Глава 1. Введение

О программе

Известно, что компьютерные системы учета, построенные по технологии клиент-сервер и разработанные с использованием таких систем проектирования, как Delphi, отличаются высоким быстродействием и хорошим качеством интерфейса. Большинство этих систем пишется «с нуля» под конкретный заказ, и в результате их стоимость оказывается высокой, а сами системы достаточно жесткими, требующими новых затрат при любой их модификации. В то же время большинство существующих на сегодняшний день специализированных систем бухгалтерского и складского учета, адаптируемых под задачу пользователя, не отличаются высоким быстродействием, имеют скромные возможности интерфейса и вызывают массу сложностей в тех случаях, когда учетные задачи хоть немного отличаются от стандартных.

Нам часто приходилось разрабатывать учетные программы на Delphi для тех наших заказчиков, которых не устраивали продукты, представленные на рынке стандартных бухгалтерских и складских программ. Опыт этих разработок говорит нам о том, что основная работа каждый раз состоит в постановке конкретной задачи, с учетом массы специфических моментов, отличающих бизнес-процессы заказчика. При этом непрерывное уточнение задачи заказчика, разработка (и доработка) системы, а также ее отладка, внедрение, ввод данных и обучение персонала, как правило, происходят параллельно, что создает массу сложностей для разработчика и приводит к удорожанию системы в целом.

Разрабатывая систему Allegro, мы старались создать инструмент для разработчика, отличающийся простотой, высоким быстродействием и высоким качеством оконного интерфейса одновременно. Мы старались создать такую среду, в которой, разработчик, начав решать проблемы заказчика с какой-нибудь одной неотложной и важной оперативной задачи, в дальнейшем имел бы возможность эволюционным путем довести конфигурацию до полного и высококачественного финансового учета во всей компании в целом.

Глава 2. Общие принципы

Текущее финансовое положение компании

Текущее финансовое положение компании – важнейшая информация для ее владельцев и для всех тех, кто принимает в компании управленческие решения. Хорошая система учета должна быть устроена таким образом, чтобы максимально быстро и верно представить текущее финансовое положение компании. Поэтому мы придаем столь большое значению двустороннему бухгалтерскому балансу – главному табло, на котором можно увидеть реальное сегодняшнее положение дел в целом.

Очень важно, чтобы структура баланса соответствовала специфике бизнеса компании. Система счетов у каждой компании может быть своя, совершенно непохожая на систему счетов другой компании. Часто невозможно бывает адекватно описать конкретную деятельность в рамках какого-либо стандартного набора (плана) счетов. Это вовсе не означает, что между балансами разных компаний вообще нет ничего общего. Бухгалтерский баланс строится по одним и тем же принципам и поэтому основные разделы балансов различных компаний, как правило, совпадают. Любая компания имеет денежные средства и другие активы, дебиторов (должников), кредиторов, собственный капитал и прибыль (убыток). Но структура этих разделов может быть самой непредсказуемой. К тому же эта структура может со временем изменяться и развиваться, если компания осваивает новые виды деятельности и расширяет свой бизнес. Добавление новых счетов в некий список (план) как правило требует добавления соответствующих статей в баланс или организацию иного способа воздействия финансовых результатов этих счетов на баланс. Это привело нас к идее заводить счета прямо в балансе, придав ему вид «дерева счетов», где на верхних уровнях мы бы видели наиболее общие и традиционные для всех компаний разделы, а на нижних уровнях – специфические счета конкретной компании. Если организовать дерево так, чтобы каждый верхний уровень счетов отображал сумму всех счетов нижнего уровня, то мы всегда сможем получить все финансовые результаты в наглядной форме и нужной степени подробности.

Поэтому первые требования к хорошей системе учета мы сформулировали так:

- Система всегда должна обеспечивать быстрый бухгалтерский баланс компании
- Счета, влияющие на финансовые результаты, должны создаваться прямо в балансе
- Система не должна требовать обязательной нумерации счетов
- Система счетов должна быть универсальной, нерегламентированной и иметь древовидную структуру.
- Счета должны иметь возможность переименовываться и перемещаться в пределах «дерева счетов», не нарушая при этом сам процесс учета

Требования к аналитике

Компания может иметь большое число поставщиков и покупателей, отгрузка товаров и услуг часто происходит отдельно от платежей, и важно всегда точно знать, кто кому сколько должен. Однако заводить отдельный счет для каждого из контрагентов хоть и возможно, но по ряду причин неудобно.

Например, в полном бухгалтерском балансе сумма счетов всех дебиторов изображается слева, а сумма счетов всех кредиторов - справа. Но часто один и тот же контрагент становится то дебитором, то кредитором, в зависимости от сальдо поставок и платежей по нему. Если мы заведем прямо в балансе счета всех поставщиков и покупателей, то нам придется перетаскивать их в процессе учета то влево, то вправо. Другой выход из этой ситуации видится в том, чтобы завести два «синтетических» счета в балансе: «Счета дебиторов» и «Счета кредиторов», а где-то вне баланса завести «аналитические счета» конкретных контрагентов. И при каждом начислении на «аналитический» счет контрагента делать одновременное начисление на балансовый «синтетический» счет. Именно такое решение применяется, например, в программе *LEADER Classic*.

Однако существуют ситуации, в которых даже создание аналитических счетов не спасает. Например, если мы хотим вести аналитический учет продаваемых или производимых компанией товаров. Список товаров очень скоро может стать настолько большим, что ориентироваться в таком количестве счетов будет просто невозможно.

Наконец, существуют ситуации, в которых по одним и тем же поставщикам, покупателям или товарам ведутся разные счета, так как учитываются ***разные финансовые отношения***. Например, мы хотим завести счета основных средств по «первоначальной стоимости их приобретения», а отдельно для ***тех же средств*** завести счета «аккумулированной амортизации». Тогда нам придется заводить по два счета на каждое основное средство. А как нам быстро вычесть для каждого объекта «аккумулированную амортизацию» из «первоначальной стоимости» и получить «остаточную (книжную) стоимость»? Хорошо было бы иметь систему, допускающую сочетания (суперпозицию) счет-объект, в которой по одному и тому же объекту аналитического учета могли бы учитываться одновременно несколько разных финансовых отношений или показателей на разных счетах.

Поэтому требования к хорошему аналитическому учету мы сформулировали так:

- Аналитический учет должен позволить избежать создания большого количества однотипных счетов.
- Аналитический учет должен позволить по одному и тому же ***объекту учета*** вести учет ***различных финансовых отношений***
- Аналитический учет должен позволить объединить некоторые счета во взаимосвязанные группы, в которых автоматически можно получить суммарный итог по группе для каждого объекта учета.

Глава 3. Подготовка к работе

Установка SQL-сервера Firebird

На сервере производится ***полная установка Firebird***, а на каждом клиентском компьютере – установка ***только клиента Firebird***. Если система ставится только на один компьютер (сетевая работа с базой не требуется), то необходима ***полная установка Firebird*** на этот компьютер. Инсталлятор Firebird доступен на нашем сайте http://www.gaapinvest.com

Для версии Firebird 1.5 Release Candidate он называется Firebird-1.5.0.3744-RC4-Win32.exe. Он позволяет установить Firebird на платформы **Windows 98/ME/2000/XP**. Если на Вашем компьютере уже установлен Firebird предыдущей версии, то его желательно сначала де инсталлировать.

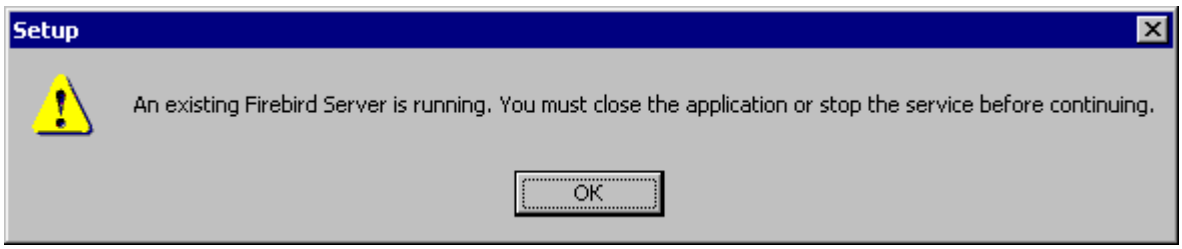
Если на Вашем компьютере установлен InterBase версии ниже 6.0, и он используется какими-то программами, то установка Firebird может нарушить работу этих программ.

Если на вашем компьютере стоит InterBase 6.0, то Firebird можно смело ставить вместо него, так как Firebird это доработанный InterBase 6.0. В этом случае прежде, чем ставить Firebird, необходимо деинсталлировать старый InterBase.

Если же на Вашем компьютере установлен Yaffil super server, и номер сборки не ниже 821, то Firebird можно вообще не ставить, так как Yaffil – это другой высококачественный сервер, потомок семейства InterBase 6.0. Программа Allegro может работать со всеми серверами версий не ниже Firebird-1.5-RC4 и Yaffil-ss-821.

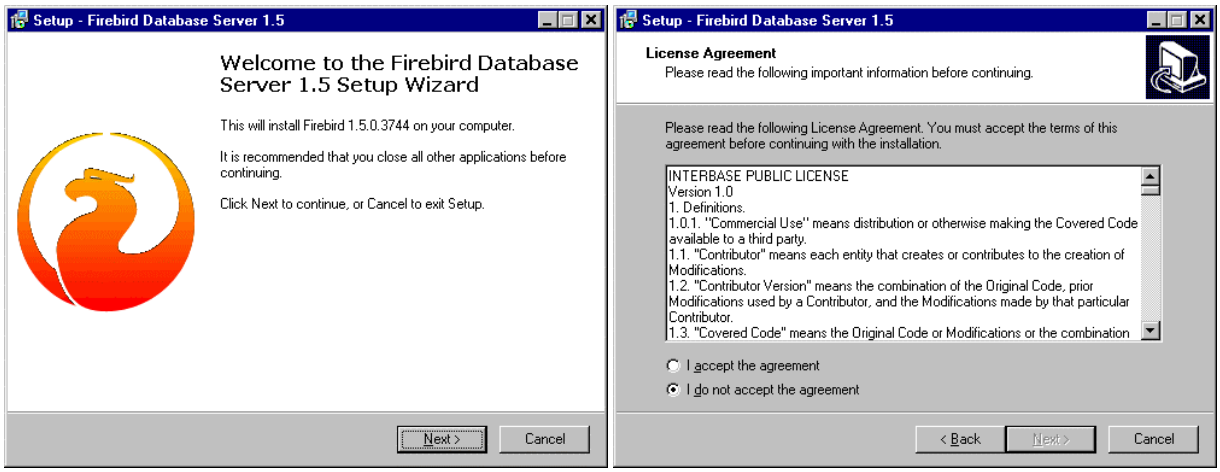
Запустите программу инсталлятора Firebird.

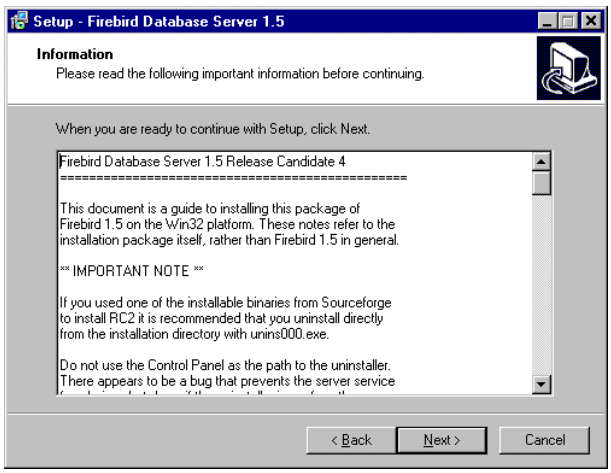
Если на компьютере уже установлен сервер баз данных Firebird, и он стартован, то инсталлятор не запустится и выдаст предупреждение о том, что прежде необходимо остановить работающий сервер Firebird. В таком случае прервите инсталляцию, остановите сервер Firebird и начните инсталляцию заново.

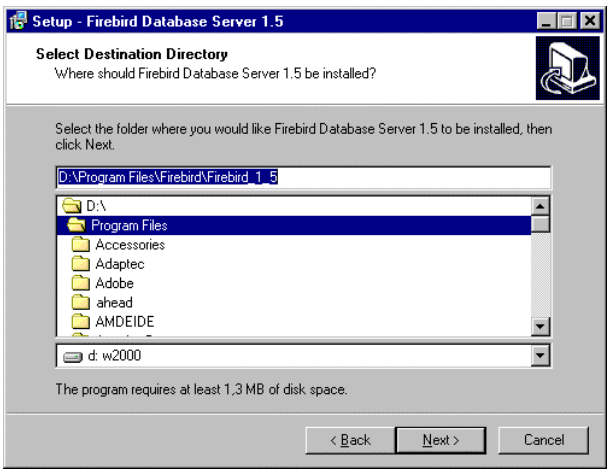


В начале инсталляции перед Вами появится окно с логотипом Firebird.
Нажмите кнопку ***Next***.

Вас попросят принять лицензионное соглашение «INTERBASE PUBLIC LICENSE».
Выберите пункт ***I accept the agreement*** и нажмите кнопку ***Next***.



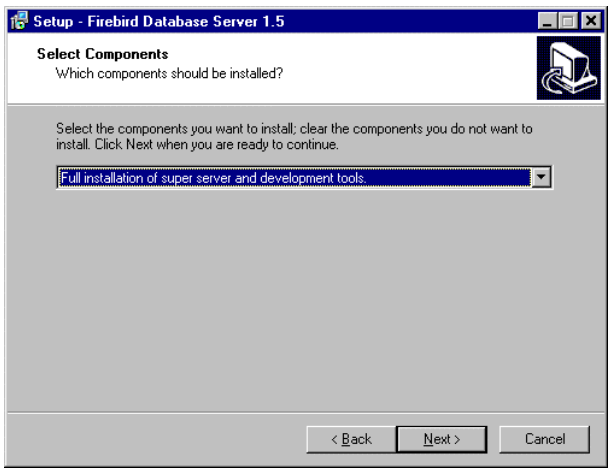


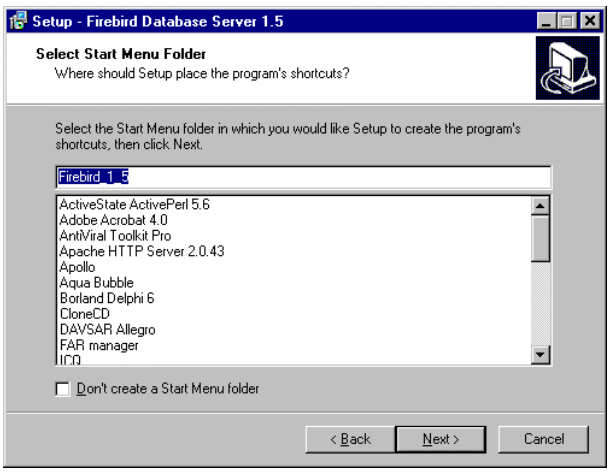


Когда появится английский текст руководства по установке Firebird, нажмите кнопку ***Next***. Задайте директорию для установки. Нажмите кнопку ***Next***.

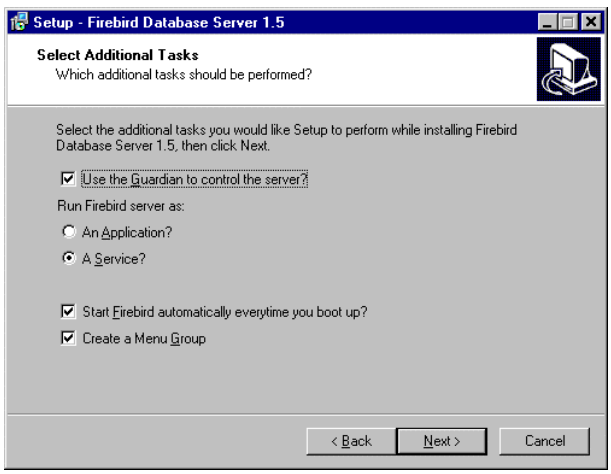
Теперь нужно выбрать, что ставить. Если мы устанавливаем Firebird на сервер, то нужно выбрать первый пункт выпадающего списка: ***Full installation of super server and development tools***.

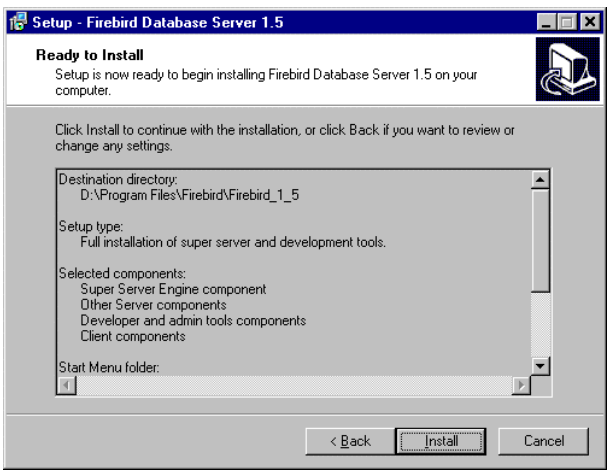
Если же мы устанавливаем только ***клиентскую часть***, то на этой стадии нужно выбрать пункт ***Minimum client install – no server, no tools***. Нажмите кнопку ***Next***. Теперь уточните название будущей папки для Firebird в меню «Пуск» и нажмите ***Next***.





Если мы делаем полную установку, то нужно указать, как ставить сервер. Дело в том, что сервер может запускаться с помощью программы Guardian, которая следит за поведением сервера и если тот «упадет», то автоматически перезапускает его. Если же Guardian не используется, то сервер запускается сам. Мы рекомендуем использовать Guardian. Поэтому птичку ***Use the Guardian to control the server*** лучше оставить.



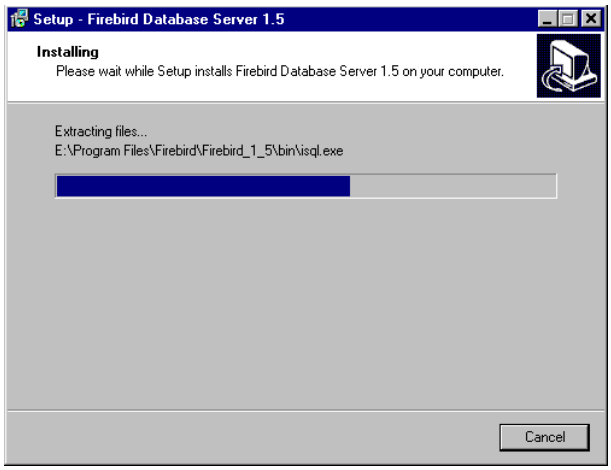


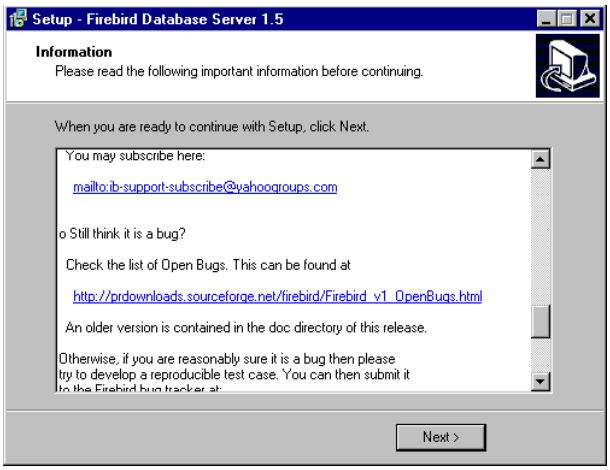
Еще следует знать, что сервер Firebird под системами Windows 98 и Windows ME работает как обычное ***приложение***, а под системами типа NT (Windows NT4, Windows 2000, Windows XP) - может работать, как ***системная служба***. Всегда желательно установить сервер именно как ***системную службу***, если есть такая возможность. Поэтому радиокнопку ***Run Firebird server as*** лучше оставить в положении ***A Service*** (служба).

Птичку ***Start Firebird automatically everytime you boot up*** также лучше оставить, чтобы сервер сам автоматически стартовал при запуске компьютера. Серверы семейства InterBase практически не потребляют ресурсов компьютера, поэтому нет особых причин запускать и останавливать сервер вручную.

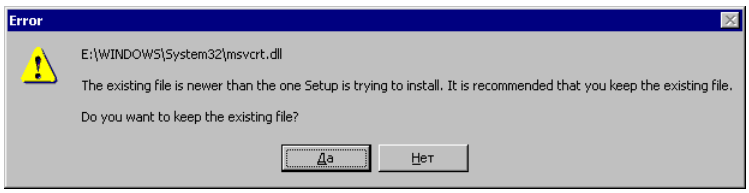
Таким образом, можно оставить все то, что предлагает инсталлятор и нажать кнопку ***Next***.

Появится страница ***Ready to Install*** (все готово к установке). Остается нажать кнопку Install и произойдет создание файлов на компьютере. Процесс инсталляции практически не занимает времени.





В процессе инсталляции может возникнуть предупреждающее окно о сохранении прежнего системного файла mvcrt.dll, если его версия более поздняя, чем та, что пытается поставить инсталлятор Firebird. В таком случае ответьте «Да», чтобы оставить более позднюю версию:



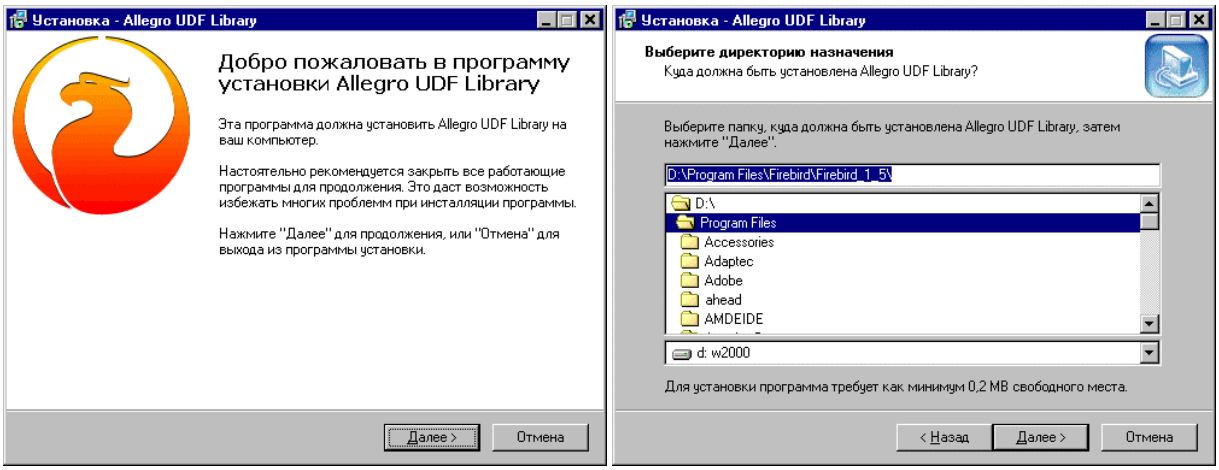
После завершения инсталляции появляется страница с различной информацией о последней версии сервера, там, в частности, можно найти ссылки на сайты разработчиков Firebird. Нажмите ***Next***.

Появится последняя страница инсталлятора с кнопкой ***Finish***. Нажмите ***Finish***. Сервер Firebird стартует сразу после установки на компьютер.

Установка библиотеки Allegro UDF Library.

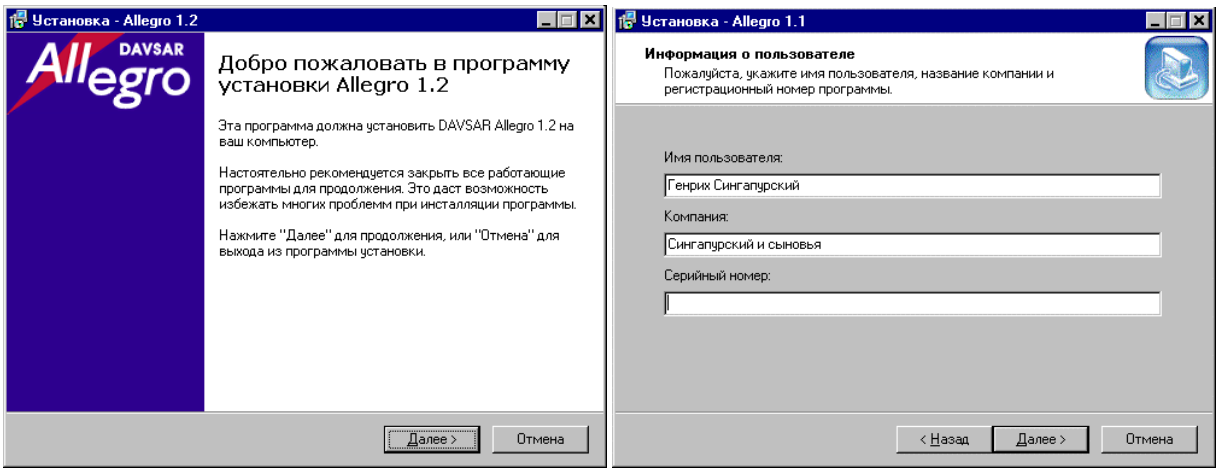
После того, как Вы установили Firebird на сервер, необходимо установить библиотеку пользовательских функций Alg_udf.dll. Для этого нужно запустить инсталлятор AllegroUdfSetup.exe, который доступен на нашем сайте: http://www.gaapinvest.com

Инсталлятор попробует сам определить директорию, в которую установлен Firebird. Если он не может определить эту директорию, то Вам нужно будет указать ее вручную:

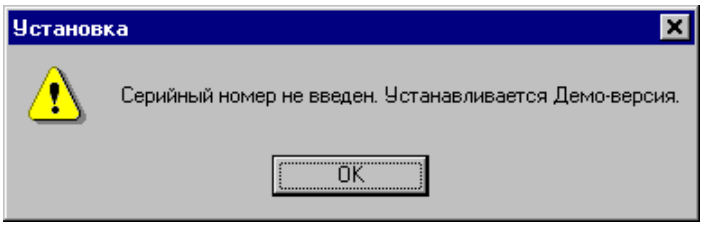


Установка программы Allegro

Инсталлятор самой программы DAVSAR Allegro называется AllegroSetup.exe и всегда доступен на нашем сайте http://www.gaapinvest.com. Размер инсталлятора программы Allegro около 4Mb.

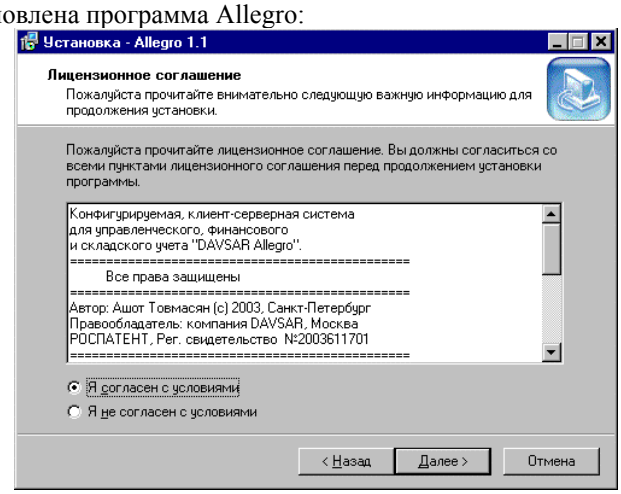


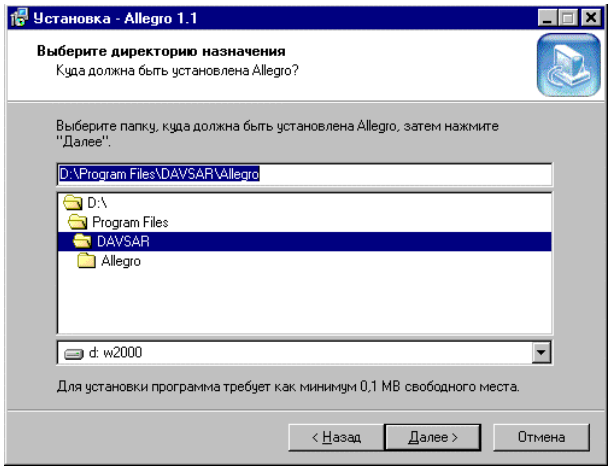
Если Вы приобрели программу Allegro, то у Вас должен быть ее серийный номер, который Вы не вправе никому передавать. В процессе инсталляции Вас попросят ввести серийный номер. Если Вы введете этот номер, то на ваш компьютер будет установлена лицензионная версия Allegro. Если Вы введете неверный серийный номер, инсталлятор сообщит Вам, что серийный номер ошибочен. Если Вы вообще не введете серийный номер, то будет установлена полнофункциональная Демо-версия Allegro с ограничением на количество документов не более 1000 в одной базе данных.

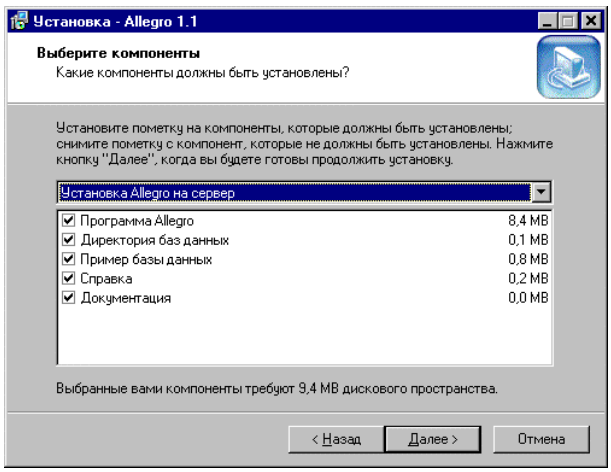


Мы рекомендуем устанавливать программу Allegro на сервер. На клиентских компьютерах достаточно создать ярлыки для запуска файла Allegro.exe по сети. Это упрощает поддержку программы. Если необходимо заменить Allegro файлом новой версии, Вам достаточно будет заменить его в одном месте (на сервере).

При инсталляции нужно принять лицензионное соглашение и указать директорию, в которую будет установлена программа Allegro:







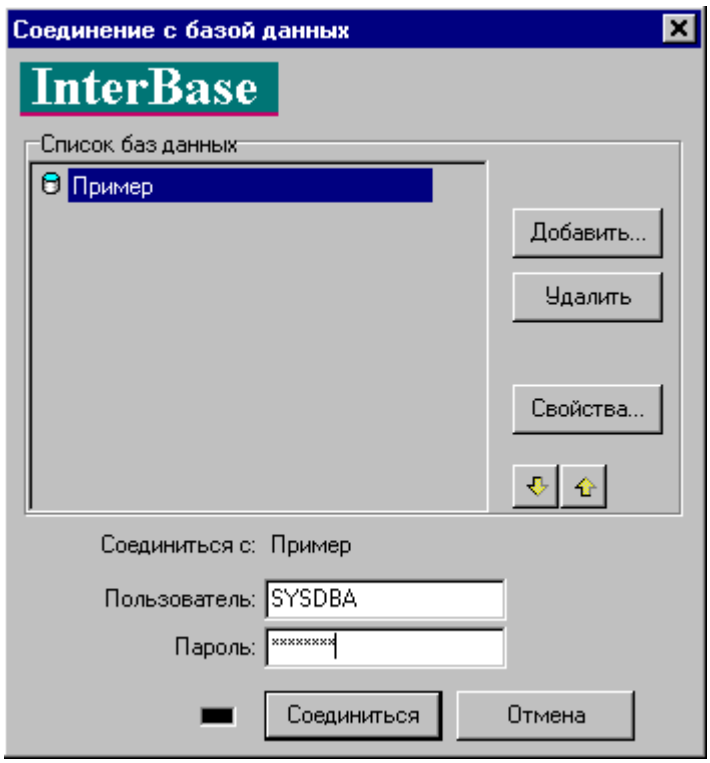
В процессе установки будут созданы поддиректории:

\bin – файл Allegro.exe и библиотеки dll
\db – файлы баз данных *.gdb
\doc – файлы документации
\help – файлы справки
\scripts – файлы скриптовых проектов оконного интерфейса

После установки Allegro на компьютер инсталлятор предложит Вам сразу запустить программу Allegro.

Первый запуск программы Allegro

При запуске Allegro появляется окно соединения с базой данных. Попробуйте сразу соединиться. Для этого наберите имя пользователя SYSDBA (администратор баз данных на сервере Firebird) и его пароль. По умолчанию пароль `masterkey` (маленькими буквами). В дальнейшем следует заменить пароль администратора.



Итак, вначале:

Пользователь: SYSDBA
Пароль: masterkey
База данных: Пример

Нажмите кнопку ***Соединиться***

В большинстве случаев, после установки Firebird, Allegro UDF Library и программы Allegro вся система сразу начинает работать.

Если не удается соединиться с базой данных

Если соединение с базой данных не удается, то возможно:

1. Не запущен сервер баз данных Firebird. Проверьте, запущен ли сервер. В системе Windows'98 значок Firebird при запущенном сервере появляется на панели задач Windows справа (рядом с системными часами). Если у Вас используется система Windows 2000/XP и Firebird установлен, как системная служба (сервис), то нужно вызвать окно «Службы» (Services) с «Панели управления» Windows и убедиться в том, что Firebird Server стартован.
2. В свойствах соединения указано неправильное имя компьютера-сервера или неверный путь к файлу базы данных. Нажмите кнопку «Свойства» в окне соединения с базой данных Allegro и проверьте правильность имени компьютера-сервера и пути к файлу базы данных. Учтите, что путь должен начинаться с буквы диска, на котором расположен файл базы данных. Файл базы данных должен

находиться на одном из дисков того же компьютера, на котором работает сервер Firebird. Файл базы данных «Пример» называется Default1.gdb и по умолчанию находится в подкаталоге \db директории, в которую установлена программа Allegro. Если Вам неизвестно имя компьютера и Allegro устанавливается на тот же компьютер, на котором работает сервер Firebird, то вместо имени компьютера можно набрать localhost или зацикливающий IP адрес 127.0.0.1. Если вы хотите работать с сервером в сети и вам известен его IP-адрес, то вместо имени компьютера можно набрать этот адрес.

3. Не работает TCP/IP протокол. Тогда можно попробовать соединиться с сервером через другой протокол. Для этого нажмите кнопку «Свойства» в окне соединения с базой данных Allegro и измените протокол на Local. Учтите, что с помощью протокола Local можно соединиться, если Allegro запущено на том же компьютере, что и сервер Firebird. С помощью протокола Local нельзя соединиться с базой данных по сети. Если у Вас в сети установлен протокол NetBeui, то можно попробовать соединиться с его помощью. Однако Вы должны позаботиться о том, чтобы в конечном итоге добиться работы с базой данных именно через TCP/IP протокол. Причем если предполагается одновременная работа нескольких пользователей с одной базой данных в сети, то все они обязаны работать с помощью одинакового протокола. Нарушение этого требования может привести к разрушению базы данных.

Глава 4. Создание новой базы данных

Из чего состоит конфигурация.

Список всех зарегистрированных баз данных хранится в файле **db.ini**. По умолчанию программа пытается найти этот файл в поддиректории **\bin**. Если файл **db.ini** находится в иной директории, чем **\bin**, путь к нему следует указать в файле **Allegro.ini**. Файл **Allegro.ini** всегда находится в поддиректории **\bin** (там же, где файл **Allegro.exe**). Если исполняемый файл **Allegro.exe** находится на сервере, то список зарегистрированных баз данных является *общим для всех пользователей*. Если исполняемый файл **Allegro.exe** установлен на клиентский компьютер, то возможно 2 решения:

1. Список зарегистрированных баз данных **db.ini** также хранится на клиентском компьютере
2. В файле **Allegro.ini** указывается сетевой путь к файлу **db.ini** на сервере

Файл **Allegro.ini**:

```
[Main]
db_ini_file=Путь к файлу db.ini
allegro_sql_file=Путь к файлу Allegro.sql (скрипт пустой базы)
allegro_serial=Серийный номер программы
images_directory=Путь к папке картинок
```

Если путь **db_ini_file**, не указан, то Allegro ищет **db.ini** в директории **\bin**.
Если путь **allegro_sql_file**, не указан, то Allegro ищет **Allegro.sql** в директории **\db**.
Если не указан серийный номер программы, Allergo стартует в демонстрационном режиме.
Если путь **images_directory**, не указан, то Allegro устанавливает путь к директории **\images**.

Файл **db.ini**:

```
[Пример]
DatabaseFileName= D:\Program Files\DAVSAR\Allegro\db\DEFAULT1.GDB
DatabaseName=127.0.0.1:D:\Program Files\DAVSAR\Allegro\db\DEFAULT1.GDB
NetProtocol=1
ServerName=127.0.0.1
ProjectDirectory=D:\MyDelphi6\Projects\Allegro\Scripts\Default1

[TechnoTrade]
DatabaseFileName=D:\Program Files\DAVSAR\Allegro\db\TECHNOTRADE.GDB
DatabaseName=localhost:D:\Program Files\DAVSAR\Allegro\db\TECHNOTRADE.GDB
NetProtocol=1
ServerName=localhost
ProjectDirectory=D:\Program Files\DAVSAR\Allegro\scripts\TechnoTrade
```

Параметры каждого зарегистрированного соединения хранятся в своем разделе. В этих параметрах указываются:

- имя или IP-адрес компьютера-сервера **ServerName**,
- путь к файлу базы данных **DatabaseFileName**, начинающийся с буквы диска,
- сетевой протокол **NetProtocol**
- путь к директории проектов **ProjectDirectory**.

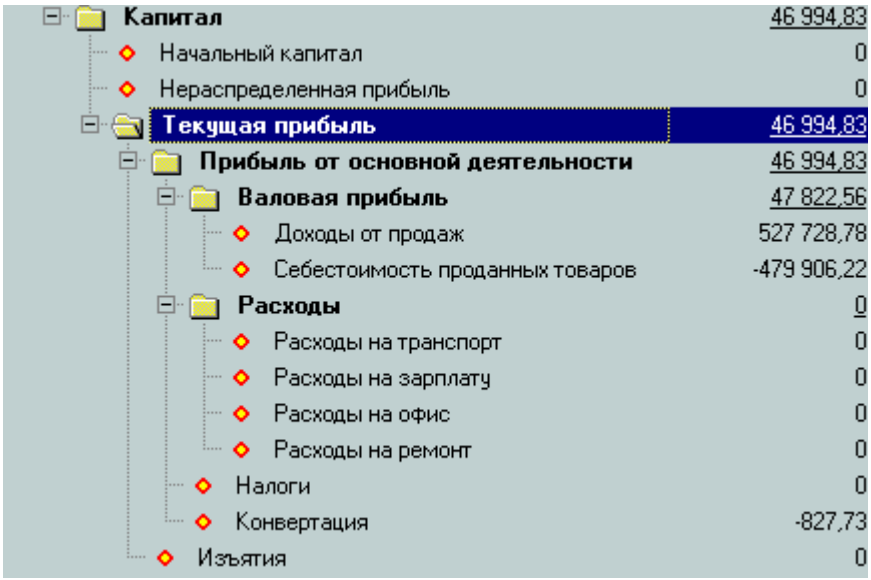
Таким образом, любое соединение содержит информацию о том, где находится файл базы данных и где находится директория скриптовых проектов. Скриптовые проекты управляют оконным интерфейсом и являются частью конкретной конфигурации. Рекомендуется для каждой конфигурации хранить скриптовые проекты в отдельной поддиректории, размещая ее в директории **\scripts**. Например, конфигурация **TechnoTrade** состоит из файла базы данных **D:\Program Files\DAVSAR\Allegro\db\TECHNOTRADE.GDB** и скриптовых проектов, хранящихся в папке **D:\Program Files\DAVSAR\Allegro\scripts\TechnoTrade**.

При создании новой базы данных «по итогом» имеющейся, создается новый файл базы, а в файле **db.ini** регистрируется новое соединение. В зарегистрированном соединении указывается путь к файлу новой базы и путь *к прежней* директории проектов.

Если мы создаем совершенно новую конфигурацию, то рекомендуется создать отдельную директорию для *будущих* скриптовых проектов и указать эту директорию в процессе создания новой базы данных.

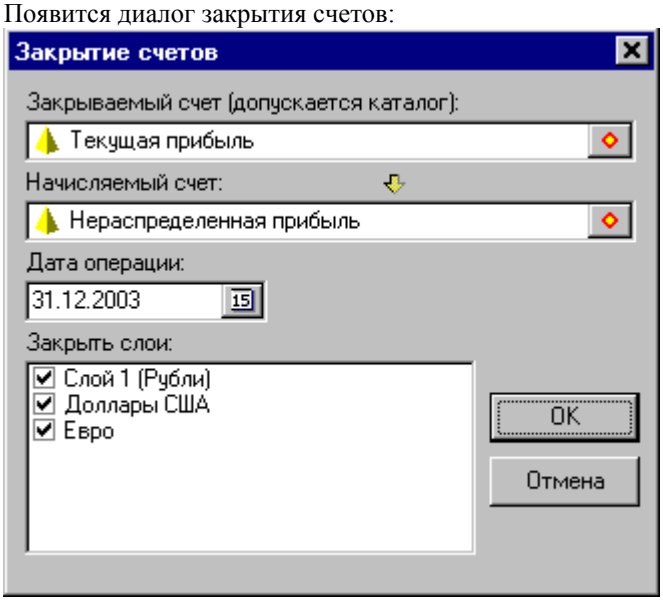
Закрытие счетов прибыли и конвертация «Нераспределенной прибыли».

Мы рекомендуем компаниям начинать новую базу данных каждый финансовый год. В этом случае применяется создание новой базы «С переносом остатков». Однако перед тем, как создавать новую базу в этом режиме, следует закрыть все счета прибыли (доходов и расходов) в текущей базе. Если новый период начинается с 1 января, то закрытие счетов прибыли следует произвести на 31 декабря предыдущего года.



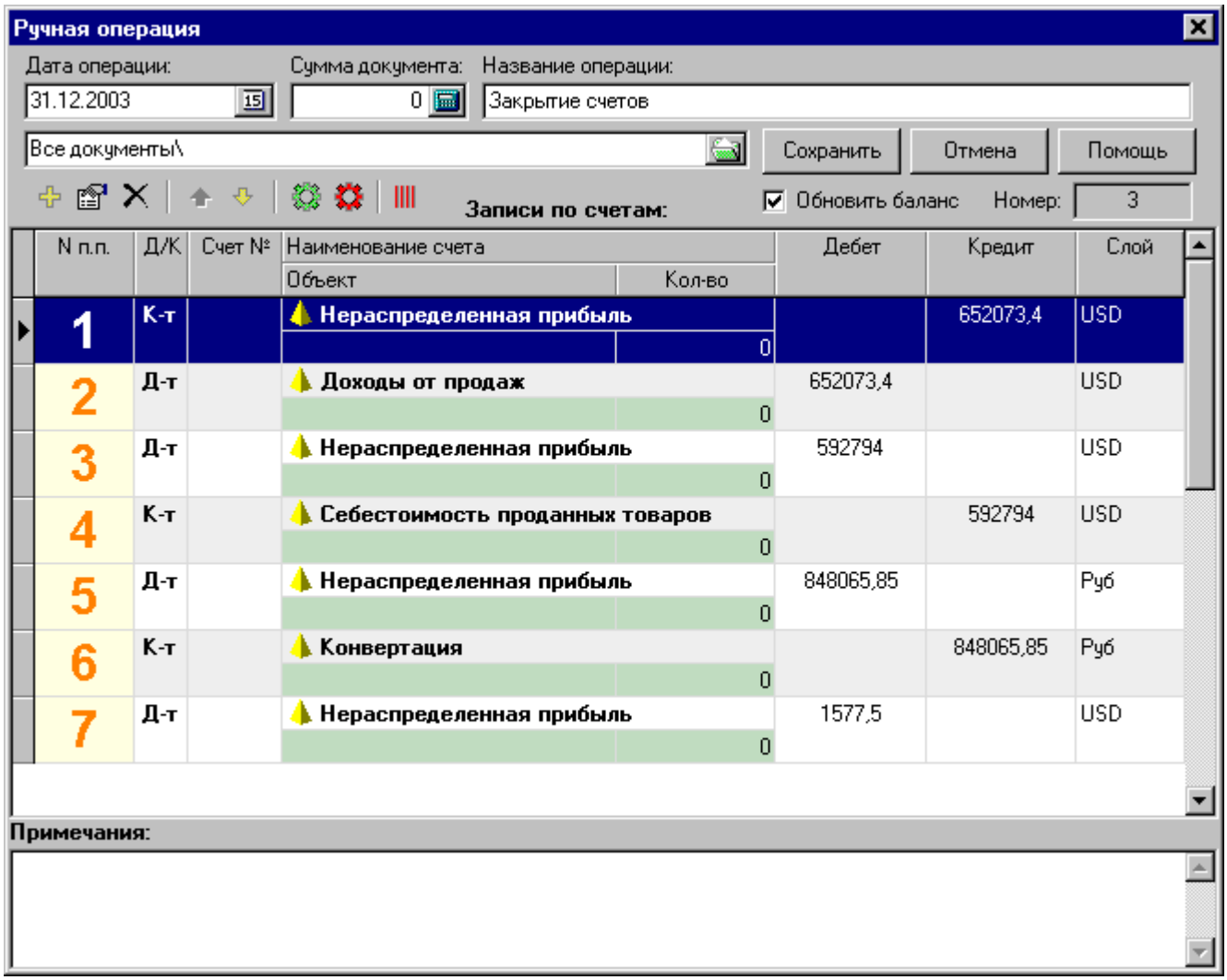
Закрытие счетов производится с помощью пункта меню **Бухгалтерия/Закрытие счетов…**

Появится диалог закрытия счетов:



Нужно указать дату закрытия, закрываемый каталог счетов и счет, на который будут начислены все остатки. Как правило - это счет «Нераспределенной прибыли». По умолчанию закрываются все слои. Операция закрытия счетов является обычной ручной операцией и после формирования с ней можно будет обращаться так же, как с любой другой ручной операцией (просматривать, редактировать или удалить). Поэтому операцию закрытия счетов можно разделить на несколько отдельных операций, например, закрыть счета сначала в одном валютном слое, а затем в другом.

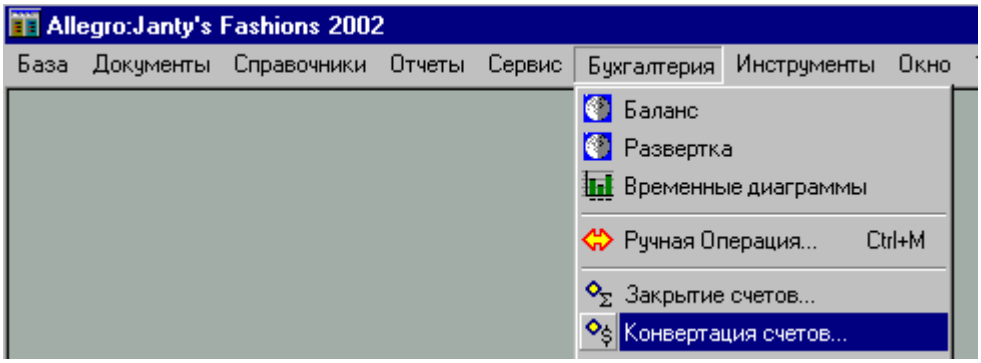
После нажатия кнопки ***ОК***, программа сформирует ручную операцию по зарытию счетов и покажет ее на экране. Останется лишь провести эту операцию, нажав кнопку ***Сохранить***:



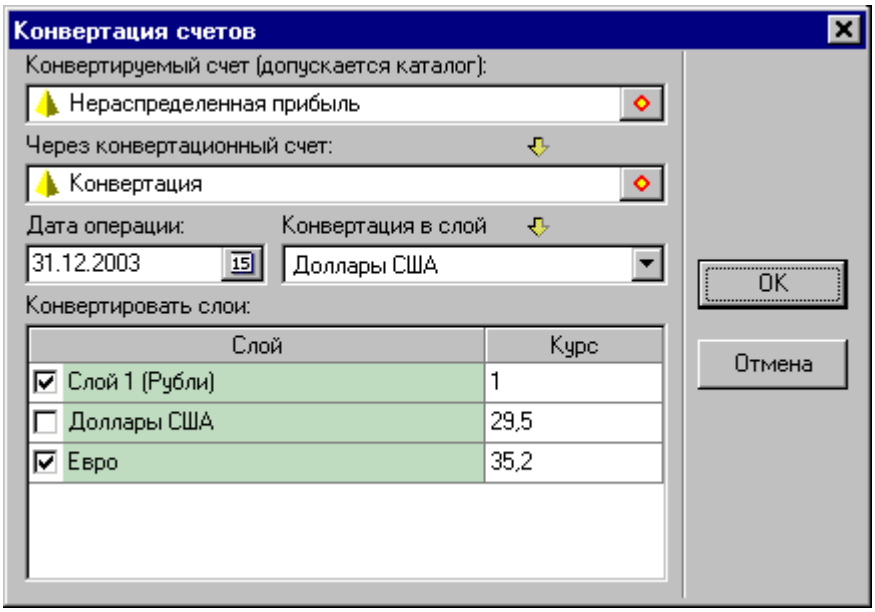
После проведения операции закрытия счетов, в таблице проводок будут созданы компенсирующие записи по счетам и остатки всех закрываемых счетов станут нулевыми. Вся текущая прибыль периода будет перенесена на счет «Нераспределенной прибыли».

Если мы ведем многовалютный учет, то на счет «Нераспределенной прибыли» могут оказаться начисленными средства в разных валютах. В дальнейшем это может привести к тому, что значение счета «Нераспределенной прибыли» в консолидированном балансе будет изменяться при изменении курсов валют, что очень нежелательно, так как изменение стоимости активов в каком-то периоде должно относиться к прибыли или убытку этого периода, а не влиять на прибыль/убыток прошлых лет. Поэтому мы рекомендуем сразу после закрытия счетов проводить ***конвертацию*** счета «Нераспределенная прибыль» через счет «Конвертация», входящий в состав каталога «Текущая прибыль». Компания должна в какой-то момент времени принять решение о том, в какой валюте фиксировать такие счета, как «Начальный Капитал» и «Нераспределенная прибыль» и придерживаться в дальнейшем этой валюты, производя конвертацию «Нераспределенной прибыли» в эту валюту после закрытия счетов текущей прибыли.

Для конвертации счета используем пункт меню **Бухгалтерия/Конвертация счетов…**



Появится диалог конвертации счета:



Курсы валют берутся из системной таблицы курсов валют на дату конвертации. Нужно указать конвертируемый счет, конвертационный счет, дату операции и валюту, в которую мы хотим отконвертировать остатки на счете. После чего нужно нажать кнопку ***OK***. Программа создаст ручную операцию и ее можно будет провести так же, как любую обычную ручную операцию.

Мастер создания новой базы данных.

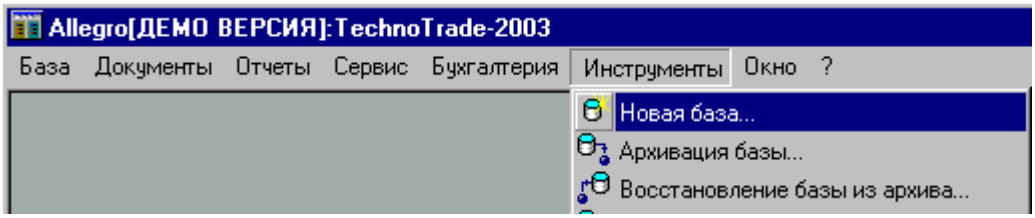
Создание новой базы данных полностью автоматизировано в Allegro. Независимо от того, какую конфигурацию Вы используете, Allegro может создать новую базу данных с метаданными именно этой конфигурации. При переносе данных в новую базу из старой можно перенести бухгалтерские остатки, какие-то документы, содержимое справочников и структуру папок проводника по документам.

Возможно также создание «пустой» базы данных, не содержащей пока никакой конфигурации вообще. При создании «пустой» базы данных Allegro создает в новой базе лишь системные таблицы и процедуры.

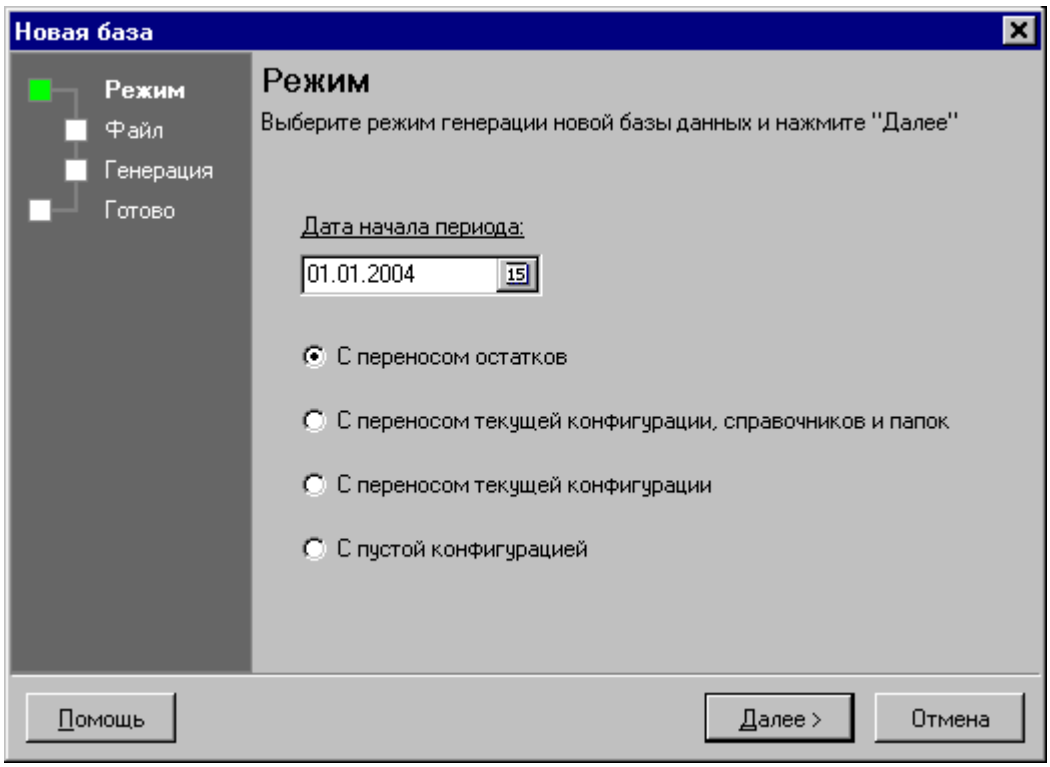
Последовательность действий, которые осуществляет Мастер создания новой базы данных такова:

- Извлекается SQL-сценарий метаданных из старой базы или, если создается «пустая» база, в качестве сценария используется файл Allegro.sql
- Создается файл новой базы данных командой CREATE DATABASE
- Исполняется скрипт SQL-сценария метаданных
- Если база создается на основе уже имеющейся, то копируются какие-то данные из старой базы в новую. В любом случае копируются все таблицы настроек. Если выбран соответствующий режим, то копируются справочники и папки проводника по документам. В случае, если переносятся бухгалтерские остатки, то запрашивается баланс на дату, предшествующую дате начала нового периода и копируется в виде остатков по счетам в новую базу.
- Новая база данных регистрируется в списке **db.ini**

Если мы уже находимся в соединении с текущей базой, то для вызова Мастера созания новой базы данных нужно использовать пункт меню **Инструменты/Новая база…** Меню Инструменты доступно, если пользователь является системным администратором сервера баз данных SYSDBA или создателем текущей базы. Если мы не находимся в соединении ни с какой базой, меню **Инструменты** также доступно.



Появится Мастер создания новой базы. Прежде всего нужно установить дату начала периода:



Если мы создаем базу данных в режиме «С переносом остатков», то выбор даты начала периода имеет большое значение. Как правило эта дата совпадает с началом финансового года. Мы рекомендуем создавать новую базу данных в начале каждого финансового года, а не накапливать все данные «вечно» в одной базе.

Создать новую базу можно в любой момент времени, «отрезая» учет с любой даты. Например, если мы создаем новый период с 1 января, это вовсе не означает, что сам процесс создания базы данных должен происходить именно 1 января. Лучше это сделать в феврале или марте, когда у нас уже будет полная уверенность в том, что все цифры на 1 января верные. Все документы, которые должны быть перенесены в новый период, автоматически будут скопированы в новую базу. Правила переноса документов задаются изначально в конфигурации и могут быть различными для разных типов документов (переносить в зависимости от даты, переносить по флагу, переносить всегда). Наиболее простое правило переноса состоит в том, что документы, содержащие даты нового периода, будут перенесены в новую базу данных. Для обеспечения ссылочной целостности будут также перенесены все документы, на которые ссылаются эти документы и так далее.

В режиме с «Переносом остатков» в новую базу данных переносятся курсы валют на даты, попадающие в новый период.

Кроме того в режиме «С переносом остатков» все остатки по счетам «на дату отреза» перенесутся в новую базу в виде начальных остатков. Остатки аналитических счетов переносятся по каждому объекту аналитики раздельно. Остатки в каждом валютном слое также переносятся раздельно. Прежде, чем создавать новую базу «С переносом остатков», нужно убедиться в том, что все счета текущей прибыли закрыты. Закрытие счетов прибыли описано в предыдущем параграфе.

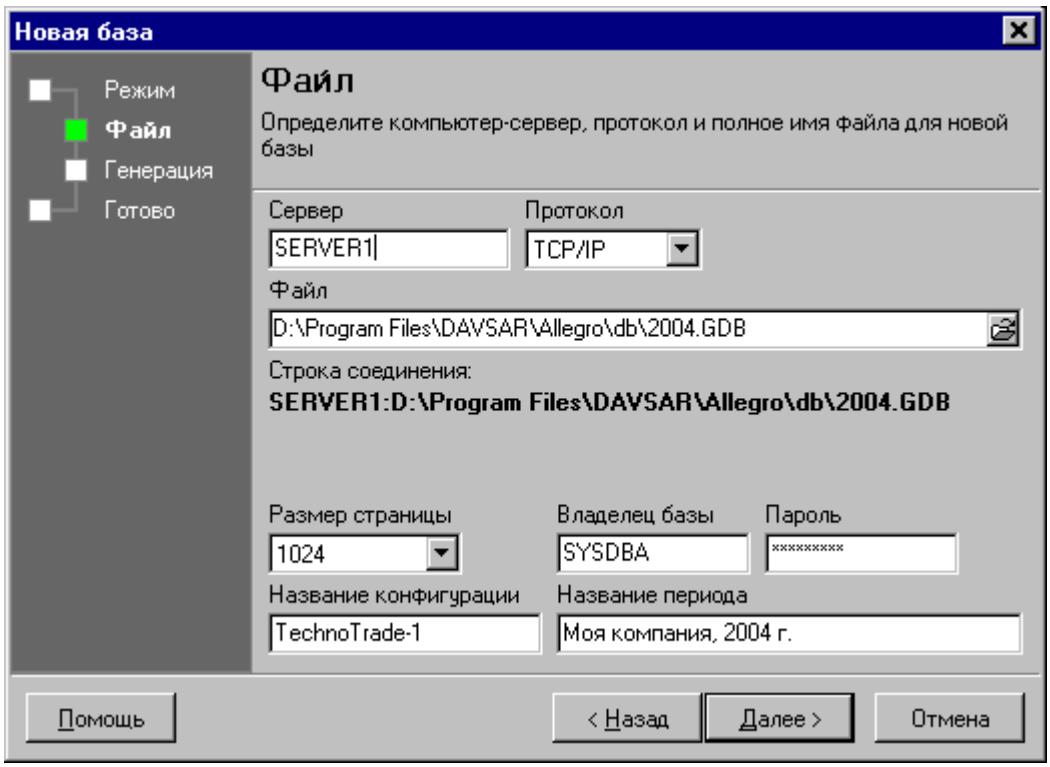
Если выбран режим «С переносом текущей конфигурации, справочников и папок», начальные остатки не создаются и никакие документы в новую базу данных не переносятся. Переносится лишь структура папок «Проводника по документам» и содержимое всех справочников.

Если выбран режим «С переносом текущей конфигурации», то не переносятся даже папки «Проводника по документам». Справочники в новой базе окажутся пустыми.

Если выбран режим «С пустой конфигурацией», то создается «пустая» база данных, в которой нет никаких метаданных кроме системных таблиц Allegro.

Таким образом, существует три режима создания новой базы на основании текущей и один режим создания базы данных с «пустой конфигурацией». После выбора режима создания базы и указания даты начала периода следует нажать кнопку ***Далее***.

Появится вторая страница Мастера.



Здесь мы должны задать имя компьютера-сервера (можно вместо имени указать IP-адрес), имя файла будущей базы данных и название периода. Учтите, что файл базы должен ***физически*** располагаться на диске компьютера-сервера. Название файла должно начинаться с буквы диска и не должно включать в себя никаких сетевых путей или символов. Рекомендуем использовать только английские буквы или цифры в названии этого файла, например:

D:\Program Files\DAVSAR\Allegro\db\2004.GDB

Название периода может быть любым, например:

Моя компания, 2004г.

Название конфигурации, название периода и дата начала периода хранятся в самой базе данных. По названию периода будет также названа регистрационная запись в файле **db.ini**.

Пользователь, создавший базу данных, считается ее ***владельцем*** (OWNER). Только владелец базы и администратор SYSDBA имеют привилегии в дальнейшем эту базу администрировать. Для простоты мы рекомендуем создавать базы от имени администратора SYSDBA.

Пароль пользователя SYSDBA по умолчанию:

masterkey

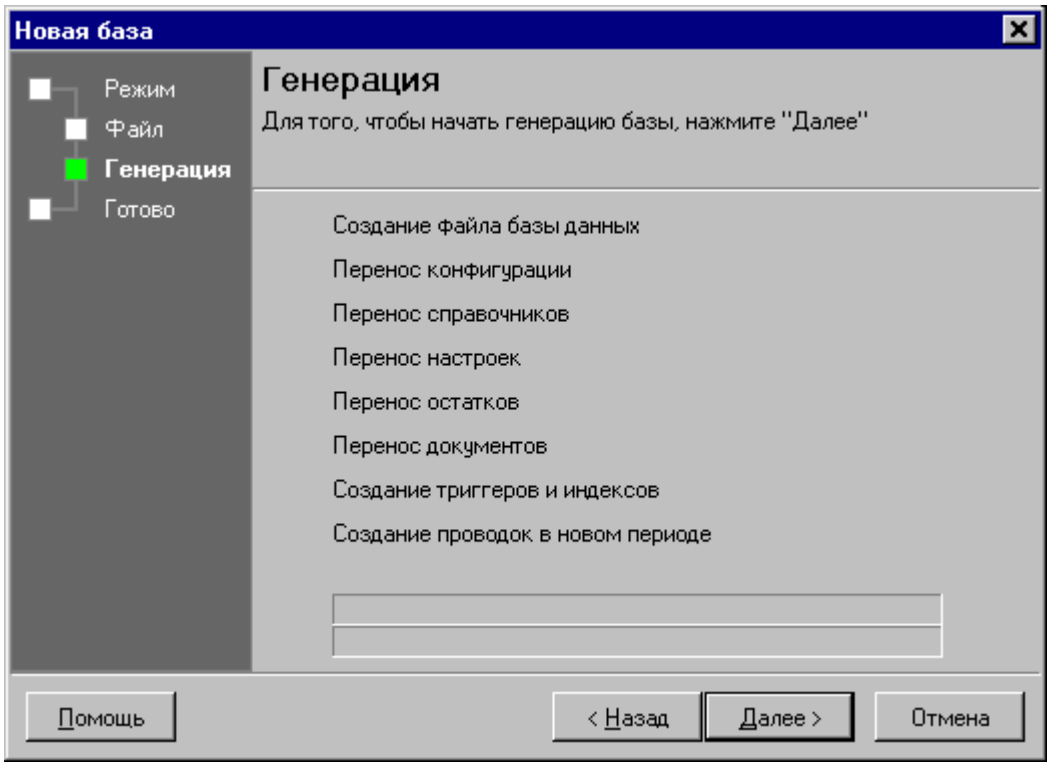
Базы данных InterBase имеют страничную организацию. Размер страницы определяется при создании базы и может в дальнейшем быть изменен, если произвести архивацию базы, а затем восстановление из архива, указав при восстановлении новый размер. Оптимальный размер страницы зависит от множества факторов (требуемое быстродействие, характер данных, размер базы, тип файловой системы). Подробнее с этой темой можно ознакомиться в документации по InterBase. По умолчанию размер страницы равен 1024 байт.

При создании базы на основе имеющейся, ***Название конфигурации*** лучше не изменять, если Вы не собираетесь вносить изменения в метаданные. При создании базы «С пустой конфигурацией» нужно придумать название для будущей конфигурации, которую Вы намерены создать.

Рекомендуемый протокол связи с сервером TCP/IP. Иные протоколы можно использовать лишь в исключительных случаях. Одновременное соединение нескольких клиентов с одной базой данных при помощи различных протоколов может привести к нежелательным последствиям, вплоть до разрушения базы данных.

После того, как мы определили все параметры, нажимаем кнопку ***Далее***.

Программа сообщит о том, что все готово к созданию новой базы данных.



После нажатия кнопки ***Далее*** начнется процесс создания базы данных. По завершении процесса новая база данных будет добавлена в список зарегистрированных баз в **db.ini**. Для того, чтобы с ней соединиться, нужно воспользоваться пунктом меню **База/Соединиться**.

Глава 5. Администрирование баз данных

Основные рекомендации

Базы данных InterBase самодостаточны, и в особом администрировании, как правило, не нуждаются. Часто такие базы годами работают без всяких жалоб со стороны пользователей. Однако существуют технические и организационные причины для того, чтобы не полагаться исключительно на везение, особенно, если Вы разработчик конфигурации или другое лицо, отвечающее за поддержку системы. Дело в том, что практически *ни один пользователь добровольно не создает резервных копий* своих файлов, если его к этому *специально не принудить*. В то же время, как легко догадаться, жесткие диски иногда просто «летят» и никуда от этого не деться. На предприятиях нередки случаи внезапного отключения питания и другие неприятности, связанные с ненормальным завершением работы операционной системы сервера (зависшие приложения, приложения, содержащие ошибки «пожирания ресурсов», вирусы, наконец). Так что никто не может быть застрахован от того, что в один прекрасный день с таким трудом накопленная, важнейшая учетная информация окажется испорченной либо безвозвратно утраченной…

Поэтому мы настоятельно рекомендуем с самого начала внедрения системы, наладить **порядок создания** резервных копий базы данных и **приучить к этому пользователя**, доведя эту операцию чуть ли не до автоматизма. Лучшее, что можно сделать, это наладить порядок, при котором в конце каждого рабочего дня производится архивация базы, и полученный так называемый BACKUP-файл затем дополнительно сжимается с помощью обычного архиватора ZIP или RAR и копируется на лазерный диск (несколько дисков).

Чем короче интервал обязательных архиваций базы, тем меньшее количество данных будет потеряно при ее внезапном крушении.

Другим важным аспектом администрирования является проблема «чистки мусора». Сервер InterBase относится к так называемым многоверсионникам. Это означает, что при совместной работе пользователей создаются копии записей в таблицах, что позволяет разделять транзакции без блокировки записей. Лишние копии записей в дальнейшем удаляются сервером в процессе «корпоративной чистки», однако некоторое количество «мусора», тем не менее, на капливается в базе данных. Для устранения этого мусора используется операция SWEEP, которая может выполняться сервером автоматически время от времени или производиться вручную.

При работе сервер InterBase добавляет новые страницы в базу данных, по мере надобности. Эти страницы увеличивают размер файла базы данных. После удаления данных размер файла базы данных не уменьшается. Если нужно уменьшить размер файла базы данных, то нужно применить последовательно архивацию (BACKUP) и восстановление (RESTORE) базы данных. В процессе архивации к тому же удаляется весь «мусор» и некоторые ошибки, а при восстановлении резервируется минимально необходимое количество страниц в файле. Мы рекомендуем время от времени (хотя бы раз в полгода) производить архивацию/восстановление базы данных. Иногда архивация/восстановление могут спасти базу данных в ситуации, если в ней возникли какие-то ошибки, например, разрушились индексы каких-то таблиц.

Существуют редкие ситуации, когда в базе данных возникают фатальные ошибки, например, ошибки нумерации страниц. В этом случае архивация базы данных не работает. При этом, если есть соединение с базой, и данные продолжают быть доступны, мы рекомендуем создать новую базу «По итогам имеющейся», взяв в качестве начала периода его значение в прежней базе. Это позволит «перекачать» все данные из старой базы в новую и таким способом спасти данные и избавиться от ошибки.

Наконец, возможны ошибки контрольных сумм и другие ошибки в базе данных, от которых не удается никак избавиться, так как вообще не удается соединиться с базой данных. Это очень редкая ситуация. Но если такое случилось – не отчаивайтесь. Возможно, базу данных удастся «починить» с помощью специальных утилит, осуществляющих функцию VALIDATE DATABASE, которые не входят в состав системы Allegro, но существуют в других приложениях, предназначенных для администрирования баз данных InterBase. Такие приложения (многие из них распространяются свободно) можно найти на ресурсах Интернет, посвященных InterBase серверу. Одним из лучших таких ресурсов мы считаем сайт www.ibase.ru

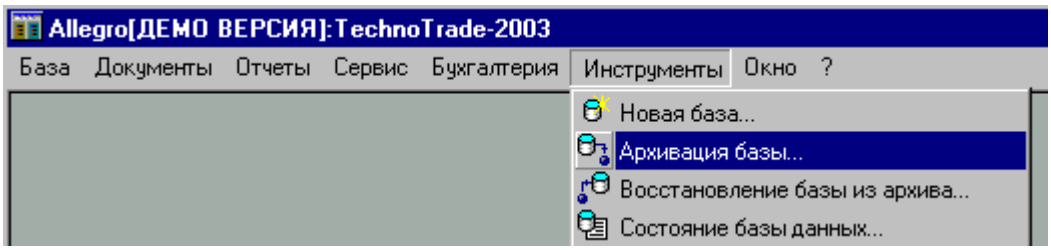
Обязательная архивация базы данных производится также перед инсталляцией новой версии сервера InterBase. После установки новой версии сервера база восстанавливается из архива.

Подробнее об архивации базы, ее восстановлении и чистке «мусора» читайте в следующих параграфах.

Архивация базы данных

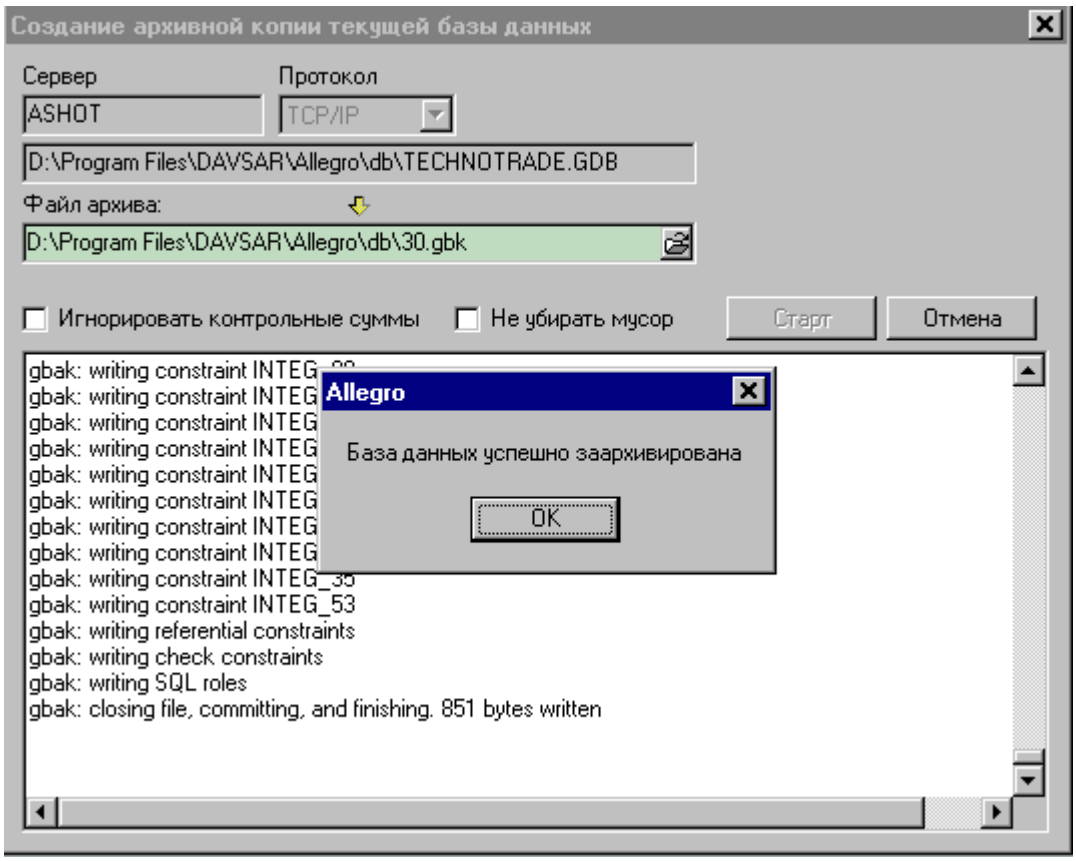
При архивации (BACKUP) базы данных создается файл с расширением GBK. При архивации происходит избавление от «мусора» и определенное сжатие данных, так как в файл архива не записываются, к примеру, индексы. При архивации размер файла обычно уменьшается раза в два - три. Разумеется, это не сильная степень сжатия. Если Вы хотите сильно сжать архив, сожмите полученный файл каким-нибудь архиватором ZIP или RAR. База данных размером 17M после архивации занимает 8M, а после сжатия RAR-ом ее размер может сократиться до 1M. Важно то, что архивация может производиться в тот момент, *когда к базе данных подключены пользователи*. Простое копирование файла базы данных в этих условиях недопустимо. Поэтому и применяется именно архивация (BACKUP), которую осуществляет сервер InterBase, игнорируя при этом неподтвержденные транзакции пользователей.

Для архивации базы данных используем пункт меню **Инструменты/Архивация базы…**



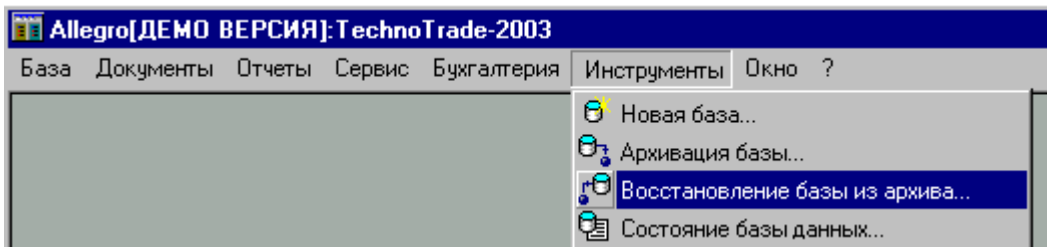
Появится диалог, в котором нужно указать полный сетевой путь к файлу архива. Файл архива имеет по умолчанию расширение GBK и может располагаться где угодно (не обязательно на сервере). После этого нужно нажать кнопку ***Старт***.

При успешном завершении процесса создания архива, мы увидим сообщение:



Восстановление базы данных из архива

При восстановлении (RESTORE) базы данных из файла архивной копии восстанавливаются все метаданные, данные и индексы. Для восстановления базы данных используем пункт меню **Инструменты/Восстановление базы из архива…**



В появившемся диалоге нужно указать имя BACKUP-файла, из которого будет производиться восстановление базы данных. При восстановлении файл базы данных создается заново. Поэтому существуют два режима восстановления из архива:

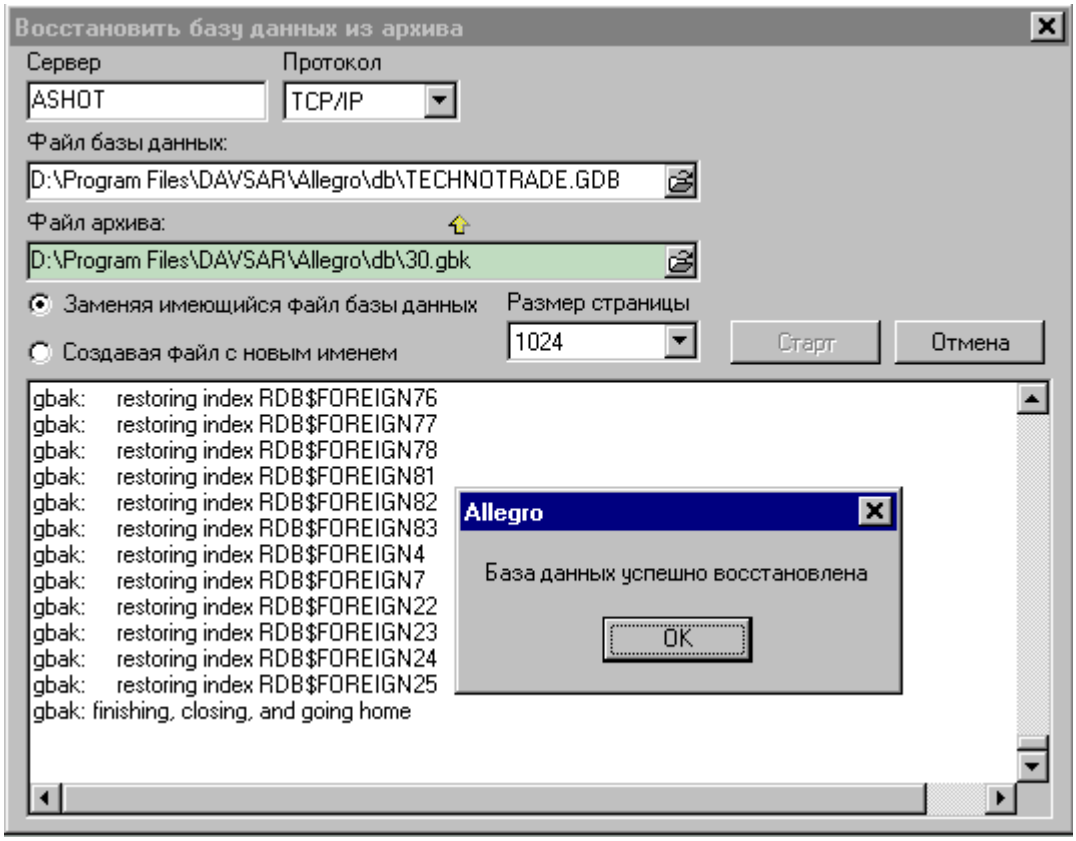
- Заменяя имеющийся файл базы данных
- Создавая файл с новым именем

По умолчанию задан второй режим. Для того чтобы восстановить базу, заменяя имеющийся файл базы данных, нужно убедиться, что все пользователи отключены от базы данных. Иначе операционная система сообщит о невозможности создания файла.

При восстановлении базы данных из архива можно задать новый размер страницы. По умолчанию это 1024 байт.

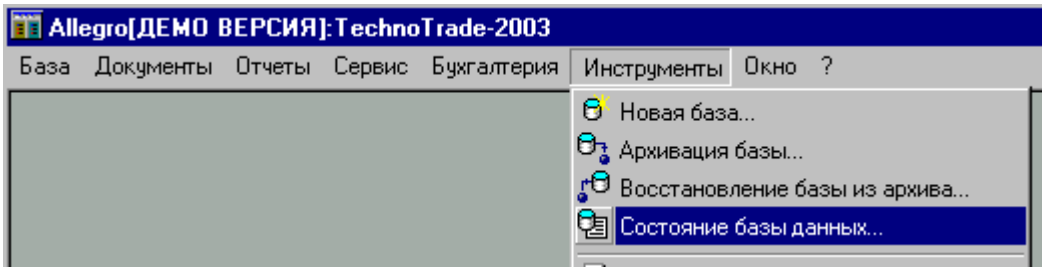
Для запуска процесса восстановления из архива нажимаем кнопку ***Старт***.

При успешном завершении процесса восстановления базы данных из архива, мы увидим сообщение.

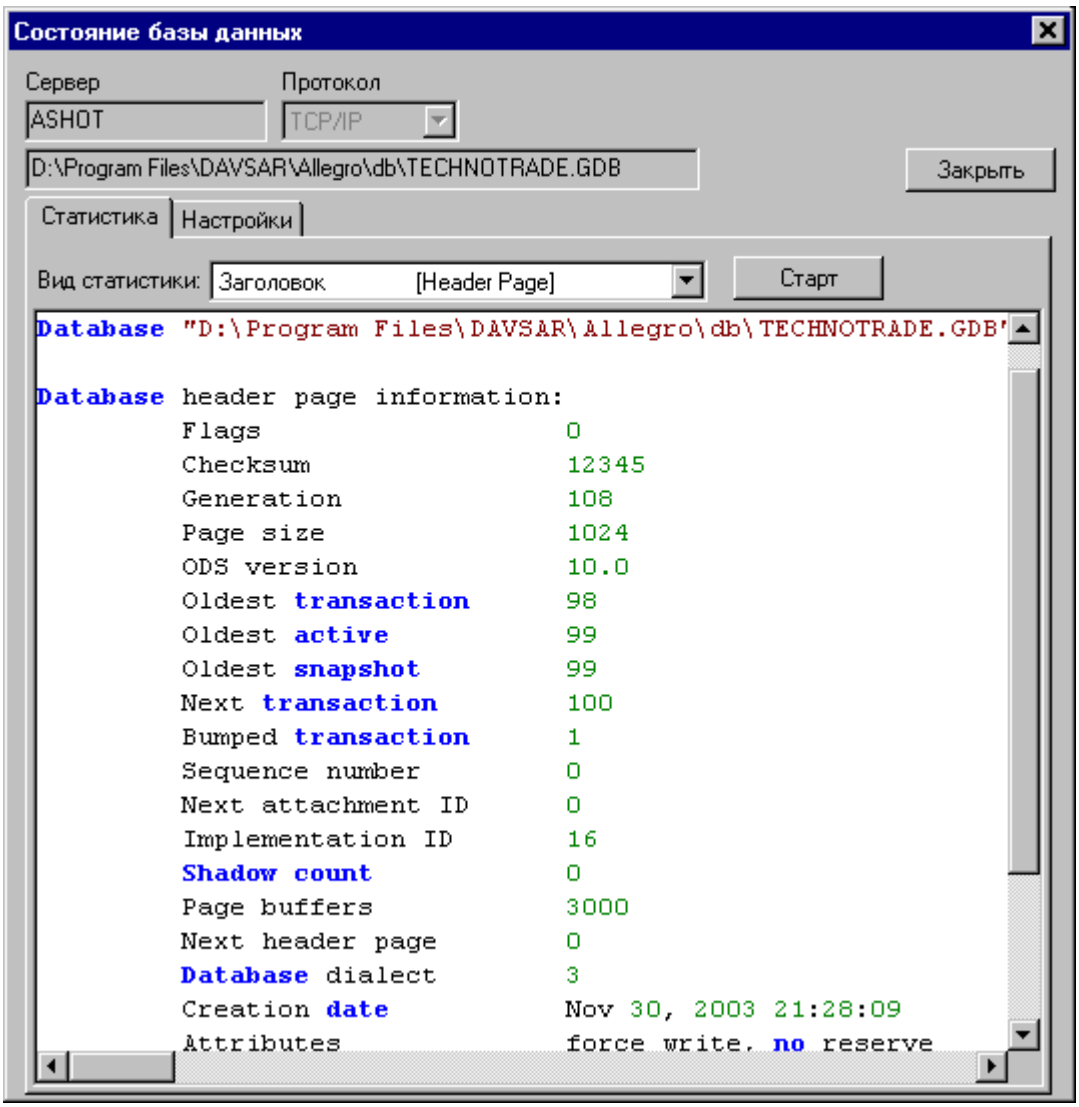


Состояние базы данных, уборка «мусора»

Для того чтобы взглянуть на текущее состояние базы данных или произвести уборку «мусора», используем пункт меню **Инструменты/Состояние базы данных…**

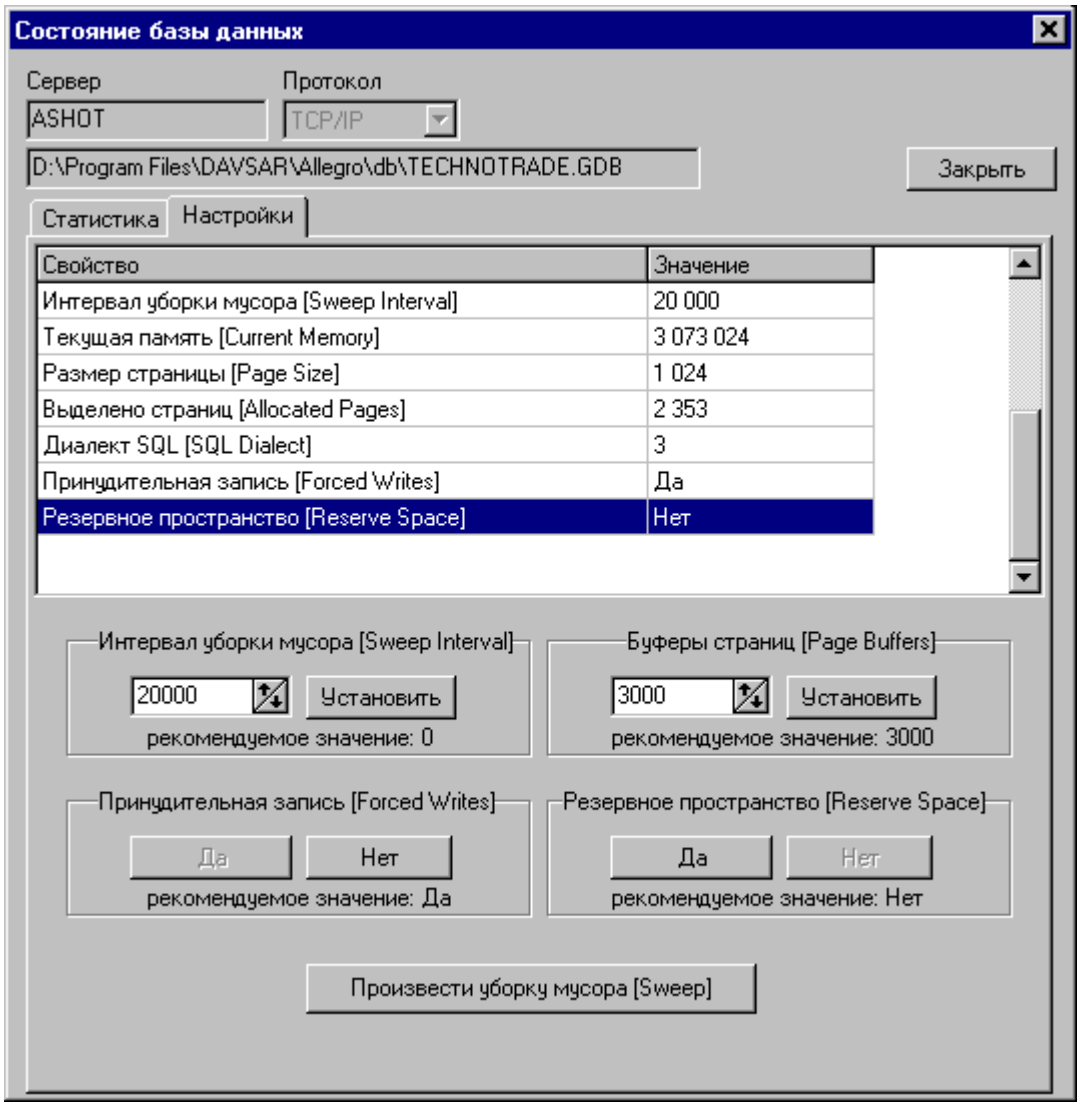


Появится окно диалога, в котором нужно выбрать вид статистики, который мы хотим просмотреть и нажать кнопку ***Старт***.



Если между параметрами Next transaction и Oldest transaction накопился большой разрыв, то следует произвести уборку мусора SWEEP. Если значение параметра Sweep Interval равно 20 000 (по умолчанию это так), то уборка мусора производится автоматически сервером, когда разрыв между номерами транзакций достигает значения 20 000. Вы можете изменить значение этого параметра. Нужно учесть, что автоматическая уборка мусора на некоторое время «подвешивает» сервер, что может помешать работе пользователей. Поэтому многие

применяют занчение Sweep Interval = 0. При таком значении автоматическая уборка «мусора» никогда не производится и ее нужно вызывать вручную. Для изменения значения параметра Sweep Interval или для того, чтобы запустить ручную уборку мусора, мы должны переключиться на закладку «Настройки»:



Для того чтобы запустить уборку «мусора» нужно нажать кнопку ***Произвести уборку мусора [Sweep]***. Для того чтобы изменить Интервал уборки «мусора» Sweep Interval, нужно ввести новое значение и нажать кнопку ***Установить***.

Глава 6. Баланс

Принципы бухгалтерского учета

Общепринятые принципы бухгалтерского учета: General Accepted Accounting Principles - помечаются далее в тексте как GAAP. Из различных международных стандартов англо-американская модель учета (GAAP) является наиболее приспособленной для удовлетворения информационных потребностей инвесторов (владельцев), кредиторов и менеджеров.

БАЛАНС

Баланс содержит информацию о финансовом положении предприятия в данный момент времени. Баланс имеет две стороны. Левая сторона называется "*Средства*" (Assets), а правая - "*Обязательства и Капитал*" (Liabilities and Capital).

СРЕДСТВА

Средства представляют собой ресурсы, принадлежащие предприятию и имеющие стоимостное выражение (денежные средства, оборудование и т.д.) Баланс показывает размер средств предприятия на определенную дату.

ОБЯЗАТЕЛЬСТВА

Обязательства представляют собой внешние источники ресурсов предприятия (денежно выраженные долги предприятия: кредиторам, государству, работникам и т.д.) Кредиторы имеют право **платежного иска** в отношении средств предприятия в сумме, равной долговым обязательствам предприятия. Иски обеспечиваются **всеми** средствами предприятия.

КАПИТАЛ

Второй источник формирования средств предприятия - *Капитал*. *Капитал* образуется из *Инвестиций* в предприятие и *Прибыли* предприятия.

ИНВЕСТИЦИИ

Инвесторы предоставляют предприятию ресурсы, взамен получая акции (долю в собственности на предприятие). Общая сумма капитала, предоставленная инвесторами, называется Акционерным Капиталом или просто Капиталом, если речь идет о частной компании или товариществе. Предприятие **ничего не должно** своим инвесторам. Если предприятие прекращает свое существование, то инвесторам достается то, что остается после платежей по обязательствам предприятия. Таким образом, **в отличие** от кредиторов, инвесторы обладают лишь правом **остаточного иска**.

"Акционерный капитал" существует только у корпораций - предприятий, продающих свои акции. Для так называемых **частных** предприятий и **товариществ** употребляется термин **"Капитал владельца (владельцев)"**. Часто форма капитала влияет на налогообложение. Например, в США федеральным налогом на прибыль облагаются только корпорации (соответствующие российским открытым акционерным обществам). Частные предприятия и товарищества в США федеральным налогом на прибыль не облагаются, так как предполагается, что их владельцы платят подоходный налог.

НЕРАСПРЕДЕЛЕННАЯ ПРИБЫЛЬ

Другой источник Капитала предприятия - *Прибыль*. Та часть прибыли, которая остается у предприятия после выплаты дивидендов инвесторам, называется **Нераспределенной прибылью**.

ПРИНЦИП ДВУСТОРОННОСТИ (GAAP)

Равенство

Средства = Обязательства + Капитал

называется балансовым уравнением и выражает **Принцип Двусторонности**.

Данное уравнение является основой бухгалтерского учета. Его также можно представить в форме, отражающей остаточный принцип в отношении капитала:

Капитал = Средства - Обязательства.

Балансовое уравнение должно всегда выполняться, если только не допущена бухгалтерская ошибка, что возможно при ручном учете. Наша программа устроена так, что невозможно нарушить балансовое уравнение.

ПРИНЦИП ДЕНЕЖНОГО ИЗМЕРЕНИЯ (GAAP)

Бухгалтерский учет оперирует только с данными, имеющими денежное выражение.

ПРИНЦИП АВТОНОМНОСТИ ПРЕДПРИЯТИЯ (GAAP)

Счета предприятия должны быть отделены от счетов владельцев предприятия и работников.

ПРИНЦИП НЕПРЕРЫВНОСТИ (GAAP)

Бухгалтерский учет исходит из предположения, что предприятие будет функционировать неопределенное время (никогда не будет закрыто), если отсутствуют факты, подтверждающие обратное. Вследствие того, что баланс опирается на принцип непрерывности, то он не содержит информации о том, за какую цену могли бы быть реализованы средства предприятия, если бы оно прекратило свою деятельность.

ПРИНЦИП СЕБЕСТОИМОСТИ (GAAP)

Когда предприятие приобретает те или иные средства, то расходы на их приобретение, отраженные в бухгалтерских отчетах, называются **себестоимостью** этих средств. В задачи бухгалтерского учета не входит анализ изменений рыночной стоимости средств. Вместо этого бухгалтерский учет исходит из их себестоимости. Принцип себестоимости состоит в том, что бухгалтерский учет опирается на себестоимость средств, а не на их рыночную стоимость.

ЧТО МОЖЕТ БЫТЬ ОТНЕСЕНО К СРЕДСТВАМ ПРЕДПРИЯТИЯ?

Каждая позиция, чтобы быть отнесенной к средствам, должна отвечать следующим **трем** требованиям:

- Позиция должна **контролироваться** предприятием. Если позиция находится в собственности предприятия, то она контролируется предприятием. Возможны и иные случаи контроля, например капитальная аренда. Но, как правило, средством предприятия может считаться только то, что принадлежит предприятию как собственность.

- Позиция должна представлять для предприятия определенную **ценность**. Испорченное и не подлежащее ремонту оборудование, бракованная продукция и т.п. не могут быть отнесены к средствам предприятия.

- Позиция должна быть **приобретена по измеряемой стоимости**. Так, например, если компания А **приобрела** торговую марку компании В за 1 миллион долларов, то эта торговая марка **может** быть отнесена к средствам компании А. Напротив, если благодаря высокому качеству продукции, компания А **пользуется** прекрасной репутацией на рынке, то репутация компании А **не может** быть отнесена к ее средствам.

ОБОРОТНЫЕ СРЕДСТВА

К оборотным средствам относятся деньги, а также средства, которые могут быть обращены в деньги или полностью использованы в ближайшем будущем (обычно в течение одного года):

- *Денежные средства,*
- *Ценные бумаги,*
- *Товарно-материальные средства,*
- *Счета дебиторов (не обеспеченные векселями долги предприятию),*
- *Векселя полученные,*
- *и т.п*

ОСНОВНЫЕ СРЕДСТВА

Средства, которые **предполагается** использовать дольше, чем в течение одного года, называются основными средствами.

НЕОСЯЗАЕМЫЕ СРЕДСТВА (ПРАВА СОБСТВЕННОСТИ)

- *Купленные патенты и торговые марки,*
- *Гудвил купленных предприятием компаний (превышение цены купленной компании над стоимостью ее средств благодаря деловым* связям и репутации купленной компании),
- *Страховки на длительные сроки (дольше отчетного периода),*
- *и т.п.*

ОСЯЗАЕМЫЕ СРЕДСТВА (ИМУЩЕСТВО)

- *Здания, сооружения, оборудование,*
- *Купленные месторождения полезных ископаемых,*
- *и т.п.*

Основные средства предприятия в бухгалтерском учете подвергают *амортизации*, то есть частями списывают в расход. Для учета таких операций применяется отдельный счет "*Аккумулированная амортизация*". В традиции GAAP этот счет располагают в левой стороне баланса, хоть он и имеет кредитовое сальдо.

КРАТКОСРОЧНЫЕ ОБЯЗАТЕЛЬСТВА

В этом разделе баланса группируются счета, выражающие долги предприятия, требующие погашения в течение отчетного периода:

- *счета кредиторов (не обеспеченные векселями долги компании)*
- *векселя выданные*
- *начисленная, но пока невыплаченная работникам зарплата*
- *налоги к оплате*
- *проценты по долгосрочным кредитам*
- *и т.п.*

КОЭФФИЦИЕНТ ТЕКУЩЕЙ ЛИКВИДНОСТИ

Оборотные средства и краткосрочные обязательства указываются в балансе отдельно от основных средств и долгосрочных обязательств. Это позволяет определять **ликвидность** предприятия (способность предприятия погашать свою текущую задолженность). Коэффициентом текущей ликвидности называется отношение оборотных средств к краткосрочным обязательствам:

Коэфф. текущей ликвидности = Оборотные средства / Краткосрочные обязательства

Результат обычно округляется до первой цифры после запятой. Предприятие считается **кредитоспособным**, если его коэффициент текущей ликвидности - не менее 2 (для США). Слово ликвидность происходит от английского слова liquid (жидкий, текучий).

ДОЛГОСРОЧНЫЕ ОБЯЗАТЕЛЬСТВА

В этом разделе баланса группируются счета, выражающие долги, полное погашение которых в течение отчетного периода не требуется (например, долгосрочные ссуды, погашаемые частями из года в год). В международной практике в этом разделе показывается также так называемый "Отсроченный налог на прибыль" - долг предприятия государству, образующийся благодаря использованию права на ускоренную амортизацию основных средств.

КАПИТАЛ

Раздел "Капитал" баланса иногда также называют "Капитал акционеров" или "Капитал владельцев". В этом разделе указываются источники капитала предприятия. В зависимости от организационно-правовой формы предприятия, они могут быть разными. Например, для американской корпорации:

- *обычные акции в обращении (проданные акции)*
- *акции в портфеле (собственные акции, выкупленные предприятием). Имеет отрицательный остаток.*
- *привилегированные акции (акции с фиксированным процентом)*
- *прочий акционерный капитал*
- *и т.п.*

Второй составляющей капитала любого предприятия является:

- *нераспределенная прибыль.*

Нераспределенная прибыль есть капитал, заработанный предприятием в результате эффективной деятельности и остающийся в его распоряжении после того, как акционерам была уплачена часть прибыли в виде дивидендов.

Нераспределенная прибыль = Прибыль – Дивиденды

Нераспределенная прибыль представляет собой ту часть капитала, которая накоплена не за последний отчетный период, а с момента создания предприятия.

Источники увеличения и уменьшения капитала:

- Вложения (Инвестиции), Доходы увеличивают КАПИТАЛ
- Изъятия (Дивиденды), Расходы уменьшают КАПИТАЛ

ПРИБЫЛЬ

Сумма, на которую увеличивается капитал предприятия за определенный период, **в результате его эффективной деятельности**, называется прибылью за этот период. Бухгалтерский отчет, называемый отчетом о

прибыли, содержит информацию о прибыли за определенный период времени. В отличие от баланса, отчет о прибыли относится к *периоду* времени, а не к *моменту*.

ДОХОДЫ И РАСХОДЫ

- Увеличение Капитала предприятия в результате деятельности предприятия называется Доходом.
- Уменьшение Капитала предприятия в результате деятельности предприятия называется Расходом.
- Прибыль представляет собой разность между доходами и расходами за определенный период времени.

МЕТОД НАЧИСЛЕНИЙ

Многие небольшие компании ведут учет только по поступлениям и выплатам денежных средств. Такой метод бухгалтерского учета называется бухгалтерский учет фактических денежных средств или **кассовый метод**. Этот метод не позволяет определить изменение Капитала компании. Большинство же компаний (в мире) ведет учет как денежных поступлений и выплат, так и доходов и расходов. Этот метод называется **методом начислений**.

ПРИНЦИП КОНСЕРВАТИЗМА (ОСТОРОЖНОСТИ) (GAAP)

- Увеличение капитала (Доход) признается тогда, когда оно стало **вполне определенным** событием.
- Уменьшение капитала (Расход) признается тогда, когда это становится **вполне возможным** событием.

Например, если компания выдвинула судебный иск, сумма возможных издержек сразу записывается в Расход. Когда компания выиграет дело, выигранную сумму можно будет зачислить в Доход компании.

ПРИНЦИП МАТЕРИАЛЬНОСТИ (СУЩЕСТВЕННОСТИ) (GAAP)

- Пренебрежение незначительными и маловажными событиями.
- Отражение всех важных событий.

ПРИНЦИП РЕАЛИЗАЦИИ (GAAP)

Доходы обычно учитываются, когда продукция либо услуги доставляются клиенту.

Принцип реализации отвечает на вопрос о том, **когда** следует учитывать доход. Принцип же консерватизма позволяет ответить на вопрос о том, **какая сумма** доходов подлежит учету. Возникающие при этом противоречия разрешаются введением дополнительных счетов:

- *Учет сомнительных платежей.*
- *Безнадежный долг.*

РАСХОДЫ И ИЗДЕРЖКИ НА ПРИОБРЕТЕНИЕ

Издержки на приобретение расходом **не являются**, так как *Денежные средства*, потраченные на приобретение, превращаются в *Товарно-Материальные средства* компании. Расход Товарно-материальных средств происходит в момент продажи компанией товаров или при ином их безвозвратном использовании или исчезновении. Расходы, создающие обязательства (зарплата, невыплаченная аренда, непогашенные проценты по кредитам) возникают в тот момент, когда возникают соответствующие обязательства - например, расходы на зарплату происходят в тот момент, когда работники работают, а не в тот момент, когда они получают зарплату. Для разделения во времени записей о расходах и выплатах используется метод начислений.

ПРИНЦИП СООТВЕТСТВИЯ (GAAP)

Расходы данного отчетного периода - это себестоимость осуществления хозяйственной деятельности (затраты), в результате которой появляется *Доход* данного периода.

ПРИНЦИП ОБЪЕКТИВНОСТИ (GAAP)

Суммы должны основываться на объективных данных (иметь ссылки на объективные факты), что позволяет рассчитывать на то, что два или более бухгалтеров, работающие с одними и теми же данными придут к похожим результатам. При работе с документами **предпочтение следует отдавать фактам, а не формальным документам.**

ПРИНЦИП ПЕРИОДИЗАЦИИ (GAAP)

Учет закрывается периодами. Прибыль компании - параметр, относящийся к периоду.

ПРИНЦИП ПОСТОЯНСТВА МЕТОДОВ УЧЕТА (GAAP)

Следует учитывать, что существуют общепринятые, но иногда несовместимые друг с другом методы учета.

Бухгалтерские записи

На сегодня известно множество форм бухгалтерского учета и форм бухгалтерских записей. Мы будем использовать так называемые *смешанные проводки,* изобретенные в свое время в Германии и популярные в современном англо-американском бухгалтерском учете.

Главное значение этих записей состоит в том, что любая финансовая сделка или событие имеет адекватный способ интерпретации, отражающий изменение текущего финансового положения компании в результате данной сделки или события. Поэтому иногда бухгалтерский учет называют *языком бизнеса*. Если текущее финансовое положение компании изображается в виде двустороннего баланса, то любая бухгалтерская операция непосредственно влияет на этот баланс. Без ясного понимания того, как те или иные операции влияют на баланс, качественный бухгалтерский учет невозможен.

Вспомним еще раз, как организован баланс.

Классический балансовый отчет (balance sheet) имеет две стороны. Левая сторона называется «Средства», правая сторона называется «Обязательства и Капитал».

В левой стороне перечисляются все *средства*, имеющиеся у компании на *данный момент времени*. К средствам относят: принадлежащие компании денежные средства, товары и материалы (по покупной стоимости), счета должников (дебиторов), основные средства (здания, сооружения, оборудование по остаточной стоимости) и другие ресурсы компании такие, как стоимость приобретенных патентов и торговых марок, предоплаченная аренда и т.п. Важным принципом при формировании средств компании является *принцип себестоимости*. В соответствии с этим принципом все средства учитываются по той стоимости, по которой они *были приобретены*, независимо от того, какова их текущая рыночная цена.

В правой стороне перечисляются все долги компании (обязательства) и показывается стоимость капитала компании. Зачем нужна правая часть баланса? Дело в том, что средства компании возникают из совершенно конкретных источников. Это могут быть *кредиты, инвестиции* и *нераспределенная прибыль* компании. *Кредиторы* предоставляют компании денежные и иные ресурсы в долг. Денежно выраженные долги компании кредиторам называются *обязательствами*. *Инвесторы* предоставляют компании средства в обмен на право управления компанией и извлечения прибыли из нее (в частности, в виде акций, если это корпорация). Компания ничего не должна своим инвесторам. Иски инвесторов имеют *остаточный характер* по отношению к искам кредиторов. Это означает, что если компания по каким-либо причинам будет закрыта и ее средства распроданы, то в первую очередь будут удовлетворены иски кредиторов, и только то, что останется – поделено между инвесторами (бывшими владельцами). Таким образом, в правой части баланса перечисляются все **Обязательства** (денежно выраженные долги компании) и **Капитал** (собственные средства компании).

Балансовым уравнением называется равенство:

Средства = Обязательства + Капитал.

Иногда его записывают, подчеркивая остаточный принцип в отношении капитала.

Капитал = Средства - Обязательства

Допустим, мы создали компанию 1 сентября 2003г, положив на ее счет $10 000.

После этой операции баланс компании на 1 сентября 2003г. выглядит следующим образом:

Средства:		**Обязательства и Капитал:**	
Денежные средства	*$10 000*	*Капитал владельцев*	*$10 000*
Всего, Средства	**$10 000**	**Всего, Обязательства и Капитал**	**$10 000**

Допустим, компания 10 сентября 2003г. взяла еще кредит на сумму $8 000.

Ее баланс на 10 сентября 2003г. выглядит так:

Средства:		**Обязательства и Капитал:**	
Денежные средства	*$18 000*	*Кредит*	*$8 000*
		Капитал владельцев	$10 000
Всего, Средства	**$18 000**	**Всего, Обязательства и Капитал**	**$18 000**

Как мы видим, компания в данный момент обладает денежными средствами в $18 000, из которых $8 000 составляют обязательства (взятый кредит), а $10 000 – собственный капитал (предоставленные инвестором

средства). Предположим, компания 12 сентября 2003г. приобрела автомобиль за $7 000 и заплатила за аренду офиса на полгода вперед ($6 000). Денежные средства при этом уменьшились.

Теперь ее баланс (на 12 сентября 2003г.) выглядит следующим образом:

Средства:		**Обязательства и Капитал:**	
Денежные средства	*$5 000*	Обязательства	$8 000
Автомобиль	*$7 000*	Капитал владельцев	$10 000
Предоплаченная аренда	*$6 000*		
Всего, Средства	**$18 000**	**Всего, Обязательства и Капитал**	**$18 000**

Операция приобретения не изменяет общую сумму средств компании. Просто одни средства (в данном случае, денежные) превращаются в другие (автомобиль и т.п.), что отражает *принцип себестоимости*.

Предположим, компания оказала услуг клиентам на сумму $3500. Соответственно возросли ее денежные средства.

Покажем баланс с учетом этой операции.

Баланс на 30 сентября 2003г.

Средства:		**Обязательства и Капитал:**	
Денежные средства	*$8 500*	Кредит	$8 000
Автомобиль	$7 000	Капитал владельцев	$10 000
Предоплаченная аренда	$6 000	*Услуги, предоставленные клиентам*	*$3 500*
Всего, Средства	**$21 500**	**Всего, Обязательства и Капитал**	**$21 500**

Как видно из баланса, средства компании возросли на сумму $3 500. Это увеличение средств имело своим источником *эффективную деятельность компании*. Так как обязательства компании при этом не возросли, можно говорить о том, что возрос ее капитал. Увеличение капитала за определенный период времени за счет эффективной деятельности компании называется *прибылью*, уменьшение капитала – *убытком*. Для того чтобы показать составляющие прибыли принято заводить временные счета *доходов* и *расходов*. В данном случае счет “Услуги, предоставленные клиентам” является счетом доходов.

Предположим, компания наняла работника, который проработал месяц, и компания задолжала ему $500 зарплаты. Покажем это, как *расход* компании. Заодно покажем, как делается *начисление* зарплаты в условиях, когда деньги *еще не выплачены*. При этом мы еще несколько усложним баланс, создав в нем разделы и введя некоторые обобщающие цифры.

Баланс на 30 сентября 2003г.

Средства:		**Обязательства и Капитал:**	
Оборотные средства:		**Обязательства:**	
Денежные средства	$8 500	*Невыплаченная зарплата*	*$500*
Предоплаченная аренда	$6 000	Кредит	$8 000
Всего, оборотные средства	**$14 500**	**Всего, Обязательства**	**$8 500**
Основные средства:		**Капитал**	
Автомобиль	$7 000	Капитал владельцев	$10 000
Всего, Основные средства	**$7 000**	**Прибыль**	
		Услуги, предоставленные клиентам	$3 500
		Расходы на зарплату	*- $500*
		Всего, Прибыль	**$3 000**
		Всего, Капитал	**$13 000**
Всего, Средства	**$21 500**	**Всего, Обязательства и Капитал**	**$21 500**

В какой-то момент времени (1 октября 2003г.) компания выплатила работнику его зарплату. При этом уменьшились ее денежные средства, и уменьшилась задолженность по зарплате.

Баланс на 1 октября 2003г.

Средства:		**Обязательства и Капитал:**	
Оборотные средства:		**Обязательства:**	
Денежные средства	*$8 000*	*Невыплаченная зарплата*	*$0*
Предоплаченная аренда	$6 000	Кредит	$8 000
Всего, оборотные средства	**$14 000**	**Всего, Обязательства**	**$8 000**
Основные средства:		**Капитал**	
Автомобиль	$7 000	Капитал владельцев	$10 000
Всего, Основные средства	**$7 000**	**Прибыль**	
		Услуги, предоставленные клиентам	$3 500
		Расходы на зарплату	- $500
		Всего, Прибыль	**$3 000**
		Всего, Капитал	**$13 000**
Всего, Средства	**$21 000**	**Всего, Обязательства и Капитал**	**$21 000**

Прибыль при этом не изменилась. Таким образом, начисления (в данном случае зарплаты) позволяют разделить во времени учет прибыли и учет денежных расходов, относя те и другие к тем периодам, когда они реально имели место.

Сделаем еще одно начисление. Спишем часть предоплаченной аренды офиса, которая использовалась в течение месяца, пока офис работал. За месяц компания израсходовала $1 000 из предоплаченной полугодовой аренды, которая ранее составляла $6 000, а теперь составляет $5 000.

Баланс на 1 октября 2003г.

Средства:		**Обязательства и Капитал:**	
Оборотные средства:		**Обязательства:**	
Денежные средства	$8 000	Невыплаченная зарплата	$0
Предоплаченная аренда	*$5 000*	Кредит	$8 000
Всего, оборотные средства	**$13 000**	**Всего, Обязательства**	**$8 000**
Основные средства:		**Капитал**	
Автомобиль	$7 000	Капитал владельцев	$10 000
Всего, Основные средства	**$7 000**	**Прибыль**	
		Услуги, предоставленные клиентам	$3 500
		Расходы на зарплату	- $500
		Расходы на аренду офиса	*- $1 000*
		Всего, Прибыль	**$2 000**
		Всего, Капитал	**$12 000**
Всего, Средства	**$20 000**	**Всего, Обязательства и Капитал**	**$20 000**

Как мы видим, можно *непосредственно в балансе* компании отражать все финансовые операции. Однако хорошо бы иметь историю операций с каждым счетом. Для этого были придуманы Т-счета. Т-счет берет свое название от буквы Т, которую он напоминает. Т-счет имеет две колонки. В одну колонку записываются все суммы, которые увеличивали счет, в другую – все суммы, которые уменьшали счет. Левая колонка Т-счета называется *дебет*, а правая колонка - *кредит*. Запись в левую колонку любого счета называется *записью в дебет (или дебетованием счета)*, а запись в правую колонку - *записью в кредит (или кредитованием счета)*.
Например, так выглядит Т-счет «Денежные средства» для тех операций, которые имели место в нашей компании:

«Денежные средства»

Дата	**Дебет**	**Кредит**	**Примечание**
01.09.03	$10 000		Внесение начального капитала
10.09.03	$8 000		Получение кредита
12.09.03		$6 000	Оплата аренды офиса на полгода вперед
12.09.03		$7 000	Приобретение автомобиля
30.09.03	$3 500		Доходы от предоставленных услуг
01.10.03		$500	Выплата зарплаты
итого	**$21 500**	**$13 500**	**Обороты счета**
	$8 000		**Конечный остаток**

Сумма всех записей в дебет называется *дебет-оборотом*, сумма всех записей в кредит – *кредит-оборотом* счета. При ручном учете обороты счета подчеркиваются одинарной чертой, а конечный остаток – двойной чертой.

Существует простое правило для всех Т-счетов: **все левосторонние счета баланса увеличиваются записью в левую колонку, то есть в дебет, тогда как все правосторонние счета баланса увеличиваются записью в правую колонку, то есть в кредит.** Соответственно, все левосторонние счета уменьшаются записью в правую колонку (кредит), а правосторонние уменьшаются записью в левую колонку (дебет). Как легко заметить счет «Денежные средства» – левосторонний счет. Покажем какой-нибудь правосторонний счет, например, счет «Невыплаченная зарплата», на котором учитывалась задолженность компании по зарплате перед работником:

«Невыплаченная зарплата»

Дата	Дебет	Кредит	Примечание
30.09.03		$500	Начисление зарплаты
01.10.03	$500		Выплата зарплаты
итого	**$500**	**$500**	**Обороты счета**
		$0	**Конечный остаток**

Каждая бухгалтерская операция (или, как ее еще называют, проводка или финансовая транзакция) должна всегда затрагивать одновременно минимум два счета (два и более). Причем сумма записей в дебет должна равняться сумме записей в кредит. Этот принцип называется правилом двойной записи. При ручном учете ведется журнал проводок, в который все финансовые операции вносятся последовательно – одна за другой.

Покажем, как бы выглядели наши финансовые операции в таком журнале:

Дата	Финансовая операция	Дебет	Кредит
01.09.03	**Внесение начального капитала**		
	Д-т, Денежные средства	$10 000	
	К-т, Капитал владельцев		$10 000
10.09.03	**Получение кредита**		
	Д-т, Денежные средства	$8 000	
	К-т, Кредит		$8 000
12.09.03	**Оплата аренды офиса на полгода вперед**		
	Д-т, Предоплаченная аренда	$6 000	
	К-т, Денежные средства		$6 000
12.09.03	**Приобретение автомобиля**		
	Д-т, Автомобиль	$7 000	
	К-т, Денежные средства		$7 000

Для тех, кто желает лучше познакомиться с бухгалтерским учетом, рекомендуем великолепную книгу профессора Гарвардского Университета Роберта Антони «Основы бухгалтерского учета».

В большинстве бухгалтерских компьютерных систем повторяется ручная модель учета и журналу проводок придается большое значение. Его часто используют как основной или даже единственный навигатор по бухгалтерским данным. В нашей системе журналу операций не придается большого значения. Основным навигатором является баланс. Любая операция (как ручная, так и автоматическая) ***мгновенно изменяет баланс*** компании. Простейшие проводки можно делать прямо в окне «Баланс», выбрав два счета: один в правой стороне окна, другой - в левой. После любой операции программа освежает баланс, и мы можем видеть непосредственные результаты наших финансовых операций. Выбрав любой счет в балансе, можно включить клавишей ***F3*** отображение ***Т-счета*** на противоположной панели. Выбрав любую цифру в ***Т-счете***, мы видим под ним финансовую операцию, которая эту цифру породила. Таким образом, программа осуществляет навигацию ***«от баланса»***. Навигация «от баланса» позволяет никогда не путаться в бухгалтерских данных, даже если таких данных очень много (сотни тысяч проводок), так как все заранее «разложено по полочкам». Это стало возможным благодаря способности программы Allegro запрашивать полный бухгалтерский баланс компании

достаточно быстро (за одну секунду формируется полный баланс из 30 тыс. проводок на компьютере с тактовой частотой 650 МГц).

Валютные слои.

Для работы с разными валютами в программе используются так называемые ***слои***. В каждом слое поддерживается независимый баланс. Это означает, что в любой бухгалтерской операции при использовании балансовых счетов сумма записей в дебет должна равняться сумме записей в кредит независимо, в каждом используемом в этой операции слое.

Основные регистры счетов

Все счета в нашей программе организованы в деревья. Различаются ***счета нижнего уровня*** и ***каталоги***. Счета нижнего уровня не содержат дочерних счетов и их можно использовать в бухгалтерских операциях. Каталогами называются счета, содержащие дочерние счета. Каталоги изображаются в виде папок. Бухгалтерские операции с каталогами невозможны. Каждое дерево счетов имеет корневой счет, который называется регистром. Например, ***двусторонний баланс компании*** состоит из двух ***основных регистров***: «Средства» и «Обязательства и Капитал». Корневой счет «Средства» обычно содержит два каталога счетов: «Оборотные средства» и «Основные средства». Каталог «Оборотные средства» обычно содержит различные счета или каталоги счетов оборотных средств, такие, как «Денежные средства», «Товары на складах», «Счета дебиторов» и им подобные. Никаких ограничений на степень «вложенности» каталогов счетов не существует. Нумерация счетов также не обязательна. Однако каждый счет должен иметь ***уникальное название***. Регистры можно переименовывать. Например, можно переименовать «Средства» в «Активы», а «Обязательства и Капитал» в «Пассивы», если это необходимо.

Окно «Баланс» содержит две симметричные панели: левую и правую. На каждой из этих панелей может быть отображен любой регистр счетов. По умолчанию отображаются левая и правая стороны баланса компании (левая сторона – «Средства», правая – «Обязательства и Капитал»). В верхней части окна расположен ***селектор даты баланса***. Баланс отображает финансовое положение компании на дату, которая выбрана в селекторе. Таким образом, мы можем увидеть баланс компании ***на любую выбранную дату***. Все суммы на вышестоящих счетах-каталогах автоматически образуются путем суммирования сумм на дочерних счетах нижних уровней.

В верхней части окна расположены кнопки управления режимом отображения сумм. В зависимости от выбранного режима, можно отобразить для всех счетов на дату баланса:

- Остатки
- Обороты
- Остатки по слоям

В верхней части окна расположен также ***селектор валютных слоев*** и кнопка для ***консолидирования данных по слоям***. Если включен режим консолидирования (кнопка нажата), то отображаются консолидированные данные - сумма слоев с пересчетом в текущую валюту по ближайшим к дате баланса курсам валют. Если режим консолидирования отключен, то отображаются только данные выбранного валютного слоя.

Счета можно ***перемещать*** между каталогами в процессе учета, даже если по этим счетам имеются бухгалтерские записи. Можно перемещать также целые каталоги счетов и ***изменять расположение счетов*** внутри каталога. Счета можно переименовывать и создавать новые счета, постепенно приближая финансовую картину компании к такому виду, который бы максимально наглядно и верно отражал реальное положение дел и упрощал бы интерпретацию всех балансовых показателей. Например, принято располагать счета в порядке убывания их ликвидности. Это значит, что чаще используемые счета имеет смысл переместить вверх, а реже используемые – вниз (внутри каталогов, в которых они расположены).

Дополнительные регистры счетов

Кроме основных регистров («Средства», «Обязательства и Капитал») можно создавать ***дополнительные регистры счетов***. Регистры могут быть ***балансовыми*** и ***забалансовыми***. Каждый дополнительный регистр счетов может быть привязан к одному ***справочнику*** и такой регистр называется ***аналитическим регистром***. Балансовые регистры всегда аналитические и имеют ***специальные счета остатков*** в составе основных регистров. Счета остатков балансовых регистров бывают двух видов:

- два развернутых остатка
- один простой остаток

Зачем нужны регистры счетов? Дело в том, что при большом количестве объектов учета (например, при большом количестве контрагентов или товаров) заводить для каждого объекта отдельный счет в балансе компании не всегда удобно. К тому же с одним и тем же объектом возможны одновременно разные, но близкие по смыслу финансовые отношения. Например, у фирмы может быть контрагент, одновременно являющийся и поставщиком и покупателем. И если необходимо учитывать эти разные отношения отдельно (без автоматического встречного взаимозачета), то, так или иначе, придется заводить два отдельных счета. Разумно было бы иметь такую организацию счетов, при которой мы могли бы видеть эти отношения отдельно, как остатки на двух разных счетах, но в то же время могли бы видеть и суммарный финансовый результат по данному контрагенту.

Например, все счета, служащие для учета отношений с контрагентами («Расчеты с поставщиками», «Расчеты с покупателями», «Расчеты по претензиям», «Расчеты с разными дебиторами и кредиторами» и др.) могут быть созданы в дополнительном балансовом регистре «Расчеты с контрагентами», привязанному к справочнику «Контрагенты». При создании такого регистра следует указать, что он будет иметь два счета развернутых остатков в балансе. После создания регистра, в балансе компании появятся два новых счета: «Расчеты с контрагентами (дебет)» и «Расчеты с контрагентами (кредит)». Эти два счета рекомендуется переименовать в «Счета дебиторов» и «Счета кредиторов» соответственно. После переименования счет «Счета дебиторов» рекомендуется переместить в каталог «Оборотные средства», а счет «Счета кредиторов» – в каталог «Краткосрочные обязательства». При запросе баланса система автоматически определит по каждому контрагенту остатки на каждом счете регистра и просуммирует раздельно все остатки с дебетовым и с кредитовым превышением, разнося их на счета соответствующих развернутых остатков.

Иногда полезно заводить балансовые регистры с простым остатком, участвующим в балансе компании. Например, можно завести регистр «прибыль от магазинов», привязанный к справочнику «Магазины» и его простой остаток вставить в каталог «Текущая прибыль» в каталоге «Капитал» правой части баланса компании.

Существуют ограничения на перемещение счетов между регистрами. Например, можно перемещать счета между основными регистрами баланса, однако невозможно перемещать счета из одного дополнительного регистра в другой. Внутри отдельно взятого регистра счета можно перемещать как угодно.

Для каждого регистра счетов можно установить свой цвет панели, что повышает узнаваемость регистров и облегчает их восприятие пользователем. Как уже было сказано, в каждой стороне окна «Баланс» можно отобразить любой регистр. Аналитические регистры отображаются с селектором объекта справочника и кнопкой фильтра в верхней части панели. Если в селекторе выбрать определенный объект справочника и нажать кнопку фильтра, то регистр на всех своих счетах отобразит остатки по выбранному объекту учета.

Оганизация счетов в базе данных. Системные таблицы COMPANY, ACC, ACC_ROOTS.

Информация о компаниях хранится в системной таблице **COMPANY**.

Таблица компаний **COMPANY**

Название поля	**Тип данных**	**Назначение**
COMPANY_ID	INTEGER	Внутренний ID компании
COMPANY_CODE	VARCHAR(20)	Код компании (необязателен)
NAME	VARCHAR(80)	Название компании
SMALL_IMAGE	BLOB	Значок компании в формате BMP 16x16 пикселей

При создании новой компании COMPANY_ID формируется при помощи генератора COMPANY_ID_GEN.

Информация о счетах и регистрах хранится в двух системных таблицах: **ACC** и **ACC_ROOTS**.

Таблица счетов **ACC**

Название поля	**Тип данных**	**Назначение**
ACC_ID	INTEGER	Внутренний ID счета
PARENT_ID	INTEGER	Внутренний ID родительского счета (каталога)
ACC_ORDER	INTEGER	Порядковый номер счета внутри каталога
HAS_CHILDREN	SMALLINT	Если счет является каталогом, то в этом поле записывается 1, если не является каталогом, то записывается 0
ACC_CODE	VARCHAR(80)	Пользовательский номер счета (необязателен)
NAME	VARCHAR(80)	Наименование счета (уникальное в пределах компании)
COMMENTS	BLOB	Примечания к счету
IS_SALDO	SMALLINT	Если счет представляет остаток дополнительного регистра, то в этом поле записывается 1, для обычных счетов 0
SALDO_KIND	SMALLINT	Вид остатка: 0 –дебетовый развернутый остаток, 1 – кредитовый развернутый остаток, -1 – простой остаток
SALDO_ROOT_ID	INTEGER	Только для счетов остатков регистров: внутренний ID регистра, остаток которого представляет данный счет
NO_BALANCE	SMALLINT	Признак забалансового счета. 0 – балансовый счет 1 – забалансовый счет
COMPANY_ID	INTEGER	Внутренний ID компании

Таблица регистров **ACC_ROOTS**

Название поля	**Тип данных**	**Назначение**
ACC_ID	INTEGER	Внутренний ID счета
ROOT_KIND	SMALLINT	Тип регистра: 0 – левой стороны баланса 1 – правой стороны баланса 2 – балансового регистра 3 – забалансового регистра
CLASS_ID	INTEGER	Внутренний ID класса справочника, привязанного к регистру
USE_QUANTITY	SMALLINT	Признак количественного учета: 0 – без количественного учета 1 – используется количественный учет
COLOR	VARCHAR(20)	Цвет панели
FONT_NAME	VARCHAR(50)	Название шрифта
FONT_SIZE	INTEGER	Размер шрифтра
FONT_CHARSET	INTEGER	Кодовая страница шрифта

OBJECT_TREE	SMALLINT	Отображать дерево объектов (для небольших справочников) 0 – не отображать дерево 1 – отображать дерево
RIGHT_HAND	SMALLINT	Способ отображения остатков на счетах регистра: 0 – левосторонний регистр 1 – правосторонний регистр

Все счета хранятся в таблице ACC. В поле PARENT_ID указывается идентификатор родительского счета-каталога. Если значение PARENT_ID равно 0, то такой счет является корневым счетом (регистром) и он имеет вхождение также в таблицу ACC_ROOTS.

При создании нового счета его ACC_ID формируется при помощи генератора ACC_ID_GEN.

Кроме всего прочего, в таблице ACC существует запись о «нулевом счете». Его ACC_ID = 0. Дело в том, что ссылочная целостность в базе данных организована таким образом, что поля типа «счет» в других таблицах объявлены как NOT NULL и пустым значением для них считается ACC_ID=0.

После создания каждой новой компании в триггере COMPANY_AFTER_INSERT создаются основные регистры баланса этой компании и несколько счетов (будущие основнные разделы) в каждом регистре:

Средства	**Обязательства и Капитал**
Оборотные средства	Краткосрочные обязательства
Основные средства	Долгосрочные обязательства
	Капитал

При создании *пустой конфигурации* отрабатывается сценарий базы данных ALLEGRO.SQL, в котором создается одна компания с названием «Моя компания».

Работа со счетами, системные хранимые процедуры ACC_TREE, ACC_PARENTS, ACC_ROOT

Зная устройство таблицы счетов ACC, мы можем при помощи SQL-запросов выбирать необходимые группы счетов. Предположим, что имеется складской регистр «Товары на складах», в котором заведены дочерние счета «Главный склад», «Магазин на ул. Парковой, 2», «Магазин на ул. Пушкинской, 5». Простейший способ запросить все имеющиеся склады:

```
select acc_id, parent_id, name
from acc
where parent_id = 115;
```

Результат запроса:

ACC_ID	PARENT_ID	NAME
123	115	Главный склад
117	115	Магазин на ул.Парковой 2
207	115	Магазин на ул.Пушкинской, 5

В данном случае ACC_ID = 115 - это идентификатор счета «Товары на складах», являющегося непосредственным родительским счетом (счетом-каталогом) для счетов конкретных складов. Такой запрос может нам понадобиться, например, если нужно организовать выпадающий список всех имеющихся складов в каком-либо окне, где пользователь вносит какие-то данные.

Разумеется, такой простой запрос позволяет лишь запросить все дочерние счета *одного уровня*. А как нам действовать, если нужно получить субсчета *всех уровней* для какого-то счета? Для этого имеется хранимая процедура ACC_TREE, возвращающая все сочетания старший счет-младший счет для всех уровней дерева.

Результирующий набор, поставляемый хранимой процедурой **ACC_TREE**

Название поля	**Тип данных**	**Назначение**
ACC_ID	INTEGER	Внутренний ID счета
CHILD_ID	INTEGER	Внутренний ID счета-потомка
SALDO_KIND	SMALLINT	Тип сальдо (для запросов баланса компании)
TURN_KIND	SMALLINT	Тип оборотов (для запросов баланса компании)

Процедура ACC_TREE для каждого счета ACC_ID возвращает парные сочетания его самого и всех его потомков CHILD_ID, независимо от уровня вложенности. Некоторые одинаковые комбинации ACC_ID, CHILD_ID встречаются несколько раз и отличаются значениями SALDO_KIND и TURN_KIND, которые используются при запросе баланса компании (сейчас мы их назначение рассматривать не будем). Важно то, что, используя процедуру ACC_TREE, мы можем легко получить ID всех дочерних счетов для счета с ID=115, написав такой запрос:

```
select distinct child_id
from acc_tree
where (acc_id = 115) and (child_id <> 115)
```

Условие (child_id <> 115) мы ввели для того, чтобы исключить из выходного набора сам родительский счет ACC_ID = 115. Слово `distinct` устраняет дубликаты, которые поставляются процедурой ACC_TREE и в данном случае нам не нужны.

Если мы хотим получить не только ID счетов, но и их названия, объединим результат работы процедуры ACC_TREE с таблицей счетов ACC:

```
select distinct a.acc_id, a.name
from acc_tree atr, acc a
where
  a.acc_id = atr.child_id and
  atr.acc_id = 115
```

Используя запросы, мы можем гарантировать, что в выпадающем списке «Выберите склад» всегда будут отображаться все нужные счета, даже если пользователь сам добавит новый счет в регистр (например, в случае возникновения нового склада «Магазин на ул. Чехова, 13»).

Еще необходимо упомянуть о системной хранимой процедуре ACC_PARENTS. Эта процедура используется процедурой ACC_TREE для построения всех сочетаний счетов вида старший-младший, но мы можем иногда ее применять и для своих целей. Процедура ACC_PARENTS имеет один входной параметр CHILD_ID типа INTEGER, поэтому вызывать ее следует с входным параметром. Процедура ACC_PARENTS возвращает *всех предков счета, включая его самого* плюс, признаки SALDO_KIND и TURN_KIND, которые сейчас нас не интересуют. Важно то, что один и тот же счет процедура может вернуть несколько раз, поэтому рекомендуется использовать в запросах слово `distinct`.

Результирующий набор, поставляемый хранимой процедурой **ACC_PARENTS**

Название поля	**Тип данных**	**Назначение**
PARENT_ID	INTEGER	Внутренний ID счета
SALDO_KIND	SMALLINT	Тип сальдо (для запросов баланса компании)
TURN_KIND	SMALLINT	Тип оборотов (для запросов баланса компании)

Например, запросим всех предков счета с ACC_ID = 1083

```
select distinct parent_id
from acc_parents(1083)
```

Если нам нужны не только ID счетов, но и их имена, то можем объединить результаты работы процедуры ACC_PARENTS с таблицей счетов:

```
select distinct a.acc_id, a.name
from acc_parents(1083) ap, acc a
where ap.parent_id = a.acc_id
```

Примерный результат запроса:

ACC_ID	**NAME**
1	Средства
3	Оборотные средства
1078	Денежные средства
1083	Касса, Пушкинская 5

Запрос всех предков счета, который мы привели, может понадобиться достаточно редко, чаще бывает нужно выяснить свойства регистра для какого-то выбранного счета. Для того чтобы узнать свойства регистра, к которому принадлежит счет, можно, конечно воспользоваться процедурой ACC_PARENTS, объединив результаты запроса с таблицей ACC_ROOTS:

```
select distinct
  ar.acc_id, ar.root_kind, ar.class_id, ar.use_quantity,
  ar.color, ar.font_name, ar.font_size, ar.object_tree, ar.right_hand
from acc_parents(1083) ap, acc_roots ar
where ap.parent_id = ar.acc_id
```

Иногда проще часть этой информации получить, используя системную хранимую процедуру ACC_ROOT. Эта процедура имеет входной параметр ACC_ID типа INTEGER. При вызове с помощью этого параметра процедуре мы передаем ID счета, регистр которого нас интересует.

Результирующий набор, поставляемый хранимой процедурой **ACC_ROOT**

Название поля	**Тип данных**	**Назначение**
ROOT_ID	INTEGER	Внутренний ID счета
ROOT_KIND	SMALLINT	Тип регистра: 0 – левой стороны баланса 1 – правой стороны баланса 2 – балансового регистра 3 – забалансового регистра
ROOT_QUANTITY		Признак количественного учета: 0 – без количественного учета 1 – используется количественный учет
ROOT_CLASS_ID	INTEGER	Внутренний ID класса справочника, привязанного к регистру
OBJECT_TREE	SMALLINT	Отображать дерево объектов (для небольших справочников) 0 – не отображать дерево 1 – отображать дерево
RIGHT_HAND	SMALLINT	Способ отображения остатков на счетах регистра: 0 – левосторонний регистр 1 – правосторонний регистр

Пример использования хранимой процедуры ACC_ROOT:

```
select * from acc_root(1083);
```

Этот SQL-запрос вернет нам одной строкой все важные сведения о регистре, в который входит счет 1083.

Глава 7. Справочники

Классы, атрибуты, наследование

Каждый регистр счетов привязан к одному справочнику объектов, например, «Товары». Допустим, что мы автоматизируем работу компании, торгующей металлическими изделиями (винтами и т.п.) и обувью. Очевидно, что разные товары могут иметь разные атрибуты. Атрибутами винтов могут быть «Металл», «Диаметр» и «Длина». Атрибутами обуви - «Материал», «Изделие», «Размер», «Цвет» и «Вид». Кроме винтов, возможно, нам понадобится в справочниках хранить разные другие металлические изделия, характеризующиеся «Металлом» и «Наименованием». К тому же неплохо бы иметь в качестве общего атрибута «Внутренний номер объекта» (ID), чтобы использовать этот атрибут в других таблицах для ссылки на объекты справочной системы. С учетом сказанного, нам необходимы минимум три таблицы для хранения «Товаров»:

«Обувь»

ID	Материал	Изделие	Размер	Цвет	Вид

«Винты»

ID	Металл	Диаметр	Длина

«Разный крепеж»

ID	Металл	Наименование

Как легко заметить, некоторые таблицы имеют общие по смыслу поля. Например, все три таблицы имеют поле ID. Винты и разный крепеж имеют общее поле «Металл». Напрашивается мысль о том, чтобы хранить однородную информацию в одном месте, а не в разных таблицах. И действительно, если мы введем понятия «Товар вообще» и «Крепеж вообще», то задача намного упростится. Атрибут ID характеризует любой «Товар вообще», а атрибут «Металл» характеризует любой «Крепеж вообще». Можно даже сказать, что ***классы*** «Винты» и «Разный крепеж» наследуют атрибут «Металл» от ***класса*** «Крепеж вообще». Что же у нас получилось? Фактически это справочная система с некоторыми объектно-ориентированными свойствами. Все объекты у нас разделены на ***классы***. Собственно, каждый конкретный справочник и есть класс. Классы образуют дерево с наследованием атрибутов старших классов младшими классами. Каждый класс, таким образом, имеет **собственные** атрибуты и атрибуты, **унаследованные** от класса-предка. Каждому классу соответствует одна физическая таблица в базе данных, в этой таблице класс хранит только **собственные** атрибуты. Для полной сборки любого справочника наша программа автоматически производит объединение всех таблиц классов-предков с таблицей класса справочника по полю «Номер объекта» (ID).

Например, если мы хотим создать классы «Товары», «Крепеж», «Винты», «Другой крепеж», «Обувь», то выглядит это примерно так:

«Товары»

ID
118
119
120
300
487
511

«Крепеж»

ID	Металл
118	Латунь
119	Железо
120	Железо
300	Сталь
487	Кожа
511	Кожа

«Винты»

ID	Диаметр	Длина
118	М3	20
119	М3	35
120	М4	40

«Разный крепеж»

ID	Наименование
300	Скоба «Такая-то»

«Обувь»

ID	Материал	Изделие	Размер	Цвет	Вид
487	Кожа	Сапоги	37	Черный	Женские
511	Кожа	Туфли	37,5	Коричневый	Женские

Таким образом, для строгого хранения данных, они разносятся по нескольким таблицам. И каждый объект одновременно хранится в разных таблицах. Часть атрибутов справочника «Винты» хранится в таблице класса «Крепеж». Поэтому для того, чтобы отобразить ***все*** атрибуты «винтов», как собственные, так и унаследованные от классов-предков, необходима «сборка данных» (по одной строке из разных таблиц). Например, для сборки справочника «Винты» будут объединены таблицы: «Товары», «Крепеж» и «Винты» по полю ID. А для сборки справочника «Разный крепеж» будут объединены таблицы «Товары», «Крепеж», «Разный крепеж».

К счастью, язык структурированных запросов SQL позволяет делать такие объединения таблиц достаточно быстро. Наша программа устроена так, что сборка справочников происходит автоматически путем генерации текстов SQL-запросов. Ничего программировать не нужно.

Атрибутом класса может быть число, строка, дата, мемо-поле, картинка или ссылка на объект другого класса. В приведенном нами примере, например, поля Материал, Изделие, Цвет и Вид в действительности следует организовать в виде ссылок на отдельные мелкие справочники:

«Материалы для обуви»

ID	Наименование
25	Кожа
27	Пластик
28	Резина

«Обувные изделия»

ID	Наименование
38	Сапоги
39	Кроссовки
40	Туфли

«Цвета»

ID	Наименование
95	Черн.
96	Коричн.

«Виды обуви»

ID	Наименование
70	Муж.
71	Жен.

Итак, в реальности в таблице класса «Обувь» в основном хранятся ссылки:

«Обувь»

ID	Материал	Изделие	Размер	Цвет	Вид
487	25	38	37	95	71
511	25	40	37,5	96	71

Разрешение ссылок (превращение ID в краткие наименования) программа осуществляет автоматически, присоединяя при сборке справочника еще и специальную таблицу наименований, так что пользователь видит нормальную таблицу, содержащую в колонках соответствующие названия, а не внутренние номера, которые реально хранятся в базе. Все объекты справочной системы имеют уникальные внутренние ID, которые поставляет генератор **OBJECT_ID_GEN**.

Такая организация справочной системы позволяет одновременно решить ряд проблем:

- Регистру счетов достаточно присвоить базовый класс какого-то дерева классов справочной системы и тот сможет работать одновременно со всеми объектами классов-потомков, какой бы атрибутикой те не обладали. А это означает для программиста, конфигурирующего систему под задачу пользователя, возможность быстрого и корректного анализа бухгалтерских данных *по любому атрибуту* каждого отдельно взятого класса.

- Декларативная *ссылочная целостность* (на основе ключей foreign keys) пронизывает всю систему ссылок и не позволяет, с одной стороны, случайно удалить ни один объект, используемый в других справочниках или документах, а с другой стороны предельно повышает скорость объединения таблиц при сборке

справочников. Важно и другое - сообщение о невозможности удаления объекта пользователь получает мгновенно.

- Ссылочная природа справочников позволяет автоматически подчинять два справочника друг другу, для чего окно справочника делится пополам. В этом режиме в нижней части окна отображаются закладки всех справочников, которые могут быть подчинены справочнику, отображаемому в верхней части. Остается лишь выбрать нужную закладку. Например, перебирая «обувные изделия», можно тут же видеть всю обувь, отфильтрованную по данному признаку. Или, выбрав «Фирму», автоматически видеть все ее реквизиты. И все это без потери скорости, благодаря индексам.

- Всегда можно добавить новый атрибут в класс и его автоматически приобретут все объекты классов-потомков.

- Способ хранения данных соответствует самым строгим требованиям теории реляционной баз данных (степень нормализации 3…5), что значительно упрощает использование данных в SQL-запросах и повышает непротиворечивость хранимой информации и устойчивость данных к изменению структуры **справочников**.

Форматирование наименований в справочниках, системная таблица OBJECT_NAMES

Способ хранения объектов в справочной системе позволяет легко вносить и корректировать объекты, однако создает определенные трудности при их отображении из-за «разношерстности» атрибутики разных классов. Особенно это могло бы ощущаться в документах (типа накладной), где требуется одновременно еотображение объектов совершенно разных классов, например «Винтов» и «Обуви» в одном списке.

Для решения этой проблемы программа поддерживает так называемую ***таблицу наименований объектов***, в которой централизованно хранятся наименования всех объектов, присутствующих в справочной системе:

Системная таблица **OBJECT_NAMES**

ID INTEGER	**CLASS_ID INTEGER**	**SHORT_NAME VARCHAR(64)**	**FULL_NAME VARCHAR(256)**
25	1000	Кожа	Кожа
27	1000	Пластик	Пластик
28	1000	Резина	Резина
38	1001	Сапоги	Сапоги
39	1001	Кроссовки	Кроссовки
….	….	….	….
118	1005	М3 х 20 (латунь)	Винты М3 х 20 (латунь)
300	1006	Скоба “такая-то”	Скоба “такая-то” (сталь)
487	1001	Сапоги муж. 43 (черн.)	Сапоги муж. 43 (черн., кожа)

Поддержка наименований происходит ***автоматически*** при помощи взаимосвязанной системы автоматически создаваемых хранимых процедур, срабатывающих при добавлении, удалении или изменении любого объекта. Способ форматирования наименований для каждого справочника пользователь задает в простом интерфейсе. При этом он может указать, какие поля справочника следует использовать для сборки каждого наименования и снабдить их префиксами и суффиксами. Сокращенное наименование (short_name) существует по умолчанию и ограничено длиной в 64 символа. В конфигурацию можно добавлять новые виды наименований, например, в данном случае добавлено полное наименование (full_name) длиной 256 символов. Способы форматирования наименований хранятся в таблице **CLASS_NAME_DEF**.

Пример способа форматирования:

Форматирование кратких наименований, справочник «Винты»

Позиция	**Префикс**	**Поле**	**Суффикс**
1	Винты	Диаметр	X
2		Размер	
3	(	Металл	)

В любой момент можно изменить способ форматирования наименований объектов. Программа сама внесет необходимые изменения в тексты хранимых процедур и переформатирует все объекты в соответствии с новыми правилами. Хранимая процедура, ответственная за форматирование наименований справочника создается вместе со справочником и имеет название <имя_таблицы_справочника>_SET_NAMES.

Пример текста такой процедуры:

```
CREATE PROCEDURE CUSTOMER_SET_NAMES(OBJECT_ID INTEGER)
AS
DECLARE VARIABLE ID INTEGER;
DECLARE VARIABLE ID2 INTEGER;
DECLARE VARIABLE NAME2 VARCHAR(4096);
DECLARE VARIABLE SHORT_NAME VARCHAR(4096);
DECLARE VARIABLE V1 VARCHAR(35);
DECLARE VARIABLE V2 VARCHAR(50);
BEGIN
/*Текст данной процедуры создан системой Allegro автоматически.*/
/*Не меняйте этот текст. Ваши изменения могут быть утеряны*/
  FOR SELECT
    CUSTOMER.ID,
    CUSTOMER.NAME1,
    CUSTOMER.NAME
  FROM
    OBJECT_NAMES O, CUSTOMER
  WHERE
    (O.OBJECT_ID = CUSTOMER.ID) AND (O.CLASS_ID = 2) AND
    ((:OBJECT_ID IS NULL) OR (CUSTOMER.ID = :OBJECT_ID))
  INTO :ID,:V1,:V2
  DO
  IF (ID <> 0) THEN
  BEGIN
    IF (V1 IS NULL) THEN V1 = '';
    IF (V2 IS NULL) THEN V2 = '';
    NAME2 = :V1;
    SHORT_NAME = :V2;
    IF (strlen(NAME2)>64) THEN NAME2 = substr(NAME2, 1, 64);
    IF (strlen(SHORT_NAME)>64) THEN SHORT_NAME = substr(SHORT_NAME, 1, 64);
    UPDATE OBJECT_NAMES
    SET
       NAME2 = :NAME2,
       SHORT_NAME = :SHORT_NAME
    WHERE OBJECT_ID = :ID;
  END
END
```

Аналогичное форматирование имеют *наименования документов* (см. далее), с той лишь разницей, что наименования объектов справочной системы **хранятся** в системной таблице **OBJECT_NAMES**, а наименования документов хранятся в системной таблице **DOC_JOURNAL**.

Иерархические фильтры. Системная таблица RUBRIC

Иерархические фильтры (рубрикаторы) позволяют имитировать «иерархические справочники» при том, что на самом деле данные хранятся в «плоских справочниках». Главное назначение рубрикаторов – упростить навигацию по имеющимся данным и ускорить ввод новых данных за счет того, что рубрикаторы не только фильтруют справочник по значениям в каких-то полях, но и подставляют соответствующие начальные значения в эти поля при добавлении новой записи. Рубрикаторы создаются вручную и состоят из папок, каждая из которых может добавить одно условие фильтрации вида **Атрибут=Значение** или не добавить никакой фильтрации вообще. Каждая папка имеет свой класс. Сочетая папки-классы с дочерними папками-фильтрами по каким-то полям, можно построить очень мощную систему каталогизации справочников, в которых имеется иерархия классов и наследование свойств. При этом все данные хранятся в строго нормализованных таблицах, а система фильтров в любой момент может быть дополнена или перестроена.

Интерфейс рубрикаторов в окне диалога справочника автоматически формирует SQL-запросы, объединяя условия фильтрации разных уровней дерева фильтров. При этом чем ниже мы находимся в дереве, тем все больше условий фильтрации может быть наложено.

Папки могут быть трех типов:

- Класс
- Фильтр по атрибуту
- Папка «Прочее»

Папка типа «Класс» ограничивает все справочные элементы классом, который ей присвоен. Объекты дочерних классов также входят в это множество. Папка типа «фильтр по атрибуту» добавляет одно условие фильтрации. Все объекты в этой папке будут отфильтрованы по этому атрибуту. Папка «Прочее» вставляет в SQL-запрос вложенный подзапрос вида NOT IN (SELECT ….), который отсеивает все объекты, подпавшие под «фильтры по атрибуту» и оставляет те, которые не прошли фильтрацию по атрибуту на этом уровне дерева. Проще говоря в папку «Прочее» на этом уровне попадает «все остальное». Если пользователь добавит на этом уровне дерева рубрик еще один фильтр по тому же атрибуту, в котором укажет значение из «Прочего», то соответствующие объекты «исчезнут» из папки «Прочее» и окажутся в папке, которую он создал.

Таким образом папка «Прочее» позволяет ограничить дерево созданием самых необходимых для работы и часто применяющихся каталогов, помещая в «Прочее» все остальное.

Фильтры можно создавать массово и массово удалять, что позволяет очень быстро построить требуемую каталогизацию. Преимущество фильтров перед обычными деревьями, которые часто применяют в «иерархических справочниках» в том, что:

- Не нужно иметь строгих административных циркуляров о том, что в какую папку следует помещать, так как все и так автоматически окажется в нужных папках.
- Не нужно мучительно решать, какие атрибуты объектов поместить «на верхний уровень» дерева, а какие «на нижних уровнях», так как в одном и том же дереве возможно построить альтернативные способы каталогизации и один и тот же объект окажется «во всех нужных местах».
- Возможно создание фильтров по атрибутам, которые не исключают одновременного попадания объекта в несколько папок. Например, в справочнике «Контрагенты» можно создать два логических поля: «Поставщик» и «Покупатель», а в дереве рубрик – две соответствующие рубрики, фильтрующие каждая по своему полю. Покупатели автоматически окажутся в одной папке, а поставщики – в другой. Если же кто-то из контрагентов является одновременно и поставщиком и покупателем, то он окажется в обеих папках одновременно.
- Так как в дереве фильтров не хранится никакой информации о самих объектах (даже неявно), можно не создавать папки, в которых мало элементов. Часто в обычных «иерархических справочниках» это приходится делать ради поддержания единообразия способа хранения данных.
- В папках фильтров по атрибутам типа «ссылка на другой справочник» названия могут не храниться, а браться «на лету» из кратких отформатированных наименований объектов, хранящихся в этом справочнике. Если в справочнике исправили грамматическую ошибку, она автоматически исправится и во всех рубрикаторах тоже.
- Если какое-то значение из справочника, на который ссылался атрибут, больше не используется, при удалении из справочника одновременно автоматически удалятся и все соответствующие этому значению папки фильтров.
- Все данные справочников хранятся в строгом виде, подходящем для SQL-запросов и гарантирующем строгость и быстроту любых финансовых отчетов с группировкой по любым атрибутам объектов.

Ниже приведен пример SQL-запроса, формируемого рубрикатором товаров, в котором пользователь зашел в папку марки товара «BOSCH», а затем выбрал в ней папку «Кофеварка». Жирным шрифтом отмечена та часть, которую иерархический фильтр вставил в SQL-запрос:

```
SELECT
  GOODS.ID GOODS_ID,
  GOODS.GOODS_KIND,
  O1.SHORT_NAME GOODS_KIND_LK,
  GOODS.GOODS_MARK,
  O2.SHORT_NAME GOODS_MARK_LK,
  GOODS.ARTICLE,
  GOODS.PRICE_W,
  GOODS.PRICE_R
FROM
  GOODS,
  OBJECT_NAMES O1,
  OBJECT_NAMES O2
WHERE
  GOODS.GOODS_KIND=O1.OBJECT_ID AND
  GOODS.GOODS_MARK=O2.OBJECT_ID AND
  GOODS_KIND=8 AND              /*кофеварка*/
  GOODS_MARK=17                 /*BOSCH*/
ORDER BY GOODS.ID
```

А вот пример запроса, который формируется, если пользователь выбрал папку «Прочее» среди видов товара, продолжая находиться в папке «BOSCH»:

```
SELECT
  GOODS.ID GOODS_ID,
  GOODS.GOODS_KIND,
  O1.SHORT_NAME GOODS_KIND_LK,
  GOODS.GOODS_MARK,
  O2.SHORT_NAME GOODS_MARK_LK,
  GOODS.ARTICLE,
  GOODS.PRICE_W,
  GOODS.PRICE_R
FROM
  GOODS,
  OBJECT_NAMES O1,
  OBJECT_NAMES O2
WHERE
  GOODS.GOODS_KIND=O1.OBJECT_ID AND
  GOODS.GOODS_MARK=O2.OBJECT_ID AND
  (GOODS_KIND NOT IN (SELECT OBJECT_ID FROM RUBRIC
       WHERE PARENT_ID = 45 AND
       FILTER_FIELD_NAME = 'GOODS_KIND' AND
       FILTER_KIND = 1)) AND      /*Прочее*/
  GOODS_MARK=17                  /*BOSCH*/
ORDER BY GOODS.ID
```

Интерфейс дерева рубрикаторов устроен таким образом, что если на каком-то уровне ветви дерева фильтрация по данному атрибуту уже имеется, то этот атрибут исключается из списка атрибутов, по которым пользователь может создать фильтрацию на всех нижних уровнях, если только они не дочерние к папке «Прочее». Это исключает вероятность случайного создания противоречивых условий фильтрации на разных уровнях рубрикатора. Таким образом каждый фильтр по атрибуту сокращает возможный список атрибутов для фильтрации, а каждый дочерний класс, как правило, добавляет новые атрибуты в список, так как дочерние классы справочников как правило обладают дополнительной атрибутикой по отношению к классам-предкам.

Дерево рубрикаторов придает системе классов справочников законченный и стройный вид. Само дерево рубрикаторов появилось в системе Allegro, начиная с версии 1.8 и заменило собой дерево классов справочников в левой стороне окна «Все справочники». Поэтому теперь после создания нового справочника конфигурирующий сам решает, включать его, как рубрику в дерево рубрикаторов или не включать. Особенно это актуально для множества мелких справочников первого уровня, которые раньше загромождали окно «Все справочники». Теперь их можно просто не включать в дерево рубрик.

Информация о дереве иерархических фильтров справочников хранится в таблице **RUBRIC**.

Системная таблица **RUBRIC**

Название поля	**Тип данных**	**Назначение**
ID	INTEGER	Внутренний ID рубрики
PARENT_ID	INTEGER	Внутренний ID родительской рубрики
HAS_CHILDREN	SMALLINT	Если рубрика имеет дочерние рубрики, то в этом поле записывается 1, если нет, то записывается 0
CLASS_ID FILTER_KIND	INTEGER SMALLINT	Внутренний ID класса Тип рубрики: 0 – класс, 1 - фильтр по атрибуту, 2 - папка "Прочее"
FILTER_FIELD_NAME	VARCHAR(31)	Имя поля, по которому производится фильтрация в справочнике
OBJECT_ID	INTEGER	Внутренний ID объекта ссылки, если установлен фильтр по полю типа TREFERENCE
INTEGER_VALUE	INTEGER	Значение фильтра, если оно - целое число
IS_AUTONAME	SMALLINT	Если рубрика использует автоматическое наименование, то в этом поле записывается 1, а если ручное, то записывается 0
NAME	VARCHAR(64)	Ручное наименование рубрики

Каждая рубрика имеет свой ID, который поставляется генератором **RUBRIC_ID_GEN**. Рубрики образуют бесконечные деревья. Все базовые рубрики (корневые) имеют PARENT_ID = 0.

Системные таблицы CLASS, CLASS_FIELDS, CLASS_NAME_DEF, CLASS_SETTINGS

Информация о дереве классов справочников хранится в таблице **CLASS**.

Системная таблица **CLASS**

Название поля	**Тип данных**	**Назначение**
CLASS_ID	INTEGER	Внутренний ID класса
PARENT_ID	INTEGER	Внутренний ID родительского класса
HAS_CHILDREN	SMALLINT	Если класс имеет дочерние классы, то в этом поле записывается 1, если нет, то записывается 0
NAME TABLE_NAME	VARCHAR(80) VARCHAR(31)	Название класса (уникальное) Название таблицы, в которой хранятся собственные атрибуты класса (уникальное)
IS_ABSTRACT	SMALLINT	Признак абстрактного класса. 0 – можно добавлять объекты в справочник данного класса 1 – нельзя добавлять объекты в справочник данного класса

Каждый класс имеет свой CLASS_ID, который поставляется генератором **CLASS_ID_GEN**. Классы образуют деревья. Все базовые классы (корневые) имеют PARENT_ID = 0. Имеется специальный «нулевой» класс с CLASS_ID = 0, служащий для сохранения ссылочной целостности. Информация об атрибутах классов хранится в таблице **CLASS_FIELDS**.

Системная таблица **CLASS_FIELDS**

Название поля	**Тип данных**	**Назначение**
CLASS_ID	INTEGER	Внутренний ID класса
FIELD_NAME	VARCHAR(31)	Имя поля в таблице (уникальное в сочетании с ID)
DISPLAY_NAME	VARCHAR(50)	Название атрибута (уникальное в пределах класса и всех его предков)
DISPLAY_FORMAT REF_TABLE_NAME	VARCHAR(50) VARCHAR(31)	Формат отображения чисел, дат или сборных наименований для полей ссылок, в зависимости от типа поля Только для полей ссылок: название таблицы, на которую ссылается данное поле
IS_REQUIRED	SMALLINT	Требовать значение при вводе на стороне приложения 0 – не требуется указывать значение при вводе данных 1 – требуется указывать значение при вводе данных
NO_CLONE	SMALLINT	0 – при клонировании значение данного поля копируется 1 – при клонировании значение данного поля не копируется
COLUMN_COLOR	VARCHAR(30)	Цвет колонки при отображении справочника на экране
COLUMN_ALIGNMENT	INTEGER	Выравнивание значения поля при отображении справочника в колонках сеток: 0 – влево, 1 – вправо, 2 – по центру

Сведения о способах форматирования наименований хранятся в таблице **CLASS_NAME_DEF.**

Для каждого справочника хранится способ сборки каждого типа наименований в виде *элементов* ф*орматирования,* состоящих из префикса, суффикса и используемого атрибута. Каждый элемент форматирования занимает одну строку в таблице **CLASS_NAME_DEF**.

Системная таблица **CLASS_NAME_DEF**

Название поля	**Тип данных**	**Назначение**
FOR_CLASS_ID	INTEGER	Внутренний ID класса, для которого определяется способ форматирования
OBJECT_NAME_TYPE	VARCHAR(31)	«Тип наименования» (название поля в таблице OBJECT_NAMES), для которого определен данный элемент форматирования
POS	INTEGER	Позиция элемента форматирования в строке наименования

CLASS_ID	INTEGER	Внутренний ID класса, собственный атрибут которого используется в элементе форматирования
FIELD_NAME	VARCHAR(31)	Имя поля, которое используется в элементе форматирования
PREFIX	VARCHAR(35)	Префикс
SUFFIX	VARCHAR(35)	Суффикс
DISPLAY_FORMAT	VARCHAR(50)	Формат отображения чисел, дат или сборных наименований для полей ссылок, в зависимости от типа поля

Настройки размеров окон справочной системы, размеры колонок в сетках и другие подобные сведения хранятся в таблице **CLASS_SETTINGS** отдельно для каждого справочника и режима его отображения.

Системная таблица **CLASS_SETTINGS**

Название поля	**Тип данных**	**Назначение**
CLASS_ID	INTEGER	Внутренний ID класса
SETTING_NAME	VARCHAR(31)	Имя настройки
SETTING_VALUE	BLOB	Значение настройки

Работа с таблицами справочников, хранимые процедуры CLASS_TREE, CLASS_ROOT

Предположим, у нас имеются классы справочников (таблицы):

- **Товары (GOODS)**
  - **Крепеж (METAL_ITEMS)**
    - **Винты (SCREWS)**
    - **Разный крепеж (OTHER_METAL_ITEMS)**
  - **Обувь (SHOES)**

Мы можем легко запросить все краткие наименования объектов класса «Обувь», объединив таблицу наименований объектов OBJECT_NAMES с таблицей SHOES по полю ID:

```
select o1.object_id, o1.short_name
from object_names o1, shoes s
where o1.object_id = s.id;
```

Так как все объекты в таблице OBJECT_NAMES хранят свои ID в поле OBJECT_ID, то, объединяя OBJECT_NAMES с любой таблицей класса по полю ID, мы получим наименования любого класса объектов, какой только существует в системе. Например, мы можем получить весь «Крепеж», в который войдут и «Винты» и «Разный крепеж»:

```
select o1.object_id, o1.short_name
from object_names o1, metal_items m
where o1.object_id = m.id;
```

Или даже все «Товары» :

```
select o1.object_id, o1.short_name
from object_names o1, goods g
where o1.object_id = g.id;
```

Объединяя таблицу любого класса с накопителем финансовых данных, например, с таблицей проводок ACC_TURN, мы можем получить анализ финансовой информации ***по любому атрибуту*** конкретно выбранного класса. Например, если мы хотим проанализировать остатки обуви на главном складе по цветам, то мы просто объединяем таблицу проводок ACC_TURN с таблицей SHOES и, для того, чтобы еще увидеть наименования цветов - с таблицей OBJECT_NAMES:

```
select
  sum(a.debit) – sum(a.credit) amount,
  o1.short_name shoes_color
from
  shoes s,
  acc_turn a,
  object_names o1
where
  a.object_id = s.id and
  o1.object_id = s.color and a.acc_id = 123
group by
  o1.short_name;
```

В приведенном примере мы использовали запрос с группировкой. Предполагалось, что ACC_ID главного склада равняется 123.

Если же мы хотим посмотреть распределение товаров на главном складе ***по классам***, тогда лучше всего воспользоваться колонкой CLASS_ID таблицы OBJECT_NAMES и тем, что нам известно устройство таблицы CLASS:

```
select
  sum(a.debit) – sum(a.credit) amount,
  c.name class_name
from
  acc_turn a,
  object_names o1,
  class c
where
  a.object_id = o1.object_id and
  o1.class_id = c.class_id and
  a.acc_id = 123
group by
  c.name;
```

А теперь предположим, что нам нужен список всех классов. Как нам его получить? Это очень просто – достаточно запросить таблицу CLASS:

```
select class_id, name from class c
```

А если нам нужен список ***всех классов, потомков класса*** «Товары»? Простой запрос типа

```
select class_id, name from class c
where parent_id = 100
```

нам вернет только прямых потомков класса «товары», но не все дочерние классы. Конечно, можно написать хранимую процедуру обхода дерева классов, но проще воспользоваться готовой системной процедурой **CLASS_TREE**, которая возвращает все сочетания CLASS_ID по правилу старший-младший для каждого класса и всех его потомков.

Результирующий набор, поставляемый хранимой процедурой **CLASS_TREE**

Название поля	**Тип данных**	**Назначение**
PARENT_ID	INTEGER	Внутренний ID класса-предка
CHILD_ID	INTEGER	Внутренний ID класса-потомка

Запрос вида:

```
select c.class_id, c.name
from class_tree ct, class c
where c.class_id = ct.child_id and  ct.parent_id = 100
```

вернет список всех классов, потомков класса «Товары», включая его самого.

Последняя системная хранимая процедура, имеющая отношение к справочной системе это процедура **CLASS_ROOT**. Эта процедура позволяет получить CLASS_ID корневого класса для любого выбранного класса, если нам известен его CLASS_ID. Она имеет входной параметр типа INTEGER, в котором при вызове передается CLASS_ID интересующего нас класса. Процедура возвращает один параметр: ROOT_ID, в котором передает нам CLASS_ID корневого класса. Например, для класса «Винты» корневым является класс «Товары». Использовать процедуру CLASS_ROOT несложно:

```
select c.class_id, c.name
from class_root(1005) cr, class c
where c.class_id = cr.root_id
```

Здесь предполагается, что класс «Винты» имеет CLASS_ID = 1005.

Глава 8. Типы документов

Главные и подчиненные таблицы

Документом в нашей программе называются любые данные, отражающие ***некоторый факт***, имевший место. Это может быть отдельный платеж или отгрузка товара, заключенный договор и т.д. Как правило, такие факты имеют дату или даже несколько дат (у договора может быть дата подписания и дата завершения). Например, такая информация, как прайс-лист тоже может считаться документом, если цены время от времени меняются или если для разных покупателей используются разные прайс-листы.

В отличие от содержимого справочников, не все документы переносятся в новую базу данных, если мы создаем ее «по остаткам предыдущего периода». Переносятся лишь те документы, которые нужны для работы в новом периоде.

Для хранения документов используются таблицы базы данных. Каждый ***тип документов*** хранится в своей отдельной таблице. Эта таблица-хранилище называется ***главной таблицей документа***. Есть документы, для хранения которых главной таблицы достаточно. Например, для хранения отдельного платежа нам требуется всего одна строка таблицы и ряд колонок: «плательщик», «получатель», «дата платежа», «сумма» и т.д.

Но существуют документы, требующие как минимум одной дополнительной таблицы-хранилища. Например, документ о продаже должен иметь одну строку в главной таблице («продавец», «покупатель», «дата» и т.д.) и множество строк в ***подчиненной таблице*** («товар», «количество», «цена», «сумма»).

Для некоторых документов может потребоваться более одной подчиненной таблицы, например, для хранения договора может потребоваться отдельно хранить «Спецификацию поставки», а отдельно «График поставок и платежей». Наша программа позволяет создавать несколько подчиненных таблиц первого уровня у одного документа.

Следует заметить, что под документом мы понимаем нечто большее, чем простой аналог бумажного документа. Скорее под документом подразумевается именно определенный факт. Например, при продаже товара может потребоваться распечатать счет-фактуру и накладную. Однако это вовсе не означает, что следует заводить два отдельных ***типа документов*** в конфигурации. Достаточно завести тип документов «Продажа товара», а распечатывать два разных документа на основании одной и той же информации. Разумеется, атрибуты документа типа «Продажа товара» (поля главной и подчиненной таблиц) должны быть подобраны таким образом, чтобы там умещалась вся необходимая для печати бумажных документов информация.

Все документы в программе имеют уникальный внутренний номер ID, который поставляется генератором DOC_ID_GEN. Поле ID является связующим между главной таблицей и подчиненными таблицами документа. Каждая подчиненная таблица имеет вдобавок уникальное поле N, значение для которого поставляется генератором с именем <имя_подчиненной_таблицы>_N_GEN. При создании подчиненной таблицы, этот генератор создается в базе данных автоматически. Уникальное поле N позволяет организовать ссылки одних документов на *позиции* других документов. Кроме того, главная таблица имеет поле DIR_ID, в котором хранится уникальный внутренний номер папки, в которой находится документ в «Проводнике документов». Каждый документ находится в какой-то конкретной папке. Если вызвать окно «Проводника документов», то в нем можно создавать новые папки и перемещать документы между папками. Это сделано для удобства навигации по документам и для поддержания определенного порядка в делах. В программе имеется также окно «Поиска документов», в котором можно искать документы по некоторым общим признакам. Подробнее навигация по документам описана в соответствующей главе.

Предположим, мы хотим создать тип документа «Продажа товара». Для этого нам нужно придумать название для главной таблицы (пусть это будет SALE) и для подчиненной (пусть это будет SALE_ITEM). После создания этих двух объектов метаданных, у нас возникнут две таблицы:

SALE

ID INTEGER	DIR_ID INTEGER

SALE_ITEM

ID INTEGER	N INTEGER

Теперь мы можем добавить атрибуты «Продавец», «Покупатель», «Дата», «№ накладной», «№счета-фактуры» в главную таблицу:

SALE

ID INTEGER	DIR_ID INTEGER	VENDOR INTEGER	CUSTOMER INTEGER	SALE_DATE DATE	NAKL_NO INTEGER	FACTURE_NO INTEGER
1120	10	1012	2033	01.05.2003	144	144

А в подчиненную таблицу добавить «Товар», «Количество», «Цена», «Сумма»:

SALE_ITEM

ID INTEGER	N INTEGER	GOODS INTEGER	QUANTITY INTEGER	PRICE DECIMAL(10,2)	ITEM_AMOUNT DECIMAL(18,2)
1120	1	3021	1	120,00	120,00
1120	2	3052	12	100,00	1200,00
1120	3	1830	10	320,50	3205,00

Разумеется, нам нужно самим придумать названия для всех атрибутов и полей таблиц. Мы рекомендуем использовать для названий полей английские атрибутов. Поля VENDOR и CUSTOMER имеют тип ссылки на справочник «Контрагенты», а поле GOODS подчиненной таблицы имеет тип ссылки на справочник «Товары». В приведенном примере показаны таблицы, в которых уже хранится один документ с ID = 1120. В главной таблице хранится одна запись с общими данными по документу, а в подчиненной таблице три записи о проданных товарах.

Форматирование наименований документов

Для отображения документов разных типов в общих списках, например, в «Проводнике по документам» используются форматированные наименования, которые хранятся в таблице **DOC_JOURNAL**. Форматированные наименования для каждого типа документов создаются при помощи хранимой процедуры с именем <имя_главной_таблицы>_GET_NAMES. Способ форматирования наименований для каждого типа документов разработчик конфигурации задает в простом интерфейсе. При этом он может указать, какие поля документа следует использовать для сборки наименования и снабдить их префиксами и суффиксами. Наименование документа ограничено длиной 128 символов. Способы форматирования наименований хранятся в таблице DOC_NAME_DEF.

Пример способа форматирования:

Форматирование наименований, тип документа «Продажа товара»

Позиция	Префикс	Поле	Суффикс
1		Покупатель	,
2	Продажи,	№ счета-фактуры	
3	(	Металл	)

В любой момент можно изменить способ форматирования наименований документов. Программа сама внесет необходимые изменения в текст хранимой процедуры в соответствии с новыми правилами. Хранимая процедура, ответственная за форматирование наименований документа создается вместе с главной таблицей документа автоматически и имеет название <имя_главной_таблицы>_GET_NAMES.

Пример процедуры форматирования:

```
CREATE PROCEDURE SALE_GET_NAMES(ID INTEGER, DIR_ID INTEGER)
RETURNS(DOC_ID INTEGER, DOC_NAME VARCHAR(128) CHARACTER SET WIN1251)
AS
DECLARE VARIABLE NAME VARCHAR(94);
DECLARE VARIABLE V1 VARCHAR(64);
DECLARE VARIABLE V2 VARCHAR(12);
BEGIN
/*Текст данной процедуры создан системой Allegro автоматически.*/
/*Не меняйте этот текст. Ваши изменения могут быть утеряны*/
  FOR SELECT
    SALE.ID,
    O1.SHORT_NAME,
    SALE.FACTURE_NO
  FROM
    SALE,
    OBJECT_NAMES O1
  WHERE
    ((:ID IS NULL) OR (SALE.ID = :ID)) AND
    ((:DIR_ID IS NULL) OR (SALE.DIR_ID = :DIR_ID)) AND
    O1.OBJECT_ID = SALE.CUSTOMER
  INTO :DOC_ID,:V1,:V2
  DO
  BEGIN
    IF (V1 IS NULL) THEN V1 = '';
    IF (V2 IS NULL) THEN V2 = '';
    NAME = :V1||', '||'Продажи, СФ № '||:V2;
    DOC_NAME = NAME;
    SUSPEND;
  END
END
```

Возвращаемый процедурой набор зависит от входных параметров, переданных в процедуру при вызове:

Входные параметры	Возвращаемый набор

ID	DIR_ID	DOC_NAME
NULL	NULL	Все наименования документов этого типа
NULL	13	Все наименования документов в папке DIR_ID=13
11820	NULL	Наименование одного документа с ID=11820

Например, мы можем запросить наименования всех документов «Продажи товара», которые находятся в папке DIR_ID = 13:

```
select * from sale_get_names(NULL, 13);
```

Системные таблицы DOC_TYPE, DOC_TABLES, DOC_FIELDS, DOC_NAME_DEF, DOC_DIR

Информация о типах документов хранится в системной таблице **DOC_TYPE**.

Системная таблица **DOC_TYPE**

Название поля	Тип данных	Назначение
DOC_TYPE_ID	INTEGER	Внутренний ID типа документов
NAME	VARCHAR(50)	Название типа документов
LARGE_IMAGE	BLOB	Пиктограмма размером 32x32 для «Проводника документов»
SMALL_IMAGE	BLOB	Пиктограмма размером 16x16 для «Проводника документов»
COMMENTS_FIELD_NAME	VARCHAR(31)	Название поля примечаний к документу (если есть)
DATE_FIELD_NAME	VARCHAR(31)	Название поля даты документа (если есть)
AMOUNT_FIELD_NAME	VARCHAR(31)	Название поля суммы документа (если есть)
DOC_ENUM_FIELD_NAME	VARCHAR(31)	Название поля номера документа (если есть)
PROJECT_NAME	VARCHAR(255)	Имя файла проекта оконного интерфейса, работающего с данным типом документов
RUN_MODE	INTEGER	Способ запуска проекта оконного интерфейса 0 – Стандартный 1 – Модальное окно 2 – Множество экземпляров
RUN_FROM_EXPLORER	INTEGER	Запуск оконного интерфейса из контекстного меню «Создать документ» в «Проводнике по документам». 0 – документ невозможно создать из «Проводника по документам» 1 – документ можно создать из «Проводника по документам»
COPY_ALWAYS	SMALLINT	Флаг «Всегда переносить в новый период»
DOC_TIMEOUT	INTEGER	Точность захвата документов относительно «даты отреза периода» в сутках при переносе документов этого типа в новый период
COPY_FLAG_FIELD_NAME	VARCHAR(31)	Название поля, хранящего флаг управления переносом документов в новый период (если есть)

Информация о таблицах-хранилищах документов хранится в системной таблице **DOC_TABLES.**

Системная таблица **DOC_TABLES**

Название поля	Тип данных	Назначение
TABLE_NAME	VARCHAR(31)	Название таблицы в базе данных
DOC_TYPE_ID	INTEGER	Внутренний ID типа документов
NAME	VARCHAR(50)	Название таблицы для пользователя
IS_DETAIL	SMALLINT	Признак подчиненной таблицы 0 – Главная таблица 1 – Подчиненная таблица

Если документ состоит из двух таблиц (главной и подчиненной), то одной записи в системной таблице DOC_TYPE соответствует две записи в системной таблице DOC_TABLES. Уникальные внутренние номера типов документов поставляются системным генератором **DOC_TYPE_ID_GEN**.

Информация об атрибутах документов хранится в системной таблице **DOC_FIELDS.**

Системная таблица **DOC_FIELDS**

Название поля	**Тип данных**	**Назначение**
TABLE_NAME	VARCHAR(31)	Название таблицы в базе данных
FIELD_NAME	VARCHAR(31)	Название поля в таблице
DOC_TYPE_ID	INTEGER	Внутренний ID типа документов
DISPLAY_NAME	VARCHAR(50)	Название атрибута
REF_TABLE_NAME	VARCHAR(31)	Только для полей ссылок: название таблицы, на которую ссылается данное поле
DISPLAY_FORMAT	VARCHAR(20)	Формат отображения чисел, дат или сборных наименований для полей ссылок, в зависимости от типа поля

Сведения о способах форматирования наименований документов хранятся в таблице **DOC_NAME_DEF.** Наименования документов используются при отображении их в «Проводнике по документам». Например, мы можем сделать так, чтобы все документы о продажах высвечивались в виде «Накладная №…». Часто удобно так настроить форматирование наименований документов, чтобы название покупателя содержалось в составе названия документа, например «Альфа, Продажи, СФ №…»

Для каждого документа хранится способ сборки наименований в виде *элементов* ф*орматирования,* состоящих из префикса, суффикса и используемого атрибута. Каждый элемент форматирования занимает одну строку в таблице **DOC_NAME_DEF**. При форматировании имен документов можно использовать только атрибуты главной таблицы.

Системная таблица **DOC_NAME_DEF**

Название поля	**Тип данных**	**Назначение**
TABLE_NAME	VARCHAR(31)	Название главной таблицы документа
POS	INTEGER	Позиция элемента форматирования в строке наименования
FIELD_NAME	VARCHAR(31)	Имя поля, которое используется в элементе форматирования
PREFIX	VARCHAR(35)	Префикс
SUFFIX	VARCHAR(35)	Суффикс
DISPLAY_FORMAT	VARCHAR(50)	Формат отображения чисел, дат или сборных наименований для полей ссылок, в зависимости от типа поля

Папки «Проводника по документам» хранятся в системной таблице **DOC_DIR.**

Системная таблица **DOC_DIR**

Название поля	**Тип данных**	**Назначение**
ID	INTEGER	Внутренний ID папки. Корневая папка «Все документы» имеет ID = 1
PARENT_ID	INTEGER	Внутренний ID родительской папки. Корневая папка «Все документы» имеет PARENT_ID = 0
NAME	VARCHAR(64)	Название папки
HAS_CHILDREN	SMALLINT	0 – Папка не имеет вложенных папок 1 – Папка имеет вложенные папки

Уникальные внутренние номера ID папок поставляются системным генератором **DOC_DIR_ID_GEN**.

Системная таблица DOC_JOURNAL

Все документы регистрируются в общем журнале - системной таблице **DOC_ JOURNAL**.
Системная таблица **DOC_JOURNAL**

Название поля	**Тип данных**	**Назначение**
ID	INTEGER	Внутренний ID документа
DOC_DIR_ID	INTEGER	Внутренний ID папки
DOC_NAME	VARCHAR(128)	Отформатированное наименование документа
DOC_TYPE_ID	INTEGER	Внутренний ID типа документов
DOC_NO	VARCHAR(80)	Номер документа
DOC_DATE	TIMESTAMP	Дата документа
DOC_AMOUNT	DECIMAL(18, 5)	Сумма документа

После создания нового типа документов, его главная таблица снабжается тремя триггерами. Например, текст триггера AFTER INSERT для документа с названием главной таблицы **SALE** выглядит так:

```
CREATE TRIGGER SALE_SN_INSERT FOR SALE
ACTIVE AFTER INSERT POSITION 200
AS
BEGIN
/*Текст этого триггера создан системой Allegro.*/
  EXECUTE PROCEDURE SALE_SET_NAMES(NEW.ID);
END
```

После вставки, изменения или удаления документа соответствующий триггер вызывает процедуру **<имя_главной_таблицы>__SET_NAMES**, передавая в нее ID документа в качестве параметра. Текст процедуры **<имя_главной_таблицы>__SET_NAMES** создается системой Allegro автоматически, и зависит от настроек в «Дополнительных свойствах» документа. Например, эта процедура может выглядеть так:

```
CREATE PROCEDURE SALE_SET_NAMES(DOC_ID INTEGER)
AS
BEGIN
/*Текст данной процедуры создан системой Allegro автоматически.*/
/*Не меняйте этот текст. Ваши изменения могут быть утеряны*/
  IF (DOC_ID IS NULL) THEN
    DELETE FROM DOC_JOURNAL WHERE DOC_TYPE_ID = 112;
  ELSE
    DELETE FROM DOC_JOURNAL WHERE ID = :DOC_ID;
  IF (DOC_ID IS NULL) THEN
  INSERT INTO DOC_JOURNAL(ID, DOC_DIR_ID, DOC_NAME, DOC_TYPE_ID, DOC_NO, DOC_DATE,
DOC_AMOUNT)
    SELECT
      D.ID, D.DIR_ID, G.DOC_NAME, 112,
      NULL, D.ADATE, D.AMOUNT
    FROM
      SALE_GET_NAMES(:DOC_ID,NULL) G, SALE D
    WHERE
      D.ID = G.DOC_ID;
  ELSE
  INSERT INTO DOC_JOURNAL(ID, DOC_DIR_ID, DOC_NAME, DOC_TYPE_ID, DOC_NO, DOC_DATE,
DOC_AMOUNT)
    SELECT
      D.ID, D.DIR_ID, G.DOC_NAME, 112,
      NULL, D.ADATE, D.AMOUNT
    FROM
      SALE_GET_NAMES(:DOC_ID,NULL) G, SALE D
    WHERE
      D.ID = :DOC_ID;
END
```

Глава 9. Таблицы настроек

Назначение таблиц настроек

Таблицы настроек являются частью конфигурации и создаются в окне «Метаданные». В отличие от справочников, поля в таблицах настроек могут иметь такие типы данных, как ссылка на бухгалтерский счет или слой. Это позволяет связывать между собой какие-то счета или связывать счета с какими-то элементами справочной системы. Например, у нас в балансе компании может быть каталог складских счетов, часть которых представляет собой склады в магазинах, принадлежащих компании. Одновременно имеется каталог счетов денежных средств (касс) тех же магазинов. При этом сами магазины иногда еще выступают в качестве юридических лиц со своими реквизитами в каких-то документах. Для того чтобы связать между собой складской счет, счет кассы и запись в справочнике контрагентов, может пригодиться таблица настроек. Разработчик в этом случае создает новую таблицу настроек в окне «Метаданные», добавляет в нее три поля: склад, касса, контрагент и ряд записей, в которых устанавливает соответствующие связи. После этого он может использовать эту таблицу в SQL-запросах всякий раз, как ему понадобится использовать подобные связи. Содержимое таблиц настроек автоматически переносится в новую базу данных, если она создается «по итогам текущей» или «с переносом справочников и папок».

Таким образом, таблицы настроек являются чем-то вроде «внутренних справочников», недоступных пользователю в окне «Справочники». В отличие от справочников, таблицы настроек не поддерживают наследование. Это самые обыкновенные таблицы с той лишь разницей, что при создании новой базы данных данные в обыкновенных, созданных вручную, таблицах не переносятся в новую базу, а данные в таблицах настроек переносится, причем, автоматически.

Таблицы настроек можно использовать, например, в качестве «справочника стандартных проводок», если разработчик решит создать поддержку стандартных проводок в своей конфигурации. Также их можно использовать, если нужно иметь какой-то список элементов, не доступный для редактирование в окне «Справочники». Если необходимо обеспечить доступ пользователя к редактированию данных в таблице проводок, разработчик самостоятельно должен реализовать для этой цели оконный интерфейс, наподобие того, который он создает для документов и отчетов.

В таблицах настроек можно хранить всевозможные нормативные данные, например, налоговые ставки или в каких-то случаях даже прайс-листы.

Системные таблицы SETTING, SETTING_FIELDS

Список таблиц настроек хранится в системной таблице **SETTING**.

Системная таблица **SETTING**

Название поля	**Тип данных**	**Назначение**
SETTING_ID	INTEGER	Внутренний ID настройки
NAME	VARCHAR(50)	Название настройки
TABLE_NAME	VARCHAR(31)	Имя таблицы

Список полей таблиц настроек хранится в системной таблице **SETTING_FIELDS**

Системная таблица **SETTING_FIELDS**

Название поля	**Тип данных**	**Назначение**
SETTING_ID	INTEGER	Внутренний ID настройки
FIELD_NAME	VARCHAR(31)	Имя поля в таблице (уникальное в сочетании с ID)
DISPLAY_NAME	VARCHAR(50)	Название атрибута (уникальное в пределах настройки)
DISPLAY_FORMAT	VARCHAR(50)	Формат отображения чисел, дат или сборных наименований для полей ссылок, в зависимости от типа поля
REF_TABLE_NAME	VARCHAR(31)	Только для полей ссылок: название таблицы, на которую ссылается данное поле

Типы данных в Справочниках, Документах и Таблицах настроек.

Как уже было сказано, таблицы настрое допускают использование таких типов данных (ссылка на бухгалтерский счет или слой), которые не допускаются в справочниках. Документы допускают еще такие типы данных, как ссылка на документ или строку подчиненной таблицы документа.

Соответствие между разными объектами метаданных и разрешенными в них полями приведено в таблице:

Тип данных	**Описание**	**Справоч-ники**	**Доку-менты**	**Таблицы настроек**
DECIMAL	Число с фиксированной точкой (до 18 разрядов)	+	+	+
DOUBLE PRECISION	Число с плавающей точкой (15 значащих цифр)	+	+	+
FLOAT	Число с плавающей точкой (7 значащих цифр)	+	+	+
INTEGER	Целое	+	+	+
SMALLINT	Малое целое	+	+	+
TBOOLEAN	Логический тип (0 или 1)	+	+	+
VARCHAR	Строка переменной длины	+	+	+
CHAR	Строка фиксированной длины	+	+	+
DATE	Дата	+	+	+
TIME	Время	+	+	+
TIMESTAMP	Дата и время	+	+	+
TMEMO	Текст неопределенной длины	+	+	+
TPICTURE	Изображение	+	+	+
TREFERENCE	Справочник	+	+	+
TDOCUMENT	Документ		+	
TDOC_ITEM	Позиция документа		+	
TACCOUNT	Бухгалтерский счет		+	+
TLAYER	Слой		+	+
TANY_REFERENCE	Любой справочник		+	
TANY_DOCUMENT	Любой документ		+	

Как мы видим, наибольшими возможностями обладают документы. Справочники и таблицы настроек имеют ограничения на используемые типы данных. Наибольшие ограничения на допустимые типы данных действуют в отношении Справочников.

Глава 10. Бухгалтерские проводки

Шаблоны операций, системные таблицы TEMPLATE, TEMPLATE_DEF

Allegro позволяет осуществлять как ручные бухгалтерские проводки, так и автоматические. Автоматические проводки формируются на основе документов и ***шаблонов операций***. Каждый тип документа может иметь несколько шаблонов операций. Шаблоны операций можно легко настраивать. Изменив шаблон операции, мы всегда можем ***изменить способ проведения*** всех документов определенного типа. Это создает большую гибкость как при разработке конфигурации, так и при ее дальнейшем использовании. Каждый шаблон имеет название операции и название хранимой процедуры, которая создает набор проводок в формате журнала операций. Если мы перенастроим шаблон, программа автоматически изменит текст этой хранимой процедуры. Каждый шаблон содержит, кроме того, ***условие срабатывания***, что позволяет по-разному проводить одни и те же типы документов, в зависимости от каких-то признаков в самих документах. Кроме того, в шаблоне может быть указан ***диапазон дат***, для которых он действует. Это означает возможность провести документы до какой-то даты с помощью одного шаблона, а после этой даты – с помощью другого шаблона.

Каждый шаблон содержит имя атрибута (поля) документа, откуда берется ***дата операции***.

Каждый шаблон содержит набор «записей по счетам», в каждой из которых указывается:

- вид записи (в дебет или в кредит)
- бухгалтерский счет (счет указывается явно или в виде ссылки на поле документа, из которого следует получить внутренний номер ACC_ID счета)
- объект аналитического учета (указывается ссылка на поле документа, из которого следует брать внутренний номер ID объекта)
- сумма (указывается ссылка на поле документа, из которого берется сумма этой записи)
- слой (указывается явно или в виде ссылки на поле документа, из которого берется внутренний номер LAYER_ID слоя)
- количество (указывается ссылка на поле документа, из которого берется количество этой записи)

На самом деле даже ручные проводки осуществляются программой при помощи шаблонов. Ручные проводки хранятся в типе документов «Ручная операция» (шапки - в таблице **GAAP**, записи по счетам – в таблице **GAAP_POS**). Хранимая процедура шаблона ручных операций называется **GAAP_TEMPLATE**.

Ниже показаны структуры таблиц, в которых хранятся сведения о шаблонах операций, на основе которых программа формирует тексты соответствующих хранимых процедур.

Системная таблица **TEMPLATE**

Название поля	**Тип данных**	**Назначение**
TEMPLATE_ID	INTEGER	Внутренний номер ID шаблона
NAME	VARCHAR(50)	Название бухгалтерской операции
STOREDPROC_NAME	VARCHAR(31)	Название хранимой процедуры, возвращающей проводки в формате журнала операций
SWITCH_SOURCE_FIELD	VARCHAR(31)	Имя поля документа, в котором содержится условие проведения. Если значение этого поля в документе равно значению поля SWITCH_VALUE шаблона, то операция будет проведена.
SWITCH_VALUE	INTEGER	Значение условия проведения
OPER_DATE_SOURCE_FIELD	VARCHAR(31)	Имя поля документа, из которого берется дата операции
DOC_TYPE_ID	INTEGER	Внутренний ID типа документа
START_DATE	DATE	Стартовая дата диапазона срабатывания шаблона. Если NULL, то начало диапазона дат совпадает с глобальной датой начала периода.
END_DATE	DATE	Крайняя дата диапазона срабатывания шаблона. Если NULL, то ограничений по сроку работы шаблона нет
IS_CHANGED	SMALLINT	Признак того, что шаблон был изменен. Используется для перепроведения всех документов этого типа с использованием данного шаблона. 0 – шаблон не изменялся 1 – шаблон был изменен пользователем

Системная таблица **TEMPLATE_DEF**

Название поля	**Тип данных**	**Назначение**
TEMPLATE_ID	INTEGER	Внутренний номер ID шаблона
OPER_POS	INTEGER	Порядковый номер строки генерируемого набора
IS_CREDIT	INTEGER	Тип записи (явно) 0 – в дебет 1 – в кредит
IS_CREDIT_SOURCE_FIELD	VARCHAR(31)	Имя поля документа, в котором содержится тип записи. Если тип записи указан явно, то NULL
ACC_ID	INTEGER	Внутренний ACC_ID счета (явно)
ACC_ID_SOURCE_FIELD	VARCHAR(31)	Имя поля документа, из которого берется ACC_ID счета. Если счет указан явно, то NULL
OBJECT_ID_SOURCE_FIELD	VARCHAR(31)	Имя поля документа, из которого берется ID объекта аналитического учета.
AMOUNT_MINUS	SMALLINT	Признак того, что знак суммы следует изменить на противоположный 0 – знак не изменять 1 – изменить знак суммы
AMOUNT_SOURCE_FIELD	VARCHAR(31)	Имя поля документа, из которого берется сумма
AMOUNT_FORMULA	VARCHAR(255)	Формула для расчета суммы. Могут использоваться символы вида Sn, где n-номер строки набора, а S-сумма в этой строке. Например, 1.2*S3 будет означать сумму третьей строки набора, умноженную на 1,2
LAYER_ID	INTEGER	Внутренний LAYER_ID слоя (явно)
LAYER_ID_SOURCE_FIELD	VARCHAR(31)	Имя поля документа, из которого берется LAYER_ID. Если слой указан явно, то NULL
QUANTITY_MINUS	SMALLINT	Признак того, что знак количества следует изменить на противоположный 0 – знак не изменять 1 – изменить знак количества
QUANTITY_SOURCE_FIELD	VARCHAR(31)	Имя поля документа, из которого берется количество
QUANTITY_FORMULA	VARCHAR(255)	Формула для расчета количества. Могут использоваться символы вида Qn, где n-номер строки набора, а Q-количество в этой строке. Например, Q1 будет означать количество первой строки набора

Для примера приведем текст хранимой процедуры, автоматически сформированный шаблоном ручных операций:

```
CREATE PROCEDURE GAAP_TEMPLATE(OP_DATE1 DATE, OP_DATE2 DATE, FOR_DOC_ID INTEGER)
RETURNS(
  OPER_DATE DATE,
  ACC_ID INTEGER,
  LAYER_ID INTEGER,
  OBJECT_ID INTEGER,
  TEMPLATE_ID INTEGER,
  DOC_ID INTEGER,
  IS_CREDIT INTEGER,
  DEBIT DECIMAL(18, 2),
  CREDIT DECIMAL(18, 2),
  QDEBIT DECIMAL(18, 3),
  QCREDIT DECIMAL(18, 3),
  DOC_N INTEGER,
  OPER_POS INTEGER
)
AS
DECLARE VARIABLE AMOUNT   DECIMAL(18,2);
DECLARE VARIABLE QUANTITY DECIMAL(18,3);
DECLARE VARIABLE V1 INTEGER;
```

```
DECLARE VARIABLE V2 INTEGER;
DECLARE VARIABLE V3 INTEGER;
DECLARE VARIABLE V4 DECIMAL(18,2);
DECLARE VARIABLE V5 INTEGER;
DECLARE VARIABLE V6 DECIMAL(18,3);
DECLARE VARIABLE S1 DECIMAL(18,2);
DECLARE VARIABLE Q1 DECIMAL(18,3);
BEGIN
  TEMPLATE_ID = 1;
  FOR SELECT
    GAAP.ID,
    GAAP.OPER_DATE,
    GAAP_POS.IS_CREDIT,
    GAAP_POS.ACC,
    GAAP_POS.OBJ,
    GAAP_POS.AMOUNT,
    GAAP_POS.LAYER,
    GAAP_POS.QUANTITY,
    GAAP_POS.N
  FROM
    GAAP, GAAP_POS
  WHERE
    ((:OP_DATE1 IS NULL) OR (GAAP.OPER_DATE >= :OP_DATE1)) AND
    ((:OP_DATE2 IS NULL) OR (GAAP.OPER_DATE <= :OP_DATE2)) AND
    ((:FOR_DOC_ID IS NULL) OR (GAAP.ID = :FOR_DOC_ID)) AND
    GAAP.ID = GAAP_POS.ID
  INTO :DOC_ID, :OPER_DATE, :V1, :V2, :V3, :V4, :V5, :V6, :DOC_N
  DO
  BEGIN
    IS_CREDIT = V1;
    ACC_ID = V2;
    OBJECT_ID = V3;
    OPER_POS = 1;
    AMOUNT = V4;
    S1 = AMOUNT;
    QUANTITY = V6;
    Q1 = QUANTITY;
    LAYER_ID = V5;
    IF (IS_CREDIT = 0) THEN
    BEGIN
      DEBIT = AMOUNT; QDEBIT = QUANTITY; CREDIT = 0; QCREDIT = 0;
    END
    ELSE
    BEGIN
      DEBIT = 0; QDEBIT = 0; CREDIT = AMOUNT; QCREDIT = QUANTITY;
    END
    SUSPEND; /*waiting for fetch operation*/

  END
END
```

С помощью простого запроса к этой процедуре мы можем, например, получить все ручные проводки, датированные с 01.01.2003 по 01.03.2003:

```
select * from gaap_template('01.01.2003','01.03.2003',NULL)
```

Если же нас интересует одна отдельно взятая «Ручная операция», скажем с ID=11043, то мы можем получить ее с помощью запроса:

```
select * from gaap_template(NULL, NULL, 11043)
```

Автоматическое проведение документов, системная таблица ACC_TURN

Так как хранимые процедуры шаблонов в любой момент предоставляют нам необходимый «кусок» журнала проводок, остается лишь сохранить этот набор в таблице проводок. Все бухгалтерские проводки хранятся в таблице ACC_TURN. Если мы хотим «перепровести» уже существующий документ (например, после внесения в него изменений), то нужно сначала удалить старые проводки по данному документу, а затем вставить новые проводки, поставляемые хранимой процедурой шаблона. Программа делает все это автоматически путем вызова хранимой процедуры **<имя_главной_таблицы_документа>_CHANGED**. Эта хранимая процедура создается программой автоматически, и ее текст зависит от того, какие шаблоны имеет конкретный тип документа.

Пример хранимой процедуры, осуществляющей перепроведение документов типа «Ручная операция»:

```
CREATE PROCEDURE GAAP_CHANGED(ID INTEGER)
AS
DECLARE VARIABLE OPER_DATE DATE;
DECLARE VARIABLE PERIOD_START DATE;
DECLARE VARIABLE LAYER_ID INTEGER;
DECLARE VARIABLE DEBIT DECIMAL(18,2);
DECLARE VARIABLE CREDIT DECIMAL(18,2);
DECLARE VARIABLE MIN_DATE DATE;
BEGIN
/*Текст данной процедуры создан системой Allegro автоматически.*/
/*Не меняйте этот текст. Ваши изменения могут быть утеряны*/
  /*Выбираем дату начала периода*/
  SELECT START_DATE FROM PERIOD_PROPERTIES INTO :PERIOD_START;
  /*Удаление старых проводок*/
  DELETE FROM ACC_TURN WHERE DOC_ID = :ID;

  /*Проверка на равенство дебета кредиту в каждом слое на каждую дату*/
  FOR SELECT OPER_DATE, LAYER_ID, SUM(DEBIT) DEBIT, SUM(CREDIT) CREDIT
  FROM GAAP_TEMPLATE(NULL,NULL,:ID) T, ACC
  WHERE OPER_DATE>=:PERIOD_START AND T.ACC_ID = ACC.ACC_ID AND
            ACC.NO_BALANCE = 0
  GROUP BY OPER_DATE, LAYER_ID INTO :OPER_DATE, :LAYER_ID, :DEBIT, :CREDIT
  DO
  BEGIN
    IF (DEBIT <> CREDIT) THEN  EXCEPTION NO_BALANCE;
    /*Запоминание саммой ранней даты в составе операции*/
    IF ((MIN_DATE IS NULL) OR (MIN_DATE > OPER_DATE)) THEN
      MIN_DATE = OPER_DATE;
  END
  /*Добавление новых проводок*/
  INSERT INTO ACC_TURN (OP_DATE, ACC_ID, LAYER_ID, OBJECT_ID, TEMPLATE_ID,
                DOC_ID, IS_CREDIT, DEBIT, CREDIT,
                QUANTITY_DEBIT, QUANTITY_CREDIT)
          SELECT OPER_DATE, ACC_ID, LAYER_ID, OBJECT_ID, TEMPLATE_ID,
                DOC_ID, IS_CREDIT, SUM(DEBIT), SUM(CREDIT),
                SUM(QDEBIT), SUM(QCREDIT)
          FROM GAAP_TEMPLATE(NULL,NULL,:ID)
          WHERE OPER_DATE >= :PERIOD_START
          GROUP BY OPER_DATE, ACC_ID, LAYER_ID, OBJECT_ID, TEMPLATE_ID,
                DOC_ID, IS_CREDIT;

  /*Сдвиг даты расчетов*/
  IF (NOT (MIN_DATE IS NULL)) THEN
    EXECUTE PROCEDURE SET_CALCULATIONS(:MIN_DATE);
END
```

Обратим внимание, что при проведении документа однотипные записи по счетам «сжимаются» с помощью агрегатных функций SUM(). Это сделано для уменьшения ***количества строк*** в таблице проводок ACC_TURN и для придания операциям более компактного и читабельного вида. Например, два десятка парных

записей при проведении накладной «Продажи» сожмется до 13 результирующих записей в таблице проводок. Одна запись будет содержать общую сумму, начисленную покупателю, одна запись - сумму начисленных доходов, одна – сумму начисленных товарных расходов, а 10 остальных – суммы, списанные со счета «Товары», раздельно для каждого товара.

Итак, для перепроведения конкретного документа какого-то типа, нам достаточно вызвать хранимую процедуру **<имя_главной_таблицы_документа>_CHANGED**, передав в нее входным параметром ID документа. Например, так можно перепровести «Продажи товара», документ с ID = 11031:

```
execute procedure sale_changed(11031);
```

Разработчик для вызова этой команды может воспользоваться также процедурой встроенного языка:

```
procedure MakeEntries(const MainTableName: string; doc_id: integer;
                      ATransaction: TIBTransaction);
```

Эта процедура просто вызывает ту же SQL команду в контексте определенной транзакции. В качестве параметров нужно передать название главной таблицы документа, внутренний номер ID документа, который следует перепровести, и указатель на компонент транзакции.

Так как все действия по перепроведению документов производятся на сервере, это происходит без ощутимых временных затрат (практически мгновенно). Все проводки хранятся в таблице ACC_TURN - сердце бухгалтерской системы. Таблица организована таким образом, чтобы максимально быстро извлекать из нее все интересующие нас сведения, например, любой Т-счет. Для этого она снабжена специальными индексами:

```
PRIMARY KEY (OP_DATE,ACC_ID,LAYER_ID,OBJECT_ID,TEMPLATE_ID,DOC_ID,IS_CREDIT)
CREATE INDEX X_ACC_TURN_DOC_ID ON ACC_TURN(DOC_ID);
CREATE INDEX X_ACC_T_ASC ON ACC_TURN(ACC_ID,OP_DATE,DOC_ID,TEMPLATE_ID,LAYER_ID);
CREATE DESCENDING INDEX X_ACC_T_DESC ON ACC_TURN(ACC_ID,OP_DATE,DOC_ID,
  TEMPLATE_ID,LAYER_ID);
```

Системная таблица **ACC_TURN**

Название поля	**Тип данных**	**Назначение**
OP_DATE	DATE	Дата проводки
ACC_ID	INTEGER	Внутренний ACC_ID счета
LAYER_ID	INTEGER	Внутренний LAYER_ID слоя
OBJECT_ID	INTEGER	Внутренний ID объекта аналитики
TEMPLATE_ID	INTEGER	Внутренний TEMPLATE_ID шаблона операции
DOC_ID	INTEGER	Внутренний ID конкретного проведенного документа
IS_CREDIT	SMALLINT	Тип записи 0 – запись в дебет 1 – запись в кредит
DEBIT	DECIMAL(18,2)	Сумма дебет
CREDIT	DECIMAL(18,2)	Сумма кредит
QUANTITY_DEBIT	DECIMAL(18,3)	Количество дебет
QUANTITY_CREDIT	DECIMAL(18,3)	Количество кредит

Обратим внимание на то, что бухгалтерские записи хранятся «вертикально» в англо-американском стиле «смешанных» проводок: по одной записи на каждый счет. Такое хранение проводок значительно ускоряет получение Т-счетов, всевозможный анализ и интерпретацию данных. Например, мы с помощью одного SQL-запроса быстро (за доли секунды при десятках тысяч проводок) можем получить текущую ***себестоимость всех имеющихся товаров на складах*** по каждому товару:

```
select g.id, o.short_name,
  (sum(a.debit)-sum(a.credit))/(sum(a.quantity_debit)-sum(a.quantity_credit))
from  acc_turn a, goods g, object_names o
where a.object_id = o.object_id and  g.id = o.object_id
group by g.id, o.short_name,
having sum(a.quantity_debit)-sum(a.quantity_credit) <> 0
```

Глава 11. Многомерные регистры

Таблицы регистров

Многомерные регистры создаются конфигурирующим и могут использоваться для разных целей, например, для складского учета. При создании нового регистра создается отдельная таблица с четырьмя обязательными системными целочисленными полями:

Название поля	**Тип данных**	**Назначение**
ID	INTEGER	Внутренний ID строки таблицы регистра
DOC_ID	INTEGER	Внутренний ID документа
DOC_N	INTEGER	Внутренний ID строки позиции документа
REG_TEMPLATE_ID	INTEGER	Внутренний ID шаблона операции

Остальные поля конфигурирующий добавляет в регистр самостоятельно. Все остальные поля регистра можно условно разделить на три группы:

- ***Поля измерений*** (основные «разрезы», например - склад, товар, партия)
- ***Поле даты операции***
- Дополнительные поля (для разных целей)
- ***Поля мер*** (поля для суммирования, например – количество, стоимость)

Поле даты операции, как сразу можно заметить, не входит в четверку системных полей, но создается конфигурирующим самостоятельно. Это обусловлено тем, что в зависимости от задачи, к полю даты могут предъявляться разные требования. В каких-то случаях достаточно иметь единственное поле даты типа DATE. В других случаях может потребоваться поле типа TIMESTAMP (дата и время). Конфигурирующий также может создать два поля: поле даты типа DATE и отдельно от него - поле времени типа TIME. В каких-то случаях в качестве второго поля может использоваться поле типа INTEGER. Одним словом, здесь возможно определенное разнообразие решений и, чтобы не ограничивать возможности конфигурирующего, ему предоставлена свобода выбора в организации временного континуума операций. Единственное условие – конфигурирующий должен создать ***индекс даты операции***. В этот индекс он может включить одно поле или несколько полей, в зависимости от того, как ему видится правильный континуум операций по дате в этом регистре, однако первое поле в индексе даты операции обязательно должно иметь тип DATE или TIMESTAMP.

Для ***полей измерений*** конфигурирующий также обязан создать индекс, который называется ***индексом группировки по основным измерениям***. Этот индекс служит для ускорения выборок и группировок в регистре, а также используется при переносе остатков в новую базу в качестве списка полей, по которым будет производиться группировка начальных остатков.

И наконец, конфигурирующий должен создать ***поля мер***. Эти поля всегда имеют числовой тип, так как по ним производится суммирование в регистре.

После того, как все поля и индексы созданы, необходимо указать определенные сведения в «Дополнительных свойствах» регистра. Там нужно указать:

- Индекс даты операции
- Индекс группировки по основным измерениям
- Поля мер
- Установить (необязательно) опцию переноса остатков в новую базу, если та создается «по остаткам».

После чего регистр считается готовым к использованию и к нему можно создавать «шаблоны операций», в которы оговаривается, как «проводить» разные типы документов в этом регистре.

Проведение документов в регистрах похоже на проведение их в таблице бухгалтерских проводок **ACC_TURN**. Каждый документ при проведении вставляет в регистр записи, копируя в них информацию из каких-то своих полей. Если документ «перепроводится» в регистре, то старые записи по данному документу из регистра удаляются, и вместо них вставляются новые записи. Различие между бухгалтерскими проводками и проводками в регистре состоит в том, что:

- В регистрах нет требований двойной записи (равенства суммы записей в дебет сумме записей в кредит)
- В регистрах можно проводить документ частично (только определенные позиции), а в бухгалтерской таблице проводок **ACC_TURN** документ всегда проводится целиком.

Таблицы итогов

Таблицы итогов создаются в дополнение к регистрам и позволяют решать две задачи:

- Поддерживать мгновенные текущие итоги (например, текущее количество товаров на складах)
- Обеспечивать конфликт блокировок (lock confilct) или deadlock при попытке двух пользователей одновременно использовать общий ресурс (например, отгрузить со склада последнюю единицу товара)

К каждому регистру можно создать несколько таблиц итогов, например, к многомерному складскому регистру, включающему в поля измерений ***Склад***, ***Товар*** и ***Партию***, можно создать две таблицы итогов: «остатки партий на складах» и «остатки товаров на складах».

При создании таблицы итогов конфигурирующий должен выбрать из всех полей регистра:

- ***Ключевые поля*** (из основных измерений регистра)
- ***Поля сумм*** (из полей мер регистра)

Таблица итогов имеет всегда поля, одноименные соответствующим полям регистра, но количество полей в таблице итогов бывает меньшим. Таблицы итогов вообще не имеют системных полей типа ID. В качестве первичного ключа в них используется ключевая комбинация измерений, например «Склад + Товар».

После создания таблицы итогов необходимо создать триггеры для таблицы регистра. Эти триггеры создаются автоматически системой Allegro – конфигурирующему достаточно нажать одну кнопку. Триггеры срабатывают при любой вставке, модификации или удалении строки в таблице регистра.

Например, триггер AFTER UPDATE для регистра GOODS_REG, управляющий таблицей итогов GOODS_REG_TOTAL, может выглядеть примерно так:

```
CREATE TRIGGER GOODS_REG_RT_UPDATE FOR GOODS_REG
ACTIVE AFTER UPDATE POSITION 200
AS
BEGIN
/*Триггер обновления таблицы итогов*/
/*Текст этого триггера создан системой Allegro автоматически.*/
/*Не меняйте этот текст. Ваши изменения могут быть утеряны*/
  IF (EXISTS(
    SELECT 1 FROM GOODS_REG_TOTAL
    WHERE
      (ACC = NEW.ACC) AND
      (GOODS = NEW.GOODS)
    ))
  THEN
    UPDATE GOODS_REG_TOTAL SET
      INCOME = INCOME + NEW.INCOME - OLD.INCOME,
      INCOME_AMOUNT = INCOME_AMOUNT + NEW.INCOME_AMOUNT - OLD.INCOME_AMOUNT,
      OUTCOME = OUTCOME + NEW.OUTCOME - OLD.OUTCOME,
      OUTCOME_AMOUNT = OUTCOME_AMOUNT +
          NEW.OUTCOME_AMOUNT - OLD.OUTCOME_AMOUNT
    WHERE
      (ACC = NEW.ACC) AND
      (GOODS = NEW.GOODS);
  ELSE
    INSERT INTO GOODS_REG_TOTAL(ACC, GOODS, INCOME, INCOME_AMOUNT,
       OUTCOME, OUTCOME_AMOUNT)
    VALUES(NEW.ACC, NEW.GOODS, NEW.INCOME - OLD.INCOME, NEW.INCOME_AMOUNT -
OLD.INCOME_AMOUNT,
       NEW.OUTCOME - OLD.OUTCOME, NEW.OUTCOME_AMOUNT - OLD.OUTCOME_AMOUNT);
END
```

Системные таблицы REG, REG_FIELDS, REG_TOTAL, REG_TOTAL_FIELDS

Список таблиц регистров хранится в системной таблице **REG**.

Системная таблица **REG**

Название поля	**Тип данных**	**Назначение**
REG_ID	INTEGER	Внутренний ID регистра
NAME	VARCHAR(50)	Название регистра
TABLE_NAME	VARCHAR(31)	Имя таблицы регистра
TRIGGERS_ARE_READY	SMALLINT	Готовность триггеров, поддерживающих итоги 1 – триггеры готовы к использованию 0 – триггеры не сформированы
COPY_RESTS_TO_NEW_PERIOD	SMALLINT	Флаг переноса остатков в новую базу 1 – переносить остатки 0 – не переносить остатки
DATE_INDEX	VARCHAR(31)	Индекс даты операции
GROUP_INDEX	VARCHAR(31)	Индекс группировки по основным измерениям
SUM_FIELDS	VARCHAR(1023)	Список полей мер (через запятую)

При создании нового регистра REG_ID формируется при помощи генератора REG_ID_GEN

Список полей таблиц регистров хранится в системной таблице **REG_FIELDS**

Системная таблица **REG_FIELDS**

Название поля	**Тип данных**	**Назначение**
REG_ID	INTEGER	Внутренний ID регистра
FIELD_NAME	VARCHAR(31)	Имя поля в таблице (уникальное в сочетании с ID)
DISPLAY_NAME	VARCHAR(50)	Название атрибута (уникальное в пределах настройки)
DISPLAY_FORMAT	VARCHAR(50)	Формат отображения чисел, дат или сборных наименований для полей ссылок, в зависимости от типа поля
REF_TABLE_NAME	VARCHAR(31)	Только для полей ссылок: название таблицы, на которую ссылается данное поле

Список таблиц итогов хранится в системной таблице **REG_TOTAL**.

Системная таблица **REG_TOTAL**

Название поля	**Тип данных**	**Назначение**
REG_TOTAL_ID	INTEGER	Внутренний ID таблицы итогов
REG_ID	INTEGER	Внутренний ID регистра
NAME	VARCHAR(50)	Название таблицы итогов
TABLE_NAME	VARCHAR(31)	Название таблицы в базе данных

При создании новой таблицы итогов REG_TOTAL_ID формируется при помощи генератора REG_TOTAL_ID _GEN.

Список полей таблиц итогов хранится в системной таблице **REG_TOTAL_FIELDS**

Системная таблица **REG_TOTAL_FIELDS**

Название поля	**Тип данных**	**Назначение**
REG_TOTAL_ID	INTEGER	Внутренний ID таблицы итогов
FIELD_NAME	VARCHAR(31)	Имя поля в таблице (уникальное в сочетании с ID)
DISPLAY_NAME	VARCHAR(50)	Название атрибута (уникальное в пределах настройки)
DISPLAY_FORMAT	VARCHAR(50)	Формат отображения чисел, дат или сборных наименований для полей ссылок, в зависимости от типа поля
REF_TABLE_NAME	VARCHAR(31)	Только для полей ссылок: название таблицы, на которую ссылается данное поле

Шаблоны операций с регистрами, системные таблицы REG_TEMPLATE, REG_TEMPLATE_FIELDS

Автоматические проводки по регистрам формируются на основе документов и ***шаблонов операций с регистрами***. Каждый тип документа может иметь несколько шаблонов операций, работающих с каждым из регистров. Шаблоны легко настраивать. Изменив шаблон операции с регистром, мы всегда можем ***изменить способ проведения в регистре*** всех документов определенного типа. Это создает большую гибкость при разработке конфигурации. Каждый шаблон имеет название операции и название автоматически формируемой ***хранимой процедуры***, которая создает набор проводок в формате таблицы соответствующего многомерного регистра. Если мы перенастроим шаблон, программа автоматически изменит текст этой хранимой процедуры. Каждый шаблон содержит, кроме того, ***условие срабатывания***, что позволяет по-разному проводить одни и те же типы документов, в зависимости от каких-то признаков в самих документах. Кроме того, в шаблоне может быть указан ***диапазон дат***, для которых он действует. Это означает возможность провести документы до какой-то даты с помощью одного шаблона, а после этой даты – с помощью другого шаблона.

Каждый шаблон содержит имя атрибута (поля) документа, откуда берется ***дата операции***.

Каждый шаблон содержит набор «схем записей в регистр», в каждой из которых указывается, откуда брать данные для каждого поля регистра. Данные могут браться из полей документа или иметь фиксированные значения, указанные в шаблоне явно.

Ниже показаны структуры таблиц, в которых хранятся сведения о шаблонах операций с регистрами, на основе которых программа формирует тексты соответствующих хранимых процедур.

Системная таблица **REG_TEMPLATE**

Название поля	**Тип данных**	**Назначение**
REG_TEMPLATE_ID	INTEGER	Внутренний ID шаблона
REG_ID	INTEGER	Внутренний ID регистра
NAME	VARCHAR(50)	Название операции с регистром
STOREDPROC_NAME	VARCHAR(31)	Название хранимой процедуры, возвращающей проводки в формате таблицы регистра
SWITCH_SOURCE_FIELD	VARCHAR(31)	Имя поля документа, в котором содержится условие проведения. Если значение этого поля в документе равно значению поля SWITCH_VALUE шаблона, то операция будет проведена.
SWITCH_VALUE	INTEGER	Значение условия проведения
OPER_DATE_SOURCE_FIELD	VARCHAR(31)	Имя поля документа, из которого берется дата операции
DOC_TYPE_ID	INTEGER	Внутренний ID типа документа
START_DATE	DATE	Стартовая дата диапазона срабатывания шаблона. Если NULL, то начало диапазона дат совпадает с глобальной датой начала периода.
END_DATE	DATE	Крайняя дата диапазона срабатывания шаблона. Если NULL, то ограничений по сроку работы шаблона нет
IS_CHANGED	SMALLINT	Признак того, что шаблон был изменен. Используется для перепроведения всех документов этого типа с использованием данного шаблона. 0 – шаблон не изменялся 1 – шаблон был изменен пользователем

При создании нового шаблона значение поля REG_TEMPLATE_ID формируется при помощи генератора REG_TEMPLATE_ID _GEN.

Системная таблица **REG_TEMPLATE_FIELDS**

Название поля	**Тип данных**	**Назначение**
REG_TEMPLATE_ID	INTEGER	Внутренний номер ID шаблона
OPER_POS	INTEGER	Порядковый номер строки генерируемого набора
FIELD_NAME	VARCHAR(31)	Имя поля в таблице регистра
FIELD_SOURCE	VARCHAR(31)	Имя поля-источника в таблице документа
INVERT_FLAG	SMALLINT	Признак того, что знак числа следует изменить на противоположный («минус») 0 – знак не изменять 1 – изменить знак суммы
INTEGER_VALUE	INTEGER	Явное целое значение
VARCHAR_VALUE	VARCHAR(50)	Явное строковое значение

В качестве примера приведем текст хранимой процедуры, автоматически сформированный шаблоном проведения «документом межскладского перемещения» для складского регистра:

```
CREATE PROCEDURE DISPLACEMENT_REG_TEMPLATE
  (OP_DATE1 DATE, OP_DATE2 DATE, FOR_DOC_ID INTEGER, FOR_DOC_N INTEGER)
RETURNS(
  DOC_ID INTEGER, DOC_N INTEGER, REG_TEMPLATE_ID INTEGER, OP_DATE DATE,
  ACC INTEGER,  GOODS INTEGER,  INCOME INTEGER, INCOME_AMOUNT DECIMAL(18, 2),
  OUTCOME INTEGER, OUTCOME_AMOUNT DECIMAL(18, 2))
AS
DECLARE VARIABLE V1 INTEGER;
DECLARE VARIABLE V2 INTEGER;
DECLARE VARIABLE V3 DECIMAL(18,2);
DECLARE VARIABLE V4 INTEGER;
DECLARE VARIABLE V5 INTEGER;
BEGIN
/*Текст данной процедуры создан системой Allegro автоматически.*/
/*Не меняйте этот текст. Ваши изменения могут быть утеряны*/
  REG_TEMPLATE_ID = 101;
  DOC_N = 0;
  FOR SELECT
    DISPLACEMENT.ID,
    DISPLACEMENT.ADATE,
    DISPLACEMENT.STORE1,
    DISPLACEMENT.STORE2,
    DISPLACEMENT_ITEM.N,
    DISPLACEMENT_ITEM.AVG_ITEM_COST,
    DISPLACEMENT_ITEM.GOODS,
    DISPLACEMENT_ITEM.QUANTITY
  FROM
    DISPLACEMENT, DISPLACEMENT_ITEM
  WHERE
    DISPLACEMENT.ID = DISPLACEMENT_ITEM.ID AND
    ((:FOR_DOC_N IS NULL) OR (DISPLACEMENT_ITEM.N = :FOR_DOC_N)) AND
    ((:FOR_DOC_ID IS NULL) OR (DISPLACEMENT.ID = :FOR_DOC_ID)) AND
    ((:OP_DATE1 IS NULL) OR (DISPLACEMENT.ADATE >= :OP_DATE1)) AND
    ((:OP_DATE2 IS NULL) OR (DISPLACEMENT.ADATE - 1 < :OP_DATE2))
  INTO :DOC_ID, :OP_DATE, :V1, :V2, :DOC_N, :V3, :V4, :V5
  DO
  BEGIN
    ACC = V1;
    GOODS = V4;
    OUTCOME = V5;
    OUTCOME_AMOUNT = V3;
    INCOME = 0;
    INCOME_AMOUNT = 0;
    SUSPEND; /*waiting for fetch operation*/

    ACC = V2;
    GOODS = V4;
    INCOME = V5;
    INCOME_AMOUNT = V3;
    OUTCOME = 0;
    OUTCOME_AMOUNT = 0;
    SUSPEND; /*waiting for fetch operation*/

  END
END
```

С помощью простого запроса к этой процедуре мы можем, например, получить все записи, которые необходимо вставить в регистр для всех документов «межскладского перемещения»:

```
select * from DISPLACEMENT_REG_TEMPLATE (NULL,NULL,NULL,NULL)
```

Если нас интересуют записи для вставки в регистр для одного отдельно взятого документа «межскладского перемещения», скажем с ID=11043, то мы можем получить их с помощью запроса:

```
select * from DISPLACEMENT_REG_TEMPLATE (NULL, NULL, 11043, NULL)
```

Мы можем также получить записи для одной отдельно взятой позиции документа, например с N = 33900:

```
select * from DISPLACEMENT_REG_TEMPLATE (NULL, NULL, 11043, 33900)
```

Хранимая процедура, сформированная шаблоном может иметь различное количество выходных параметров, так как оно зависит от количества полей в конкретном регистре. Однако все такие хранимые процедуры имеют стандартные входные параметры:

Входные параметры хранимой процедурой шаблона операции с регистром:

Название параметра	**Тип данных**	**Назначение**
OP_DATE1	DATE	Начало диапазона дат (NULL – диапазон не ограничен снизу)
OP_DATE1	DATE	Конец диапазона дат (NULL – диапазон не ограничен сверху)
FOR_DOC_ID	INTEGER	Внутренний ID документа (NULL – все документы этого типа)
FOR_DOC_N	INTEGER	Внутренний номер позиции документа (NULL – все позиции)

Автоматическое проведение документов в регистрах

Так как хранимые процедуры шаблонов в любой момент предоставляют нам необходимый «кусок» записей регистра, остается лишь сохранить этот набор в таблице регистра. Если мы хотим «перепровести» уже существующий документ (например, после внесения в него изменений), то нужно сначала удалить старые записи из регистра по данному документу, а затем вставить новые записи, поставляемые хранимой процедурой шаблона. Упростить вставку записей в регистр можно с помощью вызова хранимой процедуры **<имя_главной_таблицы_документа>_R_CHANGE**. Эта хранимая процедура создается программой автоматически при настройке шаблонов записей по регистрам. Текст этой процедуры зависит от того, какие шаблоны имеет конкретный тип документа.

Идеология системы Allegro предполагает, что все необходимые для проведения данные ***уже подготовлены*** и хранятся в самих документах. В частности, если документ списывает стоимость товаров, уходящих со склада, то эта стоимость должна быть ***предварительно вычислена и сохранена*** в соответствующем поле документа. Вычисление стоимости может производиться на основе запросов к таблице регистра или таблицам итогов ***в процессе создания*** документа, однако при проведении документа с помощью хранимой процедуры **<имя_главной_таблицы_документа>_R_CHANGE** эта информация может лишь использоваться, но не модифицироваться.

Процедура **<имя_главной_таблицы_документа>_R_CHANGE** имеет три входных параметра:

Название параметра	**Тип данных**	**Назначение**
ID	INTEGER	Внутренний ID документа (NULL – все документы этого типа)
N	INTEGER	Внутренний номер позиции документа (NULL – все позиции)
REG_TEMPLATE_ID	INTEGER	Внутренний ID шаблона (NULL – все шаблоны для документа этого типа)

Параметры ID и REG_TEMPLATE_ID не могут одновременно принимать значение NULL. Если такое произошло, процедура не создаст записей в регистре.

Если параметр ID принимает значение NULL, то перепроводятся все документы этого типа одним шаблоном, для которого указан REG_TEMPLATE_ID.

Если параметр N отличен от NULL, то перепроводится одна строка подчиненной таблицы (позиция документа).

Если параметр ID отличен от NULL, а параметр REG_TEMPLATE_ID равен NULL, то конкретный документ перепроводится всеми имеющимися шаблонами во всех регистрах, которые они затрагивают.

Пример хранимой процедуры, осуществляющей перепроведение документов типа «межскладское перемещение»:

```
CREATE PROCEDURE DISPLACEMENT_R_CHANGE (ID INTEGER, N INTEGER,
      REG_TEMPLATE_ID INTEGER)
AS
DECLARE VARIABLE PERIOD_START DATE;
BEGIN
/*Текст данной процедуры создан системой Allegro автоматически.*/
/*Не меняйте этот текст. Ваши изменения могут быть утеряны*/
  IF ((ID IS NULL) AND (REG_TEMPLATE_ID IS NULL)) THEN
    EXIT;
  /*Выбираем дату начала периода*/
  SELECT START_DATE FROM PERIOD_PROPERTIES INTO :PERIOD_START;

  /*Удаляем прежние записи из регистра GOODS_REG*/
  DELETE FROM GOODS_REG
  WHERE
   ((:ID IS NULL) OR (DOC_ID = :ID)) AND
   ((:N IS NULL) OR (DOC_N = :N)) AND
   ((:REG_TEMPLATE_ID IS NULL) OR (REG_TEMPLATE_ID = :REG_TEMPLATE_ID));
  /*Вставляем новые записи в регистр*/
  IF ((REG_TEMPLATE_ID IS NULL) OR (REG_TEMPLATE_ID = 101)) THEN
    INSERT INTO GOODS_REG(ID, DOC_ID, DOC_N, REG_TEMPLATE_ID, OP_DATE,
                    ACC, GOODS, INCOME, INCOME_AMOUNT, OUTCOME, OUTCOME_AMOUNT)
    SELECT GEN_ID(GOODS_REG_ID_GEN, 1), DOC_ID, DOC_N, REG_TEMPLATE_ID, OP_DATE,
                    ACC, GOODS, INCOME, INCOME_AMOUNT, OUTCOME, OUTCOME_AMOUNT
    FROM DISPLACEMENT_REG_TEMPLATE(NULL, NULL, :ID, :N)
    WHERE OP_DATE >= :PERIOD_START;

END
```

Итак, для перепроведения конкретного документа какого-то типа, нам достаточно вызвать хранимую процедуру **<имя_главной_таблицы_документа>_R_CHANGE**, передав в нее входным параметром ID документа. Например, так можно полностью перепровести в регистрах документ «Межскладского перемещения» с ID = 11035:

```
execute procedure displacement_r_change(11035, NULL, NULL);
```

Разработчик для вызова этой команды может воспользоваться также процедурой встроенного языка:

```
procedure MakeRegEntries(const MainTableName: string;
      doc_id,doc_n, reg_template_id: integer; ATransaction: TIBTransaction);
```

Процедура ***MakeRegEntries*** вызывает эту же SQL команду в контексте определенной транзакции. В качестве параметров нужно передать название главной таблицы документа, внутренний ID документа, который следует перепровести, внутренний идентификатор позиции (N) документа, внутренний идентификатор шаблона и указатель на компонент транзакции. При вызове этой функции из текста скриптового проекта конфигурирующий в соответствующих параметрах передает вместо NULL значение 0,если не хочет уточнять какой-то из целочисленных параметров, например:

```
MakeRegEntries('DISPALCEMENT', 11035,0,0);
  //перепроведение в регистрах всех позиций документа
  //«межскладское перемещение» с ID=11035
```

Так как все действия по перепроведению документов производятся на сервере, это происходит без ощутимых затрат времени (практически мгновенно).

Глава 12. Разработка оконного интерфейса

Полезные ссылки

INTERBASE
Лучший сетевой ресурс по серверам баз данных семейства InterBase
http://www.ibase.ru/

Сервер Firebird
http://sourceforge.net/projects/firebird/
Сервер Yaffil
http://yaffil.ibase.ru/

СКРИПТЕР И ПАЛИТРЫ DREAM CONTROLS
Производитель компонентов Dream Controls
http://www.dream-com.com/
Документ, описывающий различия между языками Delphi Script и Object Pascal
http://www.dream-com.com/dscriptdesc.html

ГЕНЕРАТОР ОТЧЕТОВ XL Report
Сайт компании AfalinaSoft - производителя компонента XL Report
http://www.afalinasoft.com/
Документацию по компоненту XL Report в формате PDF можно скачать здесь
http://www.afalinasoft.com/rus/download-xl-report.html

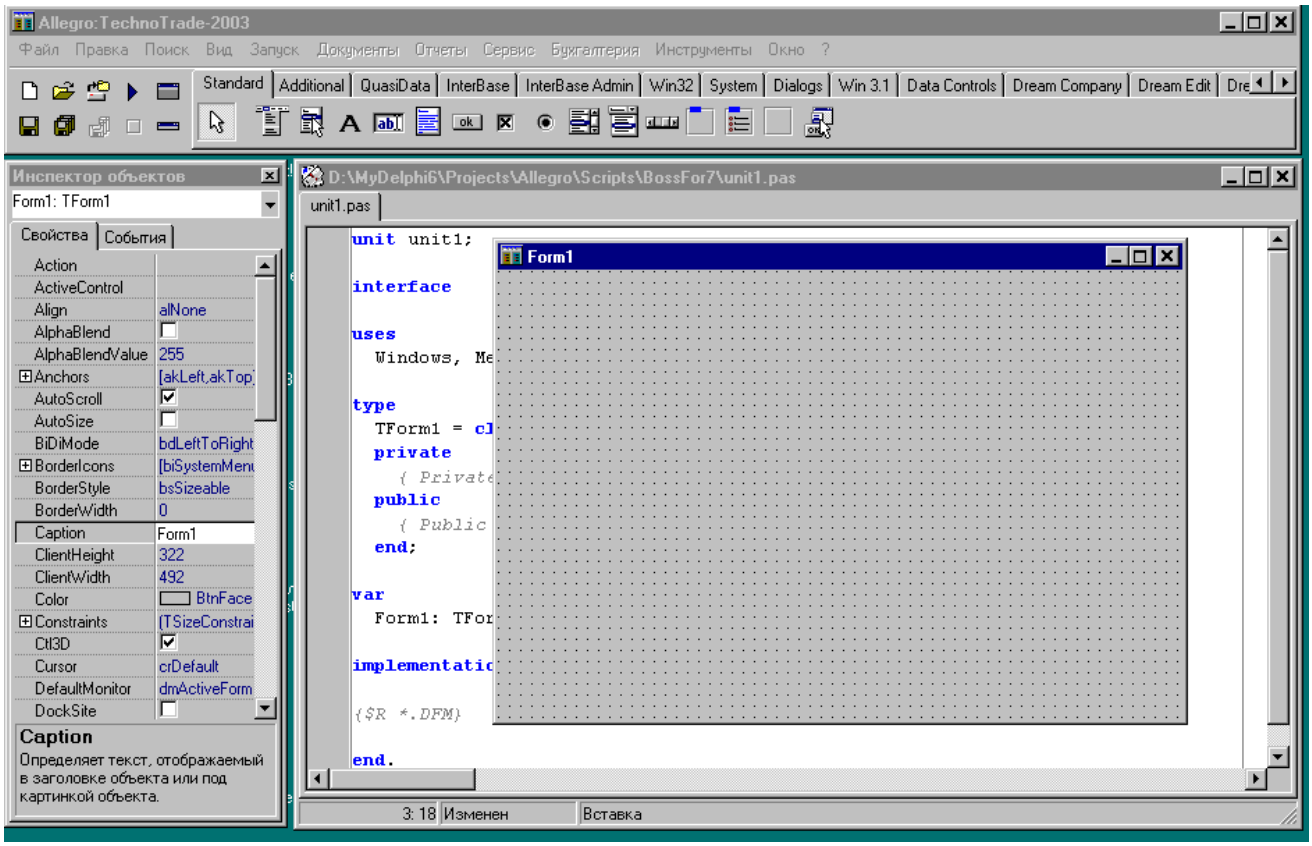
ГЕНЕРАТОР ОТЧЕТОВ Fast Report
Сайт производителя
http://www.fastreport.ru/

ЯЧЕИСТАЯ СЕТКА DbAltGrid
Сайт компании Quasidata - Производителя компонента DbAltGrid
http://www.quasidata.com/
Краткое описание компонента DbAltGrid
http://www.quasidata.com/dbaltgrid.html
Часто задаваемые вопросы по компоненту DbAltGrid
http://www.quasidata.com/faq.html

Рекомендации разработчику интерфейса

Для переключения в режим дизайна и обратно в исполняющий режим используйте пункт ***Инструменты/Дизайнер (Ctrl+D)*** Главного меню.

В режиме дизайна Главное окно Allegro сжимается по вертикали и на нем отображаются палитры компонентов VCL. Главное меню слева дополняется новыми пунктами.



Каждый интерфейс следует создавать в виде отдельного проекта (приложения).

Для этого в режиме дизайнера можно использовать пункт меню ***Файл/Новое приложение***. Проект, создаваемый таким способом, использует язык Delphi Script. Если Вы желаете работать с другим языком (VB Script, Java Script или Active Perl), используйте пункт меню Файл/Новый… и дальше выберите желаемый язык приложения. Приложение создается одновременно с главной формой Form1. После создания нового проекта отображается его главная форма, редактор текстов и Инспектор объектов в левой стороне. Теперь Вы можете добавлять компоненты на форму и писать программный код в редакторе текста. Среда проектирования напоминает среду Delphi и поэтому хорошо знакома программистам.

Вы можете добавить в проект новые формы и модули. Фактически проект (приложение) это файл, с расширением **ipr**, который хранит информацию о том, какие формы и модули входят в данный проект. Среда дизайнера устроена так, что в ней возможен запуск проектов на исполнение.

Перед запуском проекта в среде дизайнера Allegro потребует сохранения всех файлов проекта.

Рекомендуется предварительно создать ***одну общую директорию*** для всех файлов ***данной*** конфигурации, поместив ее в поддиректорию **\scripts** каталога программы Allegro. Эту директорию следует указать в качестве «Директории проектов» в «Свойствах соединения» с базой данных. Не рекомендуется, хоть и не запрещено, создавать отдельные папки для каждого проекта. Но проще все же все файлы хранить в одной папке. Так легче гарантировать уникальность названий модулей в пределах конфигурации.

Как только создан проект со своей главной формой, его можно запустить на исполнение.

При запуске ***(F9)*** происходит восстановление размеров Главного окна Allegro. Если главная форма проекта имеет стиль FormStyle = fsMDIChild, то она работает в качестве дочернего окна в Главном окне Allegro. Мы рекомендуем в большинстве случаев использовать именно этот стиль форм, так как он позволяет реализовать мультиоконный интерфейс (MDI).

При остановке проекта ***(Ctrl+F2)*** главное окно Allegro обратно сжимается и отображаются Инспектор объектов, редактор текста и формы проекта в режиме дизайна.

Рекомендуется компоненты доступа к данным (компоненты транзакций и SQL-запросов) располагать прямо на формах оконного интерфейса. Не запрещается использовать модули данных (DataModule), но в большинстве случаев это скорее излишество.

Во всех формах и из всех программных модулей доступны два встроенных глобальных компонента:

MainConnection.MainDatabase – компонент базы данных, с которой должна работать конфигурация

MainConnection.MainTransaction – транзакция «только на чтение», с которой могут работать все SQL-запросы, ограничивающиеся чтением данных.

Для организации транзакий при редактировании данных используются компоненты TIBTransaction, которые мы рекомендуем размещать непосредственно на формах главных окон проектов. В большинстве случаев Вам будет достаточно режима изоляции транзакций ReadCommitted. Для установки этого режима изоляции дважды щелкните мышью на компоненте транзакции и выберите этот режим в появившемся диалоге. Все транзакции следует подключать к встроенному глобальному компоненту базы данных **MainConnection.MainDatabase** (кроме тех случаев, когда Вам необходимо связаться с внешней базой для обмена данными между разными базами). Подключение всех транзакций к одному глобальному компоненту базы исключит возможность нескольких одновременных соединений одного и того же пользователя с базой.

Проекты оконных интерфейсов создаются, как правило, для решения трех задач:

- Предоставления пользователю интерфейса для создания и редактирования документов какого-то типа
- Предоставления пользователю интерфейса для построения отчета
- Предоставления пользователю интерфейса для служебных манипуляций данными

Если решается первая задача, то после сохранения проекта следует его «привязать» к типу документа. Это осуществляется в окне «Метаданные» в «Дополнительных свойствах документа». Там нужно указать имя файла проекта оконного интерфейса на закладке «Интерфейс» и способ вызова проекта (стандартный, модальное окно, множество экземпляров). Если это сделано, то при вызове любого документа этого типа из «Проводника по документам» автоматически запустится указанный проект.

Для того чтобы узнать программно, какой именно документ нужно загружать, если проект вызван из «проводника по документам», каждый проект имеет специальный экземпляр переменной **RunContext**, в свойстве **RunContext.Documents[0]** которой хранится эта информация. Система Allegro не исключает создание проектов оконного интерфейса, поддерживающих в одном окне одновременное редактирование нескольких логически связанных между собой документов разного типа. Поэтому в переменной **RunContext** сведения о редактируемых документах хранятся в свойстве-массиве **Documents.** Каждый элемент массива содержит идентификатор типа документа и идентификатор самого документа.

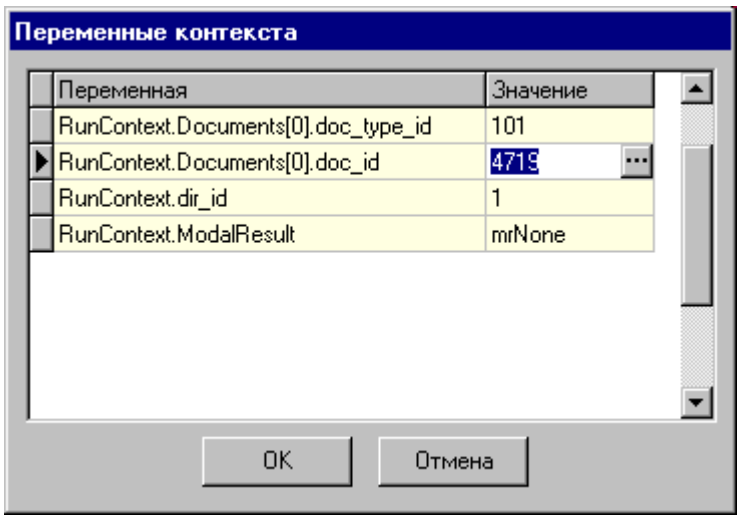
Таким образом, один проект может использоваться сразу несколькими типами документов в качестве «Проекта оконного интерфейса». Это создает большую гибкость в бизнес-логике, которую можно таким способом реализовать, но и накладывает дополнительные обязанности на конфигурирующего, если только он не использует при вызове проекта режим «модальное окно». Основной такой обязанностью является необходимость заполнять массив документов в переменной **RunContext** информацией обо всех документах, которые редактируются в данном экземпляре проекта. Это нужно для того, чтобы среда Allegro могла избежать повторного запуска одного и того же документа на редактирование, если тот уже и так находится на экране. При каждом вызове документа на редактирование Allegro проверяет массив всех переменных типа **TRunContext**, ассоциированных со всеми запущенными на данный момент экземплярами проектов и если находит, что вызываемый документ уже где-то присутствует этом массиве, то выводит соответствующее окно на передний план. И только если вызванный документ в этом массиве не найден, создается новый экземпляр проекта, экземпляр переменной **RunContext** для него и в свойство-массив **RunContext.Documents** этой переменной добавляется один элемент с информацией о вызванном документе:

RunContext.Documents[0].doc_type_id = ID типа документа
RunContext.Documents[0].doc_id = ID документа

Если создается новый документ, то при вызове проекта интерфейса внутреннему id документа присваивается значение (-1).

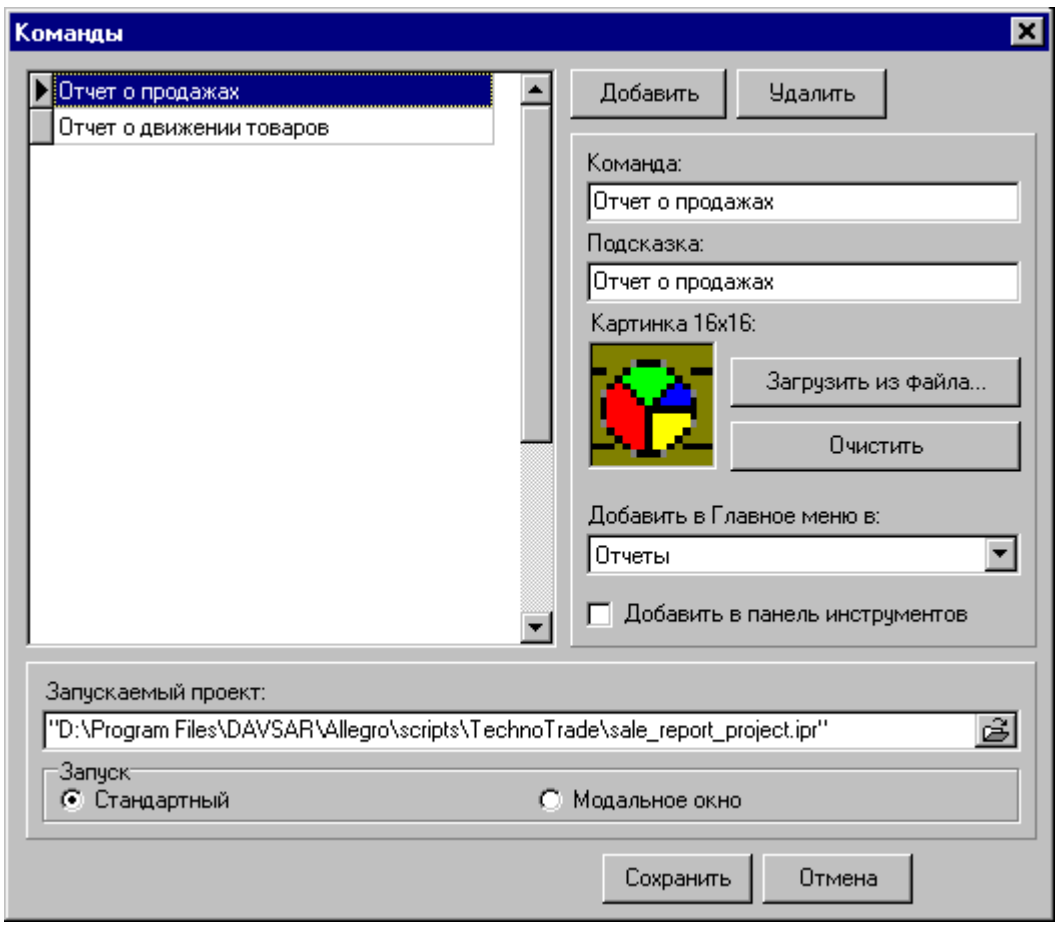
RunContext.Documents[0].doc_id = -1
RunContext.dir_id = ID текущей папки проводника

Если Вы запускаете проект в режиме дизайнера, то используется глобальный объект RunContext, свойства которого можно установить вручную с помощью пункта меню ***Запуск/Переменные контекста***:



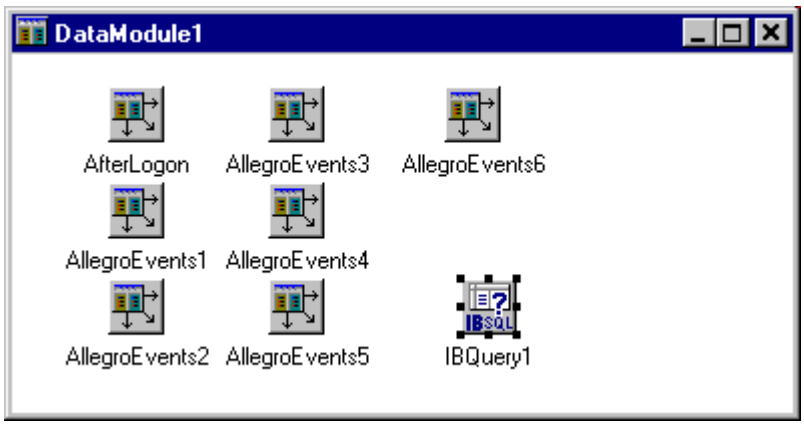
Это позволяет полностью имитировать будущий вызов документа на редактирование из «Проводника» еще на стадии запуска из под среды дизайнера.

Если же Вы создаете проект оконного интерфейса, не предназначенный для редактирования документов, например, если это какой-то проект для построения отчета или иных целей, то привязка его к среде Allegro для последующего вызова на исполнение должна осуществляться через механизм ***команд***. Для этого используйте пункт меню ***Инструменты/команды***. Появится диалог управления командами.



Глобальные модули

Кроме скриптовых проектов оконного интерфейса конфигурация может содержать ***глобальные модули***. Каждый такой модуль должен создаваться на основе модуля данных (DataModule). Модуль данных это своеобразное невидимое окно. В режиме дизайна на этом окне можно расположить какие-то невизуальные компоненты, например, таймеры (Timer), компоненты доступа к данным (IBDataSet, IBQuery, IBStoredProc) или компоненты событий (AllegroEvent):



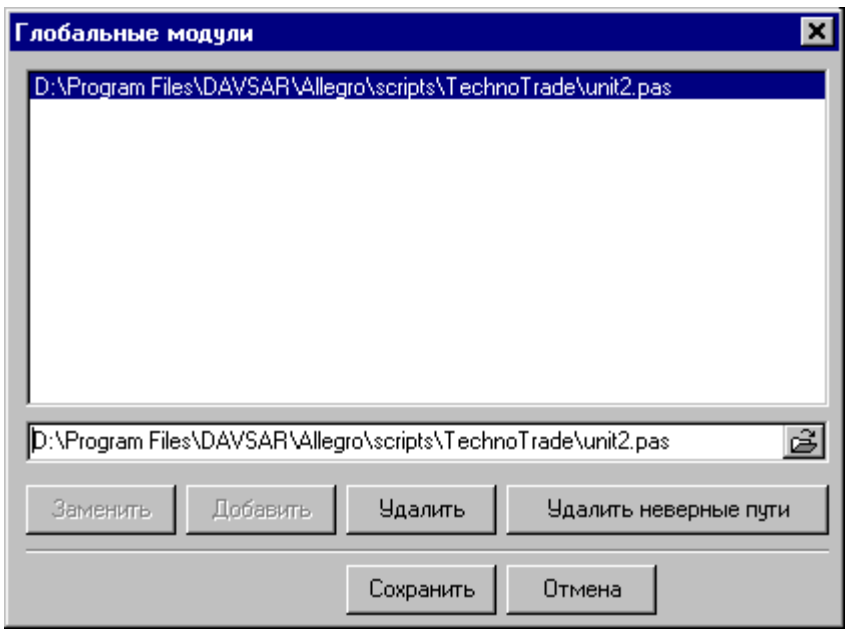
После соединения с базой данных все глобальные модули конфигурации загружаются и остаются загруженными, пока работает это соединение с базой данных. В некоторых ситуациях глобальные модули без предупреждения *временно выгружаются*. Это происходит, если:

- на экране присутствует окно «Метаданные»
- работает диалог «Директория проектов»
- работает «Мастер создания новой базы данных»

работает диалог «Восстановление базы данных из архива».

Глобальные модули *перегружаются* при каждом сохранении файлов в Дизайнере. Глобальные модули *выгружаются* (с предупреждением) при вызове диалога «Глобальные модули» и *вновь загружаются* (с сообщением) после того, как диалог завершен. В сообщениях указывается количество одновременно работающих глобальных модулей.

Каждая конфигурация имеет свой список глобальных модулей. Этот список хранится в таблице GLOBALS и для того, чтобы какой-то модуль включить в этот список. нужно использовать пункт ***Инструменты/Глобальные модули*** Главного меню. Появится диалог:



Если в «Свойствах соединения» не указана *директория проектов*, этот диалог вызвать не удастся.

Различия между языками Delphi Script и Object Pascal

Интерпретатор Dream Scripter от компании Dream Company, встроенный в Allegro, имеет ряд особенностей. Различия между языками Delphi Script и Object Pascal описаны в документе:

http://www.dream-com.com/dscriptdesc.html

Здесь мы приводим наш перевод этого документа:

Документ описывает различия между Delphi Script и Object Pascal, используемым в Delphi 5.

Все переменные в Delphi Script всегда имеют тип Variant. Приведение типов игнорируется.

Типы в объявлениях переменных игнорируются и могут быть пропущены, так что показанные ниже объявления корректны:

```
var a : integer;
var b : integerr;
var c, d;
```

Типы параметров в объявлениях процедур и функций игнорируются и могут быть пропущены, например, такой код корректен:

```
function sum(a, b) : integer;
begin
  result := a + b;
end;
```

Типы элементов массивов игнорируются и могут быть пропущены, так что эти объявления эквивалентны:

```
var x : array [1..2] of double;
var x : array [1..2];
```

Следующие ключевые слова игнорируются:
interface, implementation, program, unit

Вы можете их использовать, но они не будут иметь эффекта.
Директива ***in*** в секции ***uses*** игнорируется.

Delphi Script не поддерживает объявления типов. Он пытается их пропустить, однако некоторые сложные объявления могут вызвать сбой парсера.

Операторы ^ и @ не поддерживаются.

Вы не можете использовать имя функции для присвоения ему возвращаемого функцией значения, используйте для этой цели ключевое слово ***Result***.

Вложенные функции поддерживаются, но Вы не можете использовать переменные, объявленные в функциях верхнего уровня внутри вложенных.

Следующие ключевые слова не поддерживаются:
as, asm, class, dispinterface, exports, finalization, inherited, initialization, inline,
interface, is, library, object, out, property, record, resourcestring, set, type

Следующие директивы не поддерживаются (учтите, что некоторые из них являются устаревшими и не поддерживаются и в Delphi):
absolute, abstract, assembler, automated, cdecl, contains, default, dispid, dynamic,
export, external, far, implements, index, message, name, near, nodefault, overload,
override, package, pascal, private,protected, public,
published, read, readonly, register, reintroduce,
requires, resident, safecall, stdcall, stored, virtual, write, writeonly

Следующие функции RTL не поддерживаются в Delphi Script:
Abort, Addr, Assert, Dec, FillChar, Finalize, Hi, High, Inc, Initialize, Lo, Low, New, Ptr, SetString, SizeOf, Str, UniqueString, VarArrayRedim, VarArrayRef, VarCast, VarClear, VarCopy

Объявление открытых массивов не поддерживается.

CASE может использоваться для любых типов переменных. Так что Вы можете написать:

```
case UserName of
  'Alex', 'John' : IsAdministrator := true;
  'Peter' : IsAdministrator := false;
else
  raise('Unknown user');
end;
```

Raise может использоваться без параметров для повторной генерации последнего исключения. Вы можете также использовать строковый параметр для генерации исключения с определенным сообщением. Например:

```
Raise(Format('Invalid value : %d', [Height]));
```

Ключевое слово ***Threadvar*** интерпретируется как ***Var***
Конструкторы **[…]** для множеств не поддерживаются. Вы можете использовать функцию ***MkSet*** для создания множеств. Например:

```
Font.Style := MkSet(fsBold, fsItalic);
```

Оператор ***In*** для множеств не поддерживается. Используйте функцию ***InSet*** для проверки того, что значение является членом множества. Например:

```
if InSet(fsBold, Font.Style) then
ShowMessage('Bold');
```

Учтите, что операторы ***'+', '-', '*', '<=', '>='*** не работают корректно с множествами. Вы должны использовать вместо них логические операторы. Например,

```
ASet := BSet + CSet;
```

должно быть заменено на

```
ASet := BSet or CSet;
```

CreateObject может использоваться для создания объектов, которые сами неявно уничтожатся, если больше не используются. Так, вместо

```
procedure proc;
var l;
begin
  l := TList.Create;
  try
  // do something with l
  finally
    l.Free;
  end;
end;
```

вы можете написать

```
procedure proc;
var l;
begin
 l := CreateObject(TList);
  // do something with l
end;
```

Встроенная функция ***Evaluate*** может использоваться для интерпретации строк, как программного кода в процессе программы и исполнять код, содержащийся в строках. Например, Вы можете написать такой скрипт

```
Evaluate(ProcNames[ProcIndex]);
```

и процедуры, имена которых определены в **ProcNames[ProcIndex]** будут вызваны.

Встроенная директива ***UseUnit*** может быть использована для динамического добавления любого модуля в секцию uses во время выполнения. Например, следующий скрипт

```
if FileExists('Update.pas') then
begin
  UseUnit('Update.pas');
  Evaluate('Update.DoUpdate');
end;
```

проверяет, существует ли модуль под названием ***Update.pas*** и если это так, вызывает его метод ***DoUpdate***.

Директива ***UnloadUnit*** выгружает любой модуль, добавленный в секцию uses вызовом ***UseUni***

Встроенные глобальные переменные и константы

Имеется ряд глобальных переменных, которые можно использовать в оконных модулях.

```
var
  MainConnection: TmainConnection;
     //модуль данных (TDataModule), содержащий глобальные
     //объекты соединений (TIBDatabase и TIBTransaction)
  MainConnection.MainDatabase: TIBDatabase; //глобальный объект соединения с базой данных
  MainConnection.MainTransaction: TIBTransaction; //глобальная транзакция
  MainMetadata: Tmetadata;   //глобальный объект кеширования метаданных

  DbServerName: string;  //имя сервера, с которым установлено текущее соединение
  DbDatabaseFileName: string; //имя файла текущей базы данных
  DbProjectDirectory: string; //полный сетевой путь к конфигурационным файлам.

  DocExplorerSelectedDir_Id: integer; //значение ID текущей директории «Проводника документов»
  ManopSelectedDir_Id: integer = 1; //значение ID текущей папки интерфейса ручных операций.

  DbIniFileName: string; //путь к текущему файлу db.ini
  EmptyDbScriptFileName: string;  //имя файла SQL-скрипта пустой базы. По умолчанию Allegro.sql
  UserName: string; //имя текущего пользователя
  DbNetProtocol: integer;
    //текущий протокол связи с сервером баз данных InterBase.
    //Принимает значения 1..4. (См. ниже)

const
  NET_PROTOCOLS: array[1..4] of TProtocol = (TCP, SPX, NamedPipe, Local); //массив протоколов
```

Встроенные специализированные процедуры и функции

В программе имеется ряд встроенных специализированных процедур и функций, которые можно использовать в оконных модулях:

```
function CallDocument(var doc_type_id, doc_id, dir_id: integer;
    const params: string): TModalResult;
    //вызывает документ на экран, если у того имеется проект оконного интерфейса.

function CallDocumentEx(var doc_type_id, doc_id, dir_id: integer;
    const params: string; parent_doc_type_id, parent_doc_id: integer): TModalResult;
    //вызывает документ на экран, если у того имеется проект оконного интерфейса.
    //имеет два дополнительных параметра для передачи информации

procedure MakeEntries(const MainTableName: string; doc_id: integer;
    ATransaction: TIBTransaction);
    //перепроводит документ в бухгалтерской системе
    //с помощью вызова хранимой процедуры
    //<MainTableName>_CHANGED(doc_id) на сервере

procedure MakeRegEntries(const MainTableName: string;
    doc_id, doc_n, reg_template_id: integer; ATransaction: TIBTransaction);
    //перепроводит документ в регистрах с помощью вызова хранимой процедуры
    //<MainTableName>_R_CHANGE(doc_id, doc_n, reg_template_id) на сервере

procedure RefreshExplorer;
    //освежает «Проводник документов», если тот на экране

procedure RefreshBalance;
    //освежает «Баланс», если тот на экране
```

```
function SimpleEditDocument(doc_type_id: integer; var doc_id: integer): boolean;
    // вызывает универсальное системное окно редактирования документов

function GetLayerMask(Layer_ID: integer): boolean;
    //возвращает значение маски слоя

procedure SetLayerMask(Layer_ID: integer; Enabled: boolean);
    //устанавливает новое значение маски слоя

procedure StatusBarDisplay(LedColor: TColor; ProgressPosition,
    ProgressMin, ProgressMax: integer; const Text1, Text2, Text3: string);
    //управляет статусной строкой в главном окне

procedure TestTemplateBalance(Template_Id: integer; ShowErrFormPrompt: boolean);
    //запускает виртуальную операцию шаблона для проверки равенства дебета кредиту

function IsAllegroSystemObject(const Name: string): boolean;
    //возвращает True, если Name является именем системного объекта ядра базы данных

function GetEditFormat(const DisplayFormat: string): string;
    //возвращает переданную строку,
    //исключая из нее любые символы кроме #, 0 и точки

function UpdateTurnovers(RefreshBalance: boolean): boolean;
    //вызывает перепроведение всех бухгалтерских
    //шаблонов, у которых установлен флаг IS_CHANGED

function DocTypeSelectDlg(EnableSysDocs: boolean; var Doc_Type_Id: integer): boolean;
    //вызывает диалог выбора типа документа

function NameOfRefObject(Object_ID: integer; const NameType: string): string;
    //возвращает отформатированное наименование объекта из таблицы OBJECT_NAMES

function NameOfDocument(const MainTableName: string; Doc_Id: integer;
    ATransaction: TIBTransaction): string;
    //возвращает наименование документа, получая его вызовом
    //процедуры форматирования имени документа XXX_GET_NAMES

function UserHasGlobalPrivileges: boolean;
    // Возвращает True, если пользователь SYSDBA или
    // владелец (OWNER) текущей базы данных

function GetDocumentContext(Adoc_id: integer): TRunContext;
    //возвращает контекст документа,
    //если тот присуствует на экране и nil, если это не так.

function GetDocumentForm(Adoc_id: integer): TCustomForm;
    //возвращает указатель на форму
    //документа, если она присуствует на экране или nil, если это не так;
    //позволяет получить доступ к объектам любого активного документа.

function AlgExpandRelativePath(const FileName: string): string;
    //прибавляет к FileName путь к файлам текущей конфигурации

procedure PrepareToChangeConfiguration(OwnerRequired: boolean);
    //проверяет готовность к внесению изменений в метаданные

function ExtractSubstring(const S: string; var p: Integer; Ch: Char): string;
    //выделяет подстроку, начиная с позиции p и до того,
    //как встретится символ Ch или конец строки
```

```
procedure ResetCurrentTime;
    //сбрасывает счетчик времени, используется совместно с
    //GetCurrentTimeStr

function GetCurrentTimeStr: string;
    //отображает значение счетчика времени, используется для
    // измерения времени SQL-запросов совместно с ResetCurrentTime

function GetLocalFilePath(const S: string): string;
    //возвращает локальный путь (начиная с диска)
    //к файлу на сервере, выделяя его из сетевого пути

function StrToVarcharFilter(const FieldName, CollateStr, S: string): string;
    //преобразует строку слов, разделенных пробелами,
    //в условие поиска «по одновременному вхождению», например
    //StrToVarcharFilter('TABLE1.NAME','COLLATE PXW_CYRL','Вася Пупкин')
    //вернет такой результат:
    // 'UPPER(TABLE1.NAME COLLATE PXW_CYRL) LIKE %ВАСЯ% AND'#13'+
    // 'UPPER(TABLE1.NAME COLLATE PXW_CYRL) LIKE %ПУПКИН%'

function FileVersion(const FileName: string): string;
    //возвращает версию Allegro в виде строки.

procedure DeleteDocument(const TableName: string; Doc_Id: integer);
    //удаляет документ и все его проводки из таблицы проводок и многомерных регистров.

function GetRowCount(const TableName: string): integer;
    //возвращает количество строк в таблице.

function GetDbVersion(ADatabase: TIBDatabase): integer;
    //возвращает номер версии ядра базы.

function ForceSaldoToReg(acc_id: integer): integer;
    //возвращает счет регистра для счета развернутого
    //сальдо и сам счет для обычного счета.

procedure ConnectAndCheckVersion(ADatabase: TIBDatabase; const DatabaseCaption: string);
    //проверяет версию ядра базы данных и вызывает апдейт ядра до нужной версии.
    //Следует всегда вызывать эту функцию прежде, чем работать
    //с внешней базой данных формата Allegro.

function GetUserRole(ADatabase: TIBDatabase): string;
    //возвращает роль текущего пользователя.

procedure ReconnectIfUserHasRole(ADatabase: TIBDatabase);
    //если у пользователя имеется роль, переподключается к базе с этой ролью.

function CompatibleRoots(acc_id: integer): string;
    //возвращает в виде строки перечисленные через
    //запятую внутренние номера всех регистров счетов,
    //не нарушающих баланс в корреспонденции со счетом acc_id.

function GetFilterExpression(Rubric_id: integer; ATransaction: TIBTransaction;
    InitialValues: TStrings; var rubric_class_id: integer): string;
    //возвращает условие фильтрации
    //и идентификатор id класса по внутреннему идентификатору id рубрики.
```

```
function StoredProcedureExists(const AName: string): boolean;
    //возвращает True, если хранимая процедура с именем AName
    //существует в текущей базе данных

function TableIsEmpty(const TableName: string; ATransaction: TIBTransaction): boolean;
    //возвращает True, если таблица с именем TableName пуста
    //в контексте транзакции ATransaction

procedure ViewRelatedDocuments(const main_table_name: string; Adoc_id: integer);
   //вызывает системное окно «Документов, ссылающихся на данный документ»

function ExecuteFile(const Operation, FileName, Parameters, Directory: string; ShowCmd: Integer): integer;
  //запускает файл средствами операционной системы, (вызывает функцию ShellExecute).
```

Контекст документа RunContext

Контекст документа ***RunContext*** типа ***TRunContext*** – объект, создаваемый программой Allegro при запуске скриптового проекта функциями ***CallDocument*** и ***CallDocumentEx***. Объект ***RunContext*** доступен в скриптовом проекте документа и хранит информацию об ***id*** редактируемого (создаваемого) документа, его типе ***doc_type_id***, папке «Проводника по документам» ***dir_id*** и прочую полезную информацию. Программа Allegro хранит список всех объектов типа ***TRunContext***, связанных с запущенными в данный момент проектами и перед вызовом любого документа на редактирование проверяет, нет ли его в списке уже редактируемых. Если документ уже присутствует в списке, Allegro вместо запуска нового экземляра проекта выводит на передний план главное окно уже запущенного проекта.

```
TRunContext = class
public
  property RunMode: integer;
  property ModuleName: string; //имя модуля проекта

  property ProjectHandle: THandle;
  property ModalResult: TModalResult;
  property dir_id: integer; //папка (используется при создании нового документа)
  property Documents: TDocParams;    //список открытых документов
  property Params: string read FParams write FParams; //строка произвольных параметров
  property parent_doc_type_id: integer; //тип «родительского» документа («документа-основания»)
  property parent_doc_id: integer; //id «родительского» документа («документа-основания»)
end;


TDocParams = class
public
  function Add(doc_type_id, doc_id: integer): TDocParam;  //добавить документ в список
  property Current: TDocParam; //текущий документ

  property Items[Index: integer]: TDocParam;default;  //список документов
  property Count: integer;  //кол-во документов в списке
  procedure Clear; //очистить список документов
  procedure Delete(Index: integer); //удалить элемент списка
  procedure DeleteDoc(doc_id: integer);  // удалить документ из списка
  procedure DeleteAll(doc_type_id: integer); //удалить все документы определенного типа из списка
  procedure SetAsCurrent(doc_id: integer);  //сделать документ текущим
end;

TDocParam = class
public
  property doc_type_id: integer;  //тип документа
  property doc_id: integer;  //документ
end;
```

Встроенные объекты, импортированные из VCL

Allegro содержит встроенные объекты, импортированные из модулей библиотеки VCL Delphi. Это константы, глобальные переменные, классы, идентификаторы перечислимых типов, процедуры и функции. Все они могут использоваться в скриптовых проектах. Полное перечисление этих объектов заняло бы здесь слишком много места, поэтому мы ограничимся списком модулей библиотеки VCL Delphi6, из которых был произведен импорт объектов. Вы можете вызывать любые процедуры и функции из этих модулей.

inifiles, types, variants, actnlist, buttons,
checklst, classes, clipbrd, comctrls, controls,
dateutils, db, bactns,adodb,
dbcommon, dbctrls, dbgrids, dblookup,
dialogs, extctrls, extdlgs, filectrl,
forms, graphics, grids, imglist, mask, math,
menus, messages, mplayer,
olectrls, outline, printers, registry, stdactns,
stdctrls, system, sysutils, tabnotbk, tabs, toolwin

Из модуля Windows импортирована функция SendMessage и константы Virtual Key Codes.

В среде проектирования доступны следующие палитры компонентов:

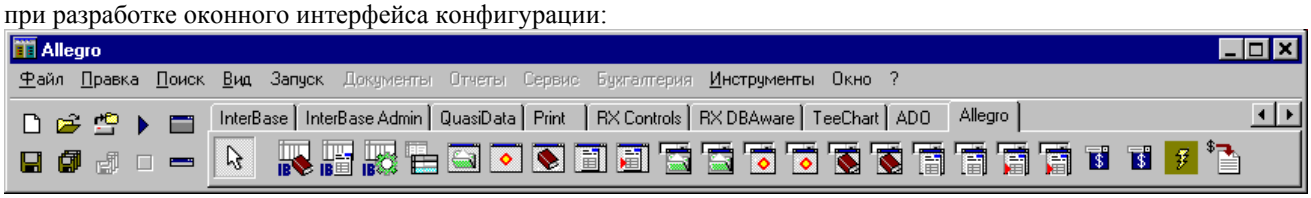
Standard, Additional, Win32, System,
Dialogs, Win 3.1, ADO, Data Controls,
InterBase, InterBase Admin, Quasidata,
Print, RxControls, RX DBAware, Allegro,
Dream Company, Dream Memo, Dream Edit,
Dream Tree, Dream Info Tree.

Некоторые полезные функции, импортированные из VCL

function Pos(Substr: string; S: string): Integer;	Ищет вхождение подстроки Substr в строке S и возвращает позицию, с которой начинается подстрока. Если вхождение не найдено, возвращает 0
function Copy(S; Index, Count: Integer): string;	Возвращает подстроку строки S, начинающуюся с позиции Index и имеющую длину Count символов
procedure Delete(var S: string; Index, *Count:Integer);	Удаляет из строки S символы, количеством Count, начиная с позиции Index
procedure Insert(Source: string; var S: string; Index: Integer);	Вставляет подстроку Source в строку S начиная с позиции Index
function Length(S): Integer;	Возвращает длину строки S в виде целого числа
function Format(const Format: string; const Args: array of const): string;	Форматирует последовательность аргументов в открытом массиве Args. Форматирование управляется строкой Format. Например, все вхождения комбинации символов %s будут заменяться соответствующими строками в массиве, а %d – целыми числами. Подробнее см. в специальной литературе по Delphi.
function FormatDateTime(const Format: string; DateTime: TDateTime): string;	Форматирует дату и время в соответствии с шаблоном. Например, ‘dd.mm.yyyy’. Подробнее см. в литературе по Delphi
function FormatFloat(const Format: string; Value: Extended): string;	Форматирует число нецелого типа при помощи шаблона, например вида ‘#,##0.00’
function DateToStr(Date: TDateTime): string;	Преобразует тип даты в строку соответствии с текущим коротким форматом даты, берущимся из региональных установок Windows при запуске программы
function IntToStr(Value: Int64): string;	Преобразует целое число в строку
function StrToInt(const S: string): Integer;	Преобразует строку в целое число. Если строка не представляет собой правильное представление целого числа, генерируется исключительная ситуация.
function StrToIntDef(const S: string; Default: Integer): Integer;	Преобразует строку в целое число. Если строка не представляет собой правильное представление целого числа, возвращает значение, переданное в параметре Default.
function DayOf(const AValue: TDateTime): Word;	Возвращает день (1…31) месяца из даты
function MonthOf(const AValue: TDateTime): Word;	Возвращает месяц (1…12) из даты
function YearOf(const AValue: TDateTime): Word;	Возвращает год (1…9999) из даты
function WeekOf(const AValue: TDateTime): Word;	Возвращает неделю (1…53) из даты
function EncodeDate(Year, Month, Day: Word): TDateTime;	Возвращает дату, сконструированную из года, месяца и дня, переданных в виде параметров
procedure DecodeDate(Date: TDateTime; var Year, Month, Day: Word);	Декодирует дату, заполняя значениями параметры года, месяца и дня.
function DaysInMonth(const AValue: TDateTime): Word;	Возвращает количество дней в месяце, в который входит дата AValue
function DaysInAMonth(const AYear, AMonth: Word): Word;	Возвращает количество дней в месяце AMonth в году AYear
function IncMonth(const Date: TDateTime; NumberOfMonths: Integer = 1): TDateTime;	Возвращает дату, сдвинутую на NumberOfMonths месяцев относительно Date. Если дата выходит за конец месяца, возвращает дату конца месяца
procedure ShowMessage(const Msg: string);	Выводит текстовое сообщение в простейшем модальном окне с кнопкой OK
function MessageDlg(const Msg: string; DlgType: TMsgDlgType; Buttons: TMsgDlgButtons; HelpCtx: Longint): Word;	Выводит диалоговое окно с сообщением Msg. Слева от сообщения изображается значок, в зависимости от параметра DlgType. В параметре Buttons нужно передать множество кнопок, которые диалог должен отобразить. Диалог возвращает значение, в зависимости от того, какая кнопка была нажата, например, mrOK или mrCancel. Подробнее см. в литературе по Delphi

Палитра компонентов Allegro

Палитра компонентов Allegro содержит специализированные компоненты, которые можно использовать при разработке оконного интерфейса конфигурации:



Эти компоненты можно разделить на несколько групп.

Компоненты доступа к данным (DataSet)

TRefQuery – запрос справочника
TDocQuery – запрос документа
TSettingQuery – запрос таблицы настроек

Усовершенствованная ячеистая сетка

TDBGridA – ячеистая сетка с дополнительными событиями, позволяющими реализовать «поиск по нажатию клавиш» и раскрашивание отдельных ячеек.

Диалоги

TRefDialog – диалог выбора из справочника.
TAccountDialog – диалог выбора счета.
TDocDirDialog – диалог выбора папки.
TDocDialog – диалог выбора из списка документа определенного типа.
TDocItemDialog – диалог выбора строки из подчиненной таблицы документа определенного типа.
TClassSelectDialog – диалог выбора класса справочника

Экранные компоненты

TRefEdit – селектор объекта (оконный элемент управления, вызывающий диалог выбора из справочника)
TDBRefEdit – селектор объекта, работающий как компонент управления данными
TAccountEdit – селектор счета (оконный элемент управления, вызывающий диалог выбора счета)
TDBAccountEdit – селектор счета, работающий как компонент управления данными
TDocDirEdit – селектор папки (оконный элемент управления, вызывающий диалог выбора папки)
TDBDocDirEdit – селектор папки, работающий как компонент управления данными
TDocEdit – селектор документа (оконный элемент управления, вызывающий диалог выбора документа)
TDBDocEdit – селектор документа, работающий как компонент управления данными
TDocItemEdit – селектор позиции документа (экранный элемент управления, вызывающий диалог выбора позиции)
TDBDocItemEdit – селектор позиции документа, работающий как компонент управления данными
TLayerComboBox – селектор слоя (оконный элемент управления)
TDBLayerComboBox – селектор слоя, работающий как компонент управления данными
TClassSelectEdit – селектор класса (оконный элемент управления, вызывающий диалог класса справочника)
TDBClassSelectEdit – селектор класса, работающий как компонент управления данными

Дерево рубрик

TRubricTreeGrid – компонент для навинации по дереву рубрик

Компонент для запроса балансов

TBalance – компонент запроса балансов

Компонент для преобразования чисел в текст

TCurrencyInWords – компонент для преобразования чисел в «Сумму прописью»

Компонент запроса справочника TRefQuery

Свойства Published	
ClassTableName	Имя таблицы класса
DetailFieldName	Если справочник подчинен другому справочнику – имя поля, по которому производится подчинение (фильтрация по ключевому значению)
DisableChildClass	Не отображать объекты классов-потомков
DisableEmptyValue	Не отображать пустой элемент
EnablePost	Сохранять результаты редактирования
MasterFieldName	Имя поля в старшем справочнике при подчинении одного справочника другому (обычно поле первичного ключа). Должно быть MASTER_VALUE, если вместо старшего справочника используется явное значение из поля MasterValue
MasterValue	Явное значение ключа для подчинения справочника, если не указан старший справочник в свойстве DataSource
MaxDisplayWidth	Максимальная ширина поля при отображении (в символах)
NamesType	Способ отображения: ATTRIBUTES – поля справочника SHORT_NAME –краткие наименования объектов Возможны и другие значения, если в конфигурации имеются дополнительные типы наименований объектов.
ShowLookupNames	Показывать все поля справочника Действует, если свойство NamesType не равно ATTRIBUTES
Transaction	Транзакция

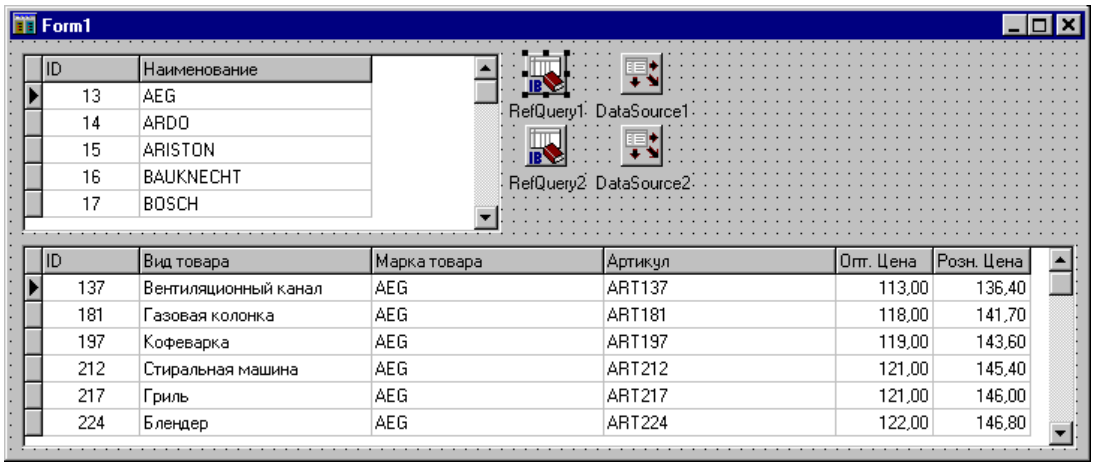
Свойства Public	**тип**	**readonly**	
Class_Id	integer		ID класса
Class_Name	string	readonly	Имя таблицы класса
FilterExpression	string		Выражение для фильтра, добавляемое в WHERE
ID_Field	TField	readonly	Компонент поля ID
Object_Id	integer		ID объекта при SelectOneObject = True
RootTableName	string	readonly	Имя таблицы корневого класса
SelectOneObject	boolean		Запрашивать только один объект

Методы	
procedure SetOrderByAlias(const alias_name: string; desc: boolean);	Упорядочивание по псевдониму поля
procedure BuildSQL	Создает тексты внутренних SQL-запросов

В большинстве случаев вместо компонентов TRefQuery лучше использовать обычные компоненты SQL-запросов TIBDataSet. Компонент TRefQuery предназначен в основном для тех ситуаций, если требуется быстро реализовать отображение какого-то справочника в сетке или же если необходимо организовать создание или редактирование элементов справочников. Если свойство EnablePost = False, то справочные элементы «редактируются», но эти изменения не запоминаются. Если свойство EnablePost = True, то изменения запоминаются, причем без возможности «отката транзакции», так как сама посылка данных на сервер осуществляется этим компонентом в отдельной, недоступной транзакции.

Пример использования компонентов справочников в режиме дизайна показан на рисунке.

Здесь справочник GOODS подчинен справочнику GOODS_MARK по полю GOODS_MARK.



Компонент запроса таблицы документа TDocQuery

Свойства Published	
Doc_ID	ID документа. Присвоив значение, запрос нужно переоткрыть, если он был открыт
MaxDisplayWidth	Максимальная ширина поля при отображении (в символах)
TableName	Имя таблицы документа. Может быть именем основной таблицы или подчиненной
Transaction	Транзакция

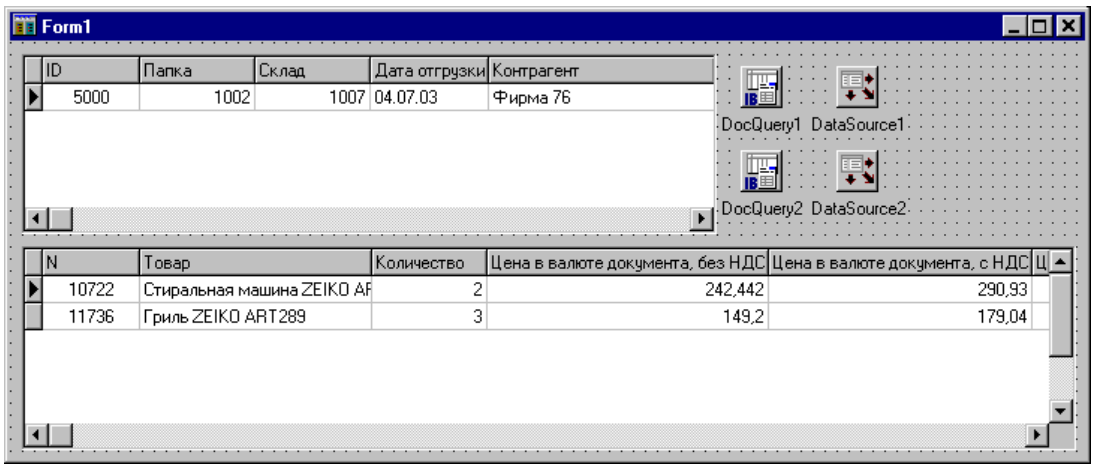
Свойства Public	**тип**	**readonly**	
IsDetail	boolean	readonly	Признак того, что это подчиненная таблица документа

Методы	
procedure BuildSQL	Создает тексты внутренних SQL-запросов

В большинстве случаев вместо компонентов TDocQuery лучше использовать обычные компоненты SQL-запросов TIBDataSet. Компонент TDocQuery предназначен в основном для тех ситуаций, если требуется быстро реализовать отображение какого-то отдельного документа.

Пример использования компонентов TDocQuery в режиме дизайна показан на рисунке.

Здесь используются два компонента. Один для запроса шапки документа из таблицы SALE, другой – для запроса позиций документа из таблицы SALE_ITEM:



Компонент запроса таблицы настроек TSettingQuery

Свойства Published	
MaxDisplayWidth	Максимальная ширина поля при отображении (в символах)
TableName	Имя таблицы настроек.
Transaction	Транзакция

Методы	
procedure BuildSQL	Создает тексты внутренних SQL-запросов

В большинстве случаев вместо компонентов TSettingQuery лучше использовать обычные компоненты SQL-запросов TIBDataSet. Компонент TSettingQuery предназначен в основном для тех ситуаций, если требуется быстро реализовать отображение какой-то таблицы настроек.

Диалог выбора из справочника TRefDialog

Этот компонент рекомендуется использовать в тех случаях, когда необходимо в процессе работы программы вызвать стандартный диалог выбора из справочника.

Свойства Published	
ClassButtons	Свойство определяет множество элементов управления, которые будут доступны в справочнике. Имеет тип TRefQueryButtons
ClassTableName	Название таблицы класса
NamesType	Способ отображения: ATTRIBUTES – поля справочника SHORT_NAME –краткие наименования объектов Возможны и другие значения, если в конфигурации имеются дополнительные типы наименований объектов.
Options	Опции диалога. Имеет тип TRefSelectDlgOptions
RootRubric_ID	Идентификатор корневой рубрики. Если значение этого свойства отлично от нуля – в окне диалога слева высвечивается дерево рубрик, начиная с этой рубрики.
SubClassButtons	Свойство определяет множество элементов управления, которые будут доступны в подчиненном справочнике, если тот отображается. Имеет тип TRefQueryButtons
SubClassTableName	Название таблицы починенного класса
Title	Заголовок для окна диалога.
Transaction	Транзакция
WindowOptions	Оконные опции. Имеет тип TWindowOptions

Свойства Public	**тип**	**readonly**	
Object_ID	integer	readonly	ID элемента, выбранного из справочника.
SubObject_ID	integer	readonly	ID элемента, выбранного из подчиненного справочника.

Методы	
function Execute: boolean	Вызывает диалог на экран. Возвращает True, если пользователь выбрал справочный элемент в диалоге и False, если он нажал кнопку «Отмена».
procedure CloseQueries	Закрывает все открытые внутренние запросы.

Объявления некоторых типов данных, используемых в свойствах этого компонента показаны ниже:

```
TRefSelectDlgOption  = (doValueRequired, doSubValueRequired, doShowSubmitted,
   doRestrictClass, doRestrictSubClass, doHideLookups, doHideSubLookups,
   doRelocateRecord, doUseWindowOptions);

TRefSelectDlgOptions = set of TRefSelectDlgOption;

{doValueRequired  - требуется выбрать значение в справочнике
 doSubValueRequired - требуется выбрать значение в подчиненном справочнике
 doShowSubmitted – показывать подчиненные справочники
 doRestrictClass – ограничить класс справочника (не отображать дочерние)
 doRestrictSubClass – ограничить класс подчиненного справочника (не отображать дочерние)
 doHideLookups – не высвечивать дополнительные поля в режиме отображения наименований
 doHideSubLookups - не высвечивать дополнительные поля подчиненного класса
 doRelocateRecord – осуществлять поиск «текущего» элемента при повторном открытии диалога
 doUseWindowOptions – использовать оконные опции

TRefQueryButton = (rqbInsert, rqbDelete, rqbEdit, rqbRefresh, rqbFilter, rqbSubmit, rqbNamesType);

TRefQueryButtons = set of TRefQueryButton;

{rqbInsert – отображать кнопку «Добавить»,
 rqbDelete – отображать кнопку «Удалить»,
 rqbEdit – отображать кнопку «Изменить»,
 rqbRefresh – отображать кнопку «Освежить»,
 rqbFilter – отображать кнопку «Фильтр»,
 rqbSubmit – отображать кнопку «Показывать подчиненные»,
 rqbNamesType – отображать селектор «Режим» (атрибуты, краткое наименование и т.п.)}

TWindowOptions = class(TPersistent)
public
  constructor Create;
published
  property Height: integer read FHeight write FHeight;
  property Width: integer read FWidth write FWidth;
  property Left: integer read FLeft write FLeft;
  property Top: integer read FTop write FTop;
  property Position: TPosition read FPosition write SetPosition;
end;
```

Использовать компонент TRefDialog несложно. Достаточно в Инспекторе объектов указать транзакцию и установить свойство ClassTableName, выбрав имя таблицы из выпадающего списка. Затем в программе следует вызвать метод Execute, а затем прочитать значение свойства Object_ID. Например, так:

```
with RefDialog1 do
if Execute then
  ShowMessage(IntToStr(Object_ID));
```

Еще проще использовать оконные компоненты – селекторы объектов TRefEdit и TDBRefEdit, которые вызывают тот же диалог выбора из справочника.

Селекторы объекта TRefEdit и TDBRefEdit

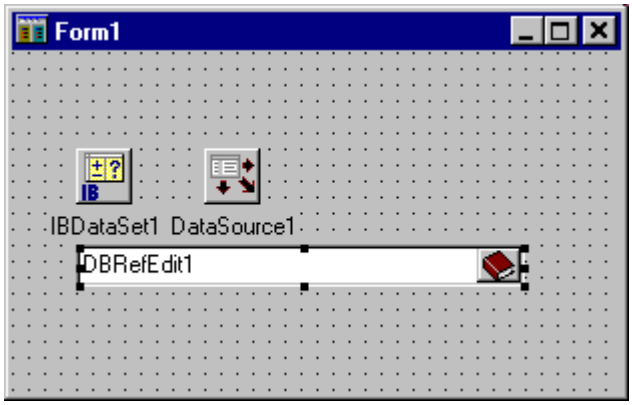
Селекторы объекта являются оконными элементами управления (контролами). Компонент TDBRefEdit является полной копией компонента TRefEdit с той лишь разницей, что он может управлять данными в компонентах типа IBDataSet, IBQuery и им подобных компонентах доступа к данным.

Компоненты всегда отображают текущее краткое наименование справочного элемента, соответствующего свойству Object_ID. Компоненты имеют также свойство SubObject_ID, которое принимает значение из подчиненного справочника, если тот отображается в диалоге.

Компоненты TRefEdit и TDBRefEdit имеют все те же свойства, что и компонент TRefDialog, кроме свойства Title, которое у этих компонентов называется DialogTitle. Кроме того они имеют массу свойств, свойственных обычным оконным компонентам (ширину, координаты и т.п.), на которых мы останавливаться здесь не будем. Оба этих компонента вызывают диалог выбора из справочника при нажатии пользователем кнопки с изображением символа справочника или при нажатии комбинации клавиш Alt+стрелка вниз.

Компонент TDBRefEdit имеет несколько дополнительных свойств для подключения к источнику данных:

Свойства Published	
DataField	Имя поля в наборе данных
DataSource	Компонент-источник данных типа TDataSource (через который происходит подключение к компоненту доступа к данным)
ShortNameField	Имя поля краткого наименования. Используется только если такое поле есть в наборе данных во избежание лишнего запроса краткого наименования для отображения.



Диалог выбора счета TAccountDialog

Этот компонент рекомендуется использовать в тех случаях, когда необходимо в процессе работы программы вызвать стандартный диалог выбора счета.

Свойства Published	
Options	Опции диалога, тип TAccountDialogOptions
Roots	Список ACC_ID регистров, которыми следует ограничить выбор счета. Перечисляются внутренние номера регистров через запятую. Если свойство пустое – отображаются все регистры всех компаний.
Title	Заголовок для окна диалога.
Transaction	Транзакция

Свойства Public	**тип**	**readonly**	
Acc_ID	integer		ACC_ID выбранного счета.

Методы	
function Execute: boolean	Вызывает диалог на экран. Возвращает True, если пользователь выбрал счет в диалоге и False, если он нажал кнопку «Отмена».

Объявления некоторых типов данных, используемых в свойствах этого компонента показаны ниже:

```
TAccountDialogOption = (doDisableFolderSelect, doForceSaldoToReg);

{ doDisableFolderSelect – запрет выбора счетов-папок
  doForceSaldoToReg – преобразовывать ACC_ID счетов сальдо
                      в счета представленных ими регистров}

TAccountDialogOptions = set of TAccountDialogOption;
```

Использовать компонент TAccountDialog несложно. Достаточно в Инспекторе объектов указать транзакцию. Затем в программе следует вызвать метод Execute, а затем прочитать значение свойства Acc_ID. Например, так:

```
with AccountDialog1 do
if Execute then
  ShowMessage(IntToStr(ACC_ID));
```

Еще проще использовать оконные компоненты – селекторы счетов TAccountEdit и TDBAccountEdit, которые вызывают тот же диалог выбора счета.

Селекторы счета TAccountEdit и TDBAccountEdit

Селекторы счетоа являются оконными элементами управления (контролами). Компонент TDBAccountEdit является полной копией компонента TAccountEdit с той лишь разницей, что он может управлять данными в компонентах типа IBDataSet, IBQuery и им подобных компонентах доступа к данным.

Компоненты всегда отображают наименование счета, соответствующего свойству Acc_ID и слева значок компании, к которой относится счет.

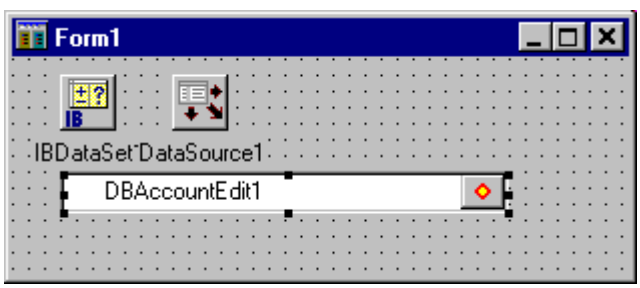
Компоненты TAccountEdit и TDBAccountEdit имеют все те же свойства, что и компонент TAccountDialog, кроме свойства Title, которое у этих компонентов называется DialogTitle. Эти компоненты имеют еще два дополнительных свойства, которых нет у компонента TAccountDialog

Свойства Public	**тип**	**readonly**	
Acc_Code	string	readonly	Код выбранного счета.
Company_ID	integer	readonly	COMPANY_ID выбранной компании

Кроме того они имеют массу свойств, свойственных обычным оконным компонентам (ширину, координаты и т.п.), на которых мы останавливаться здесь не будем. Оба этих компонента вызывают диалог выбора счета при нажатии пользователем кнопки с изображением символа счета или при нажатии комбинации клавиш Alt+стрелка вниз.

Компонент TDBAccountEdit имеет несколько дополнительных свойств для подключения к источнику данных:

Свойства Published	
DataField	Имя поля в наборе данных
DataSource	Компонент-источник данных типа TDataSource (через который происходит подключение к компоненту доступа к данным)



Диалог выбора папки TDocDirDialog

Этот компонент рекомендуется использовать в тех случаях, когда необходимо в процессе работы программы вызвать стандартный диалог выбора папки «Проводника по документам».

Свойства Published	
Title	Заголовок для окна диалога.
Transaction	Транзакция

Свойства Public	**тип**	**readonly**	
Dir_ID	integer		DIR_ID выбранной папки.
DirName	string	readonly	Название выбранной папки

Методы	
function Execute: boolean	Вызывает диалог на экран. Возвращает True, если пользователь выбрал папку в диалоге и False, если он нажал кнопку «Отмена».

Использовать компонент TDocDirDialog несложно. Достаточно в Инспекторе объектов указать транзакцию. Затем в программе следует вызвать метод Execute, а затем прочитать значение свойства Dir_ID. Например, так:

```
with DocDirDialog1 do
if Execute then
  ShowMessage(IntToStr(Dir_ID));
```

Еще проще использовать оконные компоненты – селекторы папок TDocDirEdit и TDBDocDirEdit, которые вызывают тот же диалог выбора папки.

Селекторы папки TDocDirEdit и TDBDocDirEdit

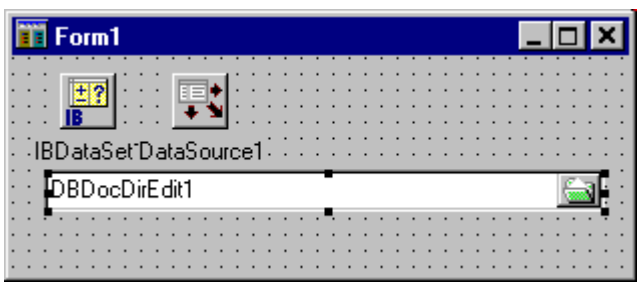
Селекторы папки являются оконными элементами управления (контролами). Компонент TDBDocDirEdit является полной копией компонента TDocDirEdit с той лишь разницей, что он может управлять данными в компонентах типа IBDataSet, IBQuery и им подобных компонентах доступа к данным.

Компоненты всегда отображают полный путь к папке, соответствующей свойству Dir_ID.
Компоненты TDocDirEdit и TDBDocDirEdit имеют все те же свойства, что и компонент TDocDirDialog, кроме свойства Title, которое у этих компонентов называется DialogTitle.

Кроме того они имеют массу свойств, свойственных обычным оконным компонентам (ширину, координаты и т.п.), на которых мы останавливаться здесь не будем. Оба этих компонента вызывают диалог выбора папки при нажатии пользователем кнопки с изображением символа папки или при нажатии комбинации клавиш Alt+стрелка вниз.

Компонент TDBDocDirEdit имеет несколько свойств для подключения к источнику данных:

Свойства Published	
DataField	Имя поля в наборе данных
DataSource	Компонент-источник данных типа TDataSource (через который происходит подключение к компоненту доступа к данным)



Селекторы слоя TLayerComboBox, TDBLayerComboBox

Селекторы слоя являются TLayerComboBox, TDBLayerComboBox обычными выпадающими списками, потомками класса TCustomComboBox и кроме обычных свойств оконных выпадающих списков имеют дополнительные свойства:

Свойства Published	
Transaction	Транзакция

Свойства Public	**тип**	**readonly**	
Layer_ID	integer		ID слоя
Text	string		краткое имя слоя

Методы	
procedure LoadItems	Вызывает загрузку списка слоев из кеша метаданных

Компонент TDBLayerComboBox предназначен для работы с наборами данных (компонентами запросов) и поэтому имеет дополнительные свойства:

Свойства Published	
DataField	Имя поля в наборе данных
DataSource	Компонент-источник данных типа TDataSource (через который происходит подключение к компоненту доступа к данным)

Методы	
function GetField: TField;	Возвращает компонент поля в наборе данных, с которым установлена связь

Для использования селектора слоя TLayerComboBox достаточно установить его свойство Transaction.

Для использования селектора слоя TDBLayerComboBox его нужно подключить к набору данных через компонент TDataSource и назначить поле DataField в Инспекторе объектов.

Разумеется, для отображения списка слоев можно воспользоваться и обычными компонентами TDBLookupComboBox или RxDBLookupCombo в сочетании с TIBQuery и TDataSource, обращаясь к системной таблице слоев при помощи SQL-запроса. Преимуществом использования TLayerComboBox и TDBLayerComboBox является то, что эти компоненты не обращаются каждый раз к базе данных, а загружают список слоев из временного кеша.

Диалог выбора документа TDocDialog

Этот компонент позволяет вызвать примитивный диалог выбора документа из списка документов определенного типа. Его не рекомендуется применять в конечных конфигурациях, применяйте этот компонент в качестве временного компонента, позволяющего быстро организовать выбор документа из спика в целях отладки.

Свойства Published	
DocTableName	Название главной таблицы документа
Title	Заголовок для окна диалога.
Transaction	Транзакция

Свойства Public	**тип**	**readonly**	
Doc_ID	integer		ID выбранного документа
DocName	string	readonly	Отформатированное наименование выбранного документа

Методы	
function Execute: boolean	Вызывает диалог на экран. Возвращает True, если пользователь выбрал документ в диалоге и False, если он нажал кнопку «Отмена».

Диалог выбора позиции документа TDocItemDialog

Этот компонент позволяет вызвать примитивный диалог выбора одной позиции документа из списка документов определенного типа. Его не рекомендуется применять в конечных конфигурациях, применяйте этот компонент в качестве временного компонента, позволяющего быстро организовать выбор позиции документа из спика в целях отладки.

Свойства Published	
DocTableName	Название подчиненной таблицы документа
Title	Заголовок для окна диалога.
Transaction	Транзакция

Свойства Public	**тип**	**readonly**	
Doc_ID	integer		ID выбранного документа
DocName	string	readonly	Отформатированное наименование выбранного документа
N	integer		Ключевое значение N выбранной позиции
DocItemName	string	readonly	Наименование позиции, состоящее из номера позиции в квадратных скобках и отформатированного наименования выбранного документа

Методы	
function Execute: boolean	Вызывает диалог на экран. Возвращает True, если пользователь выбрал документ в диалоге и False, если он нажал кнопку «Отмена».
MainTableName	Возвращает название главной таблицы документа

Селекторы документа TDocEdit, TDBDocEdit

Селекторы документа TDocEdit, TDBDocEdit являются примитивными селекторами для выбора документа из списка документов определенного типа. Их не рекомендуется применять в конечных конфигурациях, применяйте эти селекторы в качестве временных компонентов, позволяющих быстро организовать выбор документа из списка в целях отладки.

Свойства Published	
DialogTitle	Заголовок окна диалога
DocTableName	Имя главной таблицы документа
Transaction	Транзакция

Свойства Public	**тип**	**readonly**	
Doc_ID	integer		ID документа

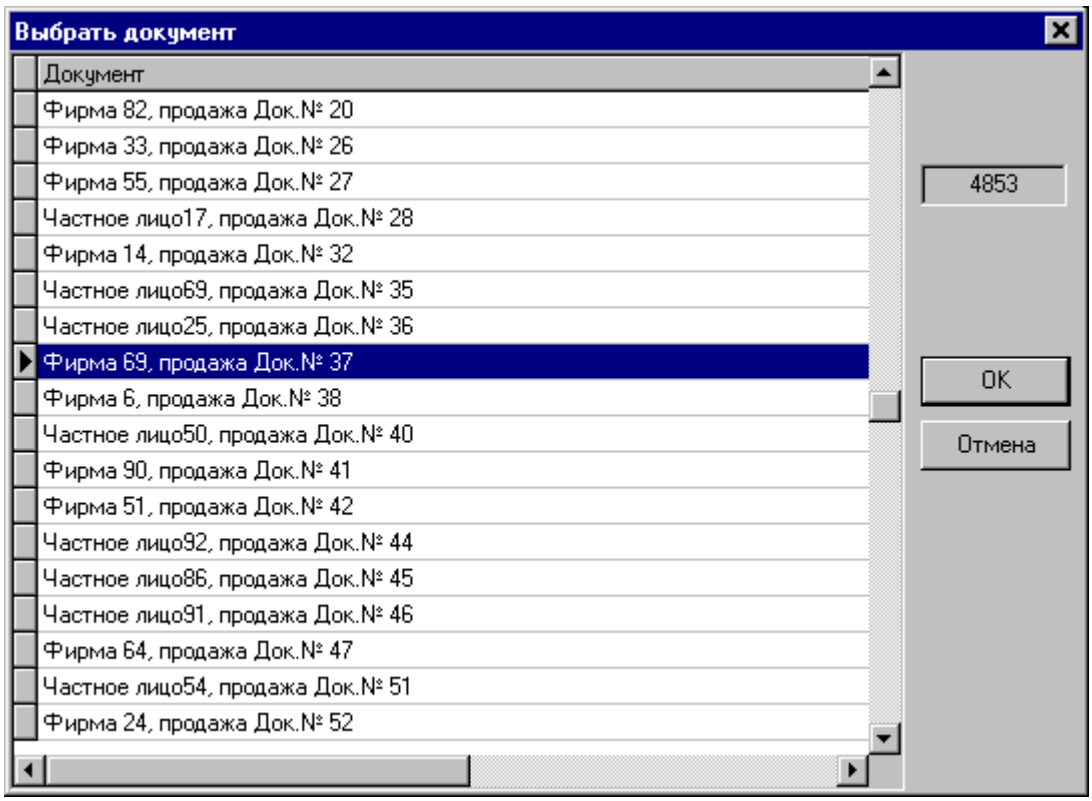
Селектор TDBDocEdit может работать с компонентами доступа к данным типа TIBDataSet, TIBQuery и имеет дополнительные свойства:

Свойства Published	
DataField	Имя поля в наборе данных

DataSource	Компонент-источник данных типа TDataSource (через который происходит подключение к компоненту доступа к данным)

Методы	
function GetField: TField;	Возвращает компонент поля в наборе данных, с которым установлена связь

Для использования TDocEdit нужно установить в Инспекторе объектов свойства Transaction и DocTableName. Когда пользователь нажмет на кнопку со значком документа или комбинацию клавиш Alt+стрелка вниз, появится диалог выбора документа:



Селекторы позиции документа TDocItemEdit, TDBDocItemEdit

Селекторы позиции документа TDocItemEdit, TDBDocItemEdit являются примитивными селекторами для выбора позиции документа из списка документов определенного типа. Их не рекомендуется применять в конечных конфигурациях, применяйте эти селекторы в качестве временных компонентов, позволяющих быстро организовать выбор позиции документа из списка в целях отладки.

Свойства Published	
DialogTitle	Заголовок окна диалога
DocTableName	Имя подчиненной таблицы документа
Transaction	Транзакция

Свойства Public	**тип**	**readonly**	
N	integer		ключевое значение N позиции документа

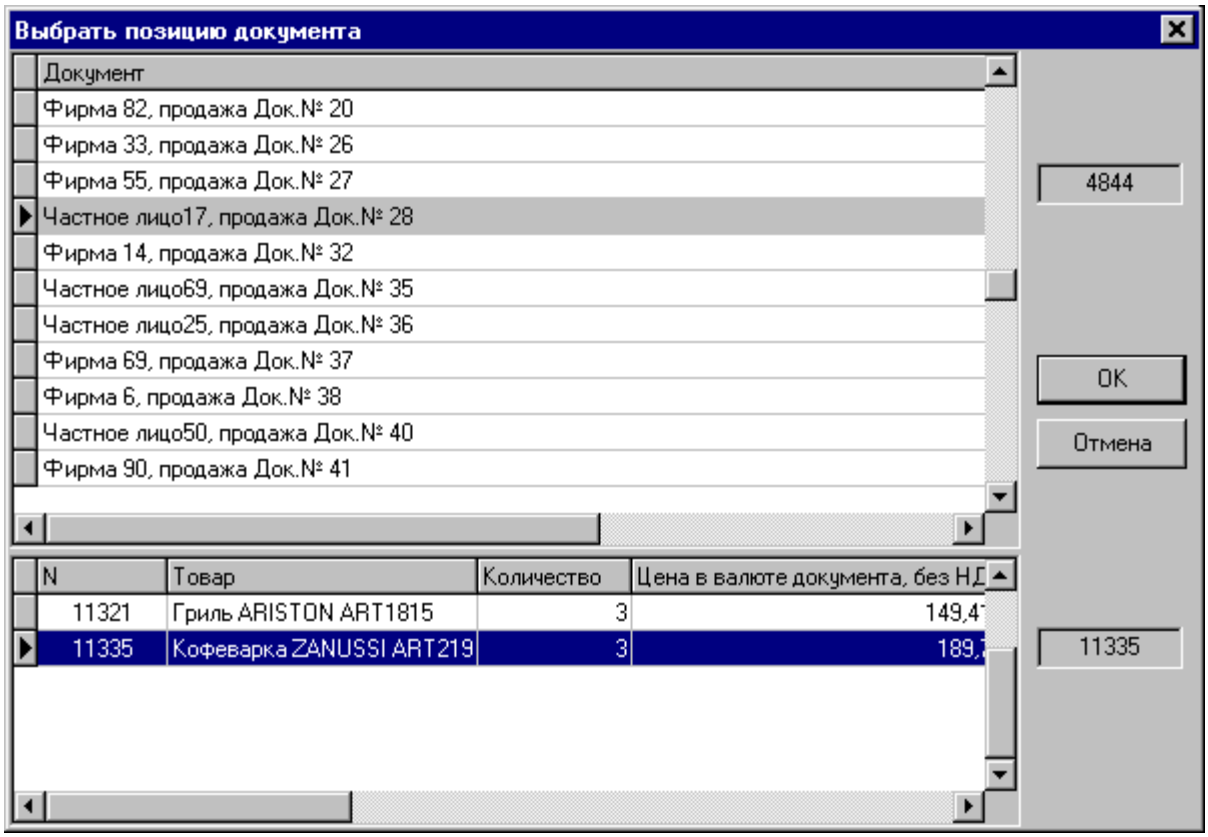
Селектор TDBDocEdit может работать с компонентами доступа к данным типа TIBDataSet, TIBQuery и имеет дополнительные свойства:

Свойства Published	
DataField	Имя поля в наборе данных
DataSource	Компонент-источник данных типа TDataSource (через который происходит

	подключение к компоненту доступа к данным)

Методы	
function GetField: TField;	Возвращает компонент поля в наборе данных, с которым установлена связь

Для использования TDocItemEdit нужно установить в Инспекторе объектов свойства Transaction и DocTableName. Когда пользователь нажмет на кнопку со значком выбора позиции документа или комбинацию клавиш Alt+стрелка вниз, появится диалог выбора позиции документа:



Ячеистая сетка TDBGridA

Компонент TDBGridA является потомком компонента TCustomDbAltGrid и поэтому имеет все те же свойства, события и методы, что и компонент TDbAltGrid от Quasidata.

дополнительные события	**Тип события**	**Назначение**
OnGetEditText	TGetEditEvent	Происходит в момент появления InplaceEditor-а
OnSetEditText	TSetEditEvent	Происходит каждый раз после нажатия клавиши в режиме редактирования в InplaceEditor-е сетки. Используется для реализации «поиска по нажатию»
OnSelectCell	TSelectCellEvent	Происходит в момент выбора ячейки
OnGetCellParams	TGetCellParamsEvent	Происходит при рисовании ячеек. Позволяет раскрашивать любые ячейки в разные цвета в процессе работы программы. Тип события позаимствован из библиотеки RX. аналогичное событие имеется в сетке TRxDBGrid

```
TGetEditEvent = procedure (Sender: TObject; ACol, ARow: Longint; var Value: string) of object;
TSetEditEvent = procedure (Sender: TObject; ACol, ARow: Longint; const Value: string) of object;
TSelectCellEvent = procedure (Sender: TObject; ACol, ARow: Longint; var CanSelect: Boolean) of object;
TGetCellParamsEvent = procedure (Sender: TObject; Field: TField;
                                 AFont: TFont; var Background: TColor; Highlight: Boolean) of object;
```

Таким образом, компонент TDBGridA наследует все преимущества ячеистой сетки с палитры Quasidata, в которой возможно многострочное расположение полей, расширяя ее рядом дополнительных полезных событий.

Подробнее о свойствах, событиях и методах самого компонента TDbAltGrid от Quasidata можно узнать на сайте производителя компонента:

http://www.quasidata.com/

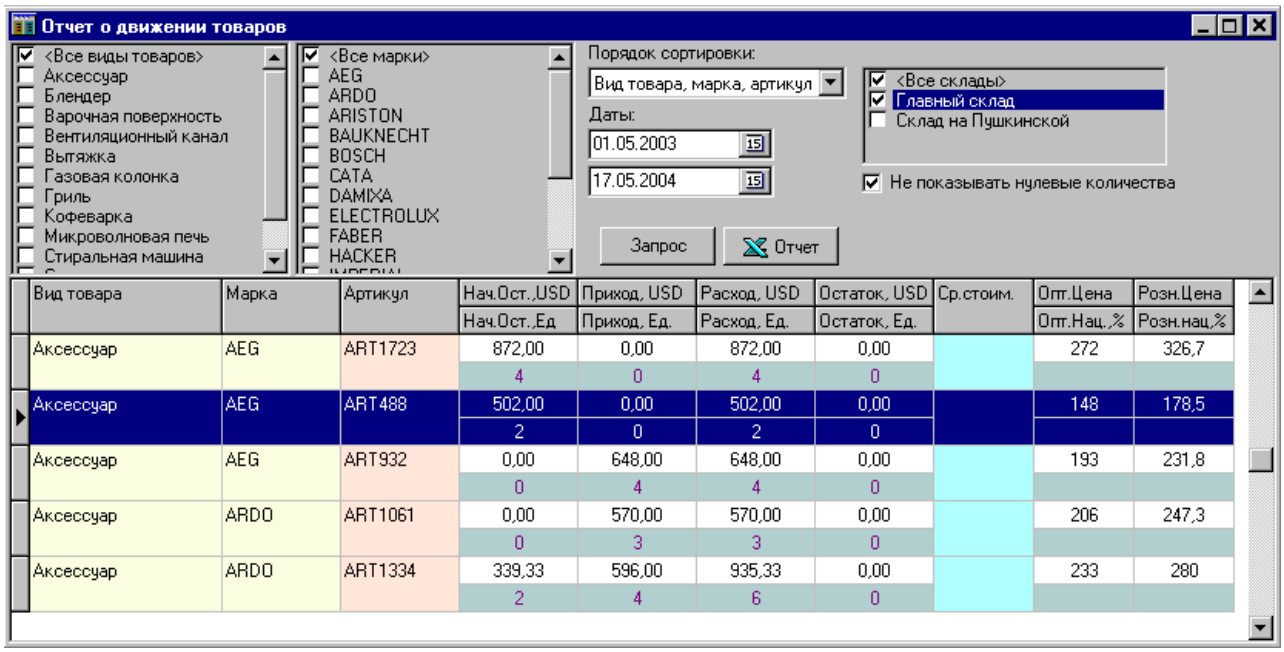
Краткое описание компонента DbAltGrid

http://www.quasidata.com/dbaltgrid.html

Часто задаваемые вопросы по компоненту DbAltGrid

http://www.quasidata.com/faq.html

При помощи ячеистых сеток можно создавать отчеты наподобие того, что показан на рисунке:



Компонент запроса балансов TBalance

Этот компонент используется самой системой Allegro для запроса ***Баланса компании*** и ***Разверток***. Вы можете также с успехом использовать этот компонент для своих целей, например, для построения косвенных отчетов на основе балансовых показателей.

Свойства Published	
ObjectBalance	Имеет значение при запросе балансов аналитических регистров. Если установлено True, то баланс запрашивается для аналитического объекта, предварительно указанного в свойстве Object_id. Если False, то баланс запрашивается для всех объектов.
Transaction	Транзакция

События	
OnProgress	TProgessEvent = procedure(Sender: TObject; const Position, Max: Integer) of object;

Свойства Public	**тип**	**readonly**	
Dates	свойство-массив элементов типа TDateTime	readonly	при вызове метода QueryBalance для каждого элемента массива будет запрошен отдельный баланс на дату, указанную в этом элементе.
DateCount	integer	readonly	Количество элементов в свойстве-массиве Dates

Object_id	integer		ID аналитического объекта, используется для запроса баланса аналитического регистра для одного объекта

Методы, подготовки запросов	
function AddDate(Value: TDateTime): integer;	Добавляет значение в свойство-массив дат Dates. Возвращает номер добавленного элемента.
procedure ClearTotals(acc_id: integer);	Освобождает внутренний буфер, в котором хранятся результаты запроса итогов, сгруппированных по объектам счета acc_id
procedure ClearAllTotals;	Освобождает все буферы, в которых хранятся результаты итогов, сгруппированных по объектам
procedure Clear;	Очищает все буферы и массив дат Dates
function SaldoToReg(Acc_id: integer): integer;	возвращает по acc_id сальдо-счета acc_id его регистра

Методы, запрашивающие данные	
procedure QueryRates;	Запрашивает курсы слоев, ближайшие к датам в массиве Dates
procedure QueryBalance;	Запрашивает балансы на даты, хранящиеся в массиве Dates
procedure QueryTotals(acc_id: integer; differential: boolean);	Запрашивает итоги по объектам счета acc_id на даты, хранящиеся в массиве Dates. Если differential=True, то запрашиваются дифференциальные итоги (разницы между итогами на соседние даты в Dates)

Методы для чтения данные из внутренних буферов компонента	
function RateOfExchange(Index, layer_id: integer): Extended;	
function CheckRates(layer_id: integer): integer;	Проверяет, присутствуют ли в результатах запроса курсов курсы слоя layer_id на все даты, хранящиеся в Dates. Если все в порядке, возвращается –1. Если какой-то курс отсутствует или равен нулю, то возвращается его индекс
5 методов для чтения из внутреннего буфера готовых результаты запроса для элемента массива дат с индексом Index, для счета acc_id, слоя layer_id, типа сальдо saldo_kind. Если SumLayers = True, то суммирует все слои, конвертируя по кросс-курсам в валюту слоя layer_id. Параметр saldo_kind = 0 используется для счетов дебетового развернутого сальдо, saldo_kind = 1 для счетов кредитового развернутого сальдо, а saldo_kind = -1 для всех обычных счетов	.
function Debit(Index, acc_id, layer_id, saldo_kind: integer; SumLayers: boolean): Extended;	Значение дебет-оборота от начала периода.
function Credit(Index, acc_id, layer_id, saldo_kind: integer; SumLayers: boolean): Extended;	Значение кредит-оборота от начала периода.
procedure DebitAndCredit(Index, acc_id, layer_id, saldo_kind: integer; SumLayers: boolean; var ADebit, ACredit: Extended);	Значение дебет и кредит-оборотов от начала периода.
function Balance(Index, acc_id, layer_id, saldo_kind, root_saldo: integer; SumLayers: boolean): Extended;	Значение остатка по счету. Используйте root_saldo = 1 для «правых» счетов. Если параметр root_saldo = 1, то возвращается разница кредит-оборот минус дебет-оборот от начала периода. Если параметр root_saldo <> 1, то возвращается дебет-оборот минус кредит-оборот от начала периода.

procedure Totals(Index, acc_id, layer_id, saldo_kind, aobject_id: integer; SumLayers: boolean; var ADebit, ACredit: Extended);	Значение итогов для конкретного объекта object_id на счете acc_id. Во внутреннем буфере хранятся результаты запроса для всех объектов этого счета.

Последовательность работы с компонентом TBalance следующая.

Подготовка:
1. Очистить все буферы методом Clear,
2. Подготовить массив дат вызовами метода AddDate,
3. Запросить курсы всех слоев метода QueryRates,
4. Проверить наличие курсов для каждого слоя в отдельности вызовами метода CheckRates,

Далее,
Если нужно получить баланс(ы) :
1. Запросить все балансы методом QueryBalance,
2. Считать результаты какими-нибудь методами чтения, например, DebitAndCredit

Если нужно получить запрос итогов, сгруппированных по объектам, то для каждого счета следует:
1. Очистить буфер методом ClearTotals
2. Вызвать QueryTotals для конкретного счета
3. Считать результаты вызовами метода Totals
4. Желательно потом очистить буфер методом ClearTotals, чтобы не занимать лишнюю память.

Для работы с компонентом TBalance нужно *заранее* иметь список счетов и *объектов*, для которых Вы хотите получить финансовые данные. Для этой цели конфигурирующий может использовать обычные запросы к системным таблицам.

Как правило компонент TBalance используется для построения косвенных отчетов на основе данных из баланса, например, ***Отчета о движении денежных средств*** или ***Отчета о движении капитала***.

Компонент для преобразования чисел в текст TCurrencyInWords

Компонент TCurrencyInWords используется для представления чисел в виде текста. Это часто бывает нужно при формировании «суммы прописью» для финансовых документов. Компонент использует правила склонения числительных русского языка, достаточно гибок и прост в употреблении.

Свойства Published	**тип**	Назначение	Значение по умолчанию
Cent1	string	правило 1 для центов	копейка
Cent2_4	string	правило 2-4 для центов	копейки
Cent5_20	string	правило 5-20 для центов	копеек
CentGender	TGender	грамматический род центов	gFeminine
CentsInWords	boolean	выводить центы прописью	
Curr1	string	правило 1 для валюты	рубль
Curr2_4	string	правило 2-4 для валюты	рубля
Curr5_20	string	правило 5-20 для валюты	рублей
CurrGender	TGender	грамматический род валюты	gMasculine
UseCents	boolean	использовать центы	True
UseFinalPoint	boolean	в конце ставить точку	
Value	Extended	**числовое значение**	**0**

Свойства Public	**тип**	**readonly**	
Text	string	readonly	**текстовое значение «прописью»**

Как видно из самих названий свойств, для задания правил склонения любых валют на русском языке достаточно ответить на ряд вопросов. Допустим, мы склоняем слово «доллар США». Ответив на вопросы

- 1 что? – доллар США

- 2-4 чего? – доллара США
- 5-20 чего? – долларов США
- Доллар США это он или она? – он (gMasculine)

можно установить нужные способы склонения для любой валюты. Способы склонения универсальны и их моно применить даже к таким видам «валюты», как «штука» или «литр».

Свойства можно установить в инспекторе объектов или во время выполнения программы. Для преобразования чисел в текст нужно свойству Value в процессе выполнения программы присвоить числовое значение, а из свойства Text считать готовое значение «прописью».

Компонент глобальных событий TAllegroEvents

Компонент TAllegroEvents используется для обработки глобальных событий в системе Allegro. В основном этот компонент предназначен для работы в составе *глобальных модулей* конфигурации. Эти модули бывают загружены постоянно.

Свойства Published	**тип**	**Назначение**
EventNames	string	список имен всех событий, на которые должен среагировать компонент, указанные через точку с запятой

События	**тип**	**Назначение**
OnEvent	TAllegroEvent	происходит при возникновении глобального события с именем, которое присутствует в свойстве EventNames

```
TAllegroEvent = procedure(const EventName: string;
   InputParams: variant; var OutputParams: variant) of object;
```

Если на событие OnEvent назначен обработчик, то при вызове этого обработчика в него передаются:

- имя события (string)
- входные параметры (Variant или Variant array)
- выходные параметры (Variant или Variant array)

Вызвать глобальное событие можно с помощью процедуры ***CallAllegroEvent***:

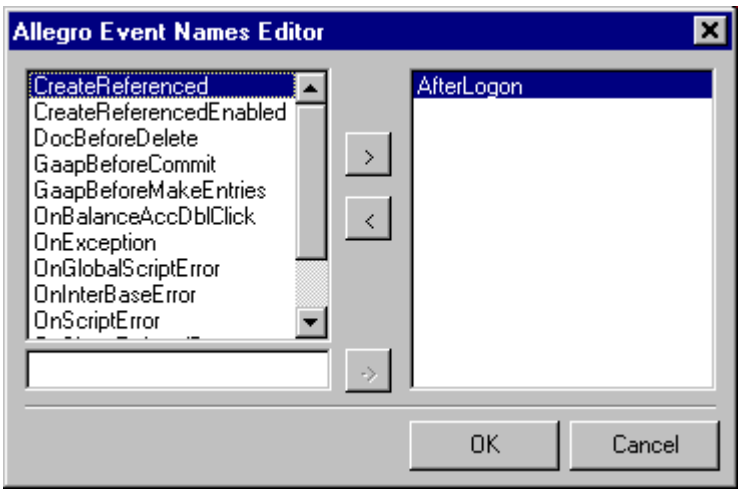
```
procedure CallAllegroEvent(const EventName: string;
   InputParams: variant; var OutputParams: variant);
```

Имеется ряд системных событий Allegro:

```
OnGlobalScriptError, OnScriptError, OnInterBaseError, OnException,
RefBeforePost, RefBeforeDelete, DocBeforeDelete
GaapBeforeMakeEntries, GaapBeforeCommit
AfterLogon, CreateReferencedEnabled, CreateReferenced,
OnBalanceAccDblClick, OnShowRelatedDocuments
```

Внимание! Если при вызове события передается один параметр, то он передается просто как тип ***Variant***, и только если число параметров больше одного, они передаются как массив ***Varraint array***.

Свойство EventNames можно редактировать в инспекторе объектов вручную, а можно вызвать редактор свойства, нажав на кнопку с многоточием. Редактор свойства позволяет не только вводить новые имена событий, но и использовать имеющиеся системные события Allegro:



Рассмотрим каждое системное событие подробнее.

Событие **OnScriptError** - происходит при ошибке в скриптовой системе, кроме ошибок в глобальных модулях.

Входные параметры	**тип**	**Назначение**
0	string	Текст сообщения об ошибке скриптера
1	string	Имя файла скриптового модуля, в котором произошла ошибка
2	string	Дополнительные сведения об ошибке, например, имя класса ошибки
3	integer	Номер строки скриптового модуля
4	integer	Номер символа в строке
Выходные параметры	**тип**	**Назначение**
	boolean	Если False (по умолчанию - так), то высвечивается системное сообщение об исключительной ситуации, если True, то это сообщение подавляется.

Событие **OnGlobalScriptError** - происходит при любой ошибке в глобальных модулях.

Входные параметры	**тип**	**Назначение**
0	string	Текст сообщения об ошибке скриптера
1	string	Имя файла скриптового модуля, в котором произошла ошибка
2	string	Дополнительные сведения об ошибке, например, имя класса ошибки
3	integer	Номер строки скриптового модуля
4	integer	Номер символа в строке
Выходные параметры	**тип**	**Назначение**
	boolean	Если False (по умолчанию - так), то высвечивается системное сообщение об исключительной ситуации, если True, то это сообщение подавляется.

Событие **OnInterBaseError** - происходит при ошибках сервера IB (например, при нарушениях уникального ключа, генерации исключительных ситуаций и т.п.).

Входные параметры	**тип**	**Назначение**
0	string	Текст сообщения об ошибке сервера
1	string	Текст сообщения Allegro об этой ошибке (возможно, переведенный на русский язык или уточненный)
2	integer	Код ошибки IBErrorCode
3	integer	Код ошибки SQLCode
Выходные параметры	**тип**	**Назначение**
	boolean	Если False (по умолчанию - так), то высвечивается системное сообщение об исключительной ситуации, если True, то это сообщение подавляется.

Событие **OnException** - происходит при ошибках самого Allegro).

Входные параметры	**тип**	**Назначение**
0	string	Текст сообщения об ошибке E.Message
1	string	Класс ошибки E.ClassName
Выходные параметры	**тип**	**Назначение**
	boolean	Если False (по умолчанию - так), то высвечивается системное сообщение об исключительной ситуации, если True, то это сообщение подавляется.

Событие **RefBeforePost** - происходит перед сохранением объекта в справочнике.

Входные параметры	**тип**	**Назначение**
0	string	Имя таблицы справочника
1	TRefQuery	Компонент справочника
Выходные параметры	**тип**	**Назначение**
	boolean	Если False (по умолчанию - так), то запись сохраняется в базе данных, если True, то сохранение отменяется (Abort).

Событие **RefBeforeDelete** - происходит перед удалением объекта из справочника.

Входные параметры	**тип**	**Назначение**
0	string	Имя таблицы справочника
1	integer	ID объекта
Выходные параметры	**тип**	**Назначение**
	boolean	Если False (по умолчанию - так), то запись удаляется, если True, то удаление записи отменяется (Abort).

Событие **DocBeforeDelete** - происходит перед удалением документа.

Входные параметры	**тип**	**Назначение**
0	TIBTransaction	Компонент транзакции
1	string	Имя таблицы документа
2	string	Отформатированное имя документа
3	integer	ID документа
Выходные параметры	**тип**	**Назначение**
	boolean	Если False (по умолчанию - так), то документ удаляется, если True, то удаление отменяется (Abort).

Событие **GaapBeforeMakeEntries** - происходит перед проведением ручной операции.

Входные параметры	**тип**	**Назначение**
0	TIBTransaction	Компонент транзакции
1	integer	ID документа
Выходные параметры	**тип**	**Назначение**
	boolean	Если False (по умолчанию - так), то операция проводится, если True, то операция отменяется (Abort).

Событие **GaapBeforeCommit** - происходит перед подтверждением транзакции ручной операции.

Входные параметры	**тип**	**Назначение**
0	TIBTransaction	Компонент транзакции
1	integer	ID документа
Выходные параметры	**тип**	**Назначение**
	boolean	Если False (по умолчанию - так), то транзакция подтверждается, если True, то ничего не происходит (Abort).

Событие **CreateReferencedEnabled** происходит каждый раз перед появлением контекстного меню «Проводника по документам».

Входные параметры	**тип**	**Назначение**
0	integer	DOC_TYPE_ID идентификатор типа текущего документа в «Проводнике по документам»
1	integer	DOC_ID идентификатор текущего документа в «Проводнике по документам»
2	integer	DIR_ID текущая папка в «Проводнике по документам»
Выходные параметры	**тип**	**Назначение**
	boolean	Если False (по умолчанию - так), то пункт меню ***Создать на основании…*** изображается невыбираемым, если конфигурация вернет True, то этот пункт контекстного меню будет выбираем и при «нажатии» на этот пункт меню пользователь спровоцирует событие **CreateReferenced** («Создать на основании»), которое конфигурация должна соответствующим образом обработать.

Событие **CreateReferenced** происходит, когда пользователь «нажал» пункт ***Создать на основании…*** контекстного меню в «Проводнике по документам». Для работы этого механизма необходимо предварительно также реализовать обработку события **CreateReferencedEnabled**, иначе этот пункт меню окажется неактивным.

Входные параметры	**тип**	**Назначение**
0	integer	DOC_TYPE_ID идентификатор типа текущего документа в «Проводнике по документам»
1	integer	DOC_ID идентификатор текущего документа в «Проводнике по документам»
2	integer	DIR_ID текущая папка в «Проводнике по документам»
Выходные параметры	**тип**	**Назначение**
	variant	Не используется

Событие **OnBalanceAccDblClick** происходит при двойном щелчке на счетах нижнего уровня в окне «Баланс».

Входные параметры	**тип**	**Назначение**
0	integer	ACC_ID идентификатор счета
1	integer	LAYER_ID идентификатор слоя
2	variant	Резервный параметр
Выходные параметры	**тип**	**Назначение**
	boolean	Если конфигурация обработала это событие, то она должна присвоить этому параметру True.

Событие **OnShowRelatedDocuments** происходит при «нажатии» на пункт меню «Ссылающиеся документы» в контекстном меню «Проводника по документам» или соответствующей кнопке в окне «Ручной операции».

Входные параметры	**тип**	**Назначение**
0	TIBTransaction	Компонент транзакции
1	string	TABLE_NAME - название главной таблицы документа
2	string	DOC_NAME – отформатированное название документа
3	integer	DOC_ID – идентификатор документа
4	integer	DOC_TYPE_ID – идентификатор типа документа
5	integer	DIR_ID – папка «Проводника по документам»
Выходные параметры	**тип**	**Назначение**
0	boolean	Если False (по умолчанию - так), то вызывается системное окно «Документы, ссылающиеся на данный», если присвоить True, то системное окно не вызывается.
1	boolean	Если конфигурация возвращает True, это означает, что исходный документ был модифицирован в контексте транзакции и следует принять решение о подтверждении или откате транзакции (используется при вызове события из окна «Ручной операции»).

Событие **AfterLogon**- происходит после соединения с базой данных. Используется для отображения заставки. Входные и выходные параметры имеют значение NULL.

Организовать вывод заставки при соединении с базой данных несложно. Нужно создать форму, разместить на ней компонент Image и загрузить в него картинку. Затем нужно поместить на эту форму таймер, в котором установить интервал отображения картинки в миллисекундах и свойству Enabled таймера присвоить False.
Затем нужно создать два обработчика – один у формы, который запустит таймер, другой у таймера, который закроет форму:

```
procedure TSplashForm.FormShow(Sender: TObject);
begin
  Timer1.Enabled := True;
end;

procedure TSplashForm.Timer1Timer(Sender: TObject);
begin
  Timer1.Enabled := False;
  Close;
end;
```

Теперь осталось создать глобальный модуль, если его еще нет, расположить на нем компонент AllegroEvents, его свойству EventNames присвоить значение AfterLogon, и создать обработчик события OnEvent:

```
procedure TDataModule1.AfterLogonEvent(const EventName: String;
   InputParams: Variant; var OutputParams: Variant);
begin
   UseUnit('SplashUnit.pas'); //загружаем модуль с формой
   SplashForm.ShowModal; //высвечиваем форму
   UnloadUnit(' SplashUnit.pas'); //выгружаем модуль с формой
end;
```

При работе с событиями Allegro нужно помнить, что если Вы назначите несколько обработчиков на событие с одинаковым именем, то сработают они все, причем в произвольном порядке.

Диалог выбора класса справочника TClassSelectDialog

Этот компонент рекомендуется использовать в тех случаях, когда необходимо в процессе работы программы вызвать стандартный диалог выбора класса справочника.

Свойства Published	
Options	Опции диалога, тип TClassDialogOptions
Root_ID	Идентификатор CLASS_ID корневого класса
Title	Заголовок для окна диалога.
Transaction	Транзакция

Свойства Public	**тип**	**readonly**	
Class_ID	integer		CLASS_ID выбранного класса.

Методы	
function Execute: boolean	Вызывает диалог на экран. Возвращает True, если пользователь выбрал класс в диалоге и False, если он нажал кнопку «Отмена».

Объявления некоторых типов данных, используемых в свойствах этого компонента показаны ниже:

```
  TClassDialogOption = (cdoDisableSelectFolders, cdoDisableSelectRoot);
{cdoDisableSelectFolders – запрет выбора классов, имеющих дочерние
  cdoDisableSelectRoot – запрет выбора корневого класса}
TClassDialogOptions = set of TClassDialogOption;
```

Использовать компонент TClassDialog несложно. Достаточно в Инспекторе объектов указать транзакцию. Затем в программе следует вызвать метод Execute, а затем прочитать значение свойства Class_ID. Например, так:

```
with ClassSelectDialog1 do
if Execute then
  ShowMessage(IntToStr(ACC_ID));
```

Еще проще использовать оконные компоненты – селекторы класса TClassSelectEdit и TDBClassSelectEdit, которые вызывают тот же диалог выбора класса справочника.

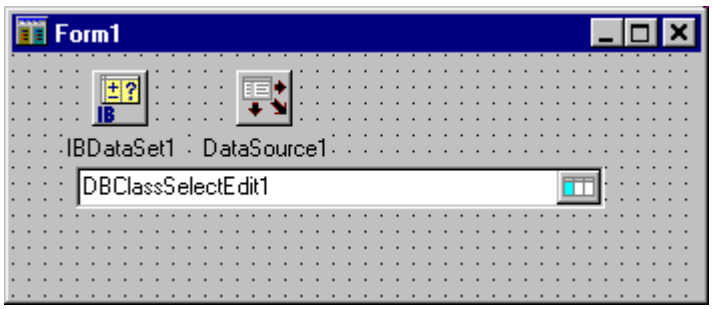
Селекторы класса справочника TClassSelectEdit и TDBClassSelectEdit

Селекторы класса являются оконными элементами управления (контролами). Компонент TDBClassSelectEdit является полной копией компонента TClassSelectEdit с той лишь разницей, что он может управлять данными в компонентах типа IBDataSet, IBQuery и им подобных компонентах доступа к данным.

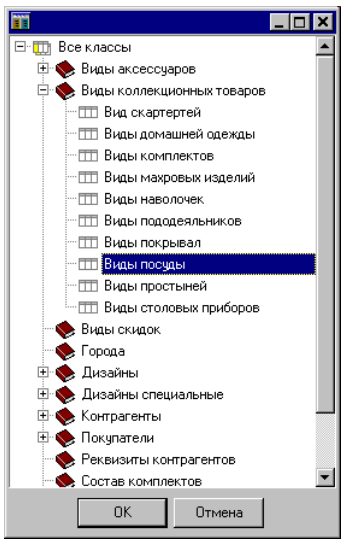
Компоненты всегда отображают наименование класса, соответствующего свойству Class_ID. Компоненты TClassSelectEdit и TDBClassSelectEdit имеют все те же свойства, что и компонент TClassSelectDialog, кроме свойства Title, которое у этих компонентов называется DialogTitle.

Кроме того они имеют массу свойств, свойственных обычным оконным компонентам (ширину, координаты и т.п.), на которых мы останавливаться здесь не будем. Оба этих компонента вызывают диалог выбора класса при нажатии пользователем кнопки с изображением таблички или при нажатии комбинации клавиш Alt+стрелка вниз. Компонент TDBClassSelectEdit имеет несколько дополнительных свойств для подключения к источнику данных:

Свойства Published	
DataField	Имя поля в наборе данных
DataSource	Компонент-источник данных типа TDataSource (через который происходит подключение к компоненту доступа к данным)



Диалог выбора класса справочника выглядит так:



Дерево рубрик TRubricTreeGrid

Этот компонент рекомендуется использовать в тех случаях, когда необходимо в процессе работы программы отображать в окне дерево рубрик и осуществлять навигацию по рубрикам. Кроме навигации компонент позволяет создавать новые рубрики, удалять рубрики или изменять их свойства.

Компонент является потомком класса деревьев TDCCustomTreeGrid и наследует большинство его свойств, о которых мы говорить здесь не будем. Ниже перечислены дополнительные свойства и методы, превращающие это дерево в дерево рубрик.

Свойства Published	
Transaction	Транзакция. Можно использовать ReadOnly транзакцию

Свойства Public	**тип**	**readonly**	
Root_Id	integer	**readonly**	Идентификатор корневой рубрики
ExpandedRubrics	string		Строка идентификаторов всех распахнутых рубрик, перечисленных через точку с запятой. Используется для запоминания степени распахнутости рубрик.
SelectedRubric_id	integer		Идентификатор текущей рубрики. Присвоение этому свойству значения раскрывает нужную ветвь, отображает соответствующую рубрику и делает ее текущей.

Методы	
procedure Open(Root_ID: integer)	Загружает корневую рубрику. Этот метод всегда следует вызывать при использовании этого компонента первым. Если в качестве параметра передать 0, то будут загружены все корневые рубрики.
procedure AddRubric	Вызывает стандартный диалог добавления рубрики
procedure AddChildRubric	Вызывает стандартный диалог добавления подрубрики
procedure AddFilterGroup	Вызывает стандартный диалог добавления подгруппы фильтров.
procedure EditRubric	Вызывает стандартный диалог редактирования свойств рубрики
procedure DeleteSelectedRubrics	Вызывает удаление выбранных рубрик
procedure RefreshNode(Node: TRubricNode)	Освежает указанный пункт дерева
function FindInTreeById(Rubric_Id: integer): TRubricNode	Ищет пункт дерева по идентификатору рубрики
function FindNodeById(ParentNode: TDCTreeNode; id: integer): TRubricNode	Ищет пункт дерева по идентификатору рубрики внутри определенного родителя
function GetRubric_ID(Node: TDCTreeNode): integer	возвращает идентификатор рубрики указанного пункта дерева.
function GetClass_ID(Node: TDCTreeNode): integer	возвращает идентификатор класса справочника указанного пункта дерева.

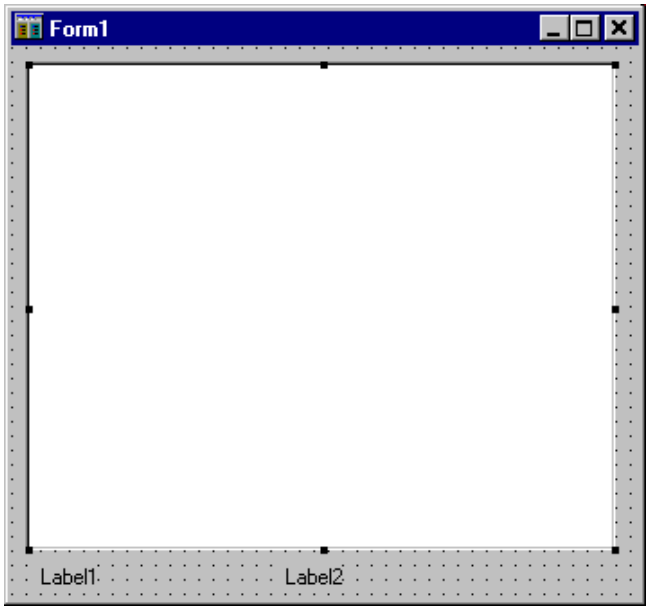
Использовать компонент TRubricTreeGrid несложно. Достаточно в Инспекторе объектов указать транзакцию. Затем в программе следует вызвать метод Open(), передав в него ID корневой рубрики. Проще всего это сделать в событии OnCreate формы:

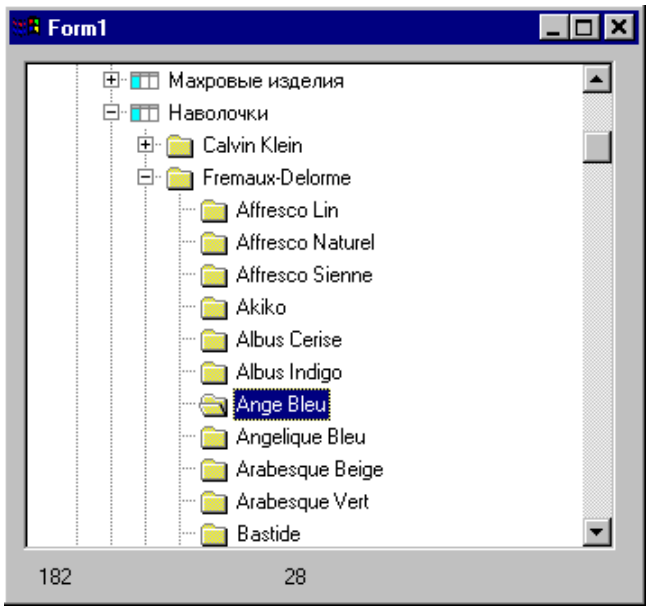
```
procedure TForm1.FormCreate(Sender: TObject);
begin
  RubricTreeGrid1.Open(0); // 0 - загружаем все дерево рубрикаторов.
end;
```

Если в дереве выбирается какой-то пункт, узнать идентификаторы класса и рубрики выбранного пункта можно в событии OnChange дерева:

```
procedure TForm1.RubricTreeGrid1Change(Sender: TObject; Node: TDCTreeNode);
begin
  Label1.Caption := IntToStr(RubricTreeGrid1.GetRubric_ID(Node)); //отображаем ID рубрики
  Label2.Caption := IntToStr(RubricTreeGrid1.GetClass_ID(Node)); //отображаем ID класса
end;
```

Как работает этот пример, показано на рисунках:





Функция GetFilterExpression позволяет получить условие фильтрации в виде куска SQL-кода, которое затем используется для управления справочниками в самой системе Allegro. Так что для эксперимента можно добавить к форме компонент TRefQuery, установить его свойство Transaction, добавить на форму TDataSource и TDBGrid, соединить это все, а в обработчик OnChange компонента дерева вписать такой код:

```
procedure TForm1.RubricTreeGrid1Change(Sender: TObject; Node: TDCTreeNode);
var
  rubric_class_id: integer;
begin
  RefQuery1.Close;
  RefQuery1.FilterExpression := GetFilterExpression(RubricTreeGrid1.GetRubric_ID(Node),
    MainConnection.MainTransaction, nil, rubric_class_id);
  RefQuery1.Class_Id := rubric_class_id;
  RefQuery1.Open;
end;
```