке **Key Fields** выберем ключевое поле ID, а в списке **Update Fields** все имеющиеся поля и нажмем кнопку **Generate SQL**.



Будут автоматически сформированы SQL-запросы **InsertSQL, ModifySQL, DeleteSQL** и **RefreshSQL**. Сохраним изменения кнопкой **OK**.

Удалим текст запроса в свойстве **DeleteSQL**. А текст из **SelectSQL** перепишем в **RefreshSQL**.

Теперь займемся компонентом **qryDetail**. Заменим в свойстве **SelectSQL** в тексте запроса лишь имя таблицы STOK_IN_ITEM на SALE_ITEM. В остальном запрос оставим тем же:

```
select
  SI.ID,
  SI.N,
  SI.GOODS,
  O.SHORT_NAME ITEM_NAME,
  SI.QUANTITY,
  SI.PRICE_L,
  SI.PRICE_L_WO_VAT,
  SI.PRICE_R,
  SI.PRICE_R_WO_VAT,
  SI.AMOUNT_L,
  SI.AMOUNT_R,
  SI.AMOUNT_S
from
  SALE_ITEM SI,
  OBJECT_NAMES O
where
  SI.GOODS = O.OBJECT_ID and
  SI.ID = :ID
order by SI.N
```

Ту же замену имени таблицы с STOK_IN_ITEM на SALE_ITEM предпримем в текстах SQL-команд в свойствах **InsertSQL, ModifySQL, DeleteSQL, RefreshSQL**.

Заменим генератор в свойстве **GeneratorField** со значения STOCK_IN_ITEM_N_GEN на SALE_ITEM_N_GEN.

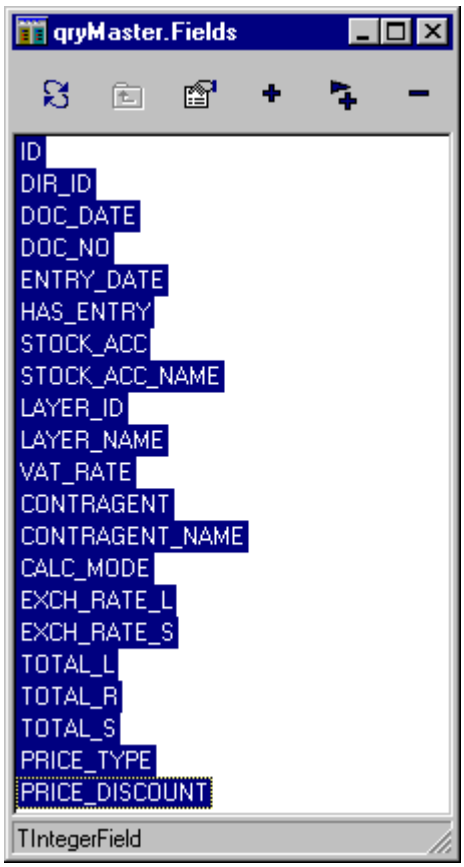
В тексте модуля **sale** удалим команды:

```
 qryMaster.FieldByName('DOC_KIND').AsInteger := 0;
 qryMaster.FieldByName('ACC_ID').AsInteger := 1012; //Поставщики
```

Вместо них вставим команды:

```
qryMaster.FieldByName('PRICE_TYPE').AsInteger := 0;
qryMaster.FieldByName('PRICE_DISCOUNT').AsCurrency := 0;
```

Дважды щелкнем на компоненте **qryMaster** и в редакторе полей удалим все прежние поля и добавим все поля заново с помощью пункта контекстного меню «Добавить поля» редактора полей:



Назначим каждому полю русское название, присваивая его свойству **DisplayLabel** в Инспекторе объектов.

Теперь перейдем к форме «шапки». Компоненты на этой форме достались нам от формы «шапки» «Поступления на склад». Некоторые компоненты мы оставим, а некоторые – удалим.

Удалим с формы **SaleHeaderForm** все компоненты, предназначенные для редактирования полей «Вид документа» и «Кредитуемый счет», так как этих полей в документе «Продажа» нет. Изменим надписи в компонентах **Label**:

«Поставщик» на «Покупатель»

«Дата поступления на склад» на «Дата отгрузки со склада»

Изменим свойство Caption компонента-птички **cbHasEntry**:

«Товар поступил на склад» на «Товар отгружен»

Добавим компонент **Label** с палитры **Standard**, назначив ему **Caption** «Скидка в цене».

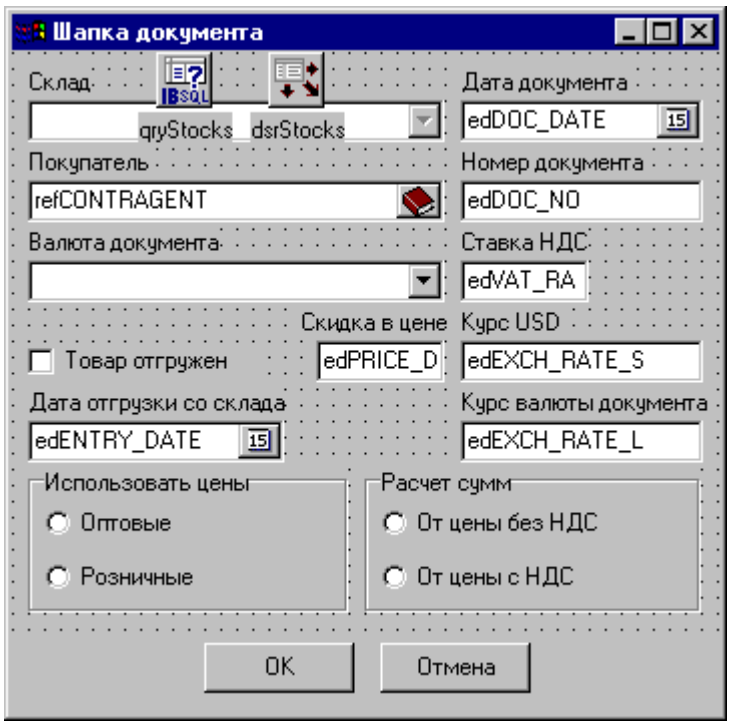
Добавим компонент **DBEdit** с палитры **DataControls** и назначим ему свойства:

свойство	**Значение**
Name	edPRICE_DISCOUNT
DataSource	SaleForm.dsrMaster
DataField	PRICE_DISCOUNT

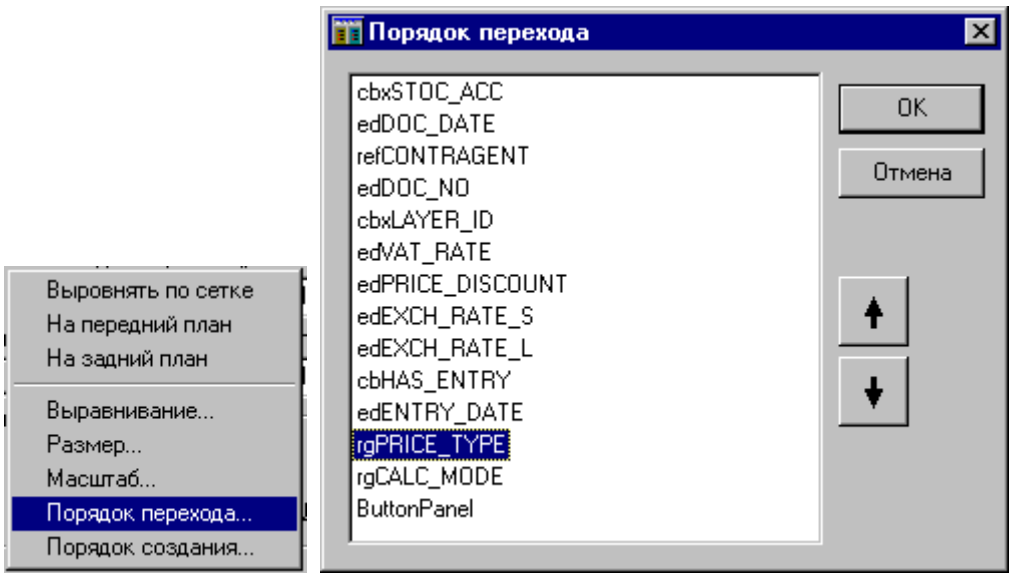
Добавим компонент **DBRadioGroup** с палитры **Data Controls** и назначим ему свойства:

свойство	**Значение**
Name	RgPRICE_TYPE
DataSource	SaleForm.dsrMaster
DataField	PRICE_TYPE
Caption	Использовать цены
Items	Оптовые Розничные
Values	0 1

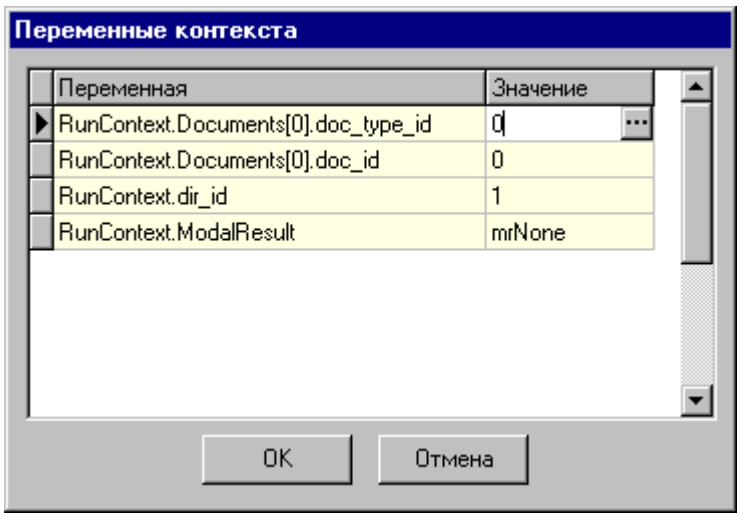
Расположим все оконные органы управления так, как показано на рисунке:



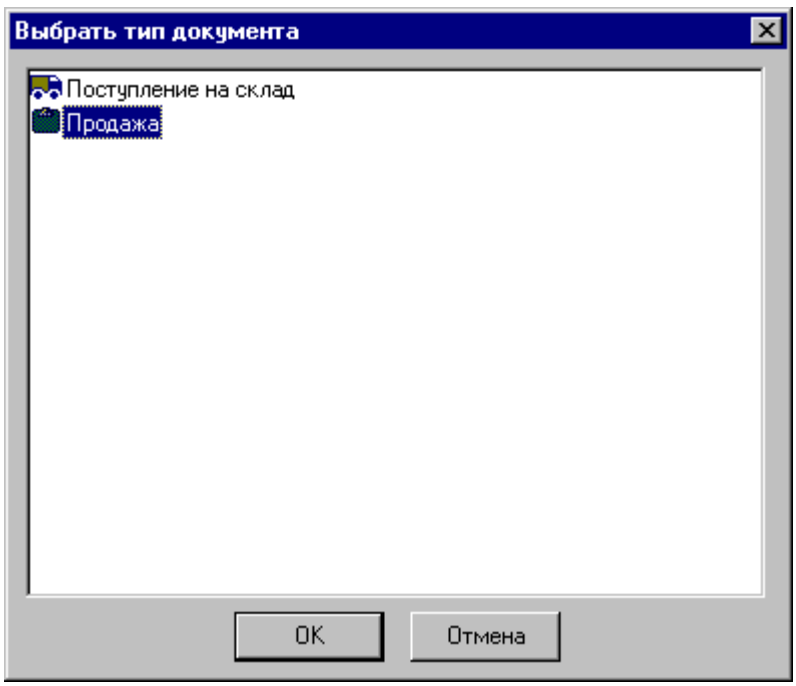
Установим по вкусу порядок перехода фокуса в этом окне, вызвав редактор свойства «Порядок перехода» (**TabOrder**) компонентов с помощью контекстного меню компонента формы:



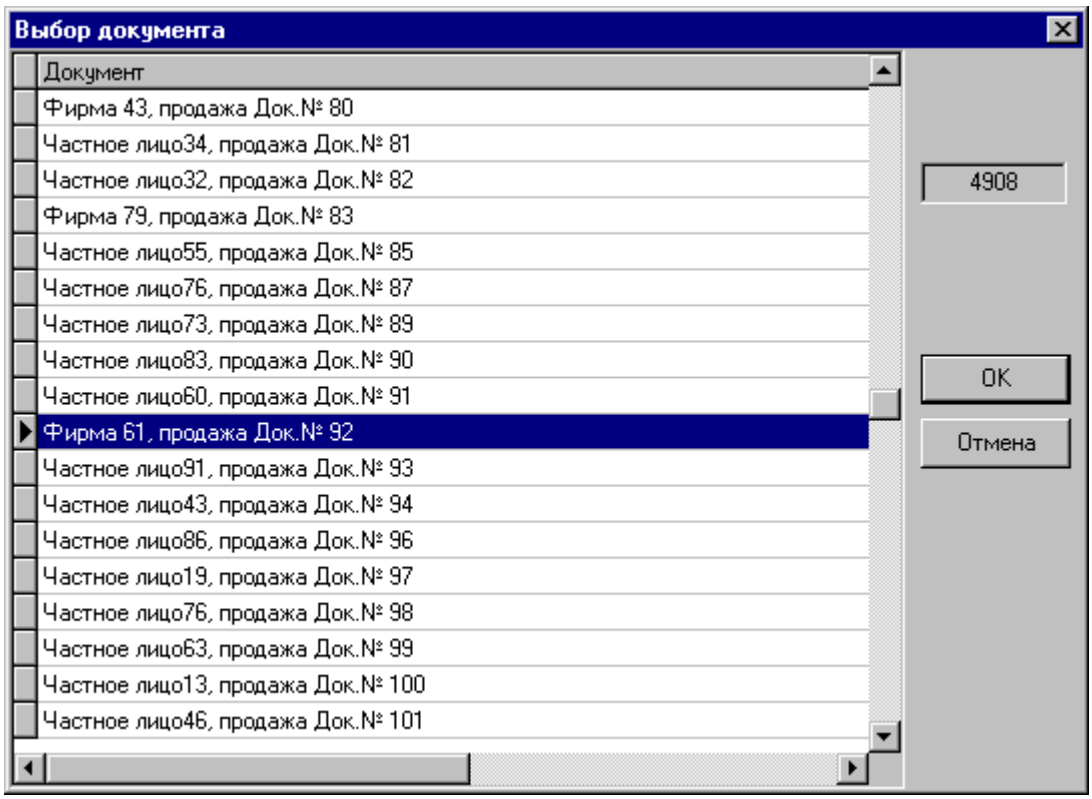
Сохраним все изменения. Установим контекст запуска проекта. Для этого с помощью меню **Запуск/Переменные контекста** вызовем окно «Переменные контекста»:



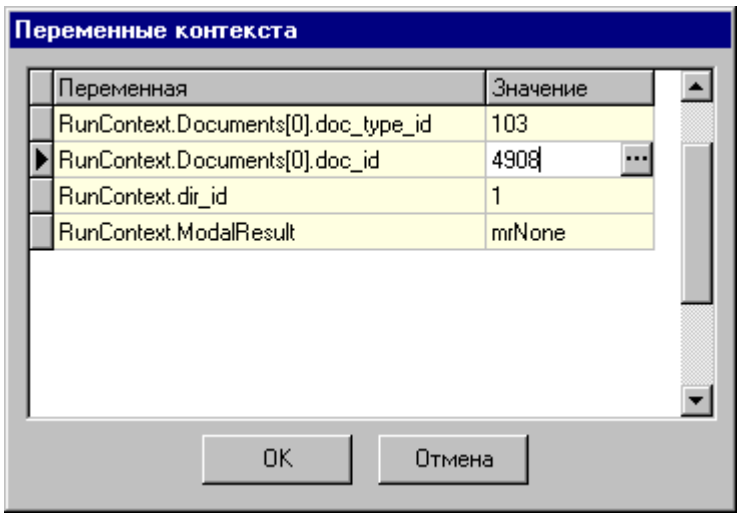
Выберем переменную **RunContext.Documents[0].doc_type_id** (id типа документа). Нажав на кнопку с многоточием , вызовем диалог выбора типа документа, выберем в нем «Продажу» и нажмем кнопку **OK**:



Выберем вторую переменную - **RunContext.Documents[0].doc_id** (id документа). Нажав на кнопку с многоточием , вызовем диалог выбора документа, выберем какой-нибудь документ и нажмем кнопку **OK**:



Значения переменных контекста установлены. Нажмем кнопку OK, чтобы сохранить их.



Запустим проект (F9).

Проверим, как работает редактирование шапки и позиций, как происходит поиск по нажатию клавиш.

Для того чтобы завершить проект интерфейса «Продажа», нам предстоит еще большая работа. Нам предстоит реализовать:

1. Копирование цен из справочника товаров в документ
2. Определение средней себестоимости товаров при сохранении документа, если товар отгружен.
3. Печать документов типа «Счета-фактуры» и «Накладной»

Начнем с копирования цены при добавлении товаров из справочника. Заказчик желает, чтобы скидка покупателю распространялась непосредственно на цену товара. Следовательно, нам не только придется копировать цену из справочника, но и изменять ее в соответствии со скидкой.

У нас имеется два способа добавить товар в документ «Продажа»:

1. Найти товар «поиском по нажатию клавиш» и вставить его из нижней сетки клавишей ***Enter***.
2. Нажать кнопку с многоточием в поле «Наименование» позиции документа и в появившемся окне диалога «Выбрать из справочника» найти нужный нам товар и нажать кнопку «Выбрать»

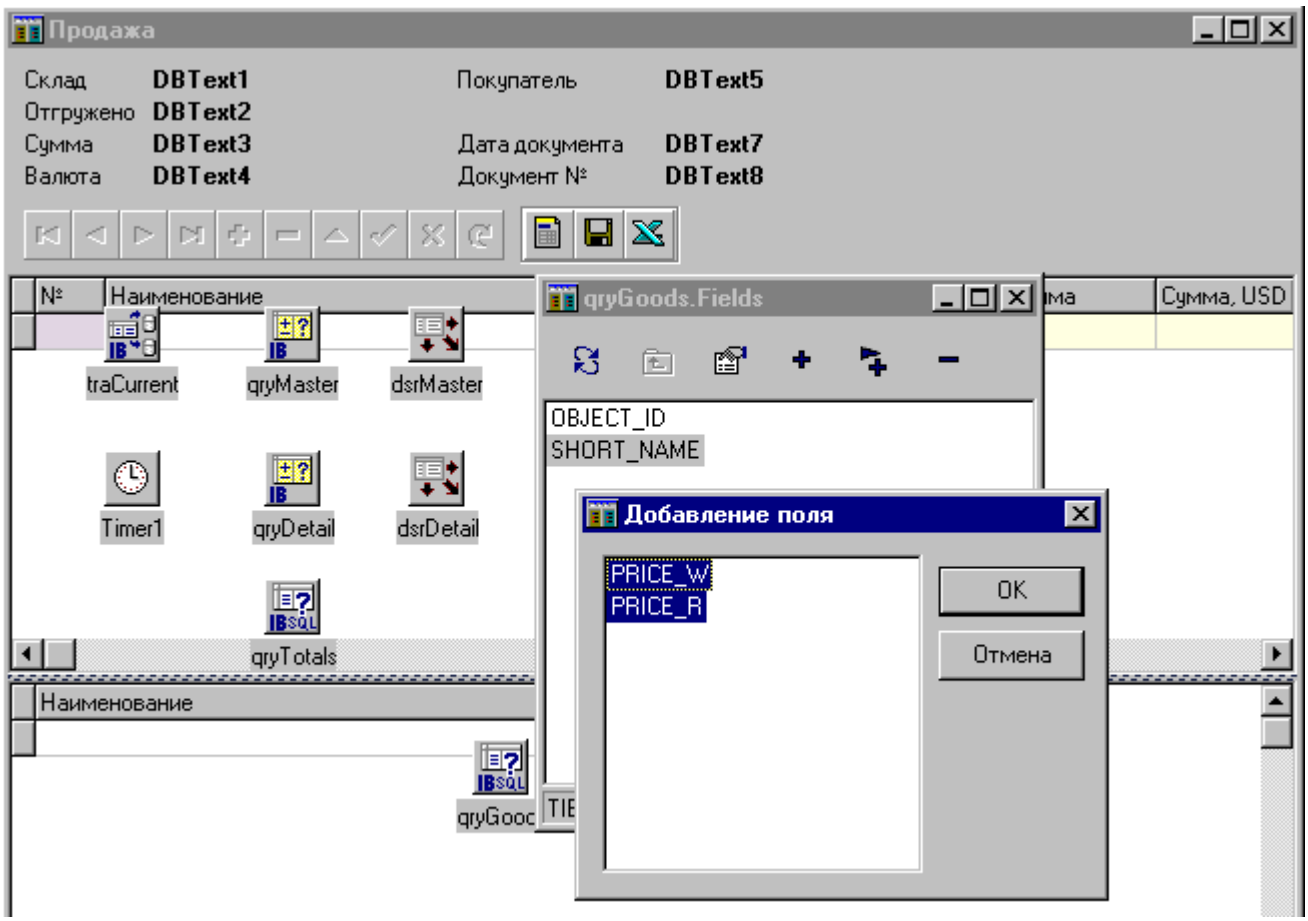
Нам нужно сделать так, чтобы вне зависимости от способа добавления товара цена, рассчитанная на основании цены в справочнике, вставлялась вместе с добавляемым товаром в позицию документа. Нам также представляется разумным, чтобы пользователь мог видеть справочные цены товаров в нижней сетке, когда он ищет их «поиском по нажатию клавиши». Поэтому мы несколько изменим текст запроса, управляющего этим поиском, добавив в него поля цен.

Итак, изменим текст запроса в свойстве **SQL** компонента **qryGoods** на:

```
SELECT O1.OBJECT_ID, O1.SHORT_NAME, G.PRICE_W, G.PRICE_R
FROM GOODS G, OBJECT_NAMES O1
WHERE G.ID = O1.OBJECT_ID
```

Фактически мы просто добавили два поля в запрос.

Дважды щелкнем на компоненте **qryGoods** и в редакторе полей с помощью контекстного меню вызовем пункт **Добавить все поля…**



Нажмем кнопку **OK**. Выбирая в редакторе полей поля, назначим им свойства в Инспекторе объектов:

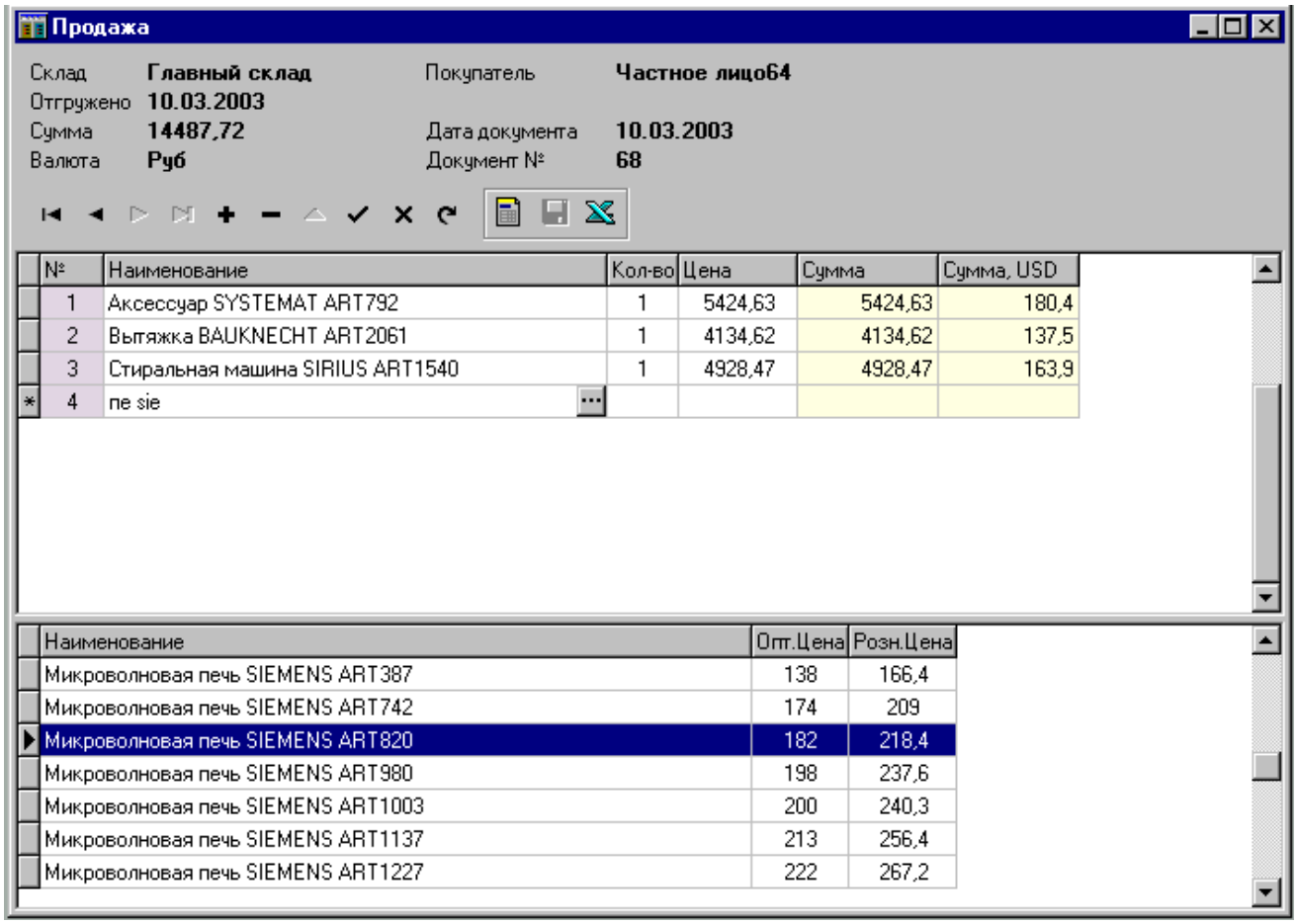
Поле	DisplayLabel	Alignment
PRICE_W	Опт.Цена	taCenter
PRICE_R	Розн.Цена	taCenter

Закроем редактор полей. Выберем нижнюю сетку **dbgGoods** и вызовем редактор колонок с помощью контекстного меню или дважды щелкнув в Инспекторе объектов на свойстве **Columns**. Добавим все колонки в редактор с помощью соответствующего пункта его контекстного меню и удалим затем колонку OBJECT_ID.

Изменим также текст модуля в части формирования текста запроса «по нажатию клавиш». Это происходит в обработчике события **OnSetEditText** сетки **dbgDetail**:

```
{Конструирование запроса из строки признаков, разделенных пробелами}
s := 'SELECT O1.OBJECT_ID, O1.SHORT_NAME, G.PRICE_W, G.PRICE_R '#13+
    'FROM GOODS G, OBJECT_NAMES O1'#13+
    'WHERE G.ID = O1.OBJECT_ID AND O1.OBJECT_ID <> 0';
```

Запустим проект. Теперь при поиске «по нажатию клавиш» в нижней сетке товары отображаются со своими ценами:



Осталось реализовать вставку нужной цены в документ при вставке товара из нижней сетки в позицию документа. Мы напишем универсальную процедуру, которая копирует цену в позицию документа. Эта процедура должна рассчитать цену на основании справочной и учесть при этом скидку, предоставляемую покупателю, а если требуется, то и пересчитать ее по курсу в другую валюту.

Остановим проект. Добавим вызов процедуры, которая выделена жирным шрифтом, в обработчик события **OnDblClick** нижней сетки:

```
{Копирование позиции из нижнего списка товаров двойным щелчком мыши}
procedure TSaleForm.dbgGoodsDblClick(Sender: TObject);
begin
  qryDetail.Edit; {Присвоение значений полям позиции}
  qryDetail.FieldByName('GOODS').AsInteger :=
    qryGoods.FieldByName('OBJECT_ID').AsInteger;
  qryDetail.FieldByName('ITEM_NAME').AsString :=
    qryGoods.FieldByName('SHORT_NAME').AsString;
```

```
  UpdatePrice(qryGoods);
  LastValue := qryDetail.FieldByName('ITEM_NAME').AsString;
  dbgDetail.SelectedField := qryDetail.FieldByName('QUANTITY');
  dbgDetail.SetFocus; //Возвращение фокуса ввода в сетку позиций
end;
```

Добавим в секцию **private** класса формы **TSaleForm** объявление процедуры (выделено жирным):

```
  private
    { Private declarations }
    procedure UpdatePrice(PriceQuery: TDataSet);
  public
    { Public declarations }
  end;
```

Добавим в раздел **implementation** модуля **sale** реализацию этой процедуры:

```
{Вставка цены}
procedure TSaleForm.UpdatePrice(PriceQuery: TDataSet);
var
  price_s, price_l: Extended;
begin
  {В зависимости от типа используемой цены}
  if qryMaster.FieldByName('PRICE_TYPE').AsInteger = 1 then
    price_s := PriceQuery.FieldByName('PRICE_R').AsCurrency
  else
    price_s := PriceQuery.FieldByName('PRICE_W').AsCurrency;
  {Применение скидки и пересчет в валюту документа}
  price_l := price_s *
    (1 - qryMaster.FieldByName('PRICE_DISCOUNT').AsCurrency/100)*
     qryMaster.FieldByName('EXCH_RATE_S').AsCurrency/
     qryMaster.FieldByName('EXCH_RATE_L').Ascurrency;
  {В зависимости от режима расчета (от цен с НДС или без НДС)
   происходит окгругление}
  if qryMaster.FieldByName('CALC_MODE').AsInteger = 1 then
    qryDetail.FieldByName('PRICE_L').AsCurrency :=
      round(price_l * 100)/100
  else
    qryDetail.FieldByName('PRICE_L_WO_VAT').AsCurrency :=
      round(price_l * 10000/
          (qryMaster.FieldByName('VAT_RATE').AsCurrency + 100))/100;
end;
```

Согласно договоренности с заказчиком, в справочнике товаров хранятся цены с НДС. Поэтому способ расчета сумм CACL_MODE (от цены с НДС или от цены без НДС) в документе «Продажа» имеет несколько иное значение, чем в документе «Поступление на склад». Если в документе «Продажа» выбран режим CALC_MODE = 0 (расчет от цен без НДС), то берется за основу справочная цена с НДС, из нее вычитается НДС, затем цена округляется до двух знаков после запятой и вставляется в документ. После таких преобразований цена с НДС в документе может уже не совпадать со справочной из-за ошибок округления. Если же выбран режим CALC_MODE = 1 (режим по умолчанию), то в документ копируются цены с НДС из справочника, а цены без НДС вычисляются с точностью до 3 знаков после запятой. В этом режиме цены с НДС в документе соответствуют справочным.

Запустим проект и убедимся, что расчет цен работает при разных валютах документа и в разных режимах «использовать цену».

Нам осталось реализовать вызов процедуры **UpdatePrice** при втором способе добавления товаров в документ – из окна диалога «Выбрать из справочника», которое вызывается нажатием кнопки с многоточием в сетке позиций. Добавим в событие **OnEditButtonClick** сетки **dbgDetail** текст, выделенный жирным шрифтом:

```
procedure TSaleForm.dbgDetailEditButtonClick(Sender: TObject);
var
  Q: TIBQuery;
begin
  with RefDialog1 do
  if Execute then //вызываем диалог на экран и проверяем, был ли выбран товар
  begin
    qryDetail.Edit; //переводим набор данных в режим редактирования

    {присваиваем значение ID товара полю GOODS набора}
    qryDetail.FieldByName('GOODS').AsInteger := Object_ID;

    {создаем run-time компонент запроса и запрашиваем цену товара}
    Q := TIBQuery.Create(nil);
    try
      Q.Transaction := self.traCurrent;
      Q.SQL.Text := 'SELECT PRICE_W, PRICE_R FROM GOODS WHERE ID = :ID';
      Q.ParamByName('ID').AsInteger := Object_ID;
      Q.Open;
      UpdatePrice(Q);
    finally
      Q.Free;
    end;

    {присваиваем значение краткое наименование товара полю ITEM_NAME набора}
    qryDetail.FieldByName('ITEM_NAME').AsString:=
                    NameOfRefObject(Object_ID, 'SHORT_NAME');

    {перемещаем фокус ввода в сетке в поле "Количество"}
    dbgDetail.SelectedField := qryDetail.FieldByName('QUANTITY');
  end;
end;
```

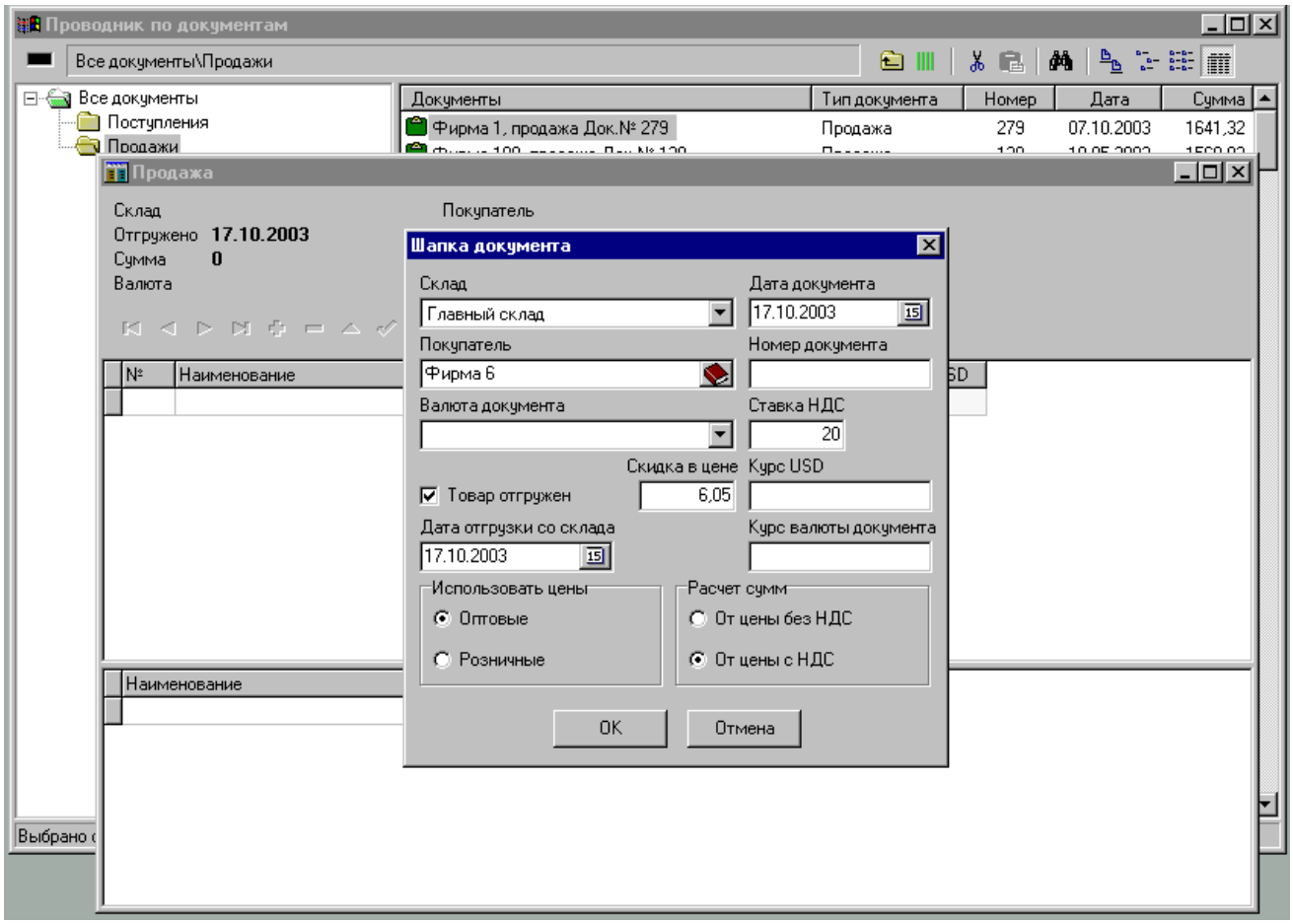
Запустим проект (F9) и убедимся, что при выборе товара из справочника в диалоге справочная цена копируется в позицию документа.

Нам еще нужно добиться того, чтобы скидка, предоставляемая покупателю, копировалась из справочника в документ сразу после выбора покупателя из справочника контрагентов. Обычно это происходит на стадии заполнения шапки документа, когда пользователь создает новую «продажу».

Для этого выберем на форме **SaleHeaderForm** компонент **refCONTRAGENT** и создадим обработчик его события **OnExecuted**. В обработчик впишем такой текст:

```
procedure TSaleHeaderForm.refCONTRAGENTExecuted(Sender: TObject);
var
  Q: TIBQuery;
begin
  {создаем run-time компонент запроса и запрашиваем скидку}
  Q := TIBQuery.Create(nil);
  try
    Q.Transaction := SaleForm.traCurrent;
    Q.SQL.Text := 'SELECT DISCOUNT FROM FIRM WHERE ID = :ID';
    Q.ParamByName('ID').AsInteger := refCONTRAGENT.Object_ID;
    Q.Open;
    {Копируем скидку в шапку документа}
    SaleForm.qryMaster.FieldByName('PRICE_DISCOUNT').AsCurrency :=
      Q.Fields[0].AsCurrency;
  finally
    Q.Free;
  end;
end;
```

Выйдем из режима дизайнера и вызовем окно «Метаданные». Выберем тип документа «Продажа» и в «Дополнительных свойствах документа» на закладке «Интерфейс» назначим проект оконного интерфейса **sale_project.ipr**. Установим птичку «создавать документ из проводника» и выберем запуск «Множество экземпляров». Нажмем кнопку «Сохранить». Вызовем «Проводник по документам» и убедимся, что теперь документы продаж вызываются двойным щелчком на экран. Попытаемся создать новый документ «Продажа» с помощью соответствующего пункта контекстного меню «Проводника по документам»:



Можно заметить, что присутствует еще ряд шероховатостей. Например, так и не решен вопрос об автоматической нумерации документов типа «Продажа». На главной форме **SaleForm** никак не отображается информация о том, какие цены используются в документе, оптовые или розничные. Попробуйте самостоятельно реализовать отображение информации об использующихся ценах на форме **SaleForm**. Попробуйте также самостоятельно реализовать автоматическую нумерацию документов. Разные компании используют разные способы нумерации документов. Постарайтесь реализовать тот способ, который принят в Вашей компании.

Хорошо бы еще при выборе валюты документа, вставлять в него курсы валют, для того чтобы не набирать их каждый раз вручную. Для этого можно было бы обратиться к системной таблице LAYER_RATES или системной хранимой процедуре LAST_EXCHANGE_RATES, запрашивающей ближайшие системные курсы на какую-то дату. Здесь возможны несколько подходов. Дата документа и дата отгрузки могут не совпадать между собой, документ может выписываться заранее и курс использоваться договорной. В любом случае, универсального решения здесь, видимо нет. Попробуйте найти свое решение проблемы и реализовать его.

Реализуем расчет средней себестоимости в документе «Продажа».

Мы с вами уже рассчитывали среднюю себестоимость, когда занимались генерацией продаж. Тогда мы создали в базе данных хранимую процедуру SALE_ITEM_UPDATE_COST, которая вычисляла среднюю стоимость каждого проданного товара и записывала ее в позиции всех документов «Продажа». Вычисление средней стоимости делалось на основе данных в таблице бухгалтерских проводок ACC_TURN.

Мы создадим новую хранимую процедуру SALE_ITEM_SET_COST на основе имеющейся процедуры, ограничив ее действие позициями одного документа.

Вызовем окно интерактивного SQL и введем такой текст (жирным шрифтом выделены отличия от процедуры SALE_ITEM_UPDATE_COST):

```
create procedure sale_item_set_cost(id integer)
as
  /*переменные для хранения полей*/
  declare variable acc_id integer;
  declare variable op_date date;
  declare variable layer_id integer;
  declare variable object_id integer;
  declare variable template_id integer;
  declare variable doc_id integer;
  declare variable debit decimal(18,2);
  declare variable credit decimal(18,2);
  declare variable quantity_debit decimal(18,3);
  declare variable quantity_credit decimal(18,3);
  /*переменные для хранения текущей суммы и количества в партии товара*/
  declare variable amount decimal(18,2);
  declare variable quantity decimal(18,3);
  declare variable avg_cost double precision;
  /*переменная для храниения текущего id товара*/
  declare variable goods integer;
begin
  goods = -1;
  select stock_acc from sale where id = :id
  into :acc_id;
  for select
    op_date, layer_id, object_id, template_id, doc_id,
    debit, credit, quantity_debit, quantity_credit
  from
    acc_turn
  where
    acc_id = :acc_id and layer_id = 2  and
    object_id in (select goods from sale_item where id = :id)
  order by
    object_id, op_date, template_id
  into
    :op_date, :layer_id, :object_id, :template_id, :doc_id,
    :debit, :credit, :quantity_debit, :quantity_credit
  do
  begin
    if (goods <> object_id) then /*начинаем очередной товар*/
    begin
      amount = 0;
      quantity = 0;
      goods = object_id;
    end
    if ((template_id = 103) and (doc_id = id)) then /*103 - Продажа*/
    begin
      if (quantity <> 0) then
        avg_cost = amount / quantity;
      else
```

```
      avg_cost = 0;
    credit = quantity_credit * avg_cost;
    /*изменяем стоимость в позициях документа "Продажи"*/
    update sale_item set cost_s = quantity * :avg_cost
    where id = :doc_id and goods = :object_id
            and cost_s <> (quantity * :avg_cost);
  end
  amount = amount + debit - credit;
  quantity = quantity + quantity_debit - quantity_credit;
 end
end
```

Выполним команду в окне ISQL, нажав Ctrl+Enter.

Обратите внимание, что мы используем вложенный запрос в условии WHERE основного запроса:

```
object_id in (select goods from sale_item where id = :id)
```

Этот второй запрос возвращает нам множество всех товаров, себестоимость которых следует уточнить. Вычисленная стоимость записывается только в документ с внутренним номером id, который передается при вызове процедуры при помощи входного параметра.

Кроме этого мы добавили условие в команду UPDATE:

and cost_s <> (quantity * :avg_cost)

Это защитит от обновления те позиции документа, себестоимость в которых уже рассчитана и рассчитана правильно. Дело в том, что мы будем вызывать созданную нами хранимую процедуру при сохранении документа «Продажа». А так как пользователь может несколько раз редактировать и сохранять документ, то не хотелось бы, чтобы это приводило к многократной перезаписи одних и тех же позиций в базе. Многоверсионный механизм транзакций сервера InterBase при каждой модификации записи в таблице создает ее копию и лишние модификации могут приводить к «замусориванию» базы данных такими копиями и неоправданному увеличению размера файла базы.

Нам осталось добавить вызов хранимой процедуры SALE_ITEM_SET_COST в проект оконного интерфейса «Продажа». Закроем окно ISQL и войдем в режим «Дизайнер».

Откроем проект **sale_project.ipr**.

Добавим на форму **SaleForm** компонент **IBQuery** с палитры **InterBase**.

Придадим в Инспекторе объектов его свойствам значения:

Свойство	**Значение**
Transaction	TraCurrent
DataSource	DsrMaster
SQL	execute procedure sale_item_set_cost(:id)
Name	QrySetItemCost

В обработчик команды **actSave** добавим текст, выделенный жирным шрифтом:

```
procedure TSaleForm.actSaveExecute(Sender: TObject);
begin
 qryTotals.Open;
 qryMaster.Edit;
 qryMaster.FieldByName('TOTAL_L').AsCurrency := qryTotals.Fields[0].AsCurrency;
 qryMaster.FieldByName('TOTAL_R').AsCurrency := qryTotals.Fields[1].AsCurrency;
 qryMaster.FieldByName('TOTAL_S').AsCurrency := qryTotals.Fields[2].AsCurrency;
 qryMaster.Post;
 qryTotals.Close;
 MakeEntries('SALE', qryMaster.FieldByName('ID').AsInteger, traCurrent);
 if qryMaster.FieldByName('HAS_ENTRY').AsInteger = 1 then
 begin
  {Вызываем перерасчет себестоимости}
  qrySetItemCost.ExecSQL;
  {повторно проводим документ}
```

```
    MakeEntries('SALE', qryMaster.FieldByName('ID').AsInteger, traCurrent);
  end;
  traCurrent.CommitRetaining; //подтвердить транзакцию
  Modified := False;
  actSave.Enabled := Modified;
  RefreshExplorer; //пересветить "Проводник по документам"
  RefreshBalance;  //пересветить "Баланс"
end;
```

Реализуем проверку текущего количества товара на складе в документе «Продажа».

При сохранении документа «Продажа», если установлена птичка «Товар отгружен», нужно проверить наличие товара на складе, чтобы убедиться, что не отгружается больше единиц товара, чем это возможно физически. Для этого после первого проведения документа, непосредственно перед расчетом средней стоимости вставим проверку результирующего количества товара на складе. И если это количество стало отрицательным, прервем выполнение команды сохранения, не дойдя до подтверждения транзакции.

Добавим еще один компонент **IBQuery** на форму **SaleForm**.

Придадим в Инспекторе объектов его свойствам значения:

Свойство	**Значение**
Transaction	traCurrent
DataSource	dsrMaster
Name	qryTestQuantity

Дважды щелкнув на свойстве SQL, вызовем редактор SQL-запросов и впишем в него текст:

```
select
  a.object_id,
  o.short_name name,
  sum(a.quantity_debit - a.quantity_credit) quantity
from
  acc_turn a, object_names o
where
  a.acc_id = :stock_acc and
  a.op_date <= :entry_date and
  a.object_id in (select goods from sale_item where id = :id) and
  a.object_id = o.object_id
group by
  a.object_id,
  o.short_name
having
  sum(a.quantity_debit - a.quantity_credit) < 0
```

Обратим внимание, что запрос параметризованный. Параметры **stock_acc, entry_date** и **id** перед открытием запроса компонент **qryTestQuantity** получит из соответствующих полей компонента **qryMaster**, благодаря тому, что свойству **DataSource** компонента **qryTestQuantity** назначен источник **dsrMaster**.

Осталось добавить активизацию этого запроса и проверку результата в обработчик команды **actSave**. Добавим объявление новой функции в секцию **private** класса формы (показано жирным шрифтом):

```
private
   { Private declarations }
   procedure UpdatePrice(PriceQuery: TDataSet);
   function Save: boolean;
  public
   { Public declarations }
  end;
```

Добавим реализацию этой функции в раздел **implementation** модуля:

```
procedure TSaleForm.Save: boolean;
begin
end;
```

Вырежем весь текст из обработчика **actSave**, вставим его в тело функции и добавим ряд операторов (добавленные операторы выделены жирным шрифтом):

```
procedure TSaleForm.Save: boolean;
begin
  Result := False;
  qryTotals.Open;
  qryMaster.Edit;
  qryMaster.FieldByName('TOTAL_L').AsCurrency := qryTotals.Fields[0].AsCurrency;
  qryMaster.FieldByName('TOTAL_R').AsCurrency := qryTotals.Fields[1].AsCurrency;
  qryMaster.FieldByName('TOTAL_S').AsCurrency := qryTotals.Fields[2].AsCurrency;
  qryMaster.Post;
  qryTotals.Close;
  MakeEntries('SALE', qryMaster.FieldByName('ID').AsInteger, traCurrent);
  if qryMaster.FieldByName('HAS_ENTRY').AsInteger = 1 then
  begin
    with qryTestQuantity do
    begin
      Open; //открываем запрос количеств
      if not qryTestQuantity.IsEmpty then //если результирующий набор не пуст
      begin
        {находим позицию в документе}
        qryDetail.Locate('GOODS', FieldByName('OBJECT_ID').AsInteger, MkSet());
        dbgDetail.SelectedField := qryDetail.FieldByName('QUANTITY');
        {выдаем сообщение}
        MessageDlg('Недостаточно товара на складе:'#13+
          FieldByName('NAME').AsString, mtError, MkSet(mbOK), 0);
        Close; //закрываем запрос количеств
        exit; //выходим из процедуры сохранения
      end
      else
        Close; //закрываем запрос количеств
    end;
    {Вызываем перерасчет себестоимости}
    qrySetItemCost.ExecSQL;
    {повторно проводим документ}
    MakeEntries('SALE', qryMaster.FieldByName('ID').AsInteger, traCurrent);
  end;
  traCurrent.CommitRetaining; //подтвердить транзакцию
  Modified := False;
  actSave.Enabled := Modified;
  RefreshExplorer; //пересветить "Проводник по документам"
  RefreshBalance;  //пересветить "Баланс"
  Result := True;
end;
```

Обратим внимание, что вначале мы присваиваем результату, который должна вернуть эта функция, значение **False**, а в самом конце – присваиваем **True**. Поэтому функция вернет **True** только в том случае, если функция сохранения довела свою работу до конца. Если количество какого-то товара на складе, входящего в данный документ, стало отрицательным – вызовется команда **exit**, которая прервет выполнение функции и та вернет значение **False**.

В обработчик **actSave** вместо прежнего текста теперь вставим вызов этой функции:

```
procedure TSaleForm.actSaveExecute(Sender: TObject);
begin
```

```
  Save;
end;
```

Изменим также обработчик события формы **OnCloseQuery**: заменим строку

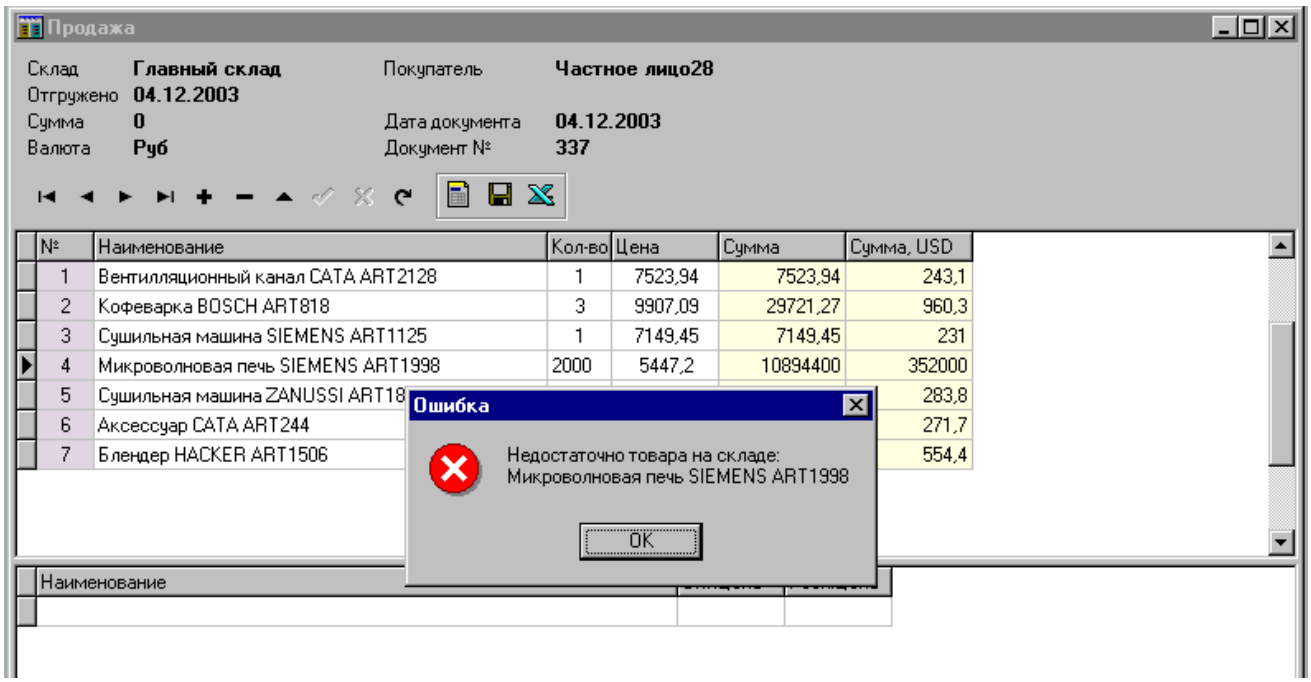
```
  mrYes:  actSaveExecute(nil); //вызов обработчика OnExecute команды actSave
```

на вызов функции Save:

```
procedure TSaleForm.FormCloseQuery(Sender: TObject; var CanClose: Boolean);
begin
  if Modified then
  case MessageDlg(
     'Сохранить изменения в документе ?', mtConfirmation,
     mkSet(mbYes, mbNo, mbCancel), 0) of
  mrYes:  CanClose := Save; //вызов функции Save
  mrCancel: Canclose := False;
  end;
end;
```

При попытке закрыть окно, если документ редактировался, возникает предложение сохранить изменения. Если пользователь ответил «Да», но при сохранении выяснилось, что товара на складе недостаточно, то сохранение не произойдет и окно не закроется, так как переменной **CanClose** в обработчике **OnCloseQuery** присвоится значение **False**.

Установим в переменных контекста какой-нибудь документ продаж. Запустим проект и попробуем значительно увеличить количество отгружаемого товара. При попытке сохранить изменения мы получим сообщение «Недостаточно товара на складе» и документ не будет сохранен:



Выйдем из режима дизайнера и создадим архивную копию базы (меню **Инструменты/Архивация базы**). Итак, мы полностью закончили интерфейс ввода документа «Продажа». Нам осталось лишь реализовать печать счетов-фактур и накладных.

Печать счетов-фактур и накладных

Нам потребуется печатать счет-фактуру и накладную. Причем счет-фактуру желательно получать как в валюте документа, так и в рублях. Вместо печати в Allegro осуществляется экспорт в Microsoft Office. Сама печать производится уже из приложения Microsoft Office. В данном случае мы используем Microsoft Excel.

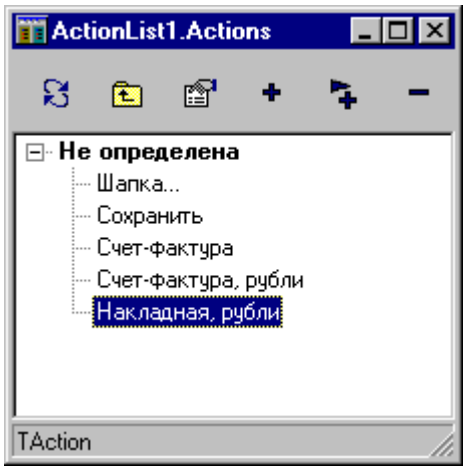
Так как мы создали интерфейс документа «Продажа» путем копирования интерфейса «Поступление на склад», пока что у нас присутствует старый отчет, вызываемый командой **actReport** «Отчет».

Войдем в режим дизайнера, откроем проект **sale_project.ipr** и дважды щелкнем на компоненте **ActionList1**.

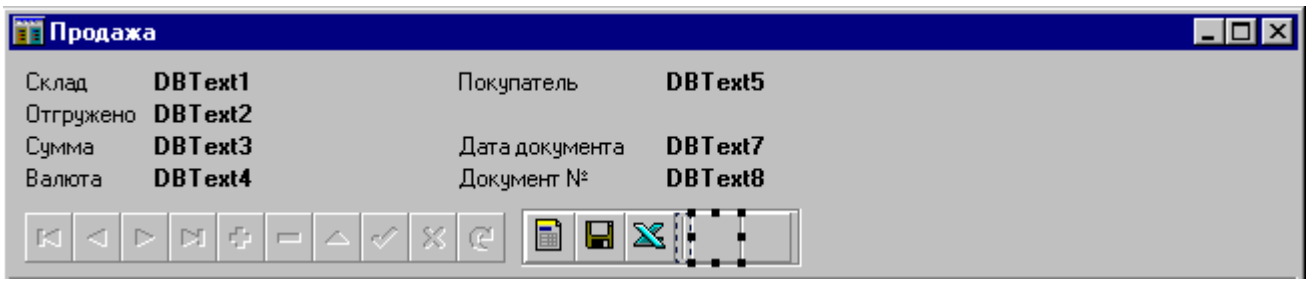
Переименуем **actReport** в **actReport1**, изменив значение свойств **Caption** и **Hint** с «Отчет» на «Счет-фактура».

Добавим еще 2 команды, назначая им свойства в Инспекторе объектов:

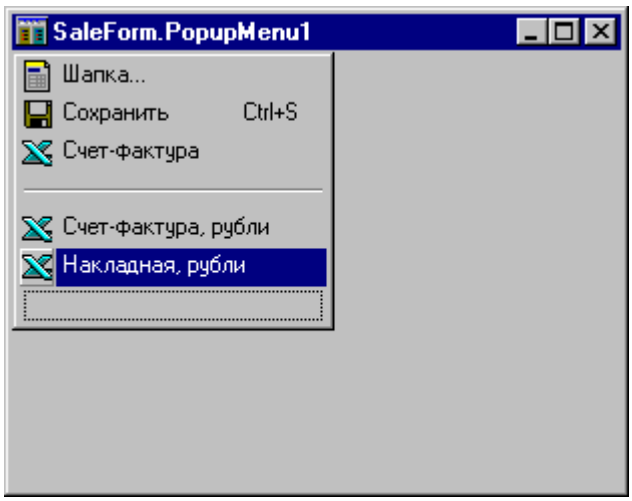
Name	Caption	Hint	ImageIndex
actReport2	Счет-фактура, рубли	Счет-фактура, рубли	2
actReport3	Накладная, рубли	Накладная, рубли	2



Увеличим ширину **Width** компонента **ToolBar1** до 140 и с помощью контекстного меню этого компонента добавим разделитель и две кнопки. Назначим добавленным кнопкам в свойстве **Action** команду **actReport2** и **actReport3**:



Дважды щелкнем на компоненте **PopupMenu1**, выберем последний (пустой) пункт и добавим клавишей Insert три новых пустых пункта. В свойство **Caption** первого запишем дефис «-», а свойствам **Action** двух следующих назначим команды **actReport2** и **actReport3**:



Дважды щелкнем на компоненте **ActionList1** и, последовательно выбирая команды **actReport2** и **actReport3**, создадим два отдельных обработчика события **OnExecute**. Впишем в них такой текст:

```
procedure TSaleForm.actReport2Execute(Sender: TObject);
begin
  XLReport2.Report;
end;

procedure TSaleForm.actReport3Execute(Sender: TObject);
begin
  XLReport3.Report;
end;
```

Для получения реквизитов фирм нам понадобится еще один SQL-запрос. Добавим на форму компонент **IBQuery** с палитры **InterBase** и придадим ему свойства в Инспекторе объектов:

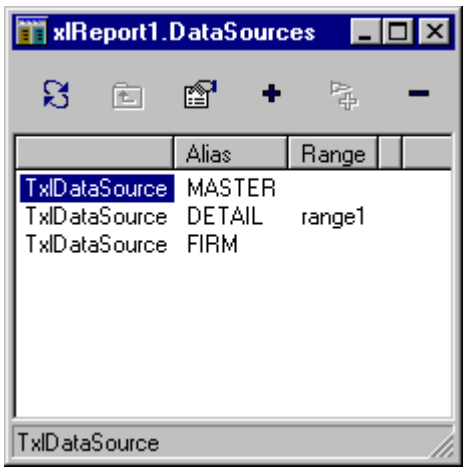
Свойство	**Значение**
Name	qryFirm
Transaction	traCurrent
DataSource	dsrMaster
SQL	select * from FIRM where ID = :CONTRAGENT

Дважды щелкнем на компоненте и добавим все поля в список «постоянных полей».

В обработчик события OnCreate формы SaleForm запишем:

```
  qryFirm.Open;
```

Выберем на форме компонент **XLReport1** и дважды щелкнем в Инспекторе на свойстве DataSources. В появившемся редакторе источников данных добавим новый источник:



Зададим ему свойства:

Свойство	Значение
Alias	FIRM
DataSet	qryFirm

Закроем редактор источников данных. У компонента **XLReport1** очистим свойство **XLSTemplate**. Теперь скопируем компонент в буфер обмена Windows с помощью клавиш ***Ctrl+C***. Дважды вставим компонент из буфера обмена Windows, нажимая комбинацию клавиш ***Ctrl+V***. На форму добавятся два новых компонента **XLReport**. Так как компонент **XLReport1** уже был настроен для работы с источниками данных **qryMaster**, **qryDetail**, **qryFirm**, то для того чтобы использовать те же настройки во всех отчетах, мы просто «клонировали» его, а не добавляли компоненты с палитры:

Сохраним все изменения.

Нам осталось создать файлы шаблонов в Microsoft Excel и вписать их имена в свойство **XSLTemplate** каждого из компонентов **XLReport**. Вы можете разработать шаблоны документов самостоятельно или воспользоваться теми файлами, что входят в пример конфигурации TechnoTrade, который доступен на нашем сайте http://www.gaapinvest.com

Файл шаблона счета-фактуры должен иметь название **facture.xls** и находиться в папке проекта **TechnoTrade**. Если Вы сами разрабатываете файл шаблона, то не забудьте, что тиражируемый регион ячеек должен иметь название range1. Желательно также определить заголовки колонок как сквозные строки, чтобы при печати многостраничного документа на каждой странице выводились заголовки столбцов.

Выберем компонент **XLReport1** и в свойстве **XLSTemplate** укажем имя файла **facture.xls**. Путь к файлу указывать не надо, компонент сам будет искать его в директории проекта. Теперь для того чтобы вызвать шаблон на редактирование, достаточно дважды щелкнуть на компоненте **XLReport1**.

В соответствующих ячейках шаблона укажем названия, состоящие из знака равенства, псевдонима источника данных (как он определен у нас в свойстве Alias) и имени поля, разделенных знаком подчеркивания:

Шапка	
Счет-фактура №	=MASTER_DOC_NO
от	=MASTER_DOC_DATE
Грузополучатель и его адрес	=FIRM_FULL_NAME
Покупатель	=FIRM_FULL_NAME
Адрес	=FIRM_JUR_ADDRESS
Идентификационный номер покупателя (ИНН)	=FIRM_INN
Позиции	
№ п/п	=DETAIL_NUM
Наименование товара	=DETAIL_ITEM_NAME
Кол-во	=DETAIL_QUANTITY

Цена за единицу измерения	=DETAIL_PRICE_L_WO_VAT
Стоимость товаров (услуг), всего без налога	=DETAIL_AMOUNT_L_WO_VAT
Налоговая ставка	=MASTER_VAT_RATE
Сумма налога	=DETAIL_VAT_L
Стоимость товаров (услуг) всего, с учетом налога	=DETAIL_AMOUNT_L

Сохраним все изменения в шаблоне и закроем Excel.

Выберем в окне **Запуск/Переменные контекста** любой документ типа «Продажа».

Запустим проект и попробуем вывести счет-фактуру, нажав на первую кнопку со значком Excel. Взглянем на получившийся документ. Мы видим, что в нем пока не выводятся поля «Стоимость товаров (услуг), всего без налога» и «Сумма налога».

Закроем Excel, отказавшись от сохранения файла.

Добавим недостающие поля в компонент запроса позиций **qryDetail**. Для этого дважды щелкнем на нем. В появившемся редакторе полей добавим (нажимая кнопку с плюсом) 4 поля типа **IBBCD**:

Название поля	**Тип поля**
AMOUNT_L_WO_VAT	IBBCD
VAT_L	IBBCD
AMOUNT_R_WO_VAT	IBBCD
VAT_R	IBBCD

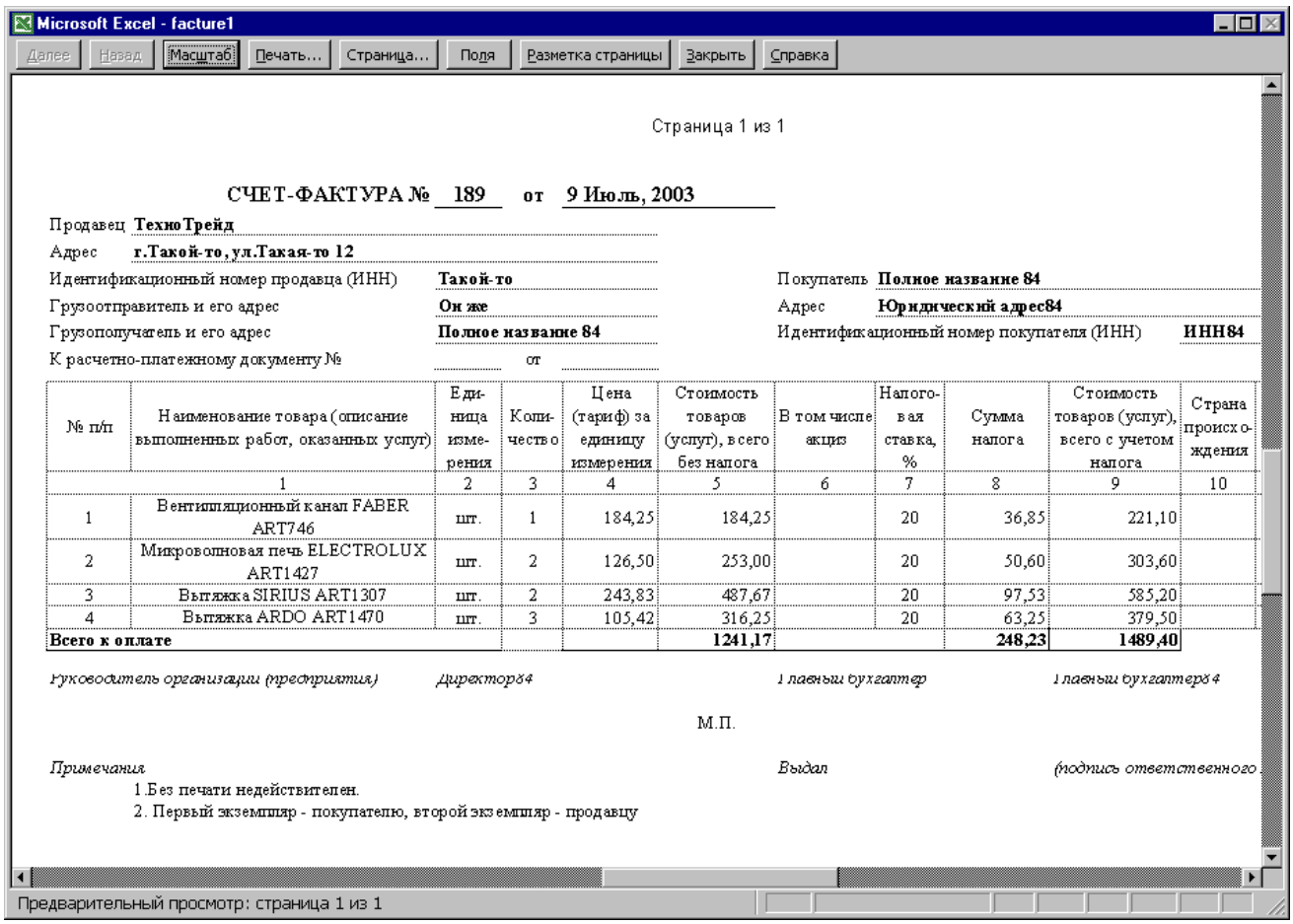
Выберем все 4 добавленные поля в редакторе полей и назначим им в Инспекторе объектов свойство **FieldKind** = **fkCalculated**. Закроем редактор полей. В обработчик события **OnCalcFields** компонента **qryDetail** добавим команды (выделено жирным):

```
procedure TSaleForm.qryDetailCalcFields(DataSet: TDataSet);
begin
  with DataSet do
  begin
    FieldByName('NUM').AsInteger := RecNo;
    FieldByName('AMOUNT_L_WO_VAT').AsCurrency :=
      FieldByName('PRICE_L_WO_VAT').AsCurrency *
      FieldByName('QUANTITY').AsCurrency;
    FieldByName('VAT_L').AsCurrency :=
      FieldByName('AMOUNT_L').AsCurrency -
      FieldByName('AMOUNT_L_WO_VAT').AsCurrency;
    FieldByName('AMOUNT_R_WO_VAT').AsCurrency :=
      FieldByName('PRICE_R_WO_VAT').AsCurrency *
      FieldByName('QUANTITY').AsCurrency;
    FieldByName('VAT_R').AsCurrency :=
      FieldByName('AMOUNT_R').AsCurrency -
      FieldByName('AMOUNT_R_WO_VAT').AsCurrency;
  end;
end;
```

Мы вычисляем сумму без НДС, просто умножая «цену без НДС» на «количество», а «сумму НДС» получаем, вычитая «сумму без НДС» из «суммы с НДС».

Запустим проект и попробуем вывести счет-фактуру еще раз.

Вызовем предварительный просмотр печати:



В данном случае мы получили печать счета-фактуры в валюте документа. Для того, чтобы получить такой же документ в рублях, закроем Excel, создадим копию файла шаблона под именем **facture_r.xls**. Запишем имя этого файла в свойство **XLSTemplate** компонента **XLReport2** и дважды щелкнем на компоненте, чтобы вызвать шаблон на редактирование.

В шаблоне заменим выражения, относящиеся к ценам и суммам в валюте документа:

	Заменить на
Цена за единицу измерения	=DETAIL_PRICE_R_WO_VAT
Стоимость товаров (услуг), всего без налога	=DETAIL_AMOUNT_R_WO_VAT
Сумма налога	=DETAIL_VAT_R
Стоимость товаров (услуг) всего, с учетом налога	=DETAIL_AMOUNT_R

Фактически, нам пришлось лишь заменить букву L на букву R. Сохраним шаблон и закроем Excel.

Запустим проект и нажмем вторую кнопку со значком Excel.

Итак, вывод счета-фактуры в рублях мы тоже закончили. Нам осталось реализовать накладные. Закроем Excel, отказавшись от сохранения файла и вернемся в режим дизайна.

В качестве накладной в этом примере мы выберем достаточно мудреную накладную, чтобы показать еще ряд нюансов - использование параметров в компоненте XLReport и формирование суммы прописью.

Заготовим шаблон товарной накладной (форма по ОКУД 330212) или возьмем его из примера конфигурации. Скопируем файл шаблона в каталог проектов и назовем его **nakl.xls**. Запишем это название в свойство **XLSTemplate** компонента **XLReport3** и дважды щелкнем на компоненте, чтобы вызвать шаблон на редактирование. В полях шаблона запишем выражения, как мы это делали в шаблоне счета-фактуры:

Шапка	
Товарная накладная №	=MASTER_DOC_NO
Дата составления	=MASTER_DOC_DATE
Грузополучатель	**=FIRM_FULL_NAME_AND_ADDRESS**
Плательщик	=FIRM_FULL_NAME

Адрес	=FIRM_JUR_ADDRESS
Идентификационный номер покупателя (ИНН)	=FIRM_INN
Позиции	
№ п/п	=DETAIL_NUM
Наименование товара	=DETAIL_ITEM_NAME
Кол-во	=DETAIL_QUANTITY
Цена за единицу измерения	=DETAIL_PRICE_R_WO_VAT
Сумма без учета НДС	=DETAIL_AMOUNT_R_WO_VAT
Ставка НДС	=MASTER_VAT_RATE
Сумма НДС	=DETAIL_VAT_R
Сумма с учетом НДС	=DETAIL_AMOUNT_R
Подвал	
Товарная накладная имеет приложение на и содержит	**=XLRParams_NUM_IN_WORDS**
Всего отпущено на сумму:	**=XLRParams_AMOUNT_IN_WORDS**

В таблице выделены жирным шрифтом три поля шаблона, для которых пока у нас нет соответствующих источников в проекте. Поле **FULL_NAME_AND_ADDRESS** просто организовать, добавив в компонент **qryFirm** вычисляемое поле типа **IBString** и задав ему свойства:

Свойство	**Значение**
FiledKind	FkCalculated
Size	250
FiledName	FULL_NAME_AND_ADDRESS

Для того чтобы это поле получало значения, нужно создать обработчик события **OnCalcFields** компонента **qryDetail**:

```
procedure TSaleForm.qryFirmCalcFields(DataSet: TDataSet);
begin
  with DataSet do
  FieldByName('FULL_NAME_AND_ADDRESS').AsString :=
    Format('%s, ИНН %s, %s',[FieldByName('FULL_NAME').AsString,
      FieldByName('INN').AsString, FieldByName('JUR_ADDRESS').AsString]);
end;
```

В подвале шаблона у нас имеются два поля, в которых нужно выводить числа прописью. Это общее количество позиций и сумма документа. Для того чтобы организовать превращение чисел в такого рода текст, нам понадобится компонент с палитры **Allegro**, умеющий это делать.

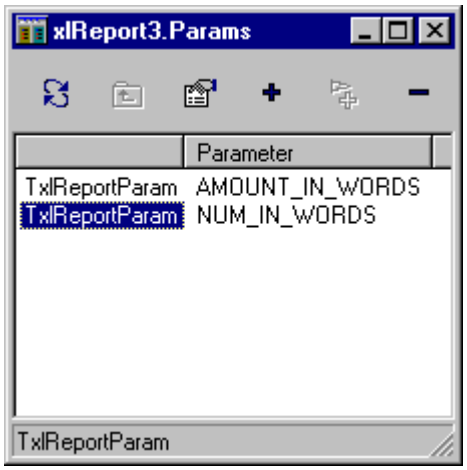
Добавим на форму **SaleForm** два компонента **CurrencyInWords** с палитры **Allegro**.

Придадим им свойства:

Свойство	**1-й компонент**	**2-й компонент**
Cent1	копейка	
Cent2_4	копейки	
Cent5_20	копеек	
Cur1	рубль	
Cur2_4	рубля	
Cur5_20	рублей	
Name	ciwAmount	ciwNum
UseCents	True	False
UseFinalPoint	True	False

Компонент **CurrencyInWords** работает просто: во время выполнения программы его свойству **Value** следует присвоить значение, а затем прочитать значение из свойства **Text**. Способ преобразования задается в свойствах, где мы устанавливаем способы склонения центов и валют, а также род центов и род валюты. Чтобы правильно задать свойство, нужно задаться вопросом такого рода: 2..4 чего? – рубля. Следовательно, Cur2_4 = рубля, 5..20 чего? – рублей, следовательно Cur5_20 = рублей.

Выберем компонент **XLReport3** и Инспекторе объектов дважды щелкнем на его свойстве **Params**. Появится редактор параметров, в котором добавим два параметра с именами: **NUM_IN_WORDS** и **AMOUNT_IN_WORDS**:

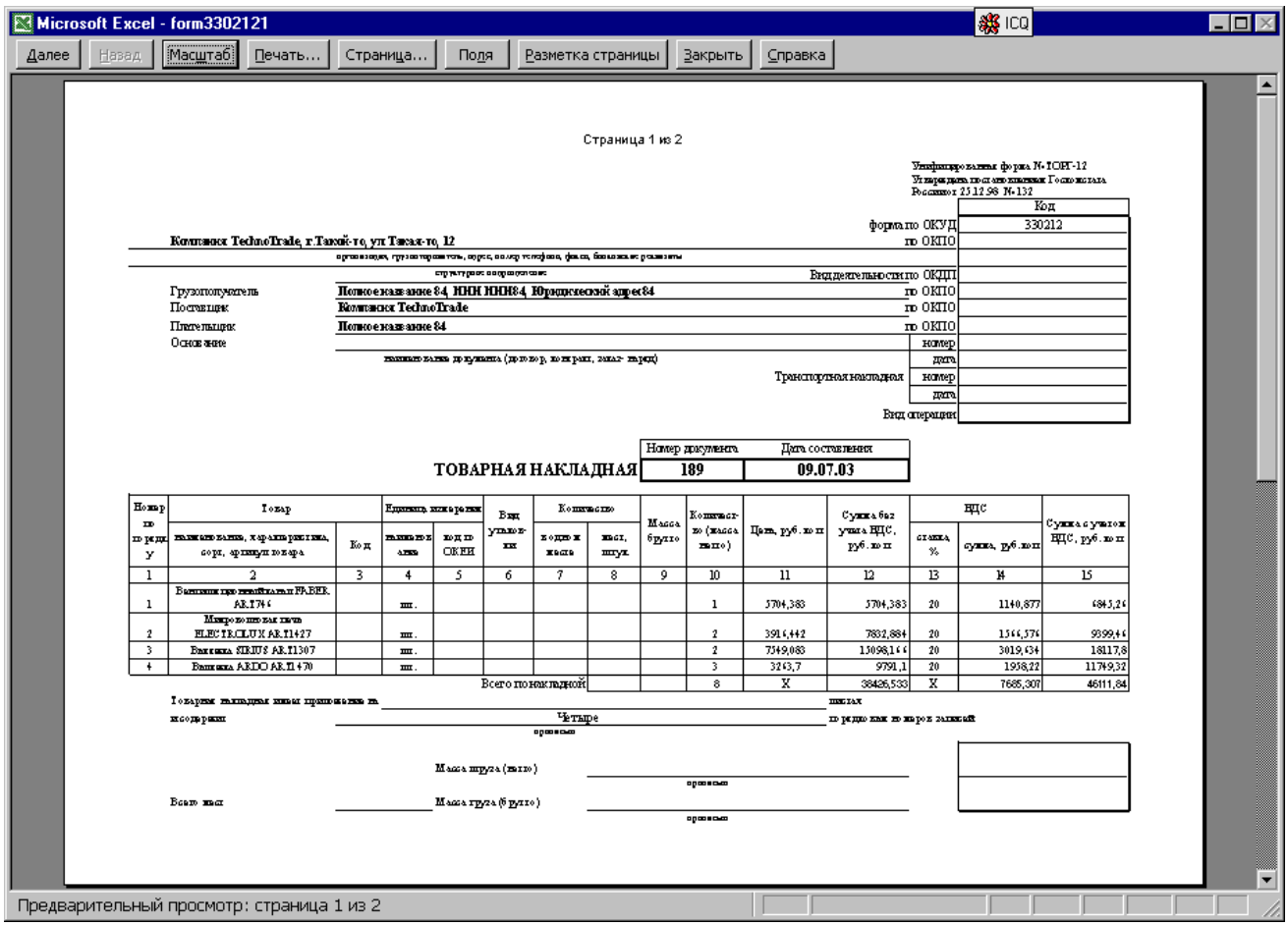


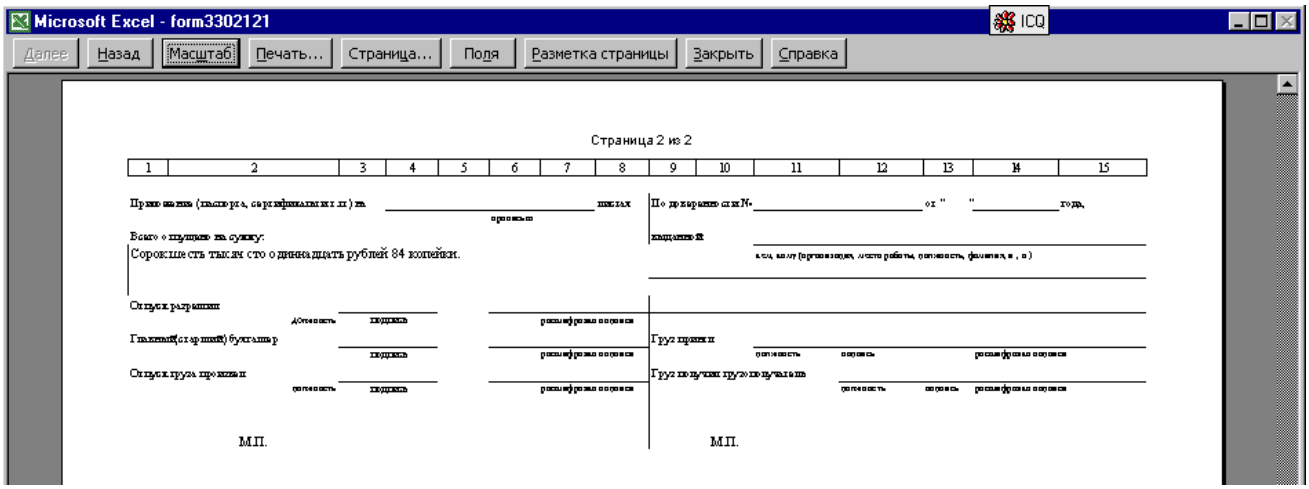
Теперь можно обращаться к этим параметрам во время выполнения программы методом **ParamByName**. В обработчик события **OnExecute** команды **actReport3** впишем текст, показанный жирным шрифтом:

```
procedure TSaleForm.actReport3Execute(Sender: TObject);
begin
  with qryDetail do
  begin
    FetchAll;
    ciwAmount.Value := qryMaster.FieldByName('TOTAL_R').AsCurrency;
    ciwNum.Value := RecordCount;
  end;
  XLReport3.ParamByName('AMOUNT_IN_WORDS').Value := ciwAmount.Text;
  XLReport3.ParamByName('NUM_IN_WORDS').Value := ciwNum.Text;
  XLReport3.Report;
end;
```

При построении отчета XLReport, найдя в шаблоне поля, начинающиеся с префикса **XLRParams_**, выведет значения параметров в соответствующие ячейки отчета.

Запустим проект и выведем накладную, нажав третью кнопку. На экране возникнет документ Excel. Вызовем предварительный просмотр печати в Excel. Накладная, скорее всего, занимает два листа:





Общее количество позиций в накладной и сумма документа отображаются прописью.
На этом мы закончили интерфейс документа «Продажа».
В заключение приведем полный листинг проекта.

Полный листинг проекта «Продажа»

```
unit sale;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  ExtCtrls, ComCtrls, Menus,DCMenuEditor,DBGridA, DBGrids, IBCustomDataSet,
  IBDatabase, DB,StdCtrls, DBCtrls,ActnList,AlgCtrls,IBQuery,xlReport,CurrInW;

type
  TSaleForm = class(TForm)
    TopPanel: TPanel;
    dbgDetail: TDBGridA;
    Splitter1: TSplitter;
    dbgGoods: TDBGrid;
    qryMaster: TIBDataSet;
    qryDetail: TIBDataSet;
    traCurrent: TIBTransaction;
    dsrMaster: TDataSource;
    dsrDetail: TDataSource;
    qryDetailID1: TIntegerField;
    qryDetailN1: TIntegerField;
    qryDetailGOODS1: TIntegerField;
    qryDetailITEM_NAME1: TIBStringField;
    qryDetailQUANTITY1: TIntegerField;
    qryDetailPRICE_L1: TIBBCDField;
    qryDetailPRICE_L_WO_VAT1: TIBBCDField;
    qryDetailPRICE_R1: TIBBCDField;
    qryDetailPRICE_R_WO_VAT1: TIBBCDField;
    qryDetailAMOUNT_L1: TIBBCDField;
    qryDetailAMOUNT_R1: TIBBCDField;
    qryDetailAMOUNT_S1: TIBBCDField;
    qryMasterID1: TIntegerField;
    qryMasterDIR_ID1: TIntegerField;
    qryMasterDOC_DATE1: TDateField;
    qryMasterDOC_NO1: TIBStringField;
    qryMasterENTRY_DATE1: TDateField;
    qryMasterHAS_ENTRY1: TSmallintField;
    qryMasterSTOCK_ACC1: TIntegerField;
    qryMasterLAYER_ID1: TIntegerField;
    qryMasterLAYER_NAME1: TIBStringField;
    qryMasterVAT_RATE1: TIBBCDField;
    qryMasterCONTRAGENT1: TIntegerField;
    qryMasterCONTRAGENT_NAME1: TIBStringField;
    qryMasterCALC_MODE1: TIntegerField;
    qryMasterTOTAL_L1: TIBBCDField;
    qryMasterTOTAL_R1: TIBBCDField;
    qryMasterTOTAL_S1: TIBBCDField;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    Label4: TLabel;
    DBText1: TDBText;
    DBText2: TDBText;
    DBText3: TDBText;
    DBText4: TDBText;
    Label5: TLabel;
    Label7: TLabel;
```

Label8: TLabel;
DBText5: TDBText;
DBText7: TDBText;
DBText8: TDBText;
qryMasterSTOCK_ACC_NAME1: TIBStringField;
PopupMenu1: TPopupMenu;
ImageList1: TImageList;
qryMasterEXCH_RATE_L1: TFloatField;
qryMasterEXCH_RATE_S1: TFloatField;
Timer1: TTimer;
ActionList1: TActionList;
actHeader: TAction;
actSave: TAction;
actReport1: TAction;
N1: TMenuItem;
N2: TMenuItem;
N3: TMenuItem;
DBNavigator1: TDBNavigator;
ToolBar1: TToolBar;
ToolButton1: TToolButton;
ToolButton2: TToolButton;
ToolButton3: TToolButton;
qryDetailNUM1: TIntegerField;
RefDialog1: TRefDialog;
qryGoods: TIBQuery;
qryGoodsOBJECT_ID1: TIntegerField;
qryGoodsSHORT_NAME1: TIBStringField;
dsrGoods: TDataSource;
qryTotals: TIBQuery;
xlReport1: TxlReport;
qryGoodsPRICE_W1: TIBBCDField;
qryGoodsPRICE_R1: TIBBCDField;
qrySetItemCost: TIBQuery;
qryTestQuantity: TIBQuery;
qryMasterPRICE_TYPE1: TIntegerField;
qryMasterPRICE_DISCOUNT1: TIBBCDField;
actReport2: TAction;
actReport3: TAction;
qryFirm: TIBQuery;
xlReport3: TxlReport;
xlReport2: TxlReport;
qryFirmID1: TIntegerField;
qryFirmDISCOUNT1: TIBBCDField;
qryFirmDEFERMENT1: TIntegerField;
qryFirmIS_OUR1: TSmallintField;
qryFirmFULL_NAME1: TIBStringField;
qryFirmR_ACCOUNT1: TIBStringField;
qryFirmINN1: TIBStringField;
qryFirmBANK1: TIBStringField;
qryFirmCITY1: TIBStringField;
qryFirmC_ACCOUNT1: TIBStringField;
qryFirmBIK1: TIBStringField;
qryFirmOKONX1: TIBStringField;
qryFirmOKPO1: TIBStringField;
qryFirmKPP1: TIBStringField;
qryFirmJUR_ADDRESS1: TIBStringField;
qryFirmREG_NO1: TIBStringField;
qryFirmREG_DATE1: TDateField;
qryFirmMANAGER1: TIBStringField;
qryFirmBOOKKEEPER1: TIBStringField;

```
    qryDetailAMOUNT_L_WO_VAT1: TIBBCDField;
    qryDetailAMOUNT_R_WO_VAT1: TIBBCDField;
    qryDetailVAT_R1: TIBBCDField;
    qryDetailVAT_L1: TIBBCDField;
    qryFirmFULL_NAME_AND_ADDRESS1: TIBStringField;
    ciwAmount: TCurrencyInWords;
    ciwNum: TCurrencyInWords;
    ToolButton4: TToolButton;
    ToolButton5: TToolButton;
    ToolButton6: TToolButton;
    N4: TMenuItem;
    N5: TMenuItem;
    N6: TMenuItem;
    procedure FormCreate(Sender: TObject);
    procedure actHeaderExecute(Sender: TObject);
    procedure actSaveExecute(Sender: TObject);
    procedure FormCloseQuery(Sender: TObject; var CanClose: Boolean);
    procedure qryMasterAfterPost(DataSet: TDataSet);
    procedure Timer1Timer(Sender: TObject);
    procedure qryDetailAfterInsert(DataSet: TDataSet);
    procedure qryDetailCalcFields(DataSet: TDataSet);
    procedure qryDetailAfterDelete(DataSet: TDataSet);
    procedure dbgDetailEditButtonClick(Sender: TObject);
    procedure dsrDetailDataChange(Sender: TObject; Field: TField);
    procedure qryDetailBeforePost(DataSet: TDataSet);
    procedure dsrMasterDataChange(Sender: TObject; Field: TField);
    procedure dbgDetailSetEditText(Sender: TObject; ACol, ARow: Integer; const Value: String);
    procedure dbgDetailKeyPress(Sender: TObject; var Key: Char);
    procedure dbgGoodsDblClick(Sender: TObject);
    procedure dbgGoodsKeyPress(Sender: TObject; var Key: Char);
    procedure actReport1Execute(Sender: TObject);
    procedure xlReport1Progress(Report: TxlReport; const Position, Max: Integer);
    procedure actReport2Execute(Sender: TObject);
    procedure actReport3Execute(Sender: TObject);
    procedure qryFirmCalcFields(DataSet: TDataSet);
  private
    { Private declarations }
    procedure UpdatePrice(PriceQuery: TDataSet);
    function Save: boolean;
  public
    { Public declarations }
  end;

var
  SaleForm: TSaleForm;

implementation
uses
  sale_header;


{$R *.DFM}

var
  Modified: boolean;
  LastValue: string;

procedure TSaleForm.FormCreate(Sender: TObject);
begin
  traCurrent.StartTransaction; //стартовать транзакцию
```

```
  Modified := False;
  actSave.Enabled := Modified;
  qryMaster.ParamByName('ID').AsInteger := RunContext.Documents[0].doc_id;
  qryMaster.Open;

  if RunContext.Documents[0].doc_id <=0 then
  begin
    qryMaster.Insert; {Начальные значения полей нового документа}
    qryMaster.FieldByName('DIR_ID').AsInteger := RunContext.dir_id;
    qryMaster.FieldByName('CALC_MODE').AsInteger := 1;
    qryMaster.FieldByName('DOC_DATE').AsDateTime := Date;
    qryMaster.FieldByName('HAS_ENTRY').AsInteger := 1;
    qryMaster.FieldByName('ENTRY_DATE').AsDateTime := Date;
    qryMaster.FieldByName('VAT_RATE').AsCurrency := 20;
    qryMaster.FieldByName('TOTAL_L').AsCurrency := 0;
    qryMaster.FieldByName('TOTAL_S').AsCurrency := 0;
    qryMaster.FieldByName('TOTAL_R').AsCurrency := 0;
    qryMaster.FieldByName('CONTRAGENT').AsInteger := 0;
    qryMaster.FieldByName('STOCK_ACC').AsInteger := 1007; //Главный склад
    qryMaster.FieldByName('PRICE_TYPE').AsInteger := 0;
    qryMaster.FieldByName('PRICE_DISCOUNT').AsCurrency := 0;
    if SaleHeaderForm.ShowModal <> mrOK then
      Timer1.Enabled := True;
  end;

  if Timer1.Enabled then
    exit;

  self.Caption := NameOfDocument('SALE', qryMaster.FieldByName('ID').AsInteger,
     traCurrent); //устанавливаем новый заголовок формы
  qryDetail.ParamByName('ID').AsInteger := RunContext.Documents[0].doc_id;
  qryDetail.Open;
  qryFirm.Open;
end;

procedure TSaleForm.actHeaderExecute(Sender: TObject);
begin
  if SaleHeaderForm.ShowModal = mrOK then
    self.Caption := NameOfDocument('SALE', qryMaster.FieldByName('ID').AsInteger,
       traCurrent); //устанавливаем новый заголовок формы
end;

procedure TSaleForm.actSaveExecute(Sender: TObject);
begin
  Save;
end;

procedure TSaleForm.FormCloseQuery(Sender: TObject; var CanClose: Boolean);
begin
  if Modified then
  case MessageDlg(
      'Сохранить изменения в документе ?', mtConfirmation,
      mkSet(mbYes, mbNo, mbCancel), 0) of
  mrYes:  CanClose := Save; //вызов функции Save
  mrCancel: Canclose := False;
  end;
end;
```

```
procedure TSaleForm.qryMasterAfterPost(DataSet: TDataSet);
begin
  Modified := True;
  actSave.Enabled := Modified;
end;

procedure TSaleForm.Timer1Timer(Sender: TObject);
begin
  Timer1.Enabled := False;
  Modified := False;
  actSave.Enabled := Modified;
  Self.Close;
end;

var
  lock_insert: boolean;

procedure TSaleForm.qryDetailAfterInsert(DataSet: TDataSet);
begin
  if not lock_insert then
  begin
    lock_insert := True;
    qryDetail.Cancel; //отменяем режим вставки
    qryDetail.Append; //вставляем в конец набора
    LastValue := '';
    dbgDetail.SelectedField := qryDetail.FieldByName('ITEM_NAME');
                  //перемещаем фокус ввода в поле наименования
  end;
  lock_insert := False;
end;

procedure TSaleForm.qryDetailCalcFields(DataSet: TDataSet);
begin
  with DataSet do
  begin
    FieldByName('NUM').AsInteger := RecNo;
    FieldByName('AMOUNT_L_WO_VAT').AsCurrency :=
      FieldByName('PRICE_L_WO_VAT').AsCurrency *
      FieldByName('QUANTITY').AsCurrency;
    FieldByName('VAT_L').AsCurrency :=
      FieldByName('AMOUNT_L').AsCurrency -
      FieldByName('AMOUNT_L_WO_VAT').AsCurrency;
    FieldByName('AMOUNT_R_WO_VAT').AsCurrency :=
      FieldByName('PRICE_R_WO_VAT').AsCurrency *
      FieldByName('QUANTITY').AsCurrency;
    FieldByName('VAT_R').AsCurrency :=
      FieldByName('AMOUNT_R').AsCurrency -
      FieldByName('AMOUNT_R_WO_VAT').AsCurrency;
  end;
end;

procedure TSaleForm.qryDetailAfterDelete(DataSet: TDataSet);
begin
  Modified := True;
  DataSet.Close;
  DataSet.Open;
  actSave.Enabled := Modified;
end;
```

```
procedure TSaleForm.dbgDetailEditButtonClick(Sender: TObject);
var
  Q: TIBQuery;
begin
  with RefDialog1 do
  if Execute then //вызываем диалог на экран и проверяем, был ли выбран товар
  begin
    qryDetail.Edit; //переводим набор данных в режим редактирования

    {присваиваем значение ID товара полю GOODS набора}
    qryDetail.FieldByName('GOODS').AsInteger := Object_ID;

    {создаем run-time компонент запроса и запрашиваем цену товара}
    Q := TIBQuery.Create(nil);
    try
      Q.Transaction := self.traCurrent;
      Q.SQL.Text := 'SELECT PRICE_W, PRICE_R FROM GOODS WHERE ID = :ID';
      Q.ParamByName('ID').AsInteger := Object_ID;
      Q.Open;
      UpdatePrice(Q);
    finally
      Q.Free;
    end;
    {присваиваем значение краткое наименование товара полю ITEM_NAME набора}
    qryDetail.FieldByName('ITEM_NAME').AsString :=
            NameOfRefObject(Object_ID, 'SHORT_NAME');

    {перемещаем фокус ввода в сетке в поле "Количество"}
    dbgDetail.SelectedField := qryDetail.FieldByName('QUANTITY');
  end;
end;

procedure TSaleForm.dsrDetailDataChange(Sender: TObject; Field: TField);
begin
  if (Field <> nil) and
     ((Field.FieldName = 'PRICE_L') or
      (Field.FieldName = 'PRICE_L_WO_VAT') or
      (Field.FieldName = 'QUANTITY'))then
  with qryDetail do
  begin
    {Если режим расчета сумм на основе цен с НДС
    и изменено поле "Цена с НДС"  или "Количество"}
    if ((Field.FieldName = 'PRICE_L')  or (Field.FieldName = 'QUANTITY')) and
       (qryMaster.FieldByName('CALC_MODE').AsInteger = 1) then
    begin

      FieldByName('AMOUNT_L').AsCurrency :=
        FieldByName('PRICE_L').AsCurrency *
        FieldByName('QUANTITY').AsInteger;
      FieldByName('PRICE_L_WO_VAT').AsCurrency :=
          round(FieldByName('PRICE_L').AsCurrency * 1000 * 100/
          (qryMaster.FieldByName('VAT_RATE').AsCurrency + 100))/1000;

      FieldByName('PRICE_R').AsCurrency :=
          FieldByName('PRICE_L').AsCurrency *
          qryMaster.FieldByName('EXCH_RATE_L').AsCurrency;
      FieldByName('PRICE_R_WO_VAT').AsCurrency :=
          round(FieldByName('PRICE_R').AsCurrency * 1000 * 100/
          (qryMaster.FieldByName('VAT_RATE').AsCurrency + 100))/1000;
      FieldByName('AMOUNT_R').AsCurrency :=
```

```
        round(FieldByName('PRICE_R').AsCurrency *
        FieldByName('QUANTITY').AsInteger);
    end
    else
    {Если режим расчета сумм на основе цен без НДС и
    изменено поле "Цена без НДС" или "Количество"}
    if ((Field.FieldName = 'PRICE_L_WO_VAT')   or (Field.FieldName = 'QUANTITY')) and
      (qryMaster.FieldByName('CALC_MODE').AsInteger = 0) then
    begin

      FieldByName('AMOUNT_L').AsCurrency :=
       round(FieldByName('PRICE_L_WO_VAT').AsCurrency * 10 *
        FieldByName('QUANTITY').AsInteger *
        (qryMaster.FieldByName('VAT_RATE').AsCurrency + 100))/1000;
      FieldByName('PRICE_L').AsCurrency :=
       round(FieldByName('PRICE_L_WO_VAT').AsCurrency * 10 *
        (qryMaster.FieldByName('VAT_RATE').AsCurrency + 100))/1000;

      FieldByName('PRICE_R_WO_VAT').AsCurrency :=
        FieldByName('PRICE_L_WO_VAT').AsCurrency *
        qryMaster.FieldByName('EXCH_RATE_L').AsCurrency;;
      FieldByName('PRICE_R').AsCurrency :=
        round(FieldByName('PRICE_R_WO_VAT').AsCurrency * 10 *
        (qryMaster.FieldByName('VAT_RATE').AsCurrency + 100))/1000;
      FieldByName('AMOUNT_R').AsCurrency :=
        round(FieldByName('PRICE_R_WO_VAT').AsCurrency * 10 *
        FieldByName('QUANTITY').AsInteger *
        (qryMaster.FieldByName('VAT_RATE').AsCurrency + 100))/1000;
    end;
    {Расчет суммы в долларах}
    FieldByName('AMOUNT_S').AsCurrency :=
          FieldByName('AMOUNT_L').AsCurrency *
          qryMaster.FieldByName('EXCH_RATE_L').AsCurrency/
          qryMaster.FieldByName('EXCH_RATE_S').Ascurrency;
  end;
end;

procedure TSaleForm.qryDetailBeforePost(DataSet: TDataSet);
begin
  qryDetail.FieldByName('ID').AsInteger := qryMaster.FieldByName('ID').AsInteger;
end;

procedure TSaleForm.dsrMasterDataChange(Sender: TObject; Field: TField);
begin
  {устанавливаем видимость колонок цены, в зависимости от режима расчета сумм}
  dbgDetail.Columns[3].Visible := qryMaster.FieldByName('CALC_MODE').AsInteger=1;
  dbgDetail.Columns[4].Visible := not dbgDetail.Columns[3].Visible;
end;

{Поиск по нажатию любой клавиши в сетке позиций}
procedure TSaleForm.dbgDetailSetEditText(Sender: TObject; ACol, ARow: Integer;
   const Value: String);
var
  s: string;
begin
  if Value = LastValue then
    exit; //защита от повторного поиска по тем же признакам
  if dbgDetail.SelectedField <> qryDetail.FieldByName('ITEM_NAME') then
    exit; //ограничение поиска нажатием клавиш в поле "Наименование"
```

```
  LastValue := Value;

  {Конструирование запроса из строки признаков, разделенных пробелами}
  s := 'SELECT O1.OBJECT_ID, O1.SHORT_NAME, G.PRICE_W, G.PRICE_R'#13+
       'FROM GOODS G, OBJECT_NAMES O1'#13+
       'WHERE G.ID = O1.OBJECT_ID AND O1.OBJECT_ID <> 0';
  if Value <> '' then
    s:=Format('%s AND'#13'%s',[s, StrToVarcharFilter('O1.SHORT_NAME','',Value)]);
  ResetCurrentTime; //сброс счетчика времени

  {включение светодиода в статусной строке Allegro}
  StatusBarDisplay(clLime,0,0,0,'','','');
  with qryGoods do
  begin
    Close;
    SQL.Text := s;
    Open; //открытие запроса
  end;

  {отображение в статусной строке времени, затраченного на поиск}
  StatusBarDisplay(clBlack,0,0,0,GetCurrentTimeStr,'','');
end;

{Нажатие клавиш Enter в списке позиций}
procedure TSaleForm.dbgDetailKeyPress(Sender: TObject; var Key: Char);
begin
  if (Key = 13) and not qryGoods.IsEmpty then
    dbgGoods.SetFocus; //Перемещение фокуса ввода в нижний список
end;

{Копирование позиции из нижнего списка товаров двойным щелчком мыши}
procedure TSaleForm.dbgGoodsDblClick(Sender: TObject);
begin
  qryDetail.Edit; {Присвоение значений полям позиции}
  qryDetail.FieldByName('GOODS').AsInteger :=
     qryGoods.FieldByName('OBJECT_ID').AsInteger;
  qryDetail.FieldByName('ITEM_NAME').AsString :=
     qryGoods.FieldByName('SHORT_NAME').AsString;
  UpdatePrice(qryGoods);
  LastValue := qryDetail.FieldByName('ITEM_NAME').AsString;
  dbgDetail.SelectedField := qryDetail.FieldByName('QUANTITY');
  dbgDetail.SetFocus; //Возвращение фокуса ввода в сетку позиций
end;

procedure TSaleForm.dbgGoodsKeyPress(Sender: TObject; var Key: Char);
begin
  if Key = 13 then
    dbgGoodsDblClick(nil) // Выбор и добавление товара в сетку позиций на Enter
  else if Key = 27 then
    dbgDetail.SetFocus; //Простое возвращение фокуса ввода в сетку позиций на Esc
end;

procedure TSaleForm.actReport1Execute(Sender: TObject);
begin
  XLReport1.Report;
end;
```

```
{Отображение прогресса в статусной строке}
procedure TSaleForm.xlReport1Progress(Report: TxlReport; const Position, Max: Integer);
begin
  if Position = 0 then
    StatusBarDisplay(clBlack, Position, 0, Max, '', '', '')
  else
    StatusBarDisplay(clLime, Position, 0, Max, '', 'Экспорт в Excel...', '');
end;

{Вставка цены}
procedure TSaleForm.UpdatePrice(PriceQuery: TDataSet);
var
  price_s, price_l: Extended;
begin
  {В зависимости от типа используемой цены}
  if qryMaster.FieldByName('PRICE_TYPE').AsInteger = 1 then
    price_s := PriceQuery.FieldByName('PRICE_R').AsCurrency
  else
    price_s := PriceQuery.FieldByName('PRICE_W').AsCurrency;
  {Применение скидки и пересчет в валюту документа}
  price_l := price_s *
    (1 - qryMaster.FieldByName('PRICE_DISCOUNT').AsCurrency/100)*
     qryMaster.FieldByName('EXCH_RATE_S').AsCurrency/
     qryMaster.FieldByName('EXCH_RATE_L').Ascurrency;
  {В зависимости от режима расчета (от цен с НДС или без НДС)
   происходит окгругление}
  if qryMaster.FieldByName('CALC_MODE').AsInteger = 1 then
    qryDetail.FieldByName('PRICE_L').AsCurrency :=
      round(price_l * 100)/100
  else
    qryDetail.FieldByName('PRICE_L_WO_VAT').AsCurrency :=
      round(price_l * 10000/
          (qryMaster.FieldByName('VAT_RATE').AsCurrency + 100))/100;
end;

procedure TSaleForm.Save: boolean;
begin
  Result := False;
  qryTotals.Open;
  qryMaster.Edit;
  qryMaster.FieldByName('TOTAL_L').AsCurrency := qryTotals.Fields[0].AsCurrency;
  qryMaster.FieldByName('TOTAL_R').AsCurrency := qryTotals.Fields[1].AsCurrency;
  qryMaster.FieldByName('TOTAL_S').AsCurrency := qryTotals.Fields[2].AsCurrency;
  qryMaster.Post;
  qryTotals.Close;
  MakeEntries('SALE', qryMaster.FieldByName('ID').AsInteger, traCurrent);
  if qryMaster.FieldByName('HAS_ENTRY').AsInteger = 1 then
  begin
    with qryTestQuantity do
    begin
      Open; //открываем запрос количеств
      if not qryTestQuantity.IsEmpty then //если результирующий набор не пуст
      begin
        {находим позицию в документе}
        qryDetail.Locate('GOODS', FieldByName('OBJECT_ID').AsInteger, MkSet());
        dbgDetail.SelectedField := qryDetail.FieldByName('QUANTITY');
        {выдаем сообщение}
        MessageDlg('Недостаточно товара на складе:'#13+
          FieldByName('NAME').AsString, mtError, MkSet(mbOK), 0);
        Close; //закрываем запрос количеств
```

```
        exit; //выходим из процедуры сохранения
      end
      else
        Close; //закрываем запрос количеств
    end;
    {Вызываем перерасчет себестоимости}
    qrySetItemCost.ExecSQL;
    {повторно проводим документ}
    MakeEntries('SALE', qryMaster.FieldByName('ID').AsInteger, traCurrent);
  end;
  traCurrent.CommitRetaining; //подтвердить транзакцию
  Modified := False;
  actSave.Enabled := Modified;
  RefreshExplorer; //пересветить "Проводник по документам"
  RefreshBalance;  //пересветить "Баланс"
  Result := True;
end;

procedure TSaleForm.actReport2Execute(Sender: TObject);
begin
  XLReport2.Report;
end;

procedure TSaleForm.actReport3Execute(Sender: TObject);
begin
  with qryDetail do
  begin
    FetchAll;
    ciwAmount.Value := qryMaster.FieldByName('TOTAL_R').AsCurrency;
    ciwNum.Value := RecordCount;
  end;
  XLReport3.ParamByName('AMOUNT_IN_WORDS').Value := ciwAmount.Text;
  XLReport3.ParamByName('NUM_IN_WORDS').Value := ciwNum.Text;
  XLReport3.Report;
end;

procedure TSaleForm.qryFirmCalcFields(DataSet: TDataSet);
begin
  with DataSet do
  FieldByName('FULL_NAME_AND_ADDRESS').AsString :=
    Format('%s, ИНН %s, %s',[FieldByName('FULL_NAME').AsString,
      FieldByName('INN').AsString, FieldByName('JUR_ADDRESS').AsString]);
end;

end.
```

```
unit sale_header;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  ExtCtrls, StdCtrls,RxDBComb, AlgCtrls, RXDBCtrl, DBCtrls,RxLookup,IBQuery, DB;

type
  TSaleHeaderForm = class(TForm)
    ButtonPanel: TPanel;
    btnOK: TButton;
    btnCancel: TButton;
```

```
    Label2: TLabel;
    Label4: TLabel;
    Label5: TLabel;
    Label6: TLabel;
    Label7: TLabel;
    Label8: TLabel;
    Label9: TLabel;
    Label10: TLabel;
    Label11: TLabel;
    edDOC_DATE: TDBDateEdit;
    cbxSTOC_ACC: TRxDBLookupCombo;
    edDOC_NO: TDBEdit;
    edVAT_RATE: TDBEdit;
    refCONTRAGENT: TDBRefEdit;
    edEXCH_RATE_S: TDBEdit;
    cbxLAYER_ID: TDBLayerComboBox;
    edEXCH_RATE_L: TDBEdit;
    edENTRY_DATE: TDBDateEdit;
    cbHAS_ENTRY: TDBCheckBox;
    rgCALC_MODE: TDBRadioGroup;
    qryStocks: TIBQuery;
    dsrStocks: TDataSource;
    Label1: TLabel;
    edPRICE_DISCOUNT: TDBEdit;
    rgPRICE_TYPE: TDBRadioGroup;
    procedure FormShow(Sender: TObject);
    procedure btnOKClick(Sender: TObject);
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
    procedure refCONTRAGENTExecuted(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  SaleHeaderForm: TSaleHeaderForm;

implementation
uses
  sale;

{$R *.DFM}

procedure TSaleHeaderForm.FormShow(Sender: TObject);
begin
  qryStocks.Open;
end;

procedure TSaleHeaderForm.btnOKClick(Sender: TObject);
begin
  with SaleForm.qryMaster do
  if InSet(State, MkSet(dsEdit, dsInsert)) then
  begin
    Post;
    RunContext.Documents[0].doc_id := FieldByName('ID').AsInteger;
  end;
  self.ModalResult := mrOK;
end;
```

```
procedure TSaleHeaderForm.FormClose(Sender: TObject; var Action: TCloseAction);
begin
  with SaleForm.qryMaster do
  if InSet(State, MkSet(dsEdit, dsInsert)) then
    Cancel;
end;

procedure TSaleHeaderForm.refCONTRAGENTExecuted(Sender: TObject);
var
  Q: TIBQuery;
begin
  {создаем run-time компонент запроса и запрашиваем скидку}
  Q := TIBQuery.Create(nil);
  try
    Q.Transaction := SaleForm.traCurrent;
    Q.SQL.Text := 'SELECT DISCOUNT FROM FIRM WHERE ID = :ID';
    Q.ParamByName('ID').AsInteger := refCONTRAGENT.Object_ID;
    Q.Open;
    {Копируем скидку в шапку документа}
    SaleForm.qryMaster.FieldByName('PRICE_DISCOUNT').AsCurrency :=
      Q.Fields[0].AsCurrency;
  finally
    Q.Free;
  end;
end;

end.
```

Глава 9. Создаем отчеты

Отчет о продажах – создаем оконный интерфейс.

Этот отчет мы построим на основе прямого SQL-запроса к документам типа «Продажа».
Включим режим «Дизайнер» и создадим новый проект. Главной форме придадим свойства:

Свойство	**Значение**
Name	SaleReportForm
Caption	Отчет о продажах
FormStyle	fsMDIChild

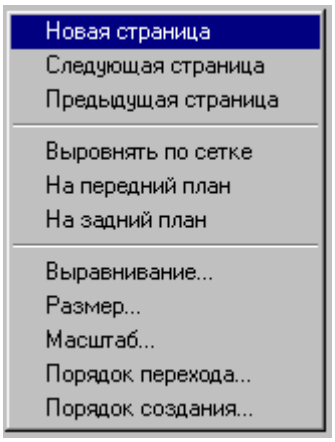
Сохраним все: модуль сохраним в файле **sale_report.pas**, а проект – в файле **sale_report_project.ipr**.
Добавим на форму компонент **Panel** с палитры **Standard**:

Свойство	**Значение**
Name	Panel1
Caption	
Alignment	alTop

Добавим компонент **PageControl** с палитры **Win32**:

Свойство	**Значение**
Name	PageControl1
Alignment	alClient

С помощью контекстного меню компонента **PageControl1** добавим новую страницу:



и назначим ей свойства:

Свойство	**Значение**
Name	tsChart
Caption	Диаграмма

Добавим еще одну новую страницу и назначим ей свойства:

Свойство	**Значение**
Name	tsGrid
Caption	Таблица

Добавим на вторую страницу компонент **DBGrid** с палитры **Data Controls** и придадим ему свойства:

Свойство	**Значение**
Name	dbgReport
Alignment	alClient

Добавим компоненты **IBQuery** и **DataSource** с палитры **InterBase**:

Придадим компоненту **IBQuery1** свойства:

Свойство	**Значение**
Name	qryReport
Transaction	MainConnection.MainTransaction

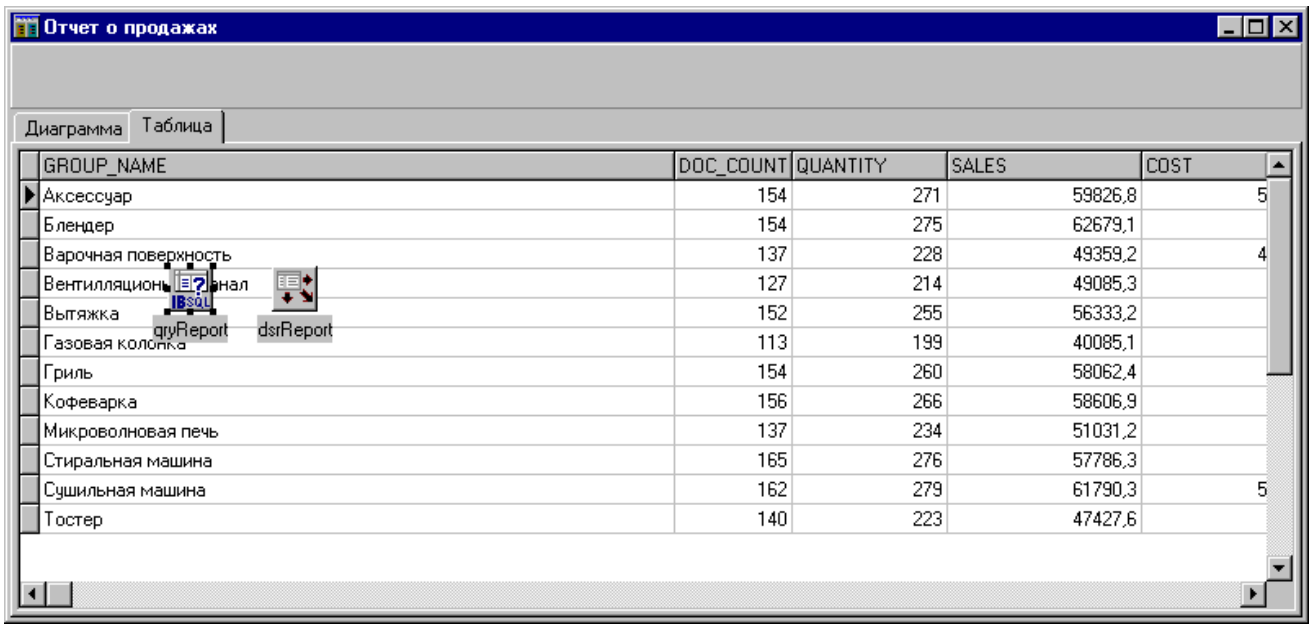
Придадим компоненту **DataSource1** свойства:

Свойство	**Значение**
Name	dsrReport
DataSet	qryReport

В свойстве **DataSource** сетки **dbgReport** укажем источник данных **dsrReport**.
Дважды щелкнем в Инспекторе объектов на свойстве SQL компонента qryReport и впишем такой запрос:

```
select
  O.SHORT_NAME GROUP_NAME,
  COUNT(*) DOC_COUNT,
  SUM(SI.QUANTITY) QUANTITY,
  SUM(SI.AMOUNT_S) SALES,
  SUM(SI.COST_S) COST,
  SUM(SI.AMOUNT_S - SI.COST_S) EARNINGS
from
  SALE_ITEM SI,
  GOODS G,
  OBJECT_NAMES O
where
  SI.GOODS = G.ID and
  G.GOODS_KIND = O.OBJECT_ID
group by
  O.SHORT_NAME
order by
  O.SHORT_NAME
```

Откроем запрос, установив у компонента **qryReport** свойство **Active = True**.



Как видите, SQL-запрос с группировкой позволяет получить очень мощный отчет за очень малое время. В данном случае мы запросили количество позиций в документах, количество проданных единиц, суммарные доходы от продаж, суммарную себестоимость всех проданных единиц, и прибыль, сгруппировав все результаты по видам товаров.

Улучшим отображение в сетке, назначив полям русские заголовки и подправив ширину полей. Для этого дважды щелкнем на компоненте **qryReport** и в появившемся редакторе полей с помощью контекстного меню добавим все поля в список. В инспекторе объектов назначим полям свойства:

Поле	Alignment	DisplayLabel	DisplayWidth	DisplayFormat
GROUP_NAME	taLeftJustify	Группа	30	
DOC_COUNT	taCenter	Док-тов	10	
QUANTITY	taCenter	Продано, Ед.	10	
SALES	taRightJustify	Доход	12	#,##0.00
COST	taRightJustify	Ср.Себест.	12	#,##0.00
EARNINGS	taRightJustify	Прибыль	12	#,##0.00

Закроем запрос **qryReport**, установив **Active = False**. Добавим в список полей вычисляемое поле **RATE** типа **Float** в редакторе полей и зададим ему свойства:

FieldKind	Alignment	DisplayLabel	DisplayWidth	DisplayFormat
fkCalculated	taCenter	Коэфф.	10	#0.00

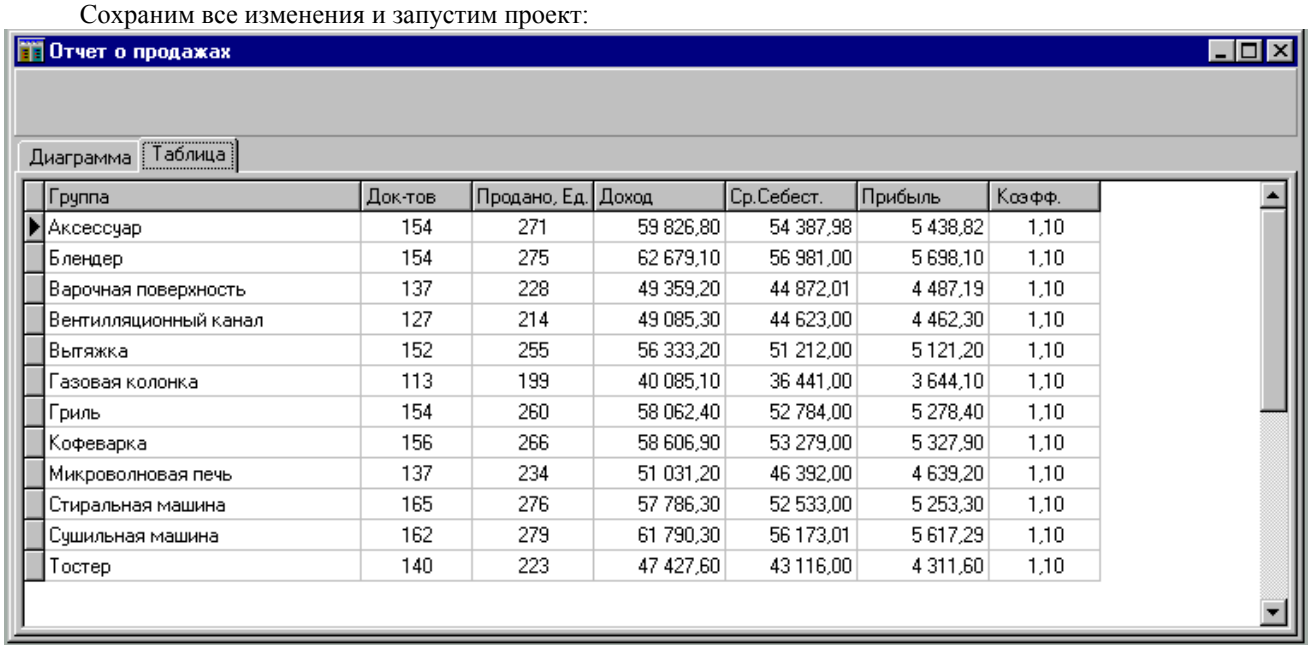
Создадим у компонента **qryReport** обработчик события **OnCalcFields**:

```
procedure TSaleReportForm.qryReportCalcFields(DataSet: TDataSet);
begin
  with DataSet do
  if FieldByName('COST').AsCurrency <> 0 then
     FieldByName('RATE').AsCurrency :=
        FieldByName('SALES').AsCurrency/FieldByName('COST').AsCurrency;
end;
```

Создадим у формы SaleReportForm обработчик OnCreate и впишем в него активизацию запроса:

```
procedure TSaleReportForm.FormCreate(Sender: TObject);
begin
  qryReport.Open;
end;
```

Сохраним все изменения и запустим проект:



Группа	Док-тов	Продано, Ед.	Доход	Ср.Себест.	Прибыль	Коэфф.
Аксессуар	154	271	59 826,80	54 387,98	5 438,82	1,10
Блендер	154	275	62 679,10	56 981,00	5 698,10	1,10
Варочная поверхность	137	228	49 359,20	44 872,01	4 487,19	1,10
Вентилляционный канал	127	214	49 085,30	44 623,00	4 462,30	1,10
Вытяжка	152	255	56 333,20	51 212,00	5 121,20	1,10
Газовая колонка	113	199	40 085,10	36 441,00	3 644,10	1,10
Гриль	154	260	58 062,40	52 784,00	5 278,40	1,10
Кофеварка	156	266	58 606,90	53 279,00	5 327,90	1,10
Микроволновая печь	137	234	51 031,20	46 392,00	4 639,20	1,10
Стиральная машина	165	276	57 786,30	52 533,00	5 253,30	1,10
Сушильная машина	162	279	61 790,30	56 173,01	5 617,29	1,10
Тостер	140	223	47 427,60	43 116,00	4 311,60	1,10

Усовершенствуем наш проект так, чтобы иметь возможность группировать результаты не только по виду товара, но и по марке. Для этого нам потребуется изменять текст запроса во время выполнения программы, подставляя в качестве поля группировки либо GOODS_KIND либо GOODS_MARK.

Добавим в текст модуля в раздел **implementation** константу:

```
const
  SQL_SALE_REPORT =
     'select'#13+
     '  O.SHORT_NAME GROUP_NAME,'#13+
     '  COUNT(*) DOC_COUNT,'#13+
     '  SUM(SI.QUANTITY) QUANTITY,'#13+
     '  SUM(SI.AMOUNT_S) SALES,'#13+
     '  SUM(SI.COST_S) COST,'#13+
     '  SUM(SI.AMOUNT_S - SI.COST_S) EARNINGS'#13+
     'from'#13+
     '  SALE_ITEM SI,'#13+
     '  GOODS G,'#13+
     '  OBJECT_NAMES O'#13+
     'where'#13+
     '  SI.GOODS = G.ID and'#13+
     '  G.%s = O.OBJECT_ID'#13+  //здесь будет подстановка имени поля
     'group by'#13+
     '  O.SHORT_NAME'#13+
     'order by'#13+
     '  O.SHORT_NAME';
```

Это практически тот же текст запроса. Обратим внимание, что в строке, выделенной жирным шрифтом вместо имени поля GOODS_KIND вставлена комбинация символов **%s**.

Мы будем использовать функцию format, которая умеет отыскивать в строке знаки процентов и подставлять вместо комбинации **%s** строковые выражения, которые можно передать ей в виде массива значений.

Для того чтобы пользователь мог задать способ группировки нам понадобится переключатель. Выберем верхнюю панель **Panel1** и придадим ее свойству значение **Alignment = alRight**. Панель прижмется к правому краю. Расширим немного панель и поставим на нее компонент **RadioGroup** с палитры **Standard** и зададим ему свойства:

Свойство	**Значение**
Caption	Способ группировки
Items	По виду товара По марке товара
ItemIndex	0
Name	rgGroupMode

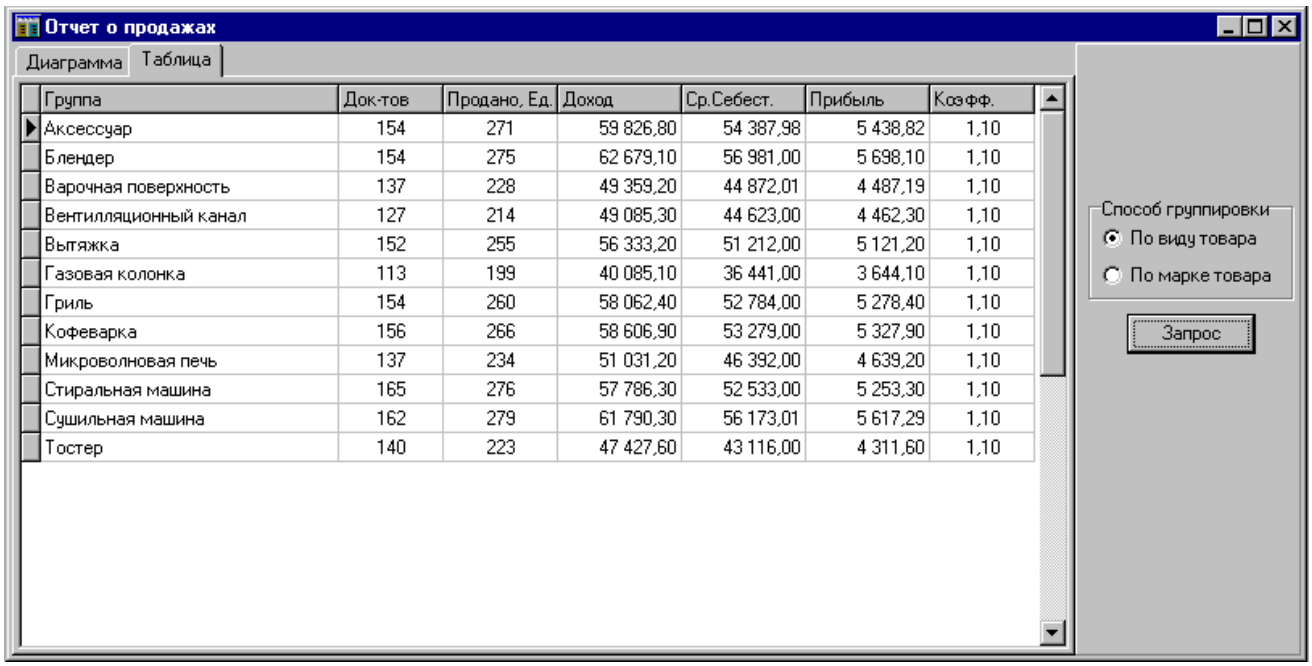
Немного ниже этого компонента расположим кнопку Button с палитры Standard и придадим ей свойства:

Свойство	**Значение**
Caption	Запрос
Name	btnQuery

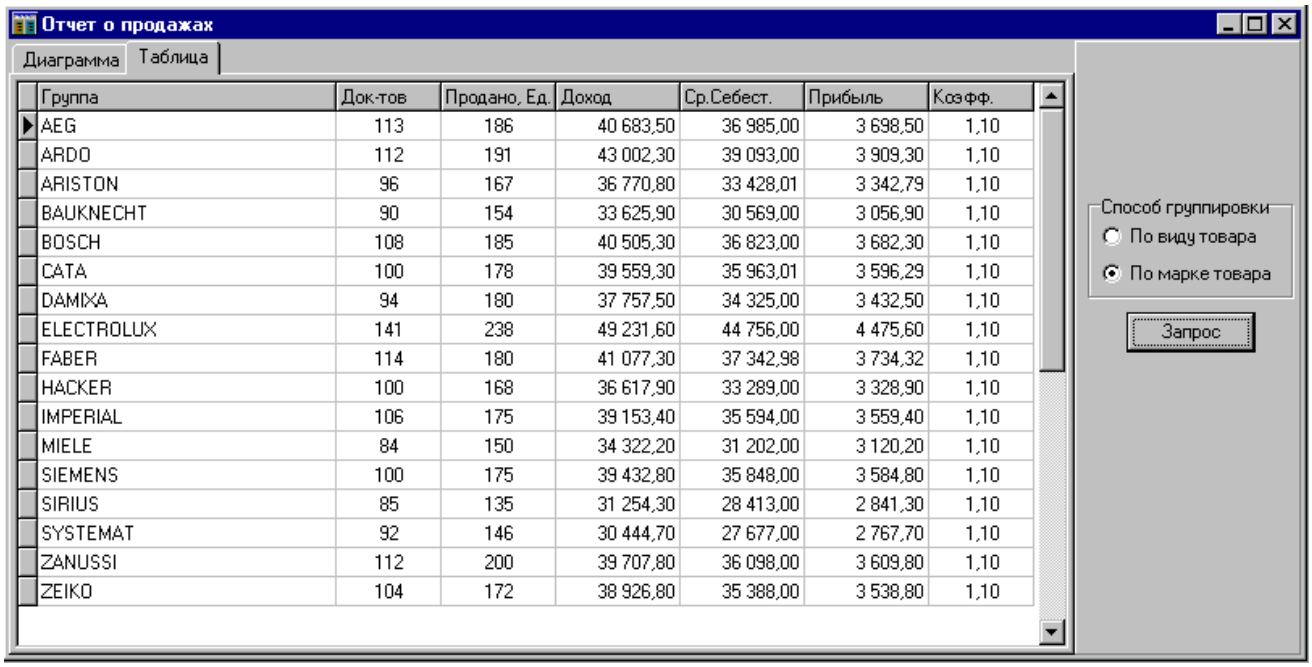
Создадим кнопке обработчик события **OnClick**:

```
procedure TSaleReportForm.btnQueryClick(Sender: TObject);
begin
  qryReport.Close;
  case rgGroupMode.ItemIndex of
  0: qryReport.SQL.Text := Format(SQL_SALE_REPORT, ['GOODS_KIND']);
  1: qryReport.SQL.Text := Format(SQL_SALE_REPORT, ['GOODS_MARK']);
  end;
  qryReport.Open;
end;
```

Запустим проект и убедимся, что после переключения «Способа группировки», нажав на кнопку «Запрос», мы получаем либо запрос с группировкой по виду товара, либо - по марке товара:



Группа	Док-тов	Продано, Ед.	Доход	Ср.Себест.	Прибыль	Коэфф.
Аксессуар	154	271	59 826,80	54 387,98	5 438,82	1,10
Блендер	154	275	62 679,10	56 981,00	5 698,10	1,10
Варочная поверхность	137	228	49 359,20	44 872,01	4 487,19	1,10
Вентилляционный канал	127	214	49 085,30	44 623,00	4 462,30	1,10
Вытяжка	152	255	56 333,20	51 212,00	5 121,20	1,10
Газовая колонка	113	199	40 085,10	36 441,00	3 644,10	1,10
Гриль	154	260	58 062,40	52 784,00	5 278,40	1,10
Кофеварка	156	266	58 606,90	53 279,00	5 327,90	1,10
Микроволновая печь	137	234	51 031,20	46 392,00	4 639,20	1,10
Стиральная машина	165	276	57 786,30	52 533,00	5 253,30	1,10
Сушильная машина	162	279	61 790,30	56 173,01	5 617,29	1,10
Тостер	140	223	47 427,60	43 116,00	4 311,60	1,10



Группа	Док-тов	Продано, Ед.	Доход	Ср.Себест.	Прибыль	Коэфф.
AEG	113	186	40 683,50	36 985,00	3 698,50	1,10
ARDO	112	191	43 002,30	39 093,00	3 909,30	1,10
ARISTON	96	167	36 770,80	33 428,01	3 342,79	1,10
BAUKNECHT	90	154	33 625,90	30 569,00	3 056,90	1,10
BOSCH	108	185	40 505,30	36 823,00	3 682,30	1,10
CATA	100	178	39 559,30	35 963,01	3 596,29	1,10
DAMIXA	94	180	37 757,50	34 325,00	3 432,50	1,10
ELECTROLUX	141	238	49 231,60	44 756,00	4 475,60	1,10
FABER	114	180	41 077,30	37 342,98	3 734,32	1,10
HACKER	100	168	36 617,90	33 289,00	3 328,90	1,10
IMPERIAL	106	175	39 153,40	35 594,00	3 559,40	1,10
MIELE	84	150	34 322,20	31 202,00	3 120,20	1,10
SIEMENS	100	175	39 432,80	35 848,00	3 584,80	1,10
SIRIUS	85	135	31 254,30	28 413,00	2 841,30	1,10
SYSTEMAT	92	146	30 444,70	27 677,00	2 767,70	1,10
ZANUSSI	112	200	39 707,80	36 098,00	3 609,80	1,10
ZEIKO	104	172	38 926,80	35 388,00	3 538,80	1,10

Отчет у нас пока строится по всем документам продаж. Хорошо бы строить его для некоторого диапазона дат, который пользователь мог бы установить произвольно. Для добавления этой возможности нам понадобятся два компонента календарей **DateEdit** с палитры **RxControls**. Добавим их на панель, расположив их один под другим. Изменим текст запроса, хранящийся в константе **SQL_SALE_REPORT**:

1. добавим в список таблиц в секции **FROM** таблицу **SALE**,
2. в секцию **WHERE** условия объединения для этой таблицы и фильтрации по полю **ENTRY_DATE**.
3. добавим еще в секцию **WHERE** условие **HAS_ENTRY=1**, чтобы в отчет входили только те документы, у которых установлена птичка «отгружено».

Изменения в тексте запроса показаны жирным шрифтом:

```
const
  SQL_SALE_REPORT =
     'select'#13+
     '  O.SHORT_NAME GROUP_NAME,'#13+
     '  COUNT(*) DOC_COUNT,'#13+
     '  SUM(SI.QUANTITY) QUANTITY,'#13+
     '  SUM(SI.AMOUNT_S) SALES,'#13+
     '  SUM(SI.COST_S) COST,'#13+
     '  SUM(SI.AMOUNT_S - SI.COST_S) EARNINGS'#13+
     'from'#13+
     '  SALE_ITEM SI,'#13+
     '  SALE S,'#13+
     '  GOODS G,'#13+
     '  OBJECT_NAMES O'#13+
     'where'#13+
     '  SI.ID = S.ID and'#13+
     '  S.HAS_ENTRY = 1 and'#13+
     '  S.ENTRY_DATE BETWEEN :DATE1 and :DATE2 and'#13+
     '  SI.GOODS = G.ID and'#13+
     '  G.%s = O.OBJECT_ID'#13+  //здесь будет подстановка имени поля
     'group by'#13+
     '  O.SHORT_NAME'#13+
     'order by'#13+
     '  O.SHORT_NAME';
```

В обработчик события **OnClick** кнопки добавим текст, показанный жирным шрифтом:

```
procedure TSaleReportForm.btnQueryClick(Sender: TObject);
begin
  qryReport.Close;
  case rgGroupMode.ItemIndex of
  0: qryReport.SQL.Text := Format(SQL_SALE_REPORT, ['GOODS_KIND']);
  1: qryReport.SQL.Text := Format(SQL_SALE_REPORT, ['GOODS_MARK']);
  end;
  qryReport.ParamByName('DATE1').AsDateTime := DateEdit1.Date;
  qryReport.ParamByName('DATE2').AsDateTime := DateEdit2.Date;
  qryReport.Open;
end;
```

Из обработчика **OnCreate** формы удалим активизацию запроса, которая нам больше не нужна, так как мы реализовали ее в обработчике нажатия кнопки. Вместо этого добавим присвоение начальных значений даты компонентам календарей:

```
procedure TSaleReportForm.FormCreate(Sender: TObject);
begin
  DateEdit1.Date := EncodeDate(YearOf(Date), MonthOf(Date), 1);
  DateEdit2.Date := Date;
end;
```

Мы устанавливаем в первом календаре первое число текущего месяца, а во втором – сегодняшнюю дату.

Над календарями поставим компонент **Label** с палитры **Standard** и в его свойство **Caption** впишем заголовок «Диапазон дат».

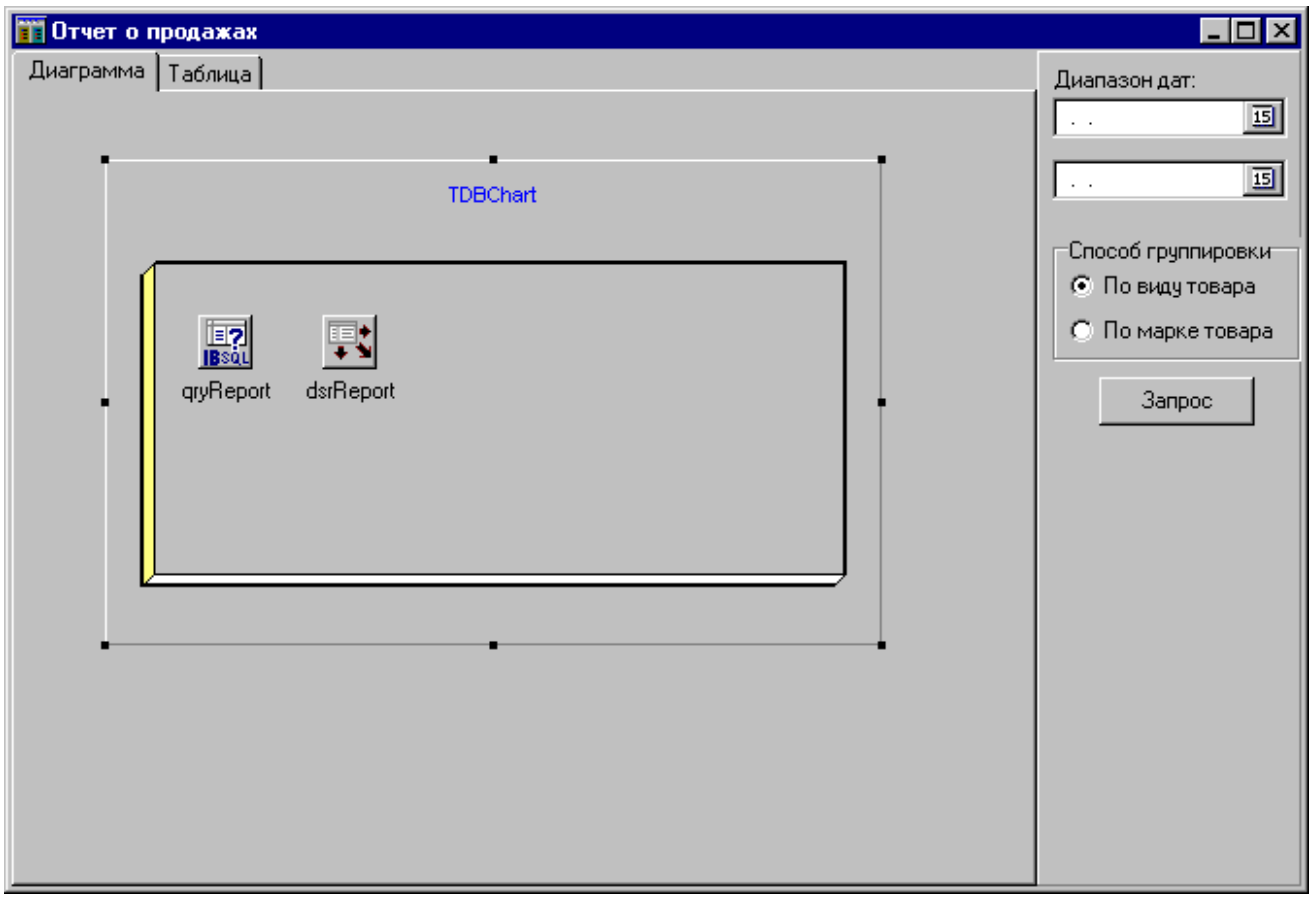
Запустим проект и убедимся, что изменение диапазона дат, задаваемых календарями, влияет на результаты запроса. Остановим проект, чтобы вернуться в режим дизайна.

Теперь мы займемся созданием диаграммы.

Щелкнем на закладке «Диаграмма» компонента **PageControl1**.

Щекнем на поверхности открывшейся в результате пустой страницы, чтобы выбрать ее, как компонент.

Поставим на нее компонент **DBChart** с палитры **DataControls**:

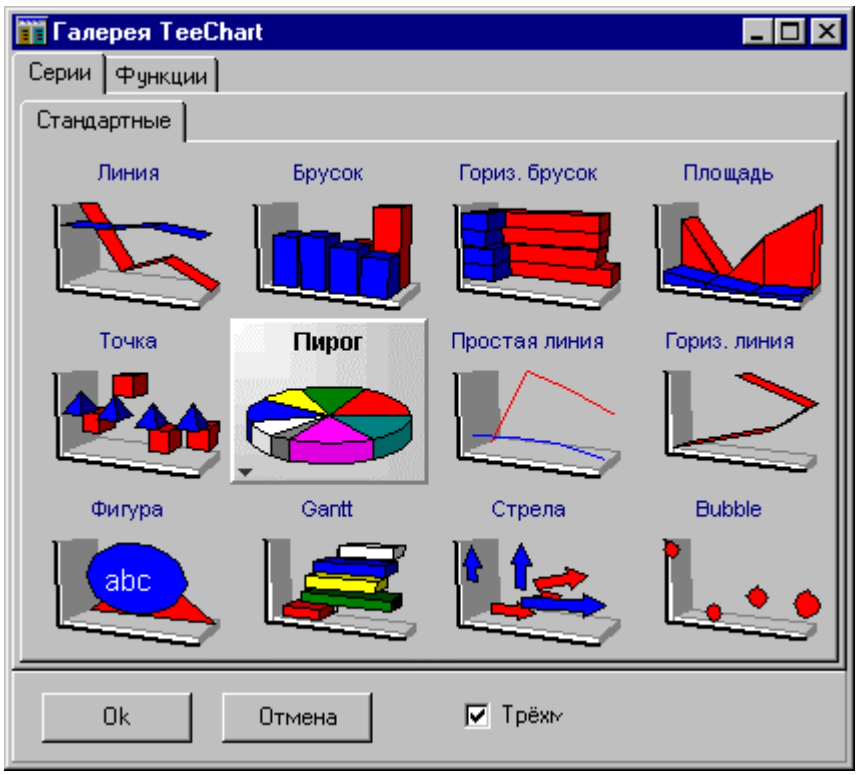


Назначим ему свойства в Инспекторе объектов:

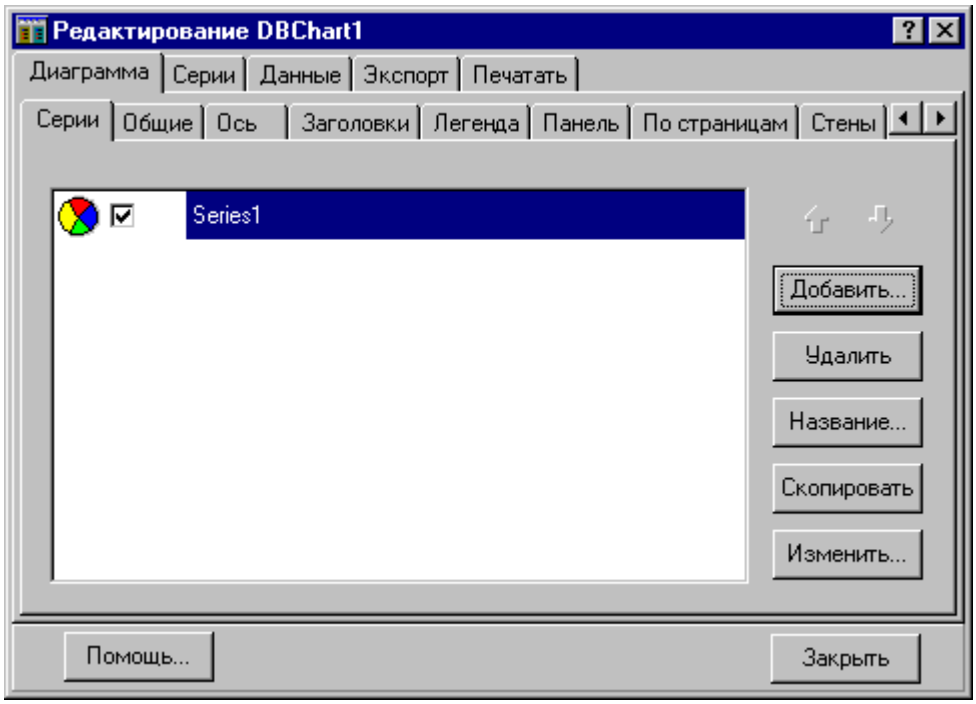
Свойство	**Значение**
Align	alClient
BevelOuter	bvNone
BorderStyle	bsSingle
Name	ReportChart

Настроим диаграмму.

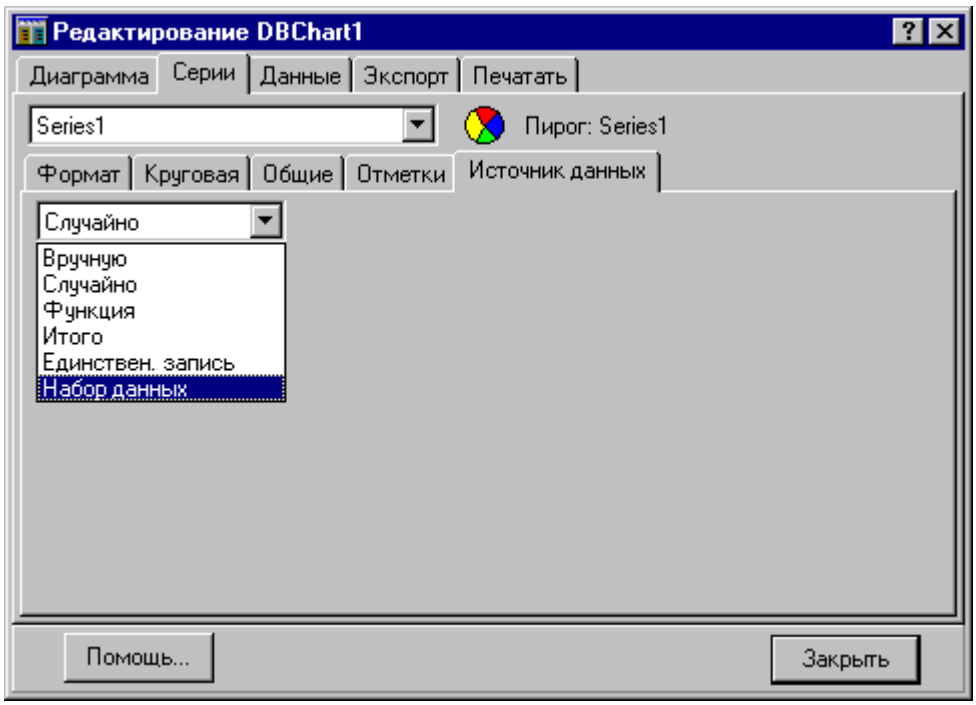
Для этого нужно дважды щелкнуть на ней. Появится редактор компонента TDBChart. Нажмем кнопку «Добавить». Появится множество вариантов будущей диаграммы, из которых мы выберем пирог и нажмем **OK**:



Этим действием мы создали серию Series1. Один компонент DBChart способен одновременно отображать множество серий, поэтому на закладке «Серии» (на которой мы находимся) для них выделено место под целый список:



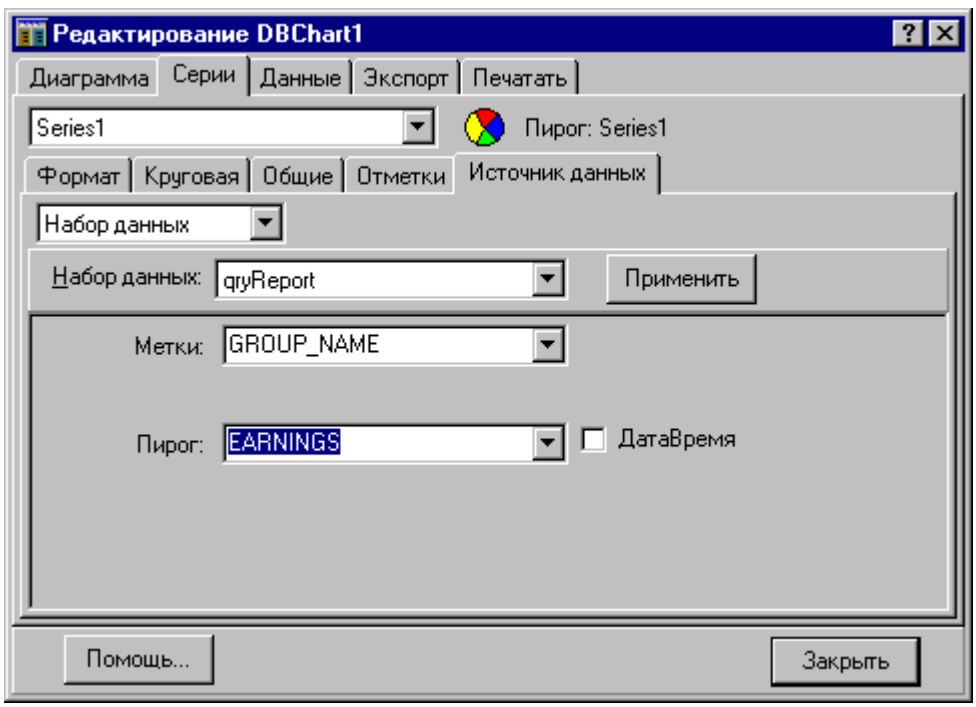
После того, как мы создали серию, нужно настроить ее свойства. Для этого выберем в верхнем ряду закладок закладку «Серии». После этого в нижнем ряду закладок выберем закладку «Источник данных» и на ней в выпадающем списке выберем «Набор данных»:



Этот выбор означает, что данные будут браться из компонента типа **TDataSet**, например, в нашем случае - это компонент запроса **qryReport**. Как только мы выбрали режим из списка, появятся новые органы управления.

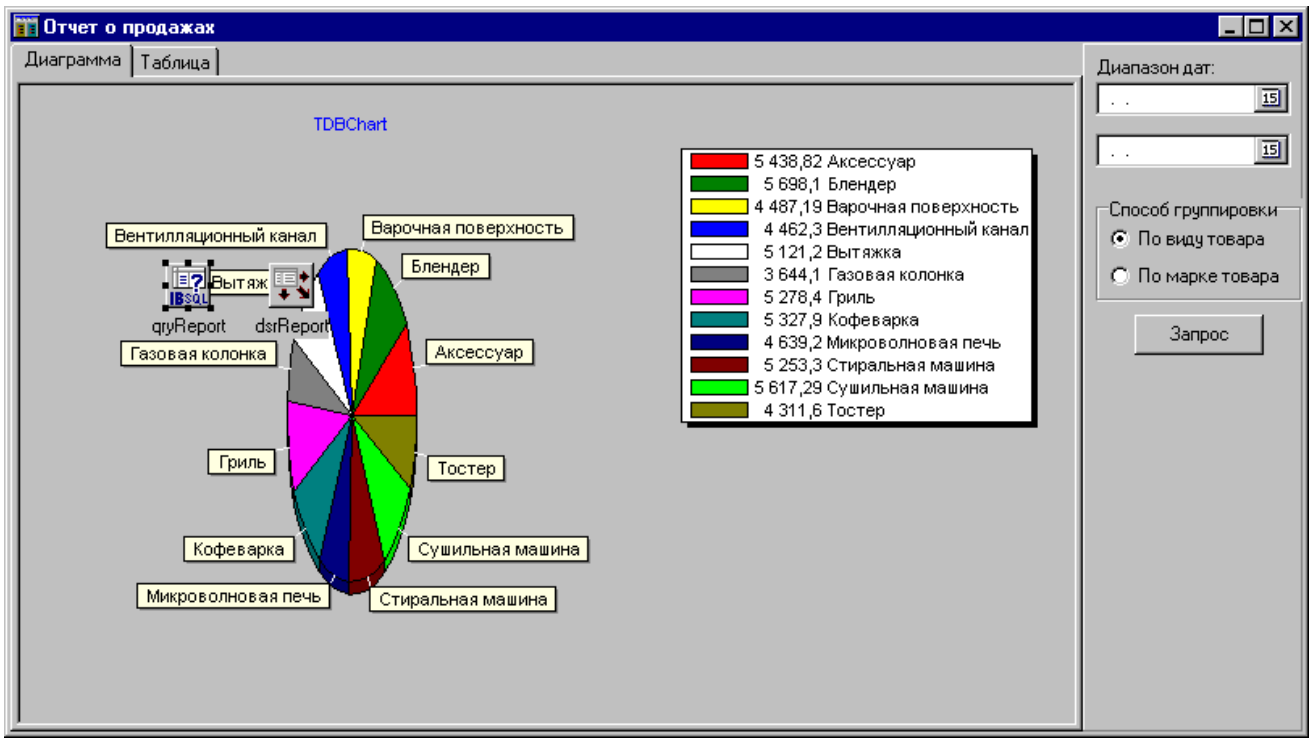
Установим следующие значения:

Свойство	**Значение**
Набор данных	qryReport
Метки	GROUP_NAME
Пирог	EARNINGS

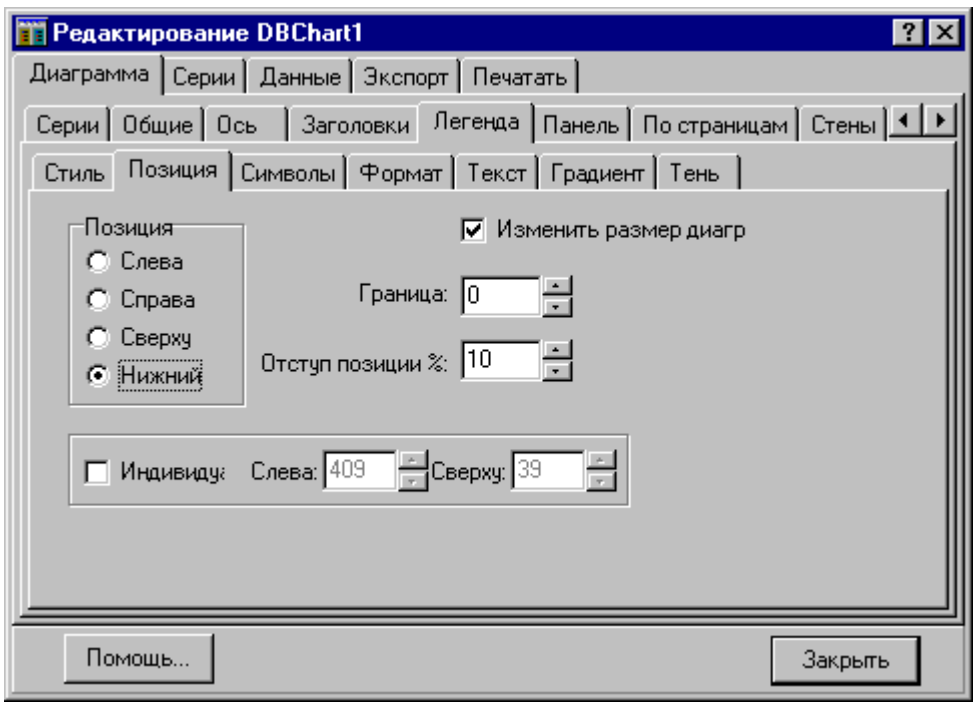


Нажмем кнопку «Применить», затем кнопку «Закрыть».

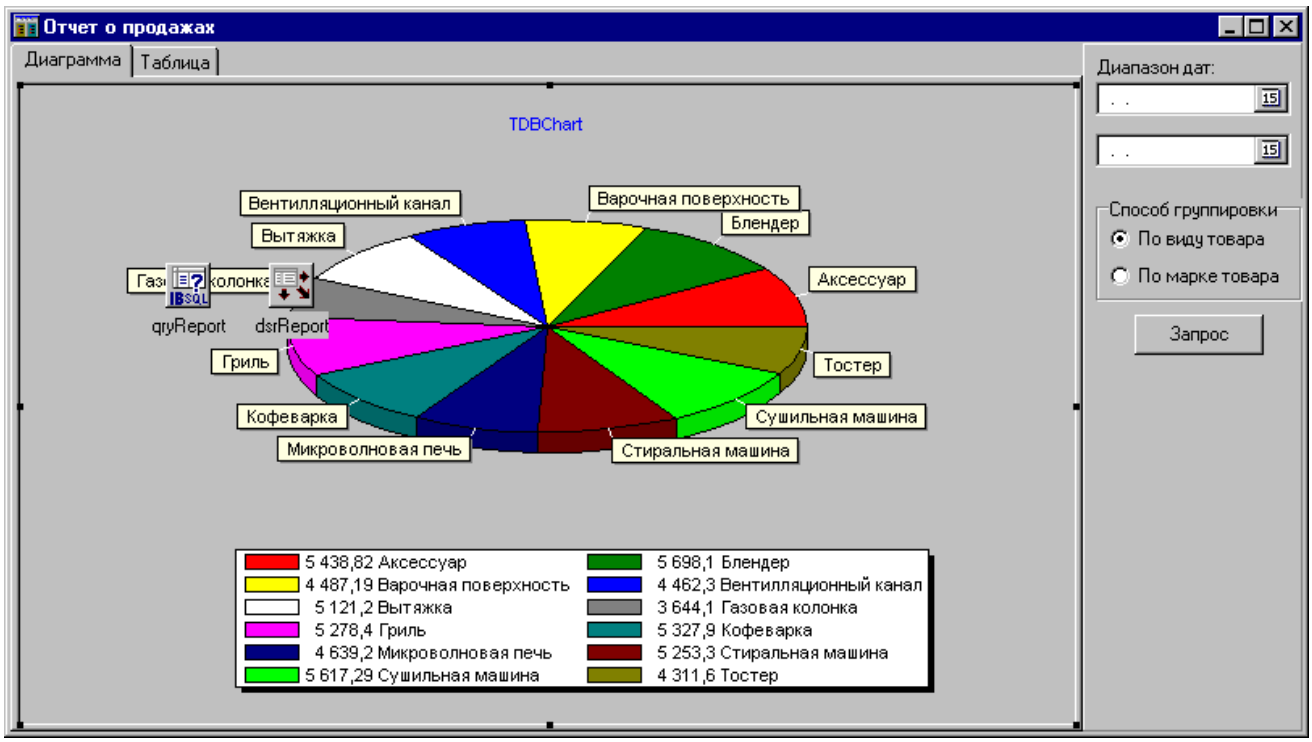
Теперь откроем запрос **qryReport**, установив свойство **Active = True** (работает тот запрос в компоненте **qryReport**, что мы создали в самом начале).



Мы видим, что весьма неудачным оказалось сочетание отображения меток вокруг пирога и расположения «легенды» справа, так как это сильно сжало диаграмму по горизонтали. Настроим расположение легенды так, чтобы она оказалась внизу. Для этого вызовем редактор диаграмм дважды щелкнем и откроем закладку «Легенда» в нижнем ряду. На страничке появится еще один ряд закладок (третий) и среди них нам нужно выбрать закладку «Позиция». На ней имеется группа радиокнопок под названием «Позиция». Выберем положение «Нижний»:



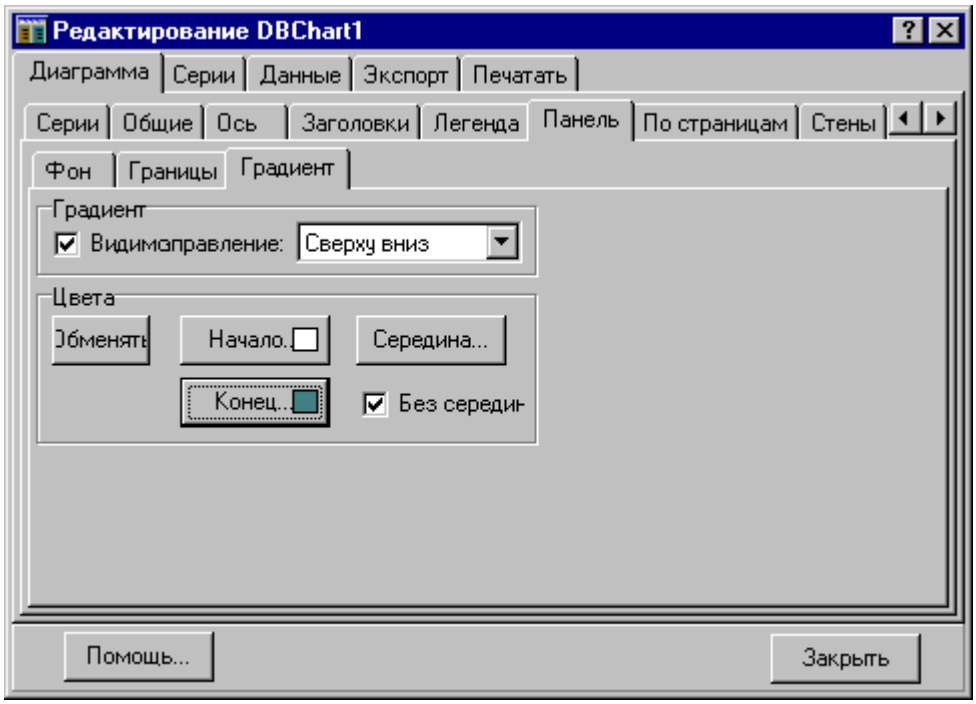
Закроем редактор. Теперь диаграмма выглядит более приемлемо:



Еще раз вызовем редактор диаграмм.

На нижней закладке «Заголовки» уберем птичку в свойстве «Видимо». Синяя надпись TDBChart в верхней части диаграммы должна исчезнуть.

На нижней закладке «Панель» выберем закладку третьего уровня «Градиент». Установим птичку «Видимо». Нажмем на кнопку «Конец» и вместо желтого цвета выберем грязно-голубой:



Закроем редактор.

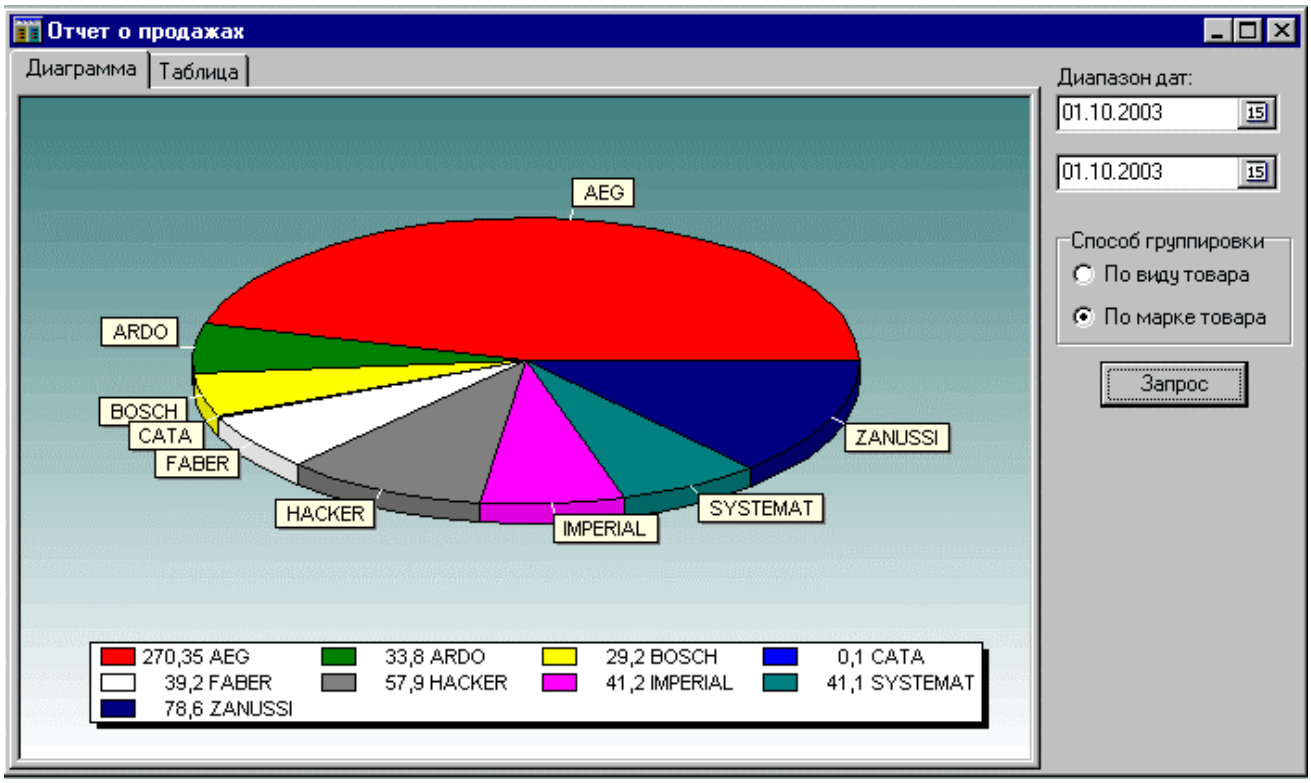
Закроем запрос **qryReport**, установив свойство **Active = False**.

Выберем панель **Panel1** и установим ее свойство внешней фаски **BevelOuter = bvNone**. Вообще желательно избегать большого количества «рельефа» в окнах, так как он, как правило, скорее отвлекает пользователя, чем помогает ему работать.

Запустим проект.

Сузим диапазон дат до одного дня, чтобы увеличить неравномерность диаграммы, выберем способ группировки по марка товара и нажмем кнопку «Запрос».

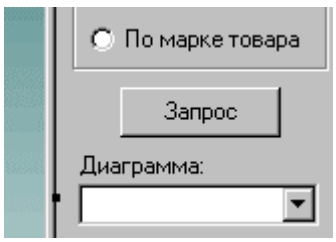
Перед нами диаграмма распределения прибыли по маркам товара:



Итак, мы получили окно с двумя закладками. На первой закладке пользователь сможет увидеть отчет в виде диаграммы, а на второй – виде таблицы. Однако в таблице пока что отображаются все сведения, а в диаграмме – лишь распределение прибыли по группам (поле **EARNINGS** запроса). Хотелось бы как-то иметь возможность отображать в диаграмме другие поля.

Для того чтобы это реализовать, добавим на панель компонент **Label**, снабдив его заголовком «Диаграмма» и компонент **ComboBox** с палитры **Standard**, установив для него такие свойства:

Свойство	**Значение**
Name	cbSeriesValues
Syle	csDropDownList



В обработчик **OnCreate** формы впишем текст, выделенный жирным шрифтом:

```
procedure TSaleReportForm.FormCreate(Sender: TObject);
var
  i: integer;
begin
  DateEdit1.Date := EncodeDate(YearOf(Date), MonthOf(Date), 1);
  DateEdit2.Date := Date;
  for i := 1 to qryReport.Fields.Count - 1 do
    cbSeriesValues.Items.Add(qryReport.Fields[i].DisplayLabel);
```

```
  cbSeriesValues.ItemIndex := 0;
end;
```

Мы просто заполняем выпадающий список **cbSeriesValues** заголовками полей запроса. Нумерация полей в компоненте запроса начинается с нуля. Мы начинаем с единицы, следовательно, пропускаем первое поле (**GROUP_NAME**). Нам оно не нужно, так как это поле группировки, а для выбора режима отображения в диаграмме потребуются только численные поля.

Изменим также обработчик события **OnClick** кнопки «Запрос»:

```
procedure TSaleReportForm.btnQueryClick(Sender: TObject);
begin
  qryReport.Close;
  case rgGroupMode.ItemIndex of
  0: qryReport.SQL.Text := Format(SQL_SALE_REPORT, ['GOODS_KIND']);
  1: qryReport.SQL.Text := Format(SQL_SALE_REPORT, ['GOODS_MARK']);
  end;

  {назначаем поле в качестве источника Y значений для диаграммы}
  Series1.YValues.ValueSource :=
    qryReport.Fields[cbSeriesValues.ItemIndex + 1].FieldName;

  qryReport.ParamByName('DATE1').AsDateTime := DateEdit1.Date;
  qryReport.ParamByName('DATE2').AsDateTime := DateEdit2.Date;
  qryReport.Open;
end;
```

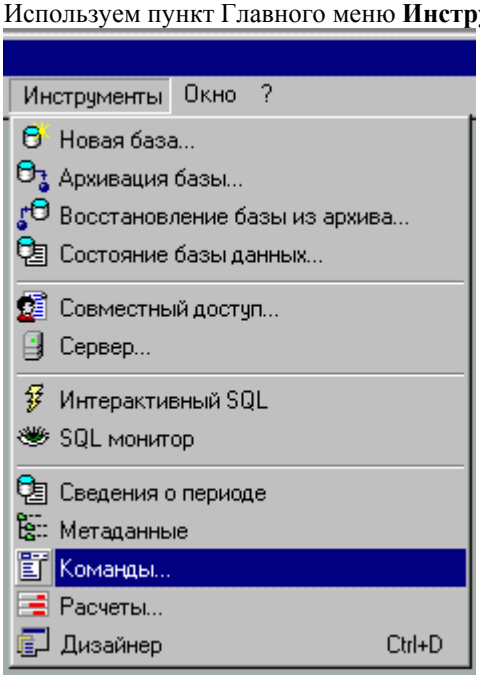
Запустим проект.

Для изменения отображаемых в диаграмме величин нужно выбрать значение из выпадающего списка и нажать кнопку «Запрос».

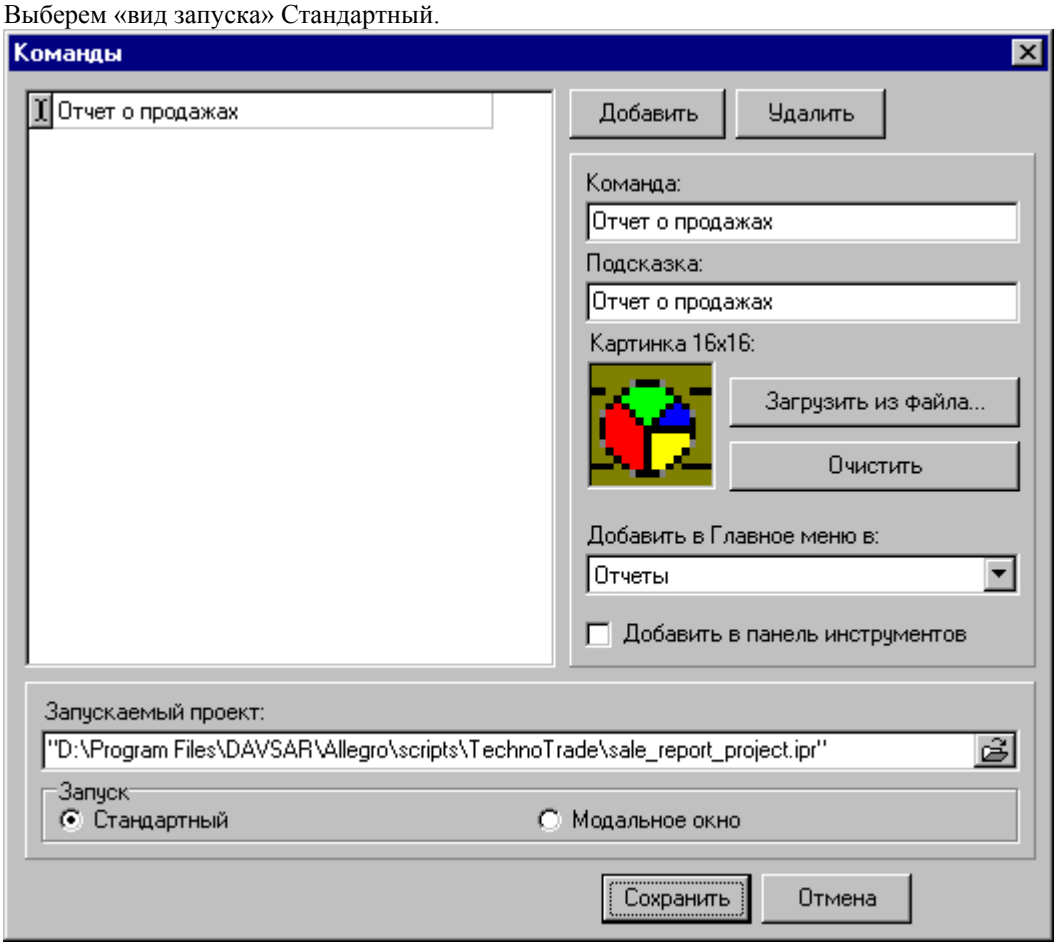
Нам осталось реализоветь экспорт набора данных в Excel и организовать вызов «Отчета о продажах» из Главного меню программы Allegro. Остановим проект и выйдем из режима «Дизайнер».

Отчет о продажах - подключаем к Главному меню.

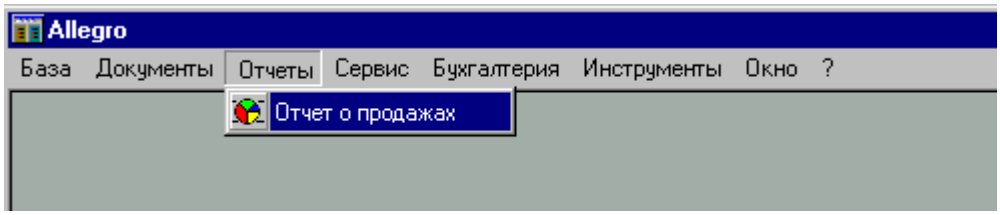
Используем пункт Главного меню **Инструменты/Команды**:



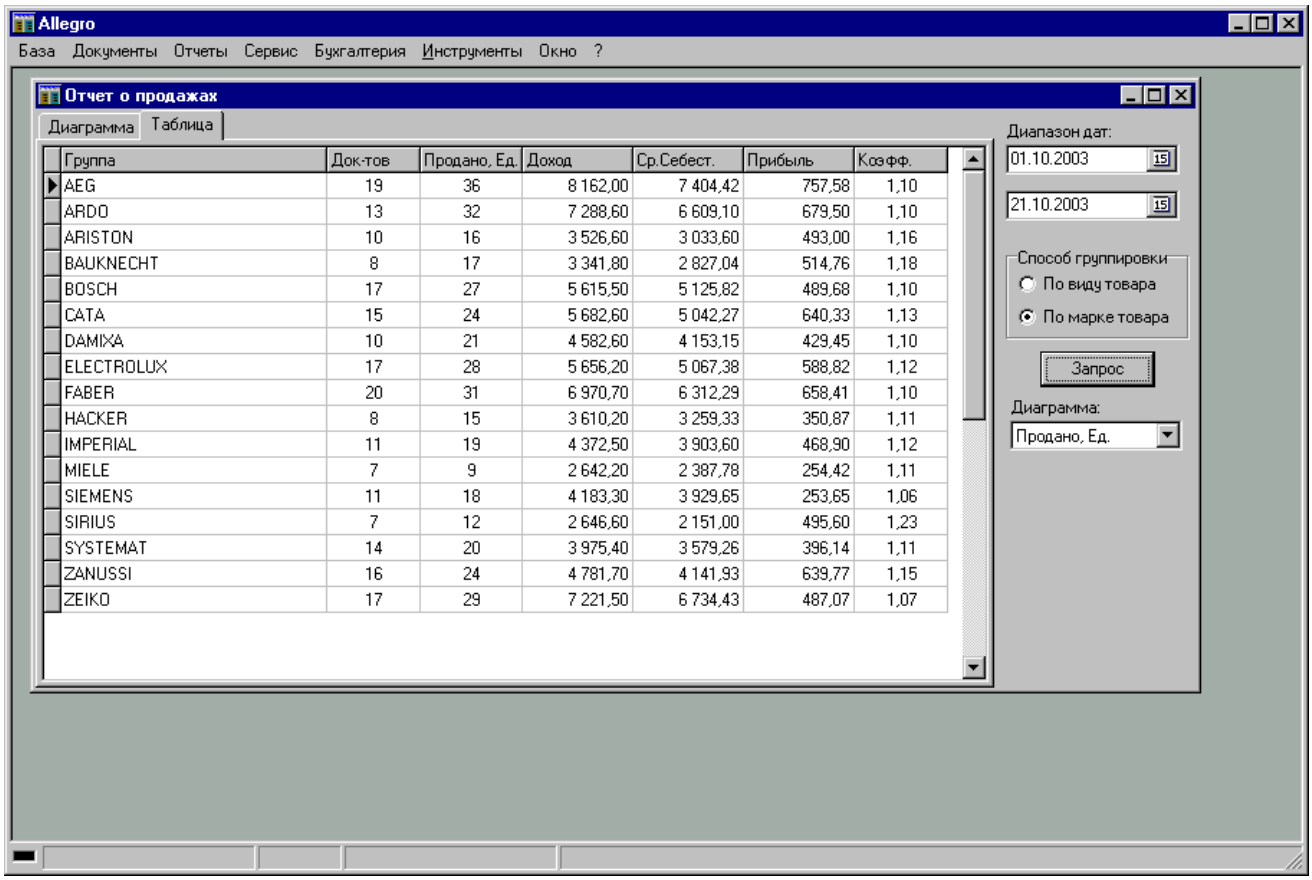
В появившемся диалоге нажмем кнопку «Добавить».
Назовем команду «Отчет о продажах».
Загрузим из файла подходящую картинку.
В выпадающем списке «Добавить в Главное меню в» выберем пункт «Отчеты».
В «запускаемом проекте» выберем файл **sale_report_prioject.ipr**.
Выберем «вид запуска» Стандартный.



Нажмем «Сохранить».
А теперь попробуем вызвать пункт меню **Отчеты**.



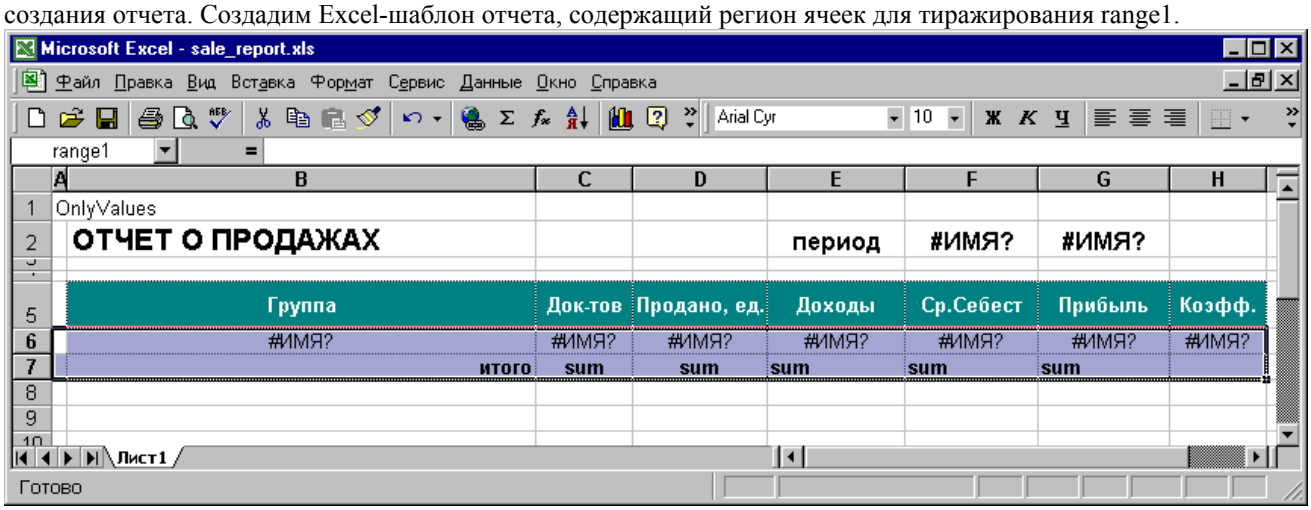
Мы видим, что в этом меню появился новый пункт. Если щелкнуть по нему мышью, то вызовется наш «Отчет о продажах». Мы встроили отчет в конфигурацию.



Группа	Док-тов	Продано, Ед.	Доход	Ср.Себест.	Прибыль	Коэфф.
AEG	19	36	8 162,00	7 404,42	757,58	1,10
ARDO	13	32	7 288,60	6 609,10	679,50	1,10
ARISTON	10	16	3 526,60	3 033,60	493,00	1,16
BAUKNECHT	8	17	3 341,80	2 827,04	514,76	1,18
BOSCH	17	27	5 615,50	5 125,82	489,68	1,10
CATA	15	24	5 682,60	5 042,27	640,33	1,13
DAMIXA	10	21	4 582,60	4 153,15	429,45	1,10
ELECTROLUX	17	28	5 656,20	5 067,38	588,82	1,12
FABER	20	31	6 970,70	6 312,29	658,41	1,10
HACKER	8	15	3 610,20	3 259,33	350,87	1,11
IMPERIAL	11	19	4 372,50	3 903,60	468,90	1,12
MIELE	7	9	2 642,20	2 387,78	254,42	1,11
SIEMENS	11	18	4 183,30	3 929,65	253,65	1,06
SIRIUS	7	12	2 646,60	2 151,00	495,60	1,23
SYSTEMAT	14	20	3 975,40	3 579,26	396,14	1,11
ZANUSSI	16	24	4 781,70	4 141,93	639,77	1,15
ZEIKO	17	29	7 221,50	6 734,43	487,07	1,07

Отчет о продажах - экспорт в Excel

Мы с вами уже реализовывали экспорт в Excel, когда разрабатывали интерфейсы документов «Поступление на склад» и «Продажа». Поэтому мы не будем подробно останавливаться на всех деталях создания отчета. Создадим Excel-шаблон отчета, содержащий регион ячеек для тиражирования range1.



Диапазон дат (период) будем выводить в отчет через параметры DATE1 и DATE2.

Добавим к форме **SaleReportForm** компонент **XLReport** спалитры **Print**, подключим его с помощью свойства DataSources к компоненту запроса **qryReport**, псевдоним **Alias = REPORT**, **Range= range1**.

Назначим компоненту **XLReport1** обработчик события **OnProgress**:

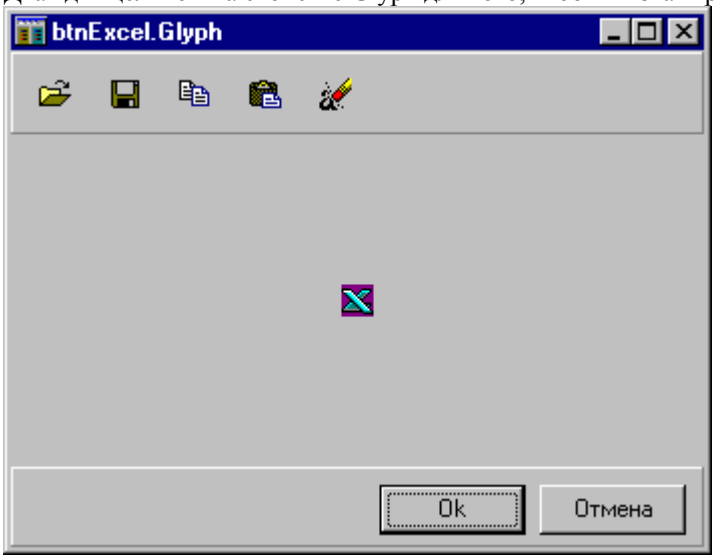
```
procedure TSaleReportForm.xlReport1Progress(Report: TxlReport; const Position, Max: Integer);
begin
  if Position = 0 then
    StatusBarDisplay(clBlack, Position, 0, Max, '', '', '')
  else
    StatusBarDisplay(clLime, Position, 0, Max, '', 'Экспорт в Excel...', '');
end;
```

Добавим на правую панель ниже кнопки «Запрос» кнопку **BitBtn** с палитры **Additional**.

Придадим этой кнопке в Инспекторе объектов свойства:

Свойство	**Значение**
Name	btnExport
Caption	Экспорт

Дважды щелкнем на свойстве Glyph для того, чтобы вызвать редактор рисунка:

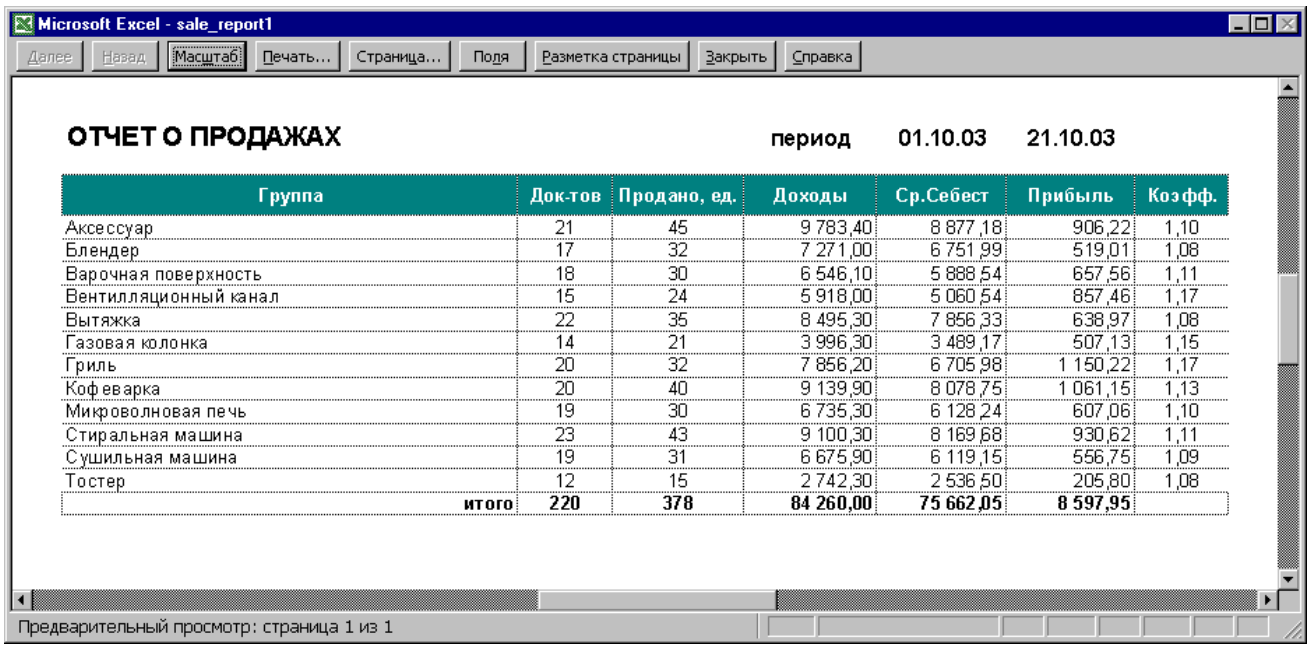


Загрузим значок Excel из файла формата bmp 16x16 пиксель. Нажмем кнопку **OK**.

Создадим у кнопки обработчик события **OnClick**:

```
procedure TSaleReportForm.btnExcelClick(Sender: TObject);
begin
  XLReport1.ParamByName('DATE1').Value := DateEdit1.Date;
  XLReport1.ParamByName('DATE2').Value := DateEdit2.Date;
  XLReport1.Report;
end;
```

Запустим проект **sale_report**. Нажмем кнопку «Запрос», затем кнопку «Экспорт». Появится Excel. Вызовем «Предварительный просмотр»:



Группа	Док-тов	Продано, ед.	Доходы	Ср.Себест	Прибыль	Коэфф.
Аксессуар	21	45	9 783,40	8 877,18	906,22	1,10
Блендер	17	32	7 271,00	6 751,99	519,01	1,08
Варочная поверхность	18	30	6 546,10	5 888,54	657,56	1,11
Вентилляционный канал	15	24	5 918,00	5 060,54	857,46	1,17
Вытяжка	22	35	8 495,30	7 856,33	638,97	1,08
Газовая колонка	14	21	3 996,30	3 489,17	507,13	1,15
Гриль	20	32	7 856,20	6 705,98	1 150,22	1,17
Кофеварка	20	40	9 139,90	8 078,75	1 061,15	1,13
Микроволновая печь	19	30	6 735,30	6 128,24	607,06	1,10
Стиральная машина	23	43	9 100,30	8 169,68	930,62	1,11
Сушильная машина	19	31	6 675,90	6 119,15	556,75	1,09
Тостер	12	15	2 742,30	2 536,50	205,80	1,08
итого	**220**	**378**	**84 260,00**	**75 662,05**	**8 597,95**	

Закроем Excel. Выйдем из режима «Дизайнер».

Мы закончили «Отчет о продажах». Приведем полный листинг проекта. Как видим, он совсем небольшой.

Полный листинг проекта «Отчет о продажах»

```
unit sale_report;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  ExtCtrls, ComCtrls,DBGrids, IBQuery, DB,IBCustomDataSet,StdCtrls,ToolEdit,
  RXDBCtrl,DBChart, Series,xlReport, Buttons;

type
  TSaleReportForm = class(TForm)
    TopPanel: TPanel;
    PageControl1: TPageControl;
    tsDiagram: TTabSheet;
    tsGrid: TTabSheet;
    DBGrid1: TDBGrid;
    qryReport: TIBQuery;
    dsrReport: TDataSource;
    qryReportGROUP_NAME1: TIBStringField;
    qryReportDOC_COUNT1: TIntegerField;
    qryReportQUANTITY1: TLargeintField;
```

```
    qryReportSALES1: TIBBCDField;
    qryReportCOST1: TIBBCDField;
    qryReportEARNINGS1: TIBBCDField;
    qryReportRATE1: TFloatField;
    rgGroupMode: TRadioGroup;
    btnQuery: TButton;
    DateEdit1: TDateEdit;
    DateEdit2: TDateEdit;
    Label1: TLabel;
    DBChart1: TDBChart;
    Series1: TPieSeries;
    Label2: TLabel;
    cbSeriesValues: TComboBox;
    xlReport1: TxlReport;
    btnExcel: TBitBtn;
    procedure qryReportCalcFields(DataSet: TDataSet);
    procedure FormCreate(Sender: TObject);
    procedure btnQueryClick(Sender: TObject);
    procedure xlReport1Progress(Report: TxlReport; const Position, Max: Integer);
    procedure btnExcelClick(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  SaleReportForm: TSaleReportForm;

implementation

{$R *.DFM}

const
  SQL_SALE_REPORT =
      'select'#13+
      '  O.SHORT_NAME GROUP_NAME,'#13+
      '  COUNT(*) DOC_COUNT,'#13+
      '  SUM(SI.QUANTITY) QUANTITY,'#13+
      '  SUM(SI.AMOUNT_S) SALES,'#13+
      '  SUM(SI.COST_S) COST,'#13+
      '  SUM(SI.AMOUNT_S - SI.COST_S) EARNINGS'#13+
      'from'#13+
      '  SALE_ITEM SI,'#13+
      '  SALE S,'#13+
      '  GOODS G,'#13+
      '  OBJECT_NAMES O'#13+
      'where'#13+
      '  SI.ID = S.ID and'#13+
      '  S.HAS_ENTRY = 1 and'#13+
      '  S.ENTRY_DATE BETWEEN :DATE1 and :DATE2 and'#13+
      '  SI.GOODS = G.ID and'#13+
      '  G.%s = O.OBJECT_ID'#13+  //здесь будет подстановка имени поля
      'group by'#13+
      '  O.SHORT_NAME'#13+
      'order by'#13+
      '  O.SHORT_NAME';
```

```
procedure TSaleReportForm.qryReportCalcFields(DataSet: TDataSet);
begin
  with DataSet do
  if FieldByName('COST').AsCurrency <> 0 then
     FieldByName('RATE').AsCurrency :=
        FieldByName('SALES').AsCurrency/FieldByName('COST').AsCurrency;
end;

procedure TSaleReportForm.FormCreate(Sender: TObject);
var
  i: integer;
begin
  DateEdit1.Date := EncodeDate(YearOf(Date), MonthOf(Date), 1);
  DateEdit2.Date := Date;
  for i := 1 to qryReport.Fields.Count - 1 do
    cbSeriesValues.Items.Add(qryReport.Fields[i].DisplayLabel);
  cbSeriesValues.ItemIndex := 0;
end;

procedure TSaleReportForm.btnQueryClick(Sender: TObject);
begin
  qryReport.Close;
  case rgGroupMode.ItemIndex of
  0: qryReport.SQL.Text := Format(SQL_SALE_REPORT, ['GOODS_KIND']);
  1: qryReport.SQL.Text := Format(SQL_SALE_REPORT, ['GOODS_MARK']);
  end;

  {назначаем поле в качестве источника Y значений для диаграммы}
  Series1.YValues.ValueSource :=
     qryReport.Fields[cbSeriesValues.ItemIndex + 1].FieldName;

  qryReport.ParamByName('DATE1').AsDateTime := DateEdit1.Date;
  qryReport.ParamByName('DATE2').AsDateTime := DateEdit2.Date;
  qryReport.Open;
end;

procedure TSaleReportForm.xlReport1Progress(Report: TxlReport; const Position, Max: Integer);
begin
  if Position = 0 then
    StatusBarDisplay(clBlack, Position, 0, Max, '', '', '')
  else
    StatusBarDisplay(clLime, Position, 0, Max, '', 'Экспорт в Excel...', '');
end;

procedure TSaleReportForm.btnExcelClick(Sender: TObject);
begin
  XLReport1.ParamByName('DATE1').Value := DateEdit1.Date;
  XLReport1.ParamByName('DATE2').Value := DateEdit2.Date;
  XLReport1.Report;
end;

end.
```

Отчет о движении товаров

Предыдущий отчет мы построили исключительно на SQL-запросе к документам. Сейчас нам предстоит сделать отчет на основе SQL-запросов к таблице проводок ACC_TURN.

Заказчик желает видеть отчет в долларах США примерно такого вида:

Товар	Начальный остаток (кол-во, сумма)	Приход (кол-во, сумма)	Расход (кол-во, сумма)	Конечный остаток (кол-во, сумма)	Средняя стоимость единицы	Опт (Тек. цена,%)	Розница (Тек. цена,%)

Заказчик желает, чтобы создать такой отчет можно было бы для произвольного диапазона дат и для каких-то определенных товаров и/или марок, например, для всех стиральных машин с марками BOSCH и SIEMENS. Еще он хочет, чтобы отчет можно было построить для какого-то одного склада или суммарно для всех складов.

Не исключено, что «монстры SQL» могут написать такой мудреный SQL-запрос, что он вернет нам весь этот отчет целиком. Но мы будем стараться придерживаться простых запросов и следить за самым главным показателем: скорость. Ничто так не расстраивает конечного пользователя, как если машина «долго думает». Особенно если это менеджер по продажам, которому нужно прямо сейчас ответить по телефону, сколько у него на складе стиральных машин.

Поэтому мы разобьем задачу на более простые задачи:

- Первым SQL-запросом выясним начальные остатки для всех товаров
- Вторым SQL-запросом выясним приход и расход для всех товаров за указанный промежуток дат
- Затем как-то все это объединим по каждому товару.
- Вычислим конечные остатки и средние стоимости единиц

Вызовем окно Интерактивного SQL.

SQL-запрос, возвращающий начальные остатки на складе 1007 на какую-то дату выглядит очень просто:

```
select object_id, sum(debit-credit) a, sum(quantity_debit – quantity_credit) q
from acc_turn
where acc_id = 1007 and op_date < ’01.06.2003’
group by object_id
```

Так же просто выглядит запрос, возвращающий приход и расход товара за определенный промежуток дат:

```
select object_id, sum(debit) ad, sum(credit) ac, sum(quantity_debit) qd, sum(quantity_credit) qc
from acc_turn
where acc_id = 1007 and op_date between ’01.06.2003’ and ’15.07.2003’
group by object_id
```

И что очень важно – такие запросы работают быстро. Однако как объединить результаты работы этих двух запросов так, чтобы для каждого товара object_id одной строкой вернуть результаты первого и второго запросов одновременно? Здесь есть один способ, который мы назвали «диагональный метод сборки сводного отчета» и рекомендуем этот метод разработчикам. Суть этого метода в том, что мы создаем в базе данных хранимую процедуру, которая выполняет несколько SQL-запросов последовательно, один за другим, возвращая результаты суммирования в разных колонках, а id объектов группировки – в одной колонке:

object_id	**a**	**q**	**Ad**	**ac**	**qd**	**qc**
xxx	xxx	xxx	0	0	0	0
xxx	xxx	xxx	0	0	0	0
xxx	xxx	xxx	0	0	0	0
xxx	0	0	xxx	xxx	xxx	xxx
xxx	0	0	xxx	xxx	xxx	xxx
xxx	0	0	xxx	xxx	xxx	xxx
xxx	0	0	xxx	xxx	xxx	xxx

Когда сканируются строки результирующего набора первого SQL-запроса, значения колонок присваиваются выходным параметрам хранимой процедуры a и q, а остальным параметрам при этом присваивается нуль. Затем сканируются строки результирующего набора второго SQL-запроса, значения колонок присваиваются выходным параметрам хранимой процедуры ad, ac, qd и qc, а нуль присваивается параметрам a и q. Хранимые процедуры сервера InterBase устроены так, что возможно осуществить еще один SQL-запрос,

работающий уже с результатами, выданными этой хранимой процедурой. Это будет простой запрос, с группировкой по object_id.

Итак, создадим хранимую процедуру:

```
create procedure goods_flow(date1 date, date2 date)
returns(object_id integer, acc_id integer, a decimal(18,2), q integer,
       ad decimal(18,2), ac decimal(18,2), qd integer, qc integer)
as
begin
  a = 0; q = 0; ad = 0; ac = 0; qd = 0; qc = 0; /*присваиваем 0*/
  /*запрос остатков*/
  for select
    a.object_id, a.acc_id,
    sum(a.debit - a.credit) a, sum(quantity_debit - quantity_credit) q
  from acc_turn a, acc
  where
    a.acc_id = acc.acc_id and
    acc.parent_id = 1005 and
    a.op_date < :date1
  group by a.object_id, a.acc_id
  into :object_id, :acc_id, :a, :q do
  suspend;

  a = 0; q = 0; /*присваиваем 0*/
  /*запрос приходов и расходов*/
  for select a.object_id, a.acc_id,
     sum(a.debit) ad, sum(a.credit) ac, sum(a.quantity_debit) qd, sum(a.quantity_credit) qc
  from acc_turn a, acc
  where
    a.acc_id = acc.acc_id and
    acc.parent_id = 1005 and
    a.op_date between :date1 and :date2
  group by a.object_id, a.acc_id
  into :object_id, :acc_id, :ad, :ac, :qd, :qc do
  suspend;
end
```

В этой процедуре мы объединили таблицу проводок ACC_TURN с таблицей счетов ACC по полю ACC_ID для того, чтобы получить результаты для всех складов. Склады у нас являются дочерними счетами к регистру «Товары», у которого ACC_ID=1005. Поэтому мы и записали условие PARENT_ID = 1005.

Теперь у нас есть хранимая процедура **goods_flow**, и мы можем сделать такой SQL-запрос:

```
select object_id,
  sum(a) a, sum(q) q, sum(ad) ad, sum(ac) ac, sum(qd) qd, sum(qc) qc
from goods_flow('01.06.2003', '15.07.2003')
where acc_id = 1007
group by object_id
```

Мы получили все необходимые нам данные, сгруппированные по товарам на складе 1007. Осталось лишь усовершенствовать запрос так, чтобы получить еще цены из справочника товаров. Значения всех остальных полей (остаток на конец и себестоимость единицы) можно будет вычислить на основе значений уже имеющихся.

Попробуем выполнить в окне Интерактивного SQL усовершенствованный запрос:

```
select
  gf.object_id,
  gk.name goods_kind_name,
  gm.name goods_mark_name,
  g.article,
  sum(gf.a) a, sum(gf.q) q, sum(gf.ad) ad, sum(gf.ac) ac, sum(gf.qd) qd, sum(gf.qc) qc,
  g.price_w, g.price_r
from
  goods_flow('01.06.2003', '15.07.2003') gf,
  goods g,
  goods_kind gk,
  goods_mark gm
where
  gf.acc_id = 1007 and
  gf.object_id = g.id and
  g.goods_kind = gk.id and
  g.goods_mark = gm.id
group by
  gf.object_id, gk.name, gm.name, g.article, g.price_w, g.price_r
order by 2,3,4
```

Мы решили, что данный отчет будет смотреться нагляднее, если вместо сборных наименований товаров изобразить три отдельные колонки: вид, марка и артикул. Поэтому мы не стали объединять набор с таблицей OBJECT_NAMES, и вместо этого объединили его с таблицами GOODS_KIND и GOODS_MARK. К тому же нам удалось при таком подходе упорядочить результирующий набор по наименованию вида, марки и артикулу при помощи выражения:

```
order by 2,3,4
```

Обратим внимание на время выполнения запроса. Как видим, оно не превышает долей секунды.

Нам осталось создать интерфейс отчета и еще немного усовершенствовать SQL-запрос, чтобы получить фильтрацию по складам, а также видам и маркам товаров.

Включим режим «Дизайнер» и создадим новый проект.

Сохраним модуль формы под названием **goods_flow.pas**, а проект под названием **goods_flow_project.ipr**.

Придадим форме свойства:

Свойство	**Значение**
Name	GoodsFlowForm
FormStyle	fsMDIChild
Caption	Отчет о движении товаров

Добавим к форме компонент **IBQuery** с палитры **InterBase** и придадим ему свойства:

Свойство	**Значение**
Name	qryReport
Transaction	MainConnection.MainTransaction

Скопируем запрос из окна ISQL в свойство **SQL** компонента **qryReport**. Для этого растянем главное окно, вызовем окно ISQL, если оно не на экране, нажмем ***Ctrl+P***, чтобы показать текст последнего SQL-запроса, выделим этот текст и скопируем в буфер обмена Windows при помощи клавиш ***Ctrl+C***. Сожмем главное окно, выберем компонент **qryReport**, дважды щелкнем на его свойства **SQL** и в появившемся редакторе запроса вставим из буфера обмена наш запрос с помощью клавиш ***Ctrl+V***.

Дважды щелкнем на компоненте **qryReport** и в Редакторе полей добавим все поля в список.

Так мы создали «постоянные поля», которым теперь можно назначить русские заголовки и способы форматирования. Назначим полям свойства в Инспекторе объектов:

Поле	Alignment	DisplayLabel	DisplayWidth	DisplayFormat	Visible
OBJECT_ID	taLeftJustify	OBJECT_ID	20		False
GOODS_KIND_NAME	taLeftJustify	Вид товара	12		True
GOODS_MARK_NAME	taCenter	Марка	12		True
ARTICLE	taLeftJustify	Артикул	12		True
A	taRightJustify	Нач.Ост.,USD	12	#,##0.00	True
Q	taCenter	Нач.Ост., Ед.	12		True
AD	taRightJustify	Приход, USD	12	#,##0.00	True
AC	taRightJustify	Расход, USD	12	#,##0.00	True
QD	taCenter	Приход, Ед.	12		True
QC	taCenter	Расход, Ед.	12		True
PRICE_W	taCenter	Опт. Цена	10		True
PRICE_R	taCenter	Розн.Цена	10		True

Расположим поля прихода сумм и количеств друг за другом. Для этого поменяем местами поля QD и AC, перетащив при помощи мыши поле QD на одну позицию вверх в редакторе полей, или просто уменьшив на единицу его свойство **Index** в Инспекторе объектов.

Нам нужно создать еще пять полей: «Конечный остаток, USD», «Конечный остаток, Ед.», «Средняя себестоимость единицы», «Оптовая наценка, %», «Розничная наценка, %».

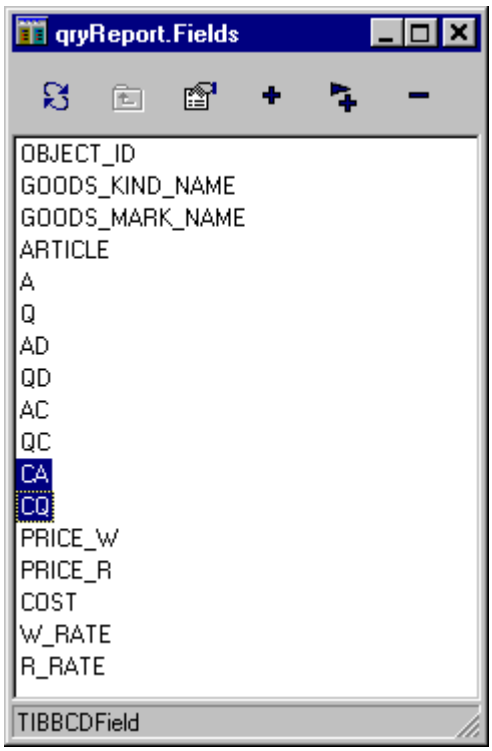
Закроем редактор полей. Закроем запрос, если он у нас открыт, установив свойство **Active = False**. Добавлять вычисляемые поля рекомендуется при закрытом запросе, иначе компонент может выдать сообщение «Поле не найдено». Вызовем снова редактор полей и создадим два новых поля типа IBBCD:

Поле	FieldKind	Alignment	DisplayLabel	DisplayWidth	DisplayFormat	Visible
CA	fkCalculated	taLeftJustify	Остаток,USD	12	#,##0.00	True
CQ	fkCalculated	taCenter	Остаток,Ед.	12		True

Создадим три новых поля типа Float:

Поле	FieldKind	Alignment	DisplayLabel	DisplayWidth	DisplayFormat	Visible
COST	fkCalculated	taRightJustify	Ср.стоим.	10	#,###.#	True
W_RATE	fkCalculated	taCenter	Опт.нац,%	10	#0.00	True
R_RATE	fkCalculated	taCenter	Розн.нац,%	10	#0.00	True

Перетащим мышью поля CA и CQ в списке так, чтобы они расположились после полей прихода:



Закроем редактор. Создадим у компонента **qryReport** обработчик события **OnCalcFields**:

```
procedure TGoodsFlowForm.qryReportCalcFields(DataSet: TDataSet);
begin
  with DataSet do
  begin
    {вычисляем остатки}
    FieldByName('CA').AsCurrency := FieldByName('A').AsCurrency +
      FieldByName('AD').AsCurrency - FieldByName('AC').AsCurrency;
    FieldByName('CQ').AsCurrency := FieldByName('Q').AsCurrency +
      FieldByName('QD').AsCurrency - FieldByName('QC').AsCurrency;

    {вычисляем среднюю стоимость единицы на складе}
    if FieldByName('CQ').AsCurrency <> 0 then
      FieldByName('COST').AsCurrency :=
        FieldByName('CA').AsCurrency / FieldByName('CQ').AsCurrency;
    {вычисляем наценки в процентах}
    if FieldByName('COST').AsCurrency <> 0 then
    begin
      FieldByName('W_RATE').AsCurrency :=
        (FieldByName('PRICE_W').AsCurrency /
         FieldByName('COST').AsCurrency - 1) *100;
      FieldByName('R_RATE').AsCurrency :=
        (FieldByName('PRICE_R').AsCurrency /
         FieldByName('COST').AsCurrency - 1)*100;
    end;
  end;
end;
```

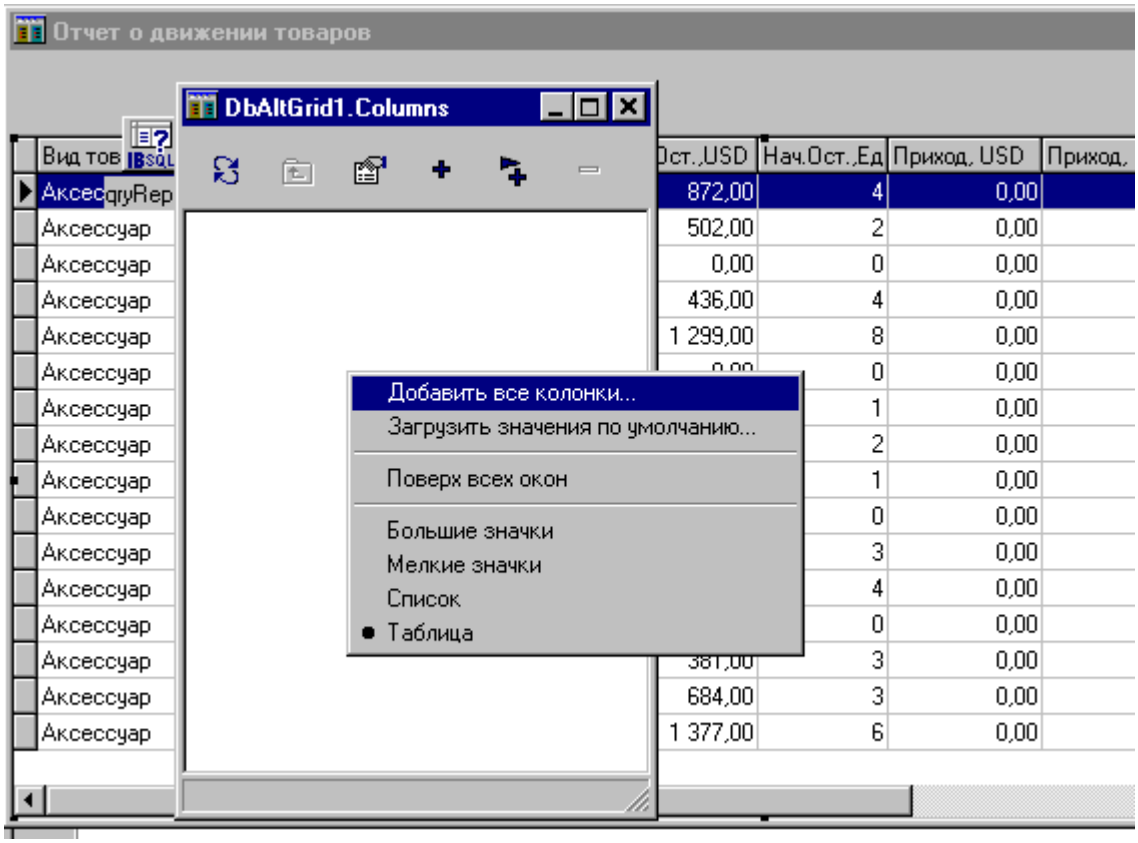
Добавим на форму компонент **Panel** с палитры **Standard**:

Свойство	**Значение**
Align	alTop
Caption	
BevelOuter	bvNone

Для отображения результатов запроса нам понадобится необычная сетка, так как полей очень много и в одну строку они явно не уместятся. Откроем запрос, установив у компонента **qryReport** свойство **Active = True**. Добавим в качестве сетки ячеистую сетку **DbAltGrid** с палитры **Quasidata** и придадим ей свойства в Инспекторе объектов:

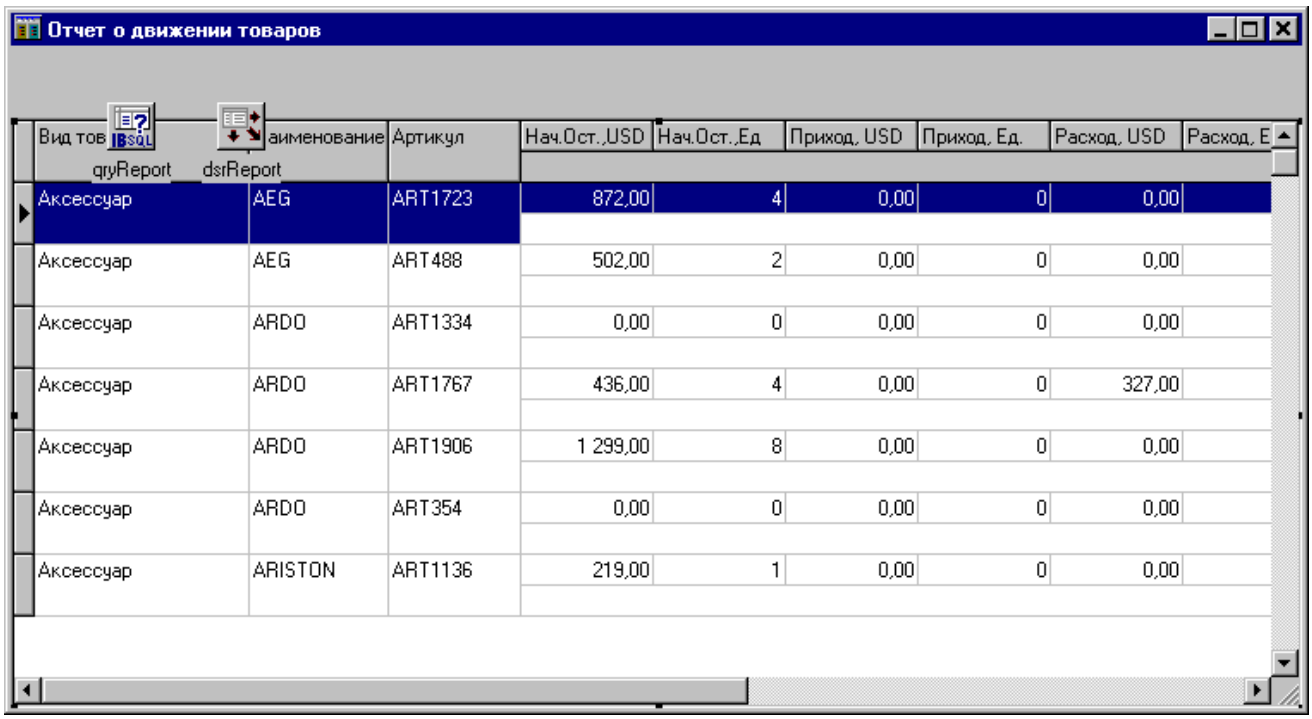
Свойство	**Значение**
Align	alClient
DataSource	dsrReport
Options	добавим dgRowSelect
AltOptions	Добавим dagColumnAltResize

Дважды щелкнем на свойстве Columns для того, чтобы вызвать редактор колонок сетки. Добавим все колонки в список при помощи соответствующего пункта контекстного меню:

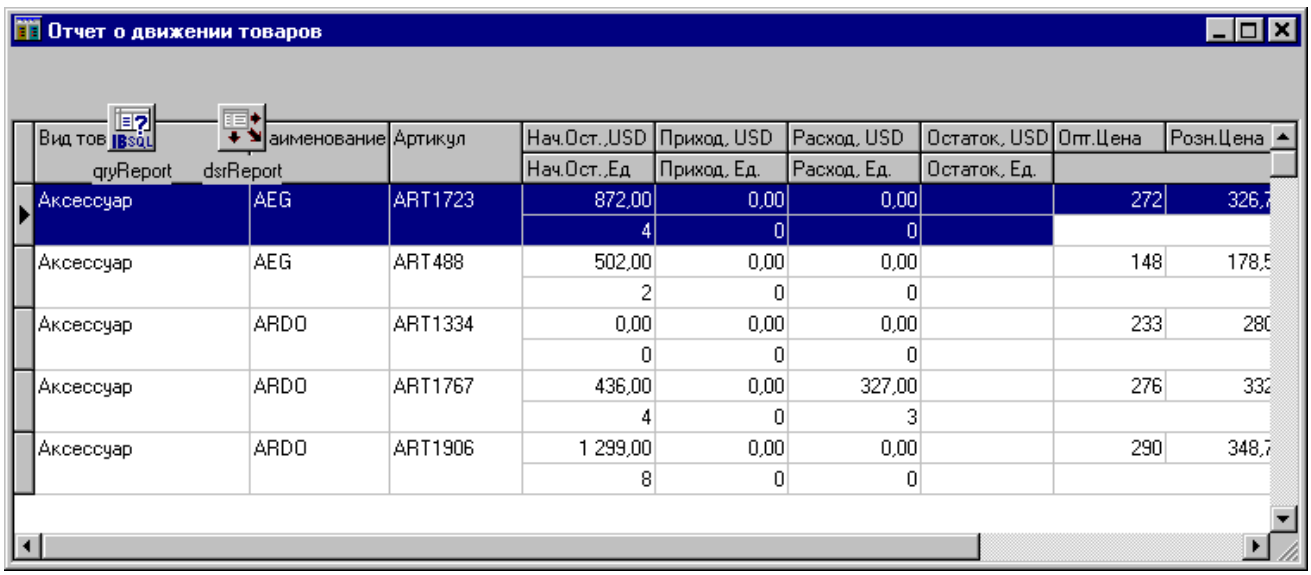


Удалим из списка колонку OBJECT_ID, так как мы не хотим ее отображать и закроем редактор колонок. Мы создали «постоянные» (persistent) колонки и теперь можем установить их свойства.

Компонент **DbAltGrid** позволяет расположить ячейки в пределах одной строки произвольно. Увеличим размер ячеек «Вид товара», «Марка» и «Артикул». Для этого подводим курсор мыши к нижнему краю заголовка колонки, и когда вид курсора мыши изменится, нажимаем левую кнопку мыши, далее, удерживая ее, «тянем» границу заголовка вниз. Ячейка увеличивает свой размер, причем дискретно. Установим размер ячеек этих трех колонок равным двойному размеру строки сетки по умолчанию:



А теперь «перетащим» заголовки колонок количеств, расположив каждое количество под соответствующей суммой (остатки под остатками, приходы под приходами и т.д.). Для «перетаскивания» заголовка колонки подводим к его центру курсор мыши, нажимаем левую кнопку и, удерживая ее, перемещаем мышью туда, куда нам нужно, после чего кнопку мыши отпускаем. Эта технология носит название **drag and drop**. В результате мы должны получить примерно такой вид сетки:



Увеличим размер ячейки колонки средней себестоимости до двойной высоты (как ячейки колонок «вид товара» и перетащим ее влево, расположив перед колонками цен.

Перетащим заголовки колонок процентов наценки, расположив каждый под соответствующей ценой.

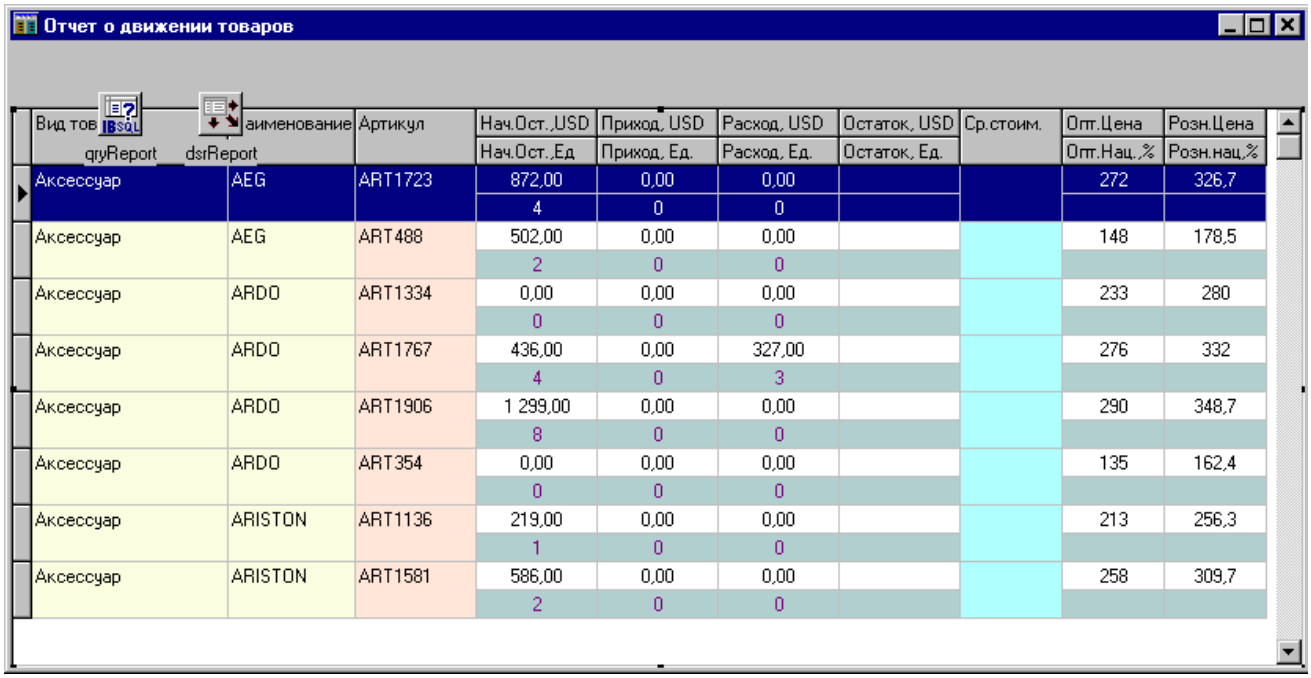
Сохраним все изменения .

Вызовем еще раз редактор колонок сетки, выберем все колонки, кроме вида товара, марки и артикула и установим в Инспекторе объектов для них выравнивание по центру **Alignment = taCenter**.

Выберем в редакторе колонок все колонки количеств (в их именах присутствует буква Q) и колонки процентов наценок (в их именах присутствует слово RATE). Одним словом, мы выбрали все колонки, ячейки которых оказались у нас в нижнем ряду. Изменим в Инспекторе объектов цвет **Color** колонки на серо-голубой, а цвет шрифта колонки **Font.Color** на **clPurple** (пурпурный).

Раскрасим по вкусу остальные колонки, устанавливая у них свойство **Color**.

Стараемся избегать кричащих цветов, так как большие поверхности ярких цветов будут утомлять глаза пользователя:



Вид тов	Наименование	Артикул	Нач.Ост.,USD / Нач.Ост.,Ед	Приход, USD / Приход, Ед.	Расход, USD / Расход, Ед.	Остаток, USD / Остаток, Ед.	Ср.стоим.	Опт.Цена / Опт.Нац.,%	Розн.Цена / Розн.нац.,%
Аксессуар	AEG	ART1723	872,00 / 4	0,00 / 0	0,00 / 0			272	326,7
Аксессуар	AEG	ART488	502,00 / 2	0,00 / 0	0,00 / 0			148	178,5
Аксессуар	ARDO	ART1334	0,00 / 0	0,00 / 0	0,00 / 0			233	280
Аксессуар	ARDO	ART1767	436,00 / 4	0,00 / 0	327,00 / 3			276	332
Аксессуар	ARDO	ART1906	1 299,00 / 8	0,00 / 0	0,00 / 0			290	348,7
Аксессуар	ARDO	ART354	0,00 / 0	0,00 / 0	0,00 / 0			135	162,4
Аксессуар	ARISTON	ART1136	219,00 / 1	0,00 / 0	0,00 / 0			213	256,3
Аксессуар	ARISTON	ART1581	586,00 / 2	0,00 / 0	0,00 / 0			258	309,7

Обратите внимание, что значения вычисляемых полей не отображаются в режиме дизайна. Только при запуске проекта тот текст, что мы прописали в обработчик **OnCalcFields** начнет работать и вычисляемые поля получат свои значения.

Теперь займемся органами управления. Нам необходима во-первых кнопка «Запрос». Добавим с палитры **Standard** кнопку **Button** на верхнюю панель нашей формы. Назначим ей свойства **Caption = Запрос** и **Name = btnQuery**. Создадим ей обработчик события **OnClick**:

```
procedure TGoodsFlowForm.btnQueryClick(Sender: TObject);
begin
  Screen.Cursor := crHourGlass; //изменяем курсор мыши на "часы"
  try
    qryReport.Close;
    qryReport.Open;
  finally
    Screen.Cursor := crDefault; //восстанваливаем курсор мыши
  end;
end;
```

Мы изменяем курсор мыши на «часы» на то время, пока отрабатывается запрос, чтобы пользователь понял, что нужно подождать. Хотя ждать ему придется недолго - наш запрос работет очень быстро. Закроем запрос в режиме дизайна, установив **Active = False**. Запустим проект (F9). После нажатия кнопки «Запрос», запрос должен открыться, а в вычисляемых полях - появиться значения.

Однако пока мы запрашиваем отчет на фиксированные даты '01.06.2003', '15.07.2003' , которые мы прописали явно в тексте запроса. Нам нужно запросить отчет за произвольный период дат. Поэтому вернемся в режим дизайна и изменим текст запроса в компоненте **qryReport**, заменив эти две даты на параметры **:DATE1** и **:DATE2**.

```
…..
  goods_flow(:DATE1, :DATE2) gf,
…..
```

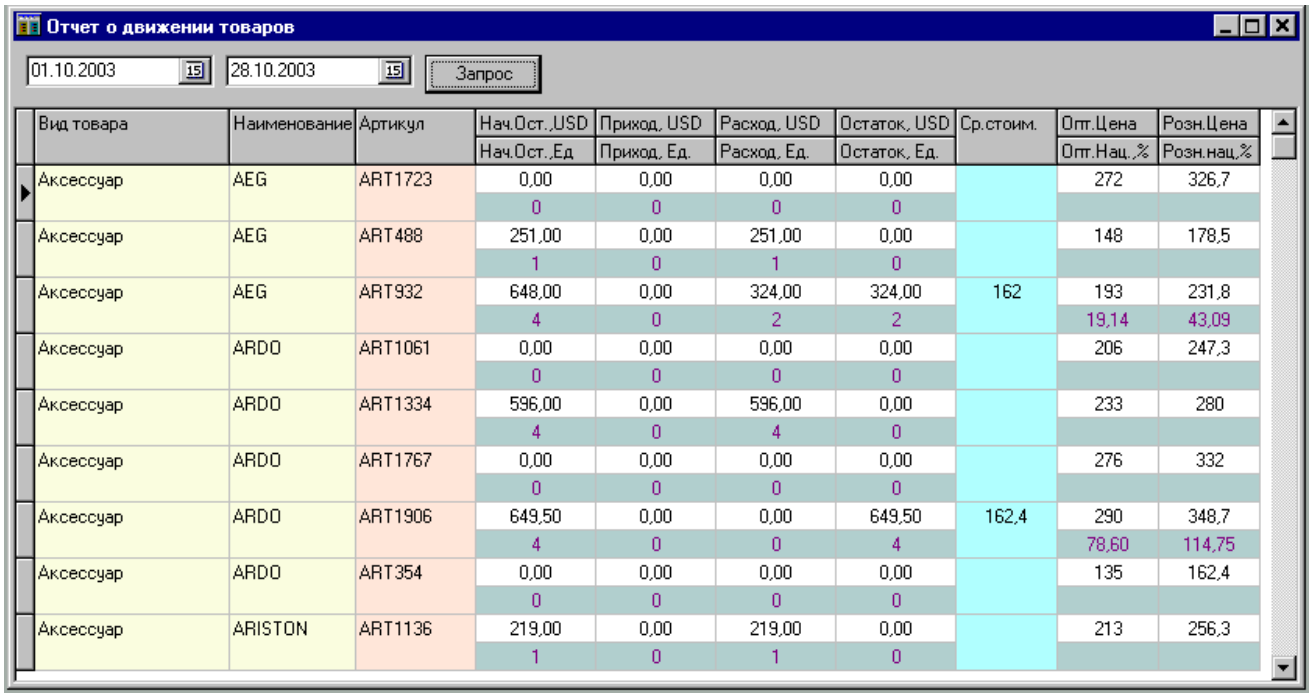
Добавим на верхнюю панель два компонента **DateEdit** с палитры **RX Controls**, и изменим текст обработчика **OnClick** кнопки «Запрос». Добавленные команды выделены здесь жирным шрифтом:

```
procedure TGoodsFlowForm.btnQueryClick(Sender: TObject);
begin
  Screen.Cursor := crHourGlass; //изменяем курсор мыши на "часы"
  try
    qryReport.Close;
    qryReport.ParamByName('DATE1').AsDateTime := DateEdit1.Date;
    qryReport.ParamByName('DATE2').AsDateTime := DateEdit2.Date;
    qryReport.Open;
  finally
    Screen.Cursor := crDefault; //восстанваливаем курсор мыши
  end;
end;
```

Создадим для формы **GoodsFlowForm** обработчик события **OnCreate**:

```
procedure TGoodsFlowForm.FormCreate(Sender: TObject);
begin
  DateEdit1.Date := EncodeDate(YearOf(Date), MonthOf(Date), 1);
  DateEdit2.Date := Date;
end;
```

Запустим проект и нажмем кнопку «Запрос»:

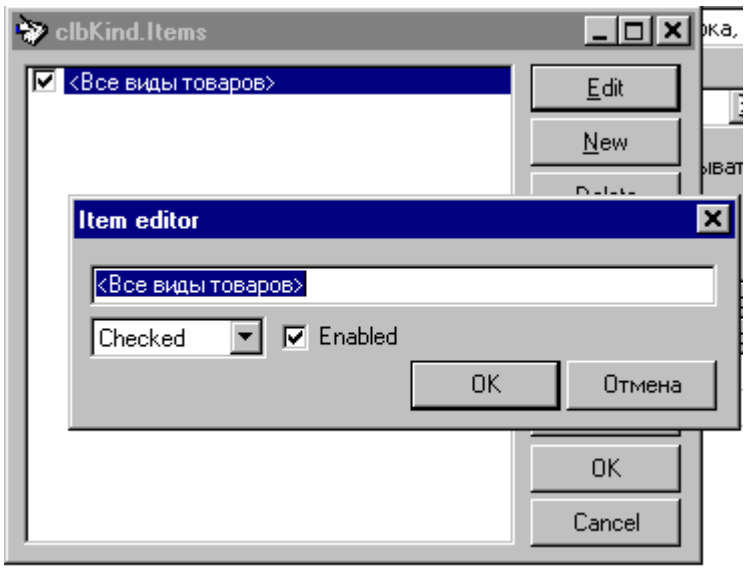


Вид товара	Наименование	Артикул	Нач.Ост.,USD / Нач.Ост.,Ед	Приход, USD / Приход, Ед.	Расход, USD / Расход, Ед.	Остаток, USD / Остаток, Ед.	Ср.стоим.	Опт.Цена / Опт.Нац.,%	Розн.Цена / Розн.нац.,%
Аксессуар	AEG	ART1723	0,00 / 0	0,00 / 0	0,00 / 0	0,00 / 0		272	326,7
Аксессуар	AEG	ART488	251,00 / 1	0,00 / 0	251,00 / 1	0,00 / 0		148	178,5
Аксессуар	AEG	ART932	648,00 / 4	0,00 / 0	324,00 / 2	324,00 / 2	162	193 / 19,14	231,8 / 43,09
Аксессуар	ARDO	ART1061	0,00 / 0	0,00 / 0	0,00 / 0	0,00 / 0		206	247,3
Аксессуар	ARDO	ART1334	596,00 / 4	0,00 / 0	596,00 / 4	0,00 / 0		233	280
Аксессуар	ARDO	ART1767	0,00 / 0	0,00 / 0	0,00 / 0	0,00 / 0		276	332
Аксессуар	ARDO	ART1906	649,50 / 4	0,00 / 0	0,00 / 0	649,50 / 4	162,4	290 / 78,60	348,7 / 114,75
Аксессуар	ARDO	ART354	0,00 / 0	0,00 / 0	0,00 / 0	0,00 / 0		135	162,4
Аксессуар	ARISTON	ART1136	219,00 / 1	0,00 / 0	219,00 / 1	0,00 / 0		213	256,3

Нам нужно еще реализовать фильтрацию в отчете. И хорошо бы иметь возможность сортровать записи в отчете, предоставив выбор способа сортировки пользователю.

Вернемся в режим дизайна. Увеличим высоту верхней панели до 152 пикселей.

Добавим три компонента **RxCheckListBox** с палитры RX **Controls**. Назовем компоненты **clbKind**, **clbMark** и **clbStock**. Они будут нам служить органами управления фильтрацией по видам товаров, маркам и складам. Выберем компонент **clbKind** и дважды щелкнем на его свойстве Items в Инспекторе объектов. Появится специальный редактор списка, в котором добавим элемент в список, нажав клавишу **New**. Назовем пункт **<Все виды товаров>** и выберем **Checked** в выпадающем списке:



Аналогично создадим один элемент списка в компоненте **clbMark** с названием **<Все марки>**.
Аналогично создадим один элемент списка в компоненте **clbStock** с названием **<Все склады>**.
Добавим на панель компонент **ComboBox** с палитры **Standard** и придадим ему свойства:

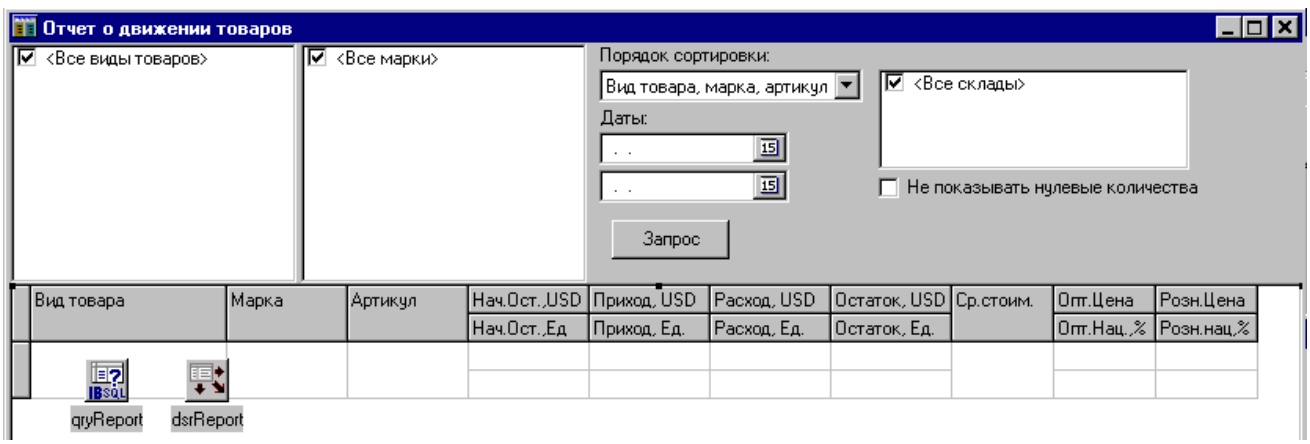
Свойство	**Значение**
Name	cbOrder
Items	Вид товара, марка, артикул Марка, вид товара, артикул
Style	csDropDownList
ItemIndex	0

Добавим два компонента **Label** с палитры **Standard** и придадим их свойствам **Caption** значения ‘Порядок сортировки’ и ‘Даты’.

Добавим компонент **CheckBox** с палитры **Standart** и придадим ему свойства:

Свойство	**Значение**
Name	cbNotZero
Caption	Не показывать нулевые количества

Расположим управляющие элементы так, как показано на рисунке:



Добавим три компонента **IBQuery** с палитры **InterBase** и придадим им свойства:

Свойство	**Значение**
Name	qryGoodsKind
Transaction	MainConnection.MainTransaction
SQL	select ID,NAME from GOODS_KIND where ID <> 0 order by NAME

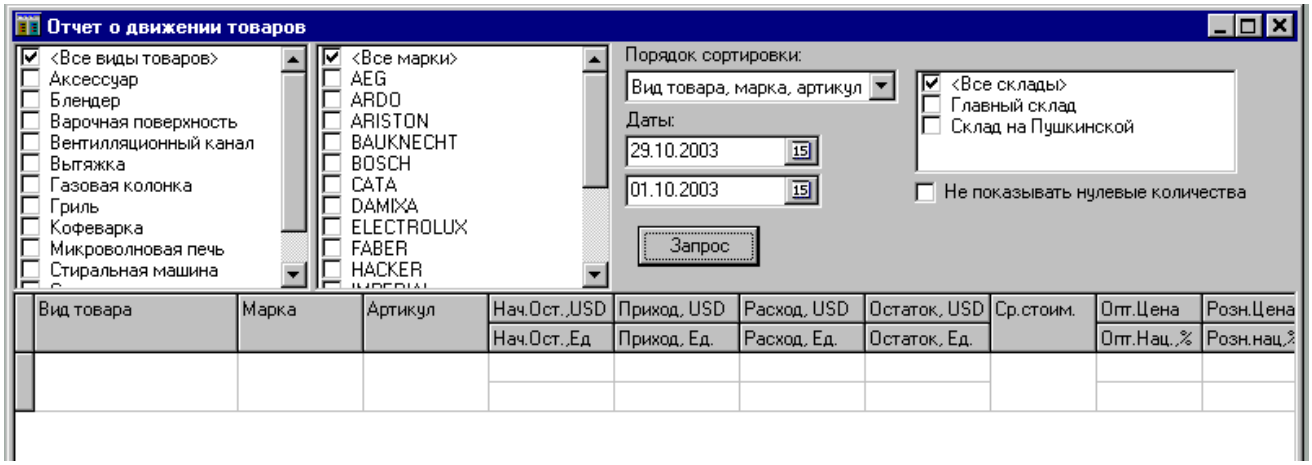
Свойство	**Значение**
Name	qryGoodsMark
Transaction	MainConnection.MainTransaction
SQL	select ID,NAME from GOODS_MARK where ID <> 0 order by NAME

Свойство	**Значение**
Name	qryStock
Transaction	MainConnection.MainTransaction
SQL	select ACC_ID,NAME from ACC where PARENT_ID = 1005 order by ACC_ORDER

Добавим в обработчик события **OnCreate** формы код, выделенный жирным шрифтом:

```
procedure TGoodsFlowForm.FormCreate(Sender: TObject);
begin
  DateEdit1.Date := EncodeDate(YearOf(Date), MonthOf(Date), 1);
  DateEdit2.Date := Date;
  with qryGoodsKind do
  begin
    Open;
    while not Eof do //заполняем список видов товаров
    begin
      clbKind.Items.Add(Fields[1].AsString);
      Next;
    end;
  end;

  with qryGoodsMark do
  begin
    Open;
    while not Eof do //заполняем список марок
    begin
      clbMark.Items.Add(Fields[1].AsString);
      Next;
    end;
  end;

  with qryStock do
  begin
    Open;
    while not Eof do //заполняем список складов
    begin
      clbStock.Items.Add(Fields[1].AsString);
      Next;
    end;
  end;
end;
```

Запустим проект:



Мы видим, что списки заполнились видами товаров и их марками из соответствующих справочников, а список складов - из системной таблицы ACC. Нам осталось организовать саму фильтрацию. Если пользователь оставит птичку на пункте «Все виды товаров», то фильтрация по видам производиться не должна, если же он уберет эту птичку и отметит птичками какие-то конкретные виды товаров, то отчет должен включить только товары отмеченных видов. Аналогично должны работать фильтры марок и складов.

Для того чтобы это реализовать нам понадобится просканировать каждый список, найти отмеченные птичками пункты и собрать строку фильтрации, которую мы вставим в SQL-запрос отчета во время выполнения программы, непосредственно перед тем, как открыть запрос.

Создадим константу в разделе **implementation**, скопировав текст запроса из компонента **qryReport** и несколько его видоизменив:

```
const

  SQL_GOODS_FLOW_REPORT =
      'select'#13+
      '  gf.object_id,'#13+
      '  gk.name goods_kind_name,'#13+
      '  gm.name goods_mark_name,'#13+
      '  g.article,'#13+
      '  sum(gf.a) a, sum(gf.q) q, sum(gf.ad) ad, sum(gf.ac) ac,'#13+
      '  sum(gf.qd) qd, sum(gf.qc) qc,'#13+
      '  g.price_w, g.price_r'#13+
      'from'#13+
      '  goods_flow(:DATE1, :DATE2) gf,'#13+
      '  goods g,'#13+
      '  goods_kind gk,'#13+
      '  goods_mark gm'#13+
      'where'#13+
      '  gf.object_id = g.id and'#13+
      '  g.goods_kind = gk.id and'#13+
      '  g.goods_mark = gm.id'#13+
      '%s'+            //здесь будет фильтр по видам, маркам и складам
      'group by'#13+
      '  gf.object_id, gk.name, gm.name, g.article, g.price_w, g.price_r'#13+
      '  %s'+          //здесь будут отсеиваться нулевые количества
      'order by %s'; //здесь будет упорядочивание
```

Как можно заметить, мы исключили явное указание склада **gf.acc_id = 1007** и добавили в трех местах **%s**.

Теперь мы создадим функцию, которая будет собирать строку фильтра, анализируя отмеченные птичками пункты определенного списка.

Добавим в секцию **private** класса формы объявление функции:

```
  { Private declarations }
  function GetFilterString(ListBox: TRxCheckListBox;
    Query: TIBQuery; const FieldName: string): string;
```

Добавим в раздел **implementation** реализацию этой функции:

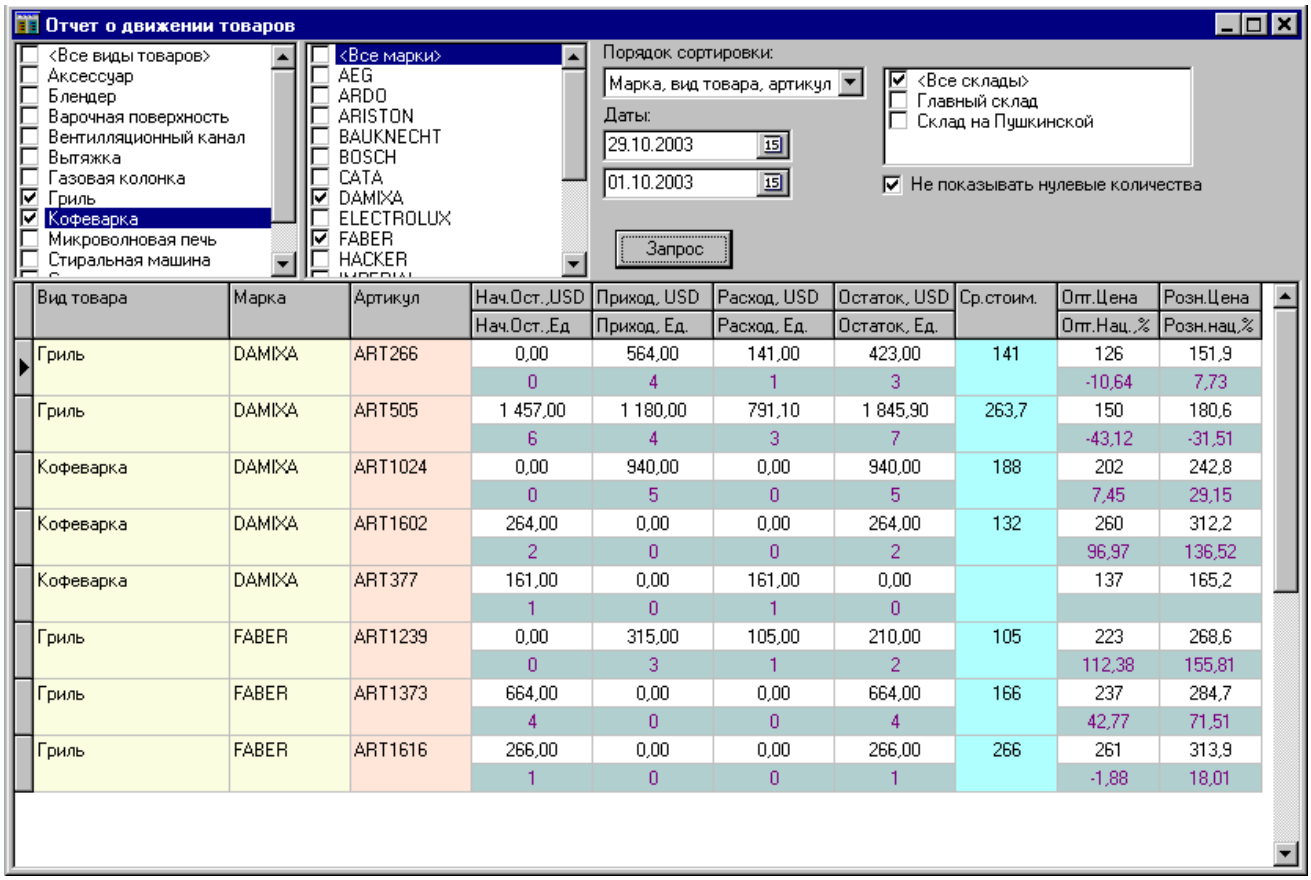
```
{Сборка строки фильтрации вида 'and <поле> in (1,5,6,11,92)' }
function TGoodsFlowForm.GetFilterString(ListBox: TRxCheckListBox;
  Query: TIBQuery; const FieldName: string): string;
begin
  Result := '';
  if ListBox.Checked[0] then
    exit;
  with Query do //сканируем набор записей
  begin
    First;
    while not Eof do
    begin
      if ListBox.Checked[RecNo] then //если соответствующий элемент
      begin                          //в списке отмечен птичкой
        if Result <> '' then Result := Result + ','; //добавляем запятую
        Result := Result + Fields[0].AsString; //добавляем в строку
      end;
      Next;
    end;
  end;
  if Result <> '' then //окончательно форматируем условие фильтрации
    Result := Format('and %s in (%s)'#13, [FieldName, Result]);
end;
```

Добавим в обработчик события **OnClick** кнопки «Запрос» код, выделенный жирным шрифтом:

```
procedure TGoodsFlowForm.btnQueryClick(Sender: TObject);
var
  s_filter, s_having, s_orderby: string;
begin
  Screen.Cursor := crHourGlass; //изменяем курсор мыши на "часы"
  try
    qryReport.Close;
    {Собираем строку по всем трем фильтрам}
    s_filter := GetFilterString(clbKind, qryGoodsKind, 'g.goods_kind') +
              GetFilterString(clbMark, qryGoodsMark, 'g.goods_mark') +
              GetFilterString(clbStock, qryStock, 'gf.acc_id');
    if cbNotZero.Checked then //не показывать нулевые количества
      s_having :=  'having sum(gf.q) <> 0 or sum(gf.qd) <> 0 or sum(gf.qc) <> 0'#13
    else
      s_having := '';

    case cbOrder.ItemIndex of
    1: s_orderby := '3,2,4';
    else
      s_orderby := '2,3,4';
    end;

    qryReport.SQL.Text := Format(SQL_GOODS_FLOW_REPORT,
                    [s_filter, s_having, s_orderby]);

    // ShowMessage(qryReport.SQL.Text); //отладочный вывод
```

```
  qryReport.ParamByName('DATE1').AsDateTime := DateEdit1.Date;
  qryReport.ParamByName('DATE2').AsDateTime := DateEdit2.Date;
  qryReport.Open;
 finally
  Screen.Cursor := crDefault; //восстанваливаем курсор мыши
 end;
end;
```

Запустим проект. Снимем птичку с пункта «Все виды товаров» и установим птички на конкретные виды. Аналогично установим фильтр на несколько марок. Установим птичку «Не показывать нулевые количества» и поменяем порядок сортировки на «Марка, вид, артикул». Нажмем кнопку «Запрос»:

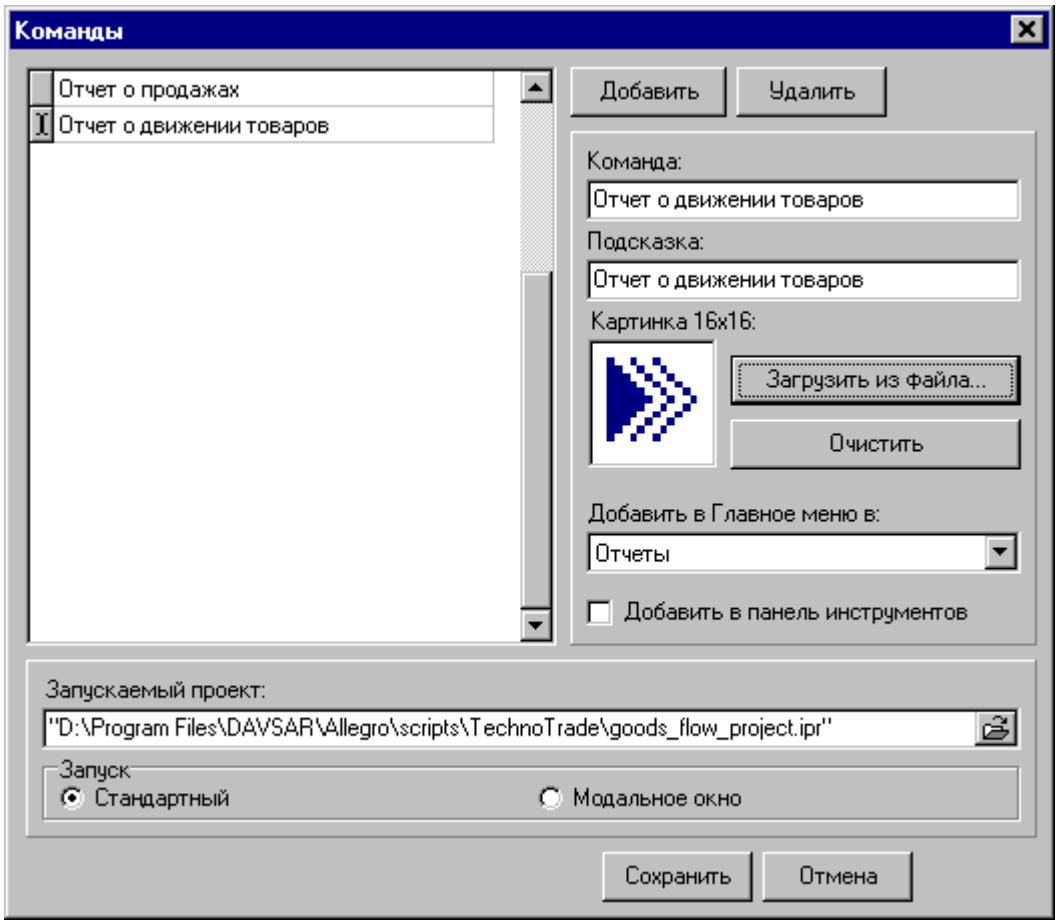


В принципе наш отчет готов (осталось реализовать лишь экспорт в Excel).

Можно подключить отчет к Главному меню Allegro, даже не выходя полностью из режима «Дизайнера». Важно лишь, чтобы главное окно было в растянутом положении.

Для подключения отчета к Главному меню вызовем окно «Команды» через меню **Инструменты/Команды** и создадим там новый пункт под названием «Отчет о движении товаров». Выберем какую-нибудь картинку формата BMP 16x16 и установим остальные поля диалога:

Команда	Отчет о движении товаров
Подсказка	Отчет о движении товаров
Добавить в главное меню	Отчеты
Запускаемый проект	D:\Program Files\DAVSAR\Allegro\scripts\TechnoTrade\goods_flow_project.ip
Способ запуска	Стандартный



Выйдем из дизайнера и попробуем вызвать наш отчет через главное меню. Запустим отчет и вызовем окно SQL-монитора. Установим в отчете фильтрацию и посмотрим, как выглядят тексты отрабатываемых SQL-запросов в SQL-мониторе. Например, мы запросили 1 вид товара и три марки, требуя ненулевые количества:

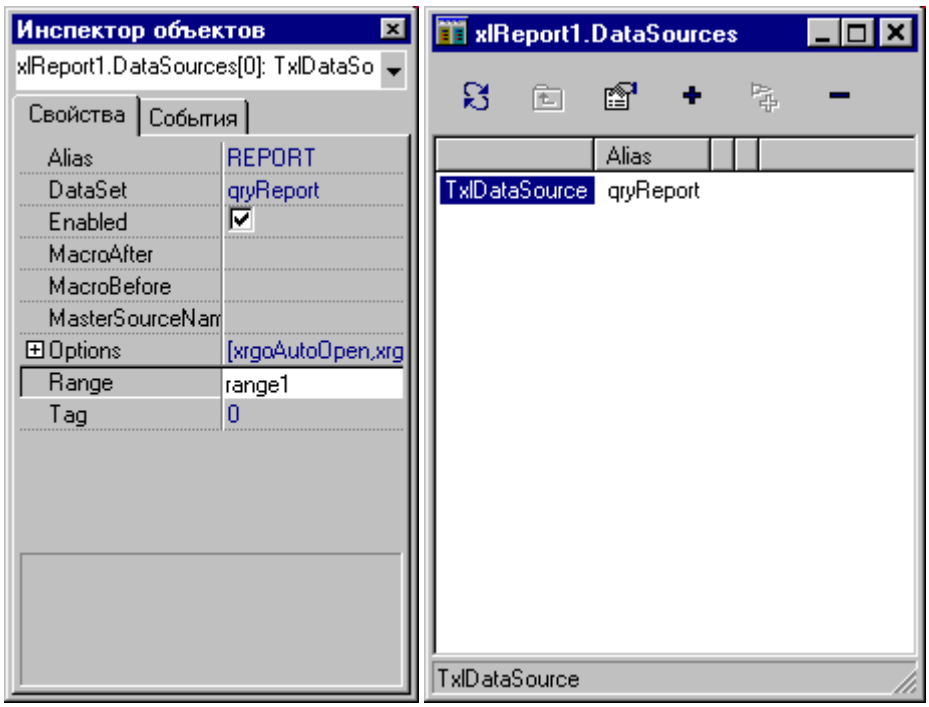
```
select
  gf.object_id,
  gk.name goods_kind_name,
  gm.name goods_mark_name,
  g.article,
  sum(gf.a) a, sum(gf.q) q, sum(gf.ad) ad, sum(gf.ac) ac,
  sum(gf.qd) qd, sum(gf.qc) qc,
  g.price_w, g.price_r
from
  goods_flow(:DATE1, :DATE2) gf,
  goods g,
  goods_kind gk,
  goods_mark gm
where
  gf.object_id = g.id and
  g.goods_kind = gk.id and
  g.goods_mark = gm.id
  and g.goods_kind in (8)
  and g.goods_mark in (25,26,27)
group by
  gf.object_id, gk.name, gm.name, g.article, g.price_w, g.price_r
having sum(gf.q) <> 0 or sum(gf.qd) <> 0 or sum(gf.qc) <> 0
order by 2,3,4
```

Отчет о движении товаров, экспорт в Excel

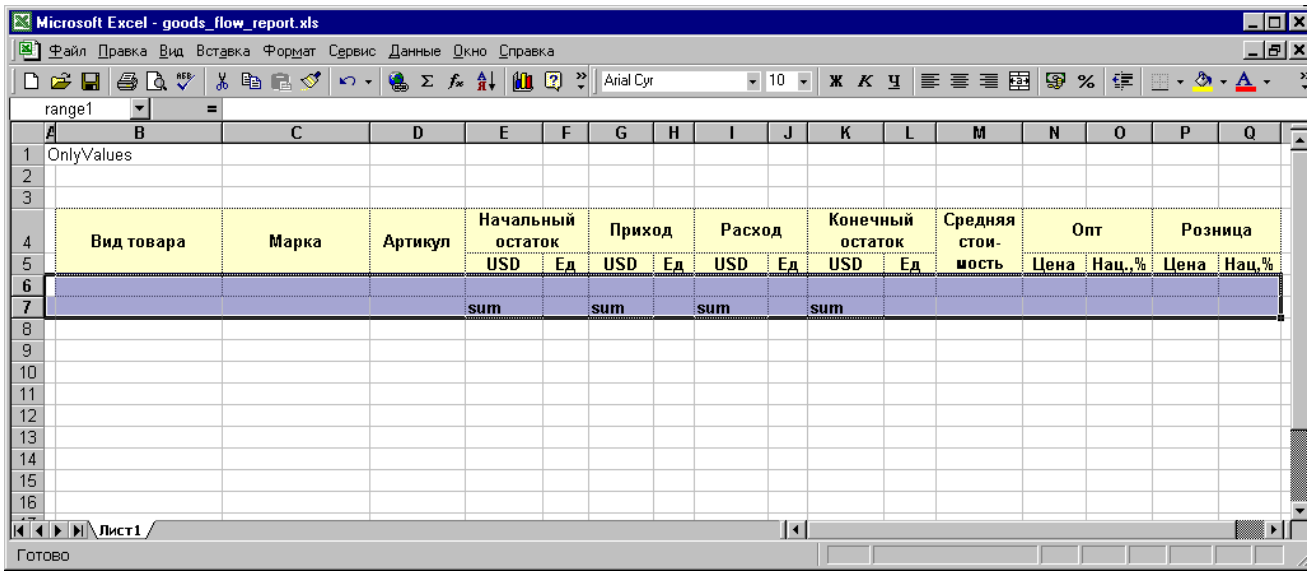
Включим режим «Дизайнер» и откроем проект **goods_flow_project.ipr**.

Добавим компонент **xlReport** с палитры **Print** и вызовем в Инспекторе объектов редактор свойства **DataSources**. В появившийся список источников данных добавим новый элемент и назначим ему свойства:

Свойство	Значение
DataSet	qryReport
Alias	REPORT
Range	range1



Закроем редактор источников данных и дважды щелкнем на компоненте xlReport1, чтобы вызвать на редактирование новый шаблон документа Excel. Удалим в нем лишние листы, оставив только «Лист1». В самой верхней левой ячейке листа напишем опцию **OnlyValues**. Ниже расположим заголовки будущих колонок отчета, оставив самую левую колонку пустой. Ниже заголовков выделим две строки (захватив левую пустую колонку) и обозначим выделенные ячейки, как регион **range1**.



Сохраним шаблон под названием **goods_flow_report.xls** и укажем название этого файла в свойстве **XLSTemplate** компонента **xlReport1**. Введем в ячейки первой строки региона range1 названия полей отчета в таком виде:

Вид товара	=REPORT_GOODS_KIND_NAME
Марка	=REPORT_GOODS_MARK_NAME
Артикул	=REPORT_ARTICLE
Начальный остаток, USD	=REPORT_A
Начальный остаток, Единиц	=REPORT_Q
Приход, USD	=REPORT_AD
Приход, Единиц	=REPORT_QD
Расход, USD	=REPORT_AC
Расход, Единиц	=REPORT_QC
Конечный остаток, USD	=REPORT_CA
Конечный остаток, Единиц	=REPORT_CQ
Средняя себестоимость	=REPORT_COST
Опт, Цена	=REPORT_PRICE_W
Опт, Наценка,%	=REPORT_W_RATE
Розница, Цена	=REPORT_PRICE_R
Розница, Наценка, %	=REPORT_R_RATE

Название каждого поля отчета состоит из псевдонима источника данных (свойство Alias) и имени поля источнике данных, разделенных знаком подчеркивания «_».

Для того чтобы не загромождать отчет нулями, установим формат # ### для всех числовых полей отчета. В колонках USD во второй строке региона введем слово **sum**. Компонент **xlReport1** выведет в эти ячейки суммы соответствующих колонок. Для всех полей отчета, кроме «Вида товара» и полей USD, зададим в формате ячеек выравнивание по горизонтали по центру.

Вызовем «Параметры страницы» шаблона. На закладке «страница» выберем альбомную ориентацию. На закладке «Колонтитулы» выберем верхний колонтитул «Страница 1 из ?». На закладке «Лист» укажем заголовки колонок отчета в качестве сквозных строк.

Вызовем предварительный просмотр страницы и раздвинем поля так, чтобы отчет помещался в лист А4. Сохраним шаблон и закроем Excel.

Добавим на форму **GoodsFlowForm** кнопку **BitBtn** с палитры **Additional**.

Свойство	Значение
Caption	Отчет
Name	btnReport

Загрузим в свойство **Glyph** пиктограмму Excel.

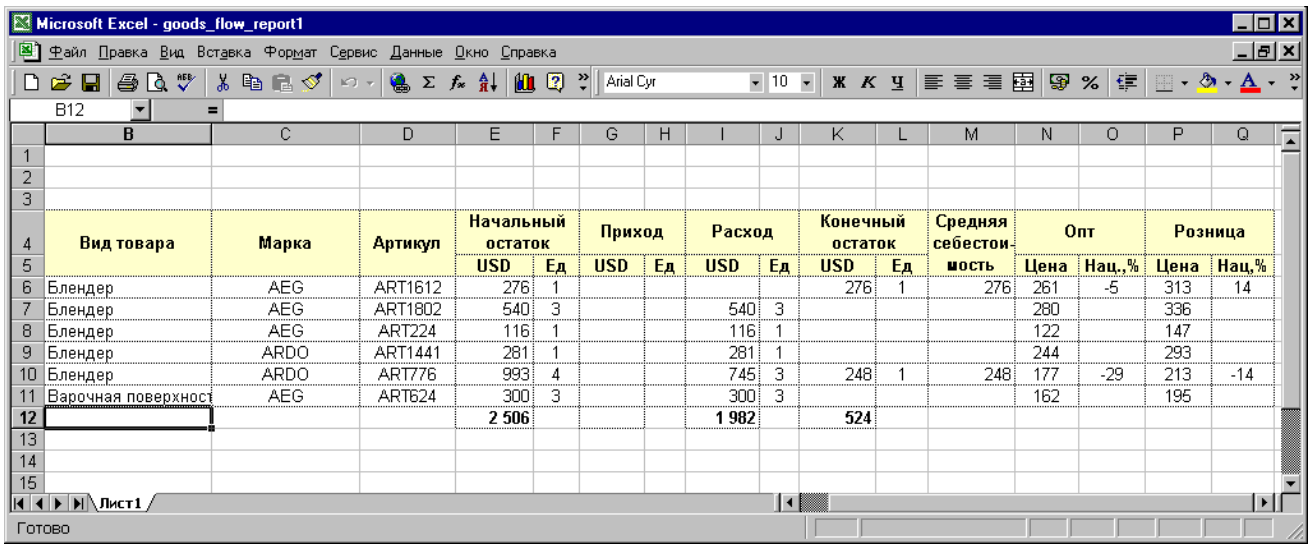
Создадим обработчик события **OnClick** для кнопки **btnReprt** и впишем в него такой текст:

```
procedure TGoodsFlowForm.btnReportClick(Sender: TObject);
begin
  xlReport1.Report;
end;
```

Создадим обработчик события OnProgress для компонента **xlReport1** и впишем в него такой текст:

```
procedure TGoodsFlowForm.xlReport1Progress(Report: TxlReport; const Position, Max: Integer);
begin
  if Position = 0 then
    StatusBarDisplay(clBlack, Position, 0, Max, '', '', '')
  else
    StatusBarDisplay(clLime, Position, 0, Max, '', 'Экспорт в Excel...', '');
end;
```

Запустим проект. Выберем какие-то виды и марки товаров. Нажмем кнопку «Запрос». Затем нажмем кнопку «Отчет». Должен появиться документ Excel примерно такого вида:



Вид товара	Марка	Артикул	Начальный остаток		Приход		Расход		Конечный остаток		Средняя себестоимость	Опт		Розница	
			USD	Ед	USD	Ед	USD	Ед	USD	Ед		Цена	Нац.,%	Цена	Нац.%
Блендер	AEG	ART1612	276	1					276	1	276	261	-5	313	14
Блендер	AEG	ART1802	540	3			540	3				280		336	
Блендер	AEG	ART224	116	1			116	1				122		147	
Блендер	ARDO	ART1441	281	1			281	1				244		293	
Блендер	ARDO	ART776	993	4			745	3	248	1	248	177	-29	213	-14
Варочная поверхност	AEG	ART624	300	3			300	3				162		195	
			2 506				**1 982**		**524**						

Наш отчет практически готов. Осталось вывести признаки фильтрации данных в отчет.
Добавим в секцию **private** класса формы объявление функции **GetFilterDescription**:

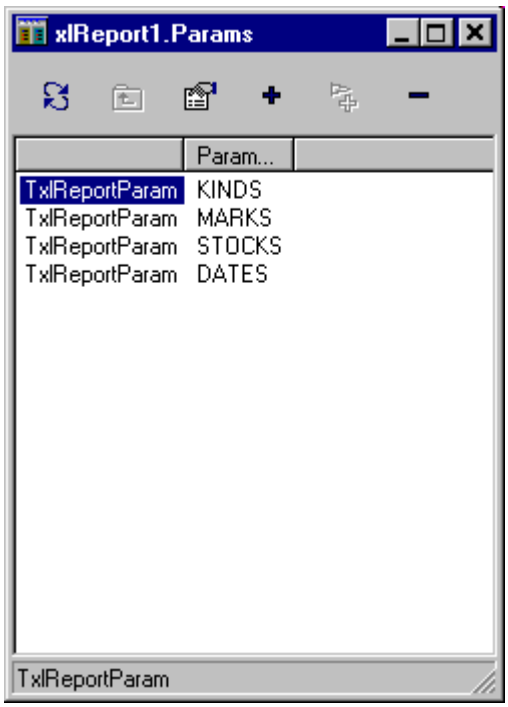
```
  private
    { Private declarations }
    function GetFilterString(ListBox: TRxCheckListBox;
      Query: TIBQuery; const FieldName: string): string;
    function GetFilterDescription(ListBox: TRxCheckListBox): string;
  public
    { Public declarations }
  end;
```

Эта функция будет возвращать нам перечисленные через запятую признаки фильтрации для вывода в отчет. Добавим в секцию **implementation** модуля реализацию этой функции:

```
{Сборка описания фильтрации для отчета}
function TGoodsFlowForm.GetFilterDescription(ListBox: TRxCheckListBox): string;
var
  i: integer;
begin
  Result := '';
  with ListBox do //сканируем список
  for i := 0 to Items.Count - 1 do
  if Checked[i] then //если соответствующий элемент в списке отмечен птичкой
  begin
    if Result <> '' then Result := Result + ', '; //добавляем запятую
    Result := Result + ListBox.Items[i]; //добавляем в строку
  end;
end;
```

Вызовем редактор свойства **Params** компонента **xlReport1**. Добавим параметры под названиями:

```
STOCKS, KINDS, MARKS, DATES
```



Изменим обработчик кнопки **btnReport**, добавив в него команды, выделенные здесь жирным шрифтом:

```
procedure TGoodsFlowForm.btnReportClick(Sender: TObject);
begin
  xlReport1.ParamByName('STOCKS').Value := 'Склады: ' + GetFilterDescription(clbStock);
  xlReport1.ParamByName('KINDS').Value := 'Виды товара: ' + GetFilterDescription(clbKind);
  xlReport1.ParamByName('MARKS').Value := 'Марки: ' + GetFilterDescription(clbMark);
  xlReport1.ParamByName('DATES').Value :=
    'Период: ' + DateToStr(DateEdit1.Date) + '..' + DateToStr(DateEdit2.Date);
  xlReport1.Report;
end;
```

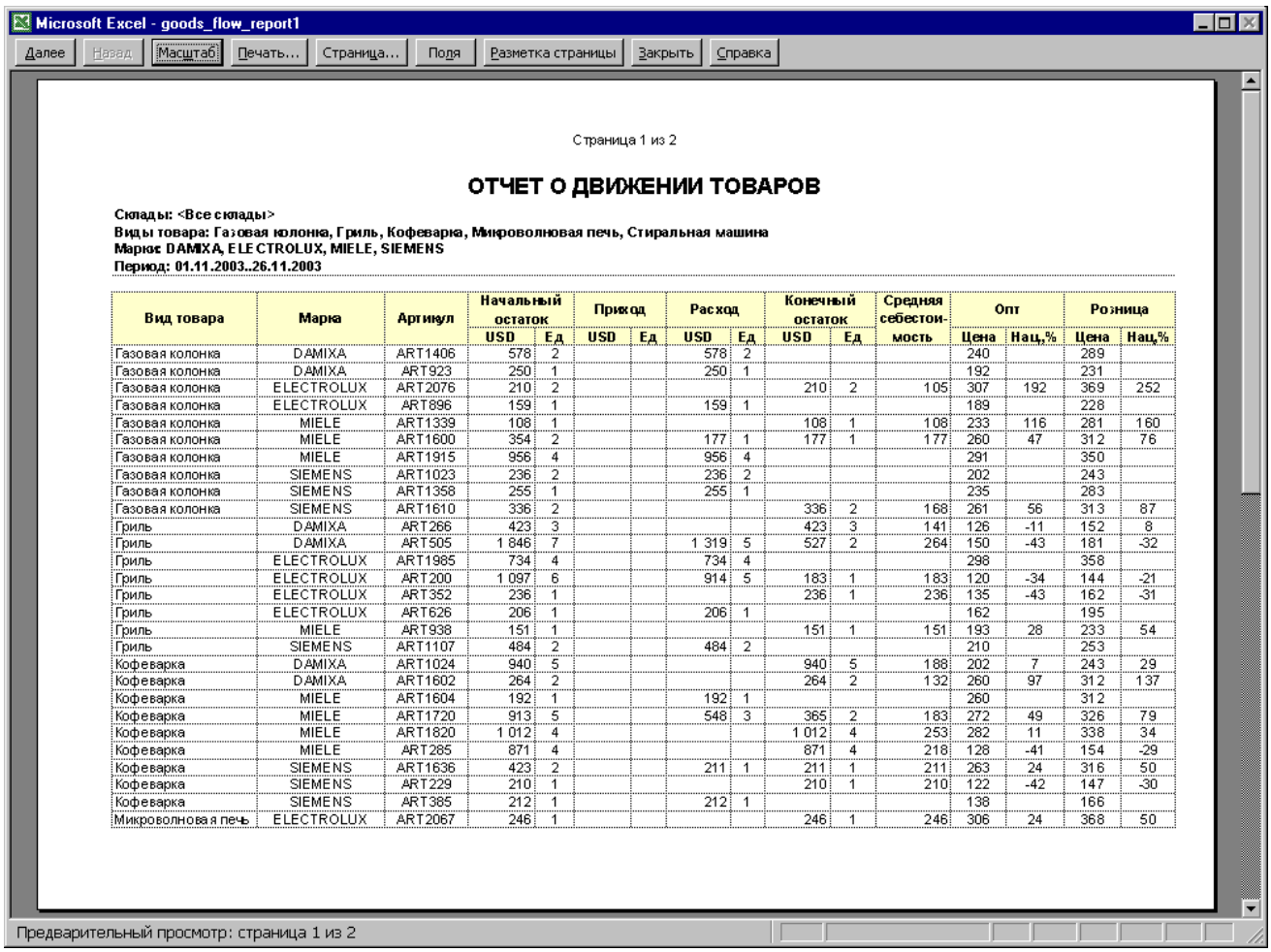
Теперь дважды щелкнем на компоненте **xlReport1** для вызова шаблона на редактирование.

Добавим пять рядов ячеек вверху отчета. Объединим ячейки в каждом ряду по всей ширине отчета. В верхней строке напишем «Отчет о движении товаров» крупным шрифтом. В остальных рядах напишем:

=XLRParams_STOCKS
=XLRParams_KINDS
=XLRParams_MARKS
=XLRParams_DATES

Придадим выравнивание влево и жирный шрифт. Сохраним шаблон и закроем Excel.

Запустим проект. Сделаем какой-нибудь запрос и выведем отчет. Вызовем «Предварительный просмотр» и попробуем распечатать отчет.



Вид товара	Марка	Артикул	Начальный остаток		Приход		Расход		Конечный остаток		Средняя себестои-мость	Опт		Розница	
			USD	Ед	USD	Ед	USD	Ед	USD	Ед		Цена	Нац,%	Цена	Нац,%
Газовая колонка	DAMIXA	ART1406	578	2			578	2				240		289	
Газовая колонка	DAMIXA	ART923	250	1			250	1				192		231	
Газовая колонка	ELECTROLUX	ART2076	210	2					210	2	105	307	192	369	252
Газовая колонка	ELECTROLUX	ART896	159	1			159	1				189		228	
Газовая колонка	MIELE	ART1339	108	1					108	1	108	233	116	281	160
Газовая колонка	MIELE	ART1600	354	2			177	1	177	1	177	260	47	312	76
Газовая колонка	MIELE	ART1915	956	4			956	4				291		350	
Газовая колонка	SIEMENS	ART1023	236	2			236	2				202		243	
Газовая колонка	SIEMENS	ART1358	255	1			255	1				235		283	
Газовая колонка	SIEMENS	ART1610	336	2					336	2	168	261	56	313	87
Гриль	DAMIXA	ART266	423	3					423	3	141	126	-11	152	8
Гриль	DAMIXA	ART505	1 846	7			1 319	5	527	2	264	150	-43	181	-32
Гриль	ELECTROLUX	ART1985	734	4			734	4				298		358	
Гриль	ELECTROLUX	ART200	1 097	6			914	5	183	1	183	120	-34	144	-21
Гриль	ELECTROLUX	ART352	236	1					236	1	236	135	-43	162	-31
Гриль	ELECTROLUX	ART626	206	1			206	1				162		195	
Гриль	MIELE	ART938	151	1					151	1	151	193	28	233	54
Гриль	SIEMENS	ART1107	484	2			484	2				210		253	
Кофеварка	DAMIXA	ART1024	940	5					940	5	188	202	7	243	29
Кофеварка	DAMIXA	ART1602	264	2					264	2	132	260	97	312	137
Кофеварка	MIELE	ART1604	192	1			192	1				260		312	
Кофеварка	MIELE	ART1720	913	5			548	3	365	2	183	272	49	326	79
Кофеварка	MIELE	ART1820	1 012	4					1 012	4	253	282	11	338	34
Кофеварка	MIELE	ART285	871	4					871	4	218	128	-41	154	-29
Кофеварка	SIEMENS	ART1636	423	2			211	1	211	1	211	263	24	316	50
Кофеварка	SIEMENS	ART229	210	1					210	1	210	122	-42	147	-30
Кофеварка	SIEMENS	ART385	212	1			212	1				138		166	
Микроволновая печь	ELECTROLUX	ART2067	246	1					246	1	246	306	24	368	50