Краткое оглавление

Подробное оглавление

Введение

Гедымин представляет собой новую технологическую платформу с открытым кодом для быстрой разработки прикладных решений. При ее проектировании учитывались современные реалии: увеличение среднего масштаба разрабатываемых решений, расширение спектра прикладных задач, которые решаются при автоматизации предприятий, новые технические условия, в которых функционирует система.

Для кого предназначено настоящее руководство

Настоящее руководство предназначено в первую очередь для разработчика, создающего прикладные программы на платформе Гедымин, а также для специалиста, занимающегося внедрением готовых решений, адаптацией и подгонкой их под нужды конкретного заказчика. Опытный пользователь и системный администратор так же найдут здесь полезную информацию по эксплуатации системы, практическому решению возникающих проблем.

Подразумевается, что читатель знаком с программированием вообще и, в частности, с такими его современными концепциями как объектно-ориентированное программирование (ООП) и быстрая разработка приложений (Rapid Application Development — RAD). Так же необходимо знание основ реляционных баз данных, клиент-серверных технологий и языка SQL. Опыт непосредственного программирования с использованием Borland Delphi, Microsoft Visual Basic или C# поможет существенно ускорить освоение платформы.

Настоящее руководство разбито на главы, каждая из которых охватывает определенный аспект платформы. В конце каждой главы приводится краткое резюме и контрольные вопросы, которые помогут определить насколько хорошо усвоен материал. В зависимости от того, к какой категории относит себя читатель: опытный пользователь, специалист по внедрению, начинающий или продвинутый разработчик, отдельные главы могут быть опущены без ущерба для усвоения всего материала.

В конце руководства находятся приложения, содержащие полезную справочную информацию: список всех глобальных объектов платформы с описанием их методов и свойств, иерархию бизнес-классов, ответы на часто возникающие вопросы, примеры решения типовых проблем и т.п.

История создания платформы

На сегодняшний день Гедымин является флагманским продуктом компании Golden Software и история его создания тесно переплетается с историей самой компании, которая была основана в далеком уже 1994 г. в г.Минске.

Первым программным продуктом, выпущенным компанией, была Анжелика — достаточно простая программа подготовки и печати платежных документов. В целях сохранения исторической достоверности, стоит сказать, что первые копии программы были проданы под именем GPWL[1], что являлось сокращением от Golden Print Works Lite. Однако, столкнувшись с тем, что рядовой белорусский пользователь был просто не в состоянии выговорить это название (хотя нам оно казалось вполне читабельным) мы были вынуждены начать поиски нового имени. Даже между отцом и матерью часто возникают разногласия при выборе имени для новорожденного. Что уже говорить о коллективе из четырех человек (имеенно столько сотрудников насчитывала в то время наша компания). В конце концов, утомившись от постоянных споров мы предоставили право выбора нашему бухгалтеру, которая положила конец нашим страданиям, безапеляционно предложив назвать программу в свою честь. Так GPWL стала Анжеликой.

Анжелика оказалась в нужном месте в нужное время. Она была современна (первая белорусская программа под Windows), проста в освоении и удобна в работе. Но, самое главное, она в совершенстве позволяла выполнять те задачи, для которых, собственно, и была предназначена: подготавливать и печатать платежные документы. Анжелика просто обязана была стать бестселлером и она стала им. Программа успешно продавалась в течение более чем десяти (!) лет. Причем, не только нашей компанией, но и многочисленными пиратами.

Анжелика положила начало целому семейству программных продуктов для ведения бухгалтерии, складского и торгового учета, расчета заработной платы, учета основных фондов предприятия и т.п. И

[1] Читается как Джи-Пи-Ви-Эль.

хотя название каждого продукта содержало в себе слово Анжелика: Анжелика-Бухгалтер, Секрет Анжелики[2], Зарплата Анжелики и т.п.[3], по сути это были отдельные приложения, каждое со своей базой данных, своим пользовательским интерфейсом и системой команд. При эксплуатации нескольких программ семейства на одном предприятии, «Анжелика-Бухгалтер» выступала в роли ядра, центральной части программного комплекса, куда экспортировались данные справочников и бухгалтерских проводок из других программ. Как правило, две любые программы комплекса (если, конечно, одна из них не являлась «Анжеликой-Бухгалтер») не могли взаимодействовать друг с другом напрямую.

Естественно, что очень скоро, такая программная организация перестала удовлетворять как нас самих, так и наших клиентов. Файл-серверная архитектура была ненадежна при эксплуатации в сетях. Адаптация программы под нужды одного заказчика зачастую приводила к тому, что переставала работать функциональность, созданная для другого заказчика. Данные дублировались в нескольких местах, что порождало настоящий ад для специалистов отдела сопровождения при их рассинхронизации. Несколько программ семейства, установленных на одном предприятии, напоминали скорее муравейник, чем грамотно построенное здание.

Первые эскизы проекта будущей платформы появились у нас в далеком уже 1997 году. Постепенно, идея приобретала все более и более четкие контуры. Начиная с 1999 года мы вплотную приступили к анализу и проектированию, а с 2000 года к кодированию.

Разработка программного обеспечения представляет собой итеративный процесс. И хотя, вышедшая весной 2005 года финальная версия платформы имела номер 1.00, на самом деле это была вторая большая итерация. Первая — датируется 2000-2003 г.г. На ней мы смогли проверить основные концепции построения системы. Определить, что именно необходимо пользователю, какой функциональности недостает, а какая является избыточной и невостребованной. На основе накопленного опыта мы внесли существенные изменения в первоначальный проект. Именно во второй «реинкарнации» мы сделали Гедымин полноценной системой программирования и быстрой разработки приложений.

Платформа с открытым кодом

Мы не сразу пришли к тому, чтобы сделать Гедымин проектом с открытым исходным кодом. Во-многом, на наше решение повлияло наблюдение за развитием многих open source систем, например, таких как: операционная система Linux, web-сервер Apache, браузер Firefox и т.д. Исходный код Гедымина, так же включает в себя открытые библиотеки JCL и SynEdit. И, конечно же, стоит упомянуть используемый Гедымином сервер баз данных Firebird, который представляет собой open source клон широко известного коммерческого SQL сервера Borland Interbase.

Мы считаем, что использование платформы с открытым исходным кодом приносит выгоду всем участникам данного процесса. Так, пользователь имеет возможность существенно снизить стоимость программной системы и может дорабатывать и развивать ее в соответствии с собственными потребностями. Разработчик, он же поставщик системы с открытым исходным кодом, получает свою выгоду ввиде богатой обратной связи, тестирования, доработки и разработки как ядра платформы, так и набора прикладных решений силами сторонних разработчиков.

Суть лицензии, по которой поставляется исходный код платформы Гедымин заключается в следующем:

1. Авторские права на исходный код платформы Гедымин принадлежат компании Golden Software of Belarus, Ltd.

---

[2] Таким «загадочным» наименованием программа автоматизации торгового учета обязана популярному в то время латиноамериканскому сериалу «Секрет Тропиканки». И хотя мы и не смотрели телевизор, название нам понравилось.

[3] Вот полный список программ, входивших в семейство Анжелика: 1) Анжелика — печать платежных документов; 2) Анжелика-Бухгалтер — бухгалтерский учет; 3) Секрет Анжелики — складской и торговый учет; 4) Средства Анжелики — учет основных средств; 5) Зарплата Анжелики — расчет заработной платы; 6) Меню Анжелики — калькуляция меню; 7) Аренда Анжелики — учет договоров аренды; 8) Договор Анжелики — составление договоров; 9) СТО Анжелики — станция технического обслуживания автомобилей; 10) Декларация Анжелики — расчет и печать налоговой декларации; 11) Хранитель Анжелики — архивное копирование; 12) Совет Анжелики — правовая информация.

2. Любой желающий может бесплатно получить исходный код платформы Гедымин[4], откомпилировать его и использовать полученный выполняемый модуль для собственных нужд (например, для автоматизации собственного предприятия) или для коммерческого использования (например, для продажи третьим лицам).
3. В случае, если некоторое лицо (организация) вносит свои изменения в исходный код платформы (исправления ошибок, доработка существующей функциональности, добавление новой функциональности), эти изменения должны быть сделаны доступными как поставщику платформы, так и любому иному пользователю платформы Гедымин.
4. Авторские права на изменения принадлежат непосредственно тому лицу (организации), которое (которая) произвело эти изменения.

Прикладные решения

Прикладные решения (называемые так же настройками) могут быть приобретены у компании Golden Software of Belarus, Ltd за отдельную плату, либо разработаны пользователем платформы самостоятельно. На сегодняшний день разработан целый спектр настроек, охватывающих такие сферы, как:

- Автоматизация бухгалтерского и налогового учета;
- Логистику и склад;
- Торговлю, включая опт, розницу и комиссионную торговлю;
- Управление торговым залом, включая подключение разнообразного торгового оборудования;
- Управление взаимоотношениями с клиентами;
- Управление кадрами предприятия и расчет заработной платы;
- Управление рестораном, включая калькуляцию блюд;
- и др.

Прикладные решения так же поставляются с открытым исходным кодом. Пользователь (покупатель) может изменять их и развивать в соответствии со своими потребностями.

Бизнес с компанией Golden Software

Условия лицензии на поставку платформы Гедымин и набора прикладных решений открывают перед партнерами компании следующие возможности по ведению собственного бизнеса:

1. Во-первых, сторонняя компания может абсолютно бесплатно воспользоваться платформой, как инструментальным средством для создания собственных прикладных решений и поставки их третьим лицам;
2. Во-вторых, сторонняя компания может стать партнером Golden Software, Ltd и распространять решения, созданные Golden Software Ltd, получая либо процент от каждой сделки, либо некоторое фиксированное вознаграждение, т.е. действовать как дистрибьютор продукции компании Golden Software, Ltd;
3. В-третьих, возможен любой смешанный вариант, основанный на двух предыдущих пунктах. Например, компания может заниматься автоматизацией предприятий определенной отрасли, устанавливая базовые настройки от Golden Software, Ltd совместно с собственными настройками, реализующими отраслевую специфику.

Инструментальные средства

При разработке платформы использовался компилятор Borland Delphi 5. Мы постарались свести к минимуму использование сторонних программных компонентов. Исходный код Гедымина включает:

---

[4] Библиотека FastReport, используемая Гедымином, не является open source проектом. Каждому, кто будет использовать исходный код платформы, придется приобрести необходимое количество лицензий непосредственно у разработчика данной библиотеки.

- Генератор отчетов FastReport версии 2.55 с небольшими исправлениями, адаптирующими его для использования в Гедымине и исправляющими некоторые мелкие баги;
- Набор компонентов Toolbar 2000;
- Библиотеку прикладных функций JCL;
- Редактор текста с подсветкой синтаксиса SynEdit;
- Популярную библиотеку zlib для упаковки данных.

Для организации доступа к серверу базы данных используются компоненты Interbase Express (IBX), которые были существенно переработаны.

Полный размер исходного кода платформы, показываемый при построении (команда Build) выполняемого модуля составляет около одного миллиона строк[5].

Обзор архитектуры

Проблемно-ориентированная среда

Гедымин представляет собой проблемно-ориентированную платформу. Это означает, что при создании прикладного решения разработчик концентрирует свои усилия именно на решении поставленной задачи, а не на создании кода отслеживающего движение мыши, управляющего размещением элементов управления на экране или отвечающего за какие-то сложные вопросы записи в базу данных.

Приложения в Гедымине разрабатываются в терминах классов проблемно ориентированных бизнес-сущностей (entity), называемых также бизнес-классами. Это одна из существенных особенностей платформы.

Вот несколько примеров классов бизнес-объектов и принципы проектирования приложений с их использованием:

Справочники

Описания таких сущностей, как товары, контрагенты, валюты, склады, объединяет наличие таких общих свойств, как внутренняя идентификация объекта в системе, необходимость поддержки иерархии и группировки элементов, необходимость поддержки вложенных таблиц. Информацию об этих объектах надо хранить, они задействованы в хозяйственных операциях предприятия и т.д. В Гедымине все такие сущности объединены в общий класс “справочник”, для которого перечисленные выше свойства и возможности поддерживаются на уровне платформы.

Для создания в Гедымине нового справочника достаточно описать необходимый набор параметров. Это делается визуально, не требуется писать ни единой строчки кода. Так, для создания справочника “товары” достаточно:

- указать его название — “товары”;
- отметить, что справочник иерархический (товары могут делиться на группы и подгруппы);
- определить другие свойства, которые должна поддерживать система для данного справочника, такие как способ нумерации элементов, автонумерация и т.п.
- задать поля элементов справочника - для товаров это может быть закупочная цена, отпускная цена, вес и т.п.

Необходимый минимум для создания (описания) бизнес-класса “справочник” на этом выполнен — теперь достаточно одним нажатием кнопки сохранить эту сущность и можно начать работать с ней в режиме использования. При этом соответствующая экранная форма для работы с созданным нами справочником генерируется системой автоматически. Естественно, разработчик может сконструировать экранную форму и самостоятельно, придав ей в точности такой вид и свойства, которые соответствуют особенностям решаемой задачи, необходимы по эргономическим соображениям и т.д.

[5] В данном количестве строк учитываются также код библиотек VCL, используемый Гедымином.

Документы

Документы — счета, накладные, заказы и т.п. — фиксируют различные события, происходящие в хозяйственной жизни организации. Важным свойством документа является его привязка ко времени. В Гедымине для этих объектов поддерживается идентификация самого события хозяйственной жизни, отражение в учетных механизмах, контроль событий и отражение событий в реальном времени. Такой набор функциональности заложен в систему и обеспечивает быструю разработку разнообразных документов.

Так, для описания документа “приходная накладная”, который фиксирует поступление товаров на предприятие, нам достаточно указать реквизиты документа:

- Организация (контрагент), от которого мы получаем товар — ссылка на справочник организаций. При этом проявляется очень важная возможность — те объекты и сущности, которые мы описываем в системе сами становятся типами данных;
- Склад, на который поступает товар — ссылка на справочник “склады”;
- Состав документа. По одной накладной может поступить несколько товаров, поэтому в документ включается вложенная таблица, в которой имеются поля типа “справочник товаров”, а также количество данного товара (число) и суммарная стоимость (тоже число).

В простейшем примере этого достаточно для описания структуры данных документа и начала работы с ним.

План счетов и регистры бухгалтерского учета

Система двойной записи бухгалтерского учета представляет собой отдельную модель учета со своей спецификой, поэтому в Гедымине план счетов, журнал хозяйственных операций, бухгалтерская проводка выделены в отдельные классы сущностей. Практика применения Гедымина для автоматизации учета очень широка и пока еще не возникало ситуации, чтобы разработчикам прикладных решений не хватило возможностей механизмов бухгалтерского учета, реализованного в Гедымине. Причем данные механизмы никак не навязывают разработчику собственно принципов ведения бухгалтерского учета.

2½ уровневая архитектура

Клиент-серверные приложения принято делить на классические двухуровневые, когда клиентская часть, реализующая логику работы программы, обращается к серверу баз данных, как правило, с использованием языка запросов SQL, и трехуровневые, в которых клиентская часть отвечает только за отображение данных и организацию диалога с пользователем, а логика программы вынесена в отдельный сервер приложения, который, в свою очередь, обращается к серверу базы данных. Обе схемы имеют как свои преимущества, так и недостатки. Очевидно, что в случае трехуровневой схемы проще внести изменения в логику работы программы — не надо заменять клиентские приложения, достаточно установить новую версию сервера. С другой стороны, в классической схеме отсутствует посредник между клиентом и сервером базы данных, что упрощает развертывание программы и увеличивает ее надежность. В Гедымине мы постарались найти компромисс между двумя вышеупомянутыми подходами. В результате получилось то, что, мы называем 2½ уровневой архитектурой. Внешне она похожа на классическую схему: клиентская часть взаимодействует с сервером базы данных. Ее основное отличие заключается в том, что в базе хранятся не только данные, но и алгоритмы, тексты макросов, экранные и печатные формы, собственно, то, что мы называем логикой работы приложения. Архитектура Гедымина вобрала в себя лучшие черты обеих вышеупомянутых схем: для развертывания системы достаточно установить соответствующее количество клиентов и сервер базы данных (в случае локальной установки инсталляция сервера базы данных не потребуется). Для изменения логики работы программы достаточно подключиться с любого (!) клиентского места под учетной записью администратора и внести необходимые изменения. Причем делать это можно «на лету», не отключая других пользователей на время внесения изменений, что особенно важно, когда речь идет о реальной системе, эксплуатирующейся на предприятии с непрерывным производством.

Принято считать, что трехуровневая схема — это схема с «тонким» клиентом., которая позволяет задействовать морально устаревшие, маломощные компьютеры. Однако, на сегодняшний день любое приложение фактически можно превратить в трехуровневое. Достаточно использовать Windows Terminal Server или аналогичное ПО других разработчиков, например Citrix Metaframe. Кроме использования устаревших или маломощных компьютеров (например наладонных) сервер терминалов позволяет

организовать работу в сети с небольшой пропускной способностью, например, когда удаленный офис соединен с головным посредством выделенной телефонной линии или через канал DSL.

Структура платформы

Основными составляющими частями платформы являются:

- Сервер базы данных (Interbase/Firebird/Yaffil);
- Система исполнения программного кода;
- Набор встроенных функций и бизнес-объектов (объектов прикладной логики);
- Инструментальные средства проектирования и быстрой разработки приложений.

Сервер базы данных с открытым кодом

При выборе сервера базы данных оценивались следующие параметры:

- Надежность при работе с большими объемами данных;
- Производительность;
- Полная поддержка стандарта SQL 99;
- Простота в обслуживании;
- Стоимость, лицензионная чистота.

После оценки всех присутствующих на рынке альтернатив выбор был сделан в пользу сервера с открытым исходным кодом Firebird/Yaffil. Этот сервер является клоном известного коммерческого сервера Borland Interbase и имеет более чем 20-ти летнюю историю использования на сотнях тысяч предприятий по всему миру. В настоящее время Firebird/Yaffil поддерживается сообществом программистов и является бесплатным программным продуктом, что позволяет существенно снизить стоимость автоматизации для предприятия.

Пост-реляционные технологии

Гедымин предоставляет разработчику возможность работать с базой данных как на уровне таблиц (реляционная модель), так и используя объектно-ориентированный подход. Как правило, запросы к таблицам на языке SQL используются для извлечения информации, особенно когда необходимо оптимизировать выборку по скорости или реализовать “хитрый” нестандартный алгоритм. Объекты (в терминах Гедымина — бизнес-объекты) используются для добавления или изменения информации. Использование объектов позволяет радикально снизить трудоемкость работы с базой данных. Так, добавление документа (например, накладной) может потребовать добавления и изменения данных в более чем десяти таблицах. При использовании же соответствующего бизнес-объекта вся операция займет не более чем три-четыре строки исходного кода.

Типы данных

При создании новых бизнес-классов, либо добавлении полей в существующие, разработчик имеет возможность использовать как простые типы данных, определенные стандартом SQL, так и сложные, реализованные в рамках платформы Гедымин. К таким, сложным типам данных, относятся:

- Ссылка;
- Множество;
- Перечисление.

Ссылка

При добавлении поля типа ссылка автоматически создается внешний ключ (FOREIGN KEY) на таблицу-справочник[6]. Разработчик имеет возможность указать поле из таблицы справочника, которое будет использоваться для отображения наименования объекта, на который указывает ссылка. В дальнейшем, система будет автоматически подставлять указанное поле в SELECT часть запроса бизнес-объекта и, соответственно, таблицу-справочник — в часть FROM. Дополнительно, можно указать условие, которое будет ограничивать подмножество записей в таблице-справочнике.

Множество

Для добавления поля типа множество необходимо указать наименование поля и выбрать таблицу-справочник, содержащую элементы множества. На основании указанных данных:

1. в основную таблицу бизнес-класса будет добавлено поле для хранения строкового отображения элементов множества;
2. будет создан триггер[7] для основной таблицы бизнес-класса, отвечающий за обновление поля, содержащего строковое отображение элементов множества;
3. будет создана т.н. кросс-таблица, каждая запись которой содержит две ссылки: на запись бизнес-объекта и на запись в таблице-справочнике, т.е. на элемент, входящий в конкретное множество.

Перечисление

При создании поля данного типа разработчик задает список элементов перечисления. Для каждого элемента указывается его наименование и код. В главную таблицу бизнес-класса добавляется поле для хранения кода выбранного элемента перечисления. При отображении данных на экране платформа сама позаботится о том, чтобы вместо кода выводилось наименование элемента.

Язык программирования

В качестве языка программирования используется усовершенствованная версия Visual Basic Script (VBScript). Этот язык отлично знаком большинству разработчиков прикладных решений. Усовершенствования, внесенные специалистами компании Golden Software, позволяют обрабатывать исключительные ситуации, возникающие при работе программы, и тем самым повышают ее надежность.

Конструктор кода

Для быстрого создания программного кода предусмотрен конструктор, который позволяет буквально составлять исходный текст функции из готовых блоков. Параметры каждого блока указываются в диалоговом окне, что позволяет используя конструктор реализовывать несложные алгоритмы даже простому смертному пользователю, далекому от программирования.

Встроенный отладчик и профайлер

Возможности встроенного отладчика позволяют:

- Просматривать/изменять значения переменных;
- Обращаться к глобальным объектам, просматривать их свойства и вызывать методы;
- Вызывать на выполение функции и процедуры;
- Устанавливать простые и условные точки останова;
- Просматривать стек вызова функций;
- Просматривать иерархию классов, их методы и свойства;
- Просматривать/изменять значения свойств объектов;

---

[6] Под справочником здесь имеется ввиду таблица, содержащая объекты на которые указывает ссылка. Очевидно, что это может быть любая таблица в базе данных, имеющая первичный ключ, например, таблица содержащая документы или бухгалтерские проводки.

[7] Данный триггер вызывается перед обновлением записи (BEFORE UPDATE).

- Выполнять профилирование кода, фиксировать время выполнения и количество вызовов отдельных функций.

Платформа позволяет изменять код что-называется «на лету». Так, если выполнение программы остановлено, вручную, с помощью точки останова или в результате возникновения ошибки или исключительной ситуации, любую функцию, в том числе и ту, которая выполняется в данный момент времени, можно изменить. После возобновления выполнения программы будет выполняться уже откорректированный код.

Редактор кода

Редактор кода, используемый в платформе Гедымин, поддерживает подсветку синтаксиса и подстановку методов или свойств объектов по первым введенным символам (подобно технологии IntelliSence).

Редактор кода программы позволяет задавать и использовать в последствии типовые шаблоны.

Предусмотрен полнотекстовый поиск по всем макросам, находящимся в базе данных, а также фильтрация списка макросов на основании заданных критериев.

Оконная система и механизм экранных форм

Реализованная в Гедымине оконная система учитывает особенности автоматизации экономической и управленческой деятельности предприятий и ориентирована на обеспечение высокой эргономичности и эффективности работы с бизнес-приложениями. В частности, система форм и элементов управления обеспечивает:

- Автоматическую связь форм и элементов управления с данными, которая позволяет организовать удобное взаимодействие между различными формами без специального программирования;
- Поведение элементов, определяемое данными;
- Специализированный набор элементов управления, ориентированный на бизнес-задачи, в т.ч.: поля ввода с функциональными кнопками (выбор, очистка, открытие значений), редактирование в одном элементе любых типов данных, эффективные и удобные динамические списки для просмотра информации из базы данных с условным форматированием и поддержкой различных вариантов сортировки и фильтрации;
- Современный эргономичный дизайн.

В числе возможностей, поддерживаемых оконной системой Гедымина:

- автоматическое формирование командного интерфейса формы в соответствии с ее назначением;
- автоматическая генерация форм;
- инструменты быстрого редактирования форм;
- возможность привязать настройки формы к конкретному пользователю.

Механизм отчетов

Возможности платформы Гедымин позволяют подготавливать как статические печатные формы, так и динамические, интерактивные документы, органично интегрированные в приложение.

Существует возможность подготовки и вывода отчета в текстовом режиме, что может быть полезно при организации печати на матричных принтерах и фискальных регистраторах.

В качестве редактора отчетов используется популярный редактор FastReport, хорошо известный многим программистам.

Управление территориально распределенными базами данных

В Гедымине реализован механизм работы с территориально распределенными информационными базами. В этом варианте обеспечивается работа единого прикладного решения с несколькими территориально разнесенными базами данных, между которыми нет постоянной связи. Интеллектуальный механизм репликации позволяет переносить целостные объекты, описывать правила

переноса объектов между базами. Переносятся только измененные данные. Система обеспечивает достаточно высокую устойчивость к сбоям и защиту от потерь данных.

Система прав доступа

В Гедымине применяется разграничение прав доступа на уровне групп пользователей. Такая схема позволяет легко настроить общую схему доступа и, впоследствии, включая конкретного пользователя в ту или иную группу управлять его правами доступа. На уровне данных поддерживается разграничение доступа для:

- классов бизнес-объектов;
- отдельных экземпляров бизнес-объектов (отдельных записей в таблице);
- полей бизнес-классов (полей в таблице);

Так же присутствуют групповые политики, которые позволяют легко настраивать доступ к различным функциям системы.

Платформа обладает встроенными возможностями по аудиту действий пользователя и мониторированию изменений данных в базе. Фиксация изменений данных происходит с помощью специальных триггеров, что позволяет фиксировать изменения, произведенные не только из настройки на платформе, но и любой внешней программой в результате прямого обращения к серверу базы данных.

Хранение настроек

Каждый пользователь, работающий с платформой, может иметь набор своих параметров, определяющих как внешний вид приложения, так и особенности его функционирования. Кроме этого существует набор глобальных параметров, общих для всех пользователей системы, и набор параметров, специфических для выбранной рабочей организации. Для хранения и доступа к параметрам существует специальная внутренняя структура — Хранилище. Разработчики, знакомые с реестром (registry) операционной системы Windows найдут много общего между ним и хранилищем платформы Гедымин. Хотя данные хранилища имеют иерархическую, древовидную структуру в базу данных они записываются как упакованный двоичный поток данных, что существенно повышает скорость при обращении к ним.

Масштабируемость

Платформа Гедымин для всех прикладных решений независимо от отраслевой специфики и фирмы разработчика обеспечивает:

- возможность использования системы от локального компьютера до десятков пользователей в локальной сети;
- использование технологии “клиент-сервер”;
- возможность развертывания работы на нескольких территориально удаленных точках с периодическим обменом информацией.

Интеграция

Гедымин рассчитан на широкую интеграцию с разными системами. В числе технологий интеграции, заложенных в платформу:

- XML-обмен;
- OLE-automation (Client и Server);
- Файловый обмен (TXT, CSV, DBF).

Реализован основанный на COM-технологии механизм, позволяющий разрабатывать внешние компоненты, расширяющие функциональность встроенного языка, на таких языках как Visual C++, MS Visual Basic, Borland Delphi.

Для импорта данных из текстовых файлов (например, из файлов, генерируемых приложением банк-клиент) существует механизм конвертации, на вход которого передается исходный текстовый файл и шаблон, заданный на специальном языке. В результате, получается набор данных, который может быть обработан в макросе.

Интернационализация

Возможности платформы позволяют перевести пользовательский интерфейс и сообщения системы на любой язык. Файл перевода хранится отдельно от выполняемого модуля и содержит данные в формате XML.

Автоматизация в точном соответствии с потребностями предприятия

Использование платформы Гедымин и созданных на ее базе решений позволяет:

- Выбрать оптимальный вариант автоматизации в точном соответствии с потребностями предприятия;
- Проводить поэтапную автоматизацию, исходя из приоритетов решаемых задач, допустимых сроков и затрат на внедрение — на базе одной и той же системы, получая реальную отдачу на каждом шаге, с минимальными затратами времени и средств;
- Значительно упростить обучение пользователей и администрирование системы;
- Развивать систему по мере роста потребностей предприятия, не останавливая при этом ее эксплуатации.

Установка и запуск

В настоящей главе подробно описана процедура установки локальной (однопользовательской) или сетевой (многопользовательской) версии платформы Гедымин, как с использованием автоматического инсталлятора, так и в ручном режиме. Кроме этого рассматриваются вопросы перехода от локальной версии к сетевой, обновления исполняемого модуля программы и структуры базы данных, а также загрузки новых настроек или обновления установленных ранее.

ВНИМАНИЕ! Установка программы возможна только при наличии локальных прав администратора на компьютер у пользователя, инсталлирующего программу.

Локальная установка

Перед инсталляцией программы рекомендуется закрыть все открытые приложения Windows.

Перед установкой проверьте наличие свободного пространства на том диске, где будет размещена программа. Для корректной работы достаточно компьютера с установленной операционной системой Windows 98/Me/2000/XP, процессором Pentium-400 и выше, 64 Мб оперативной памяти (RAM), 200 Мб свободного места на диске. Для надежной и комфортной работы рекомендуется компьютер с установленной оперативной системой Windows2000/XP, файловой системой NTFS, процессором Pentium-1500 и выше, 256 Мб оперативной памяти (RAM).

Этапы установки

Вставьте компакт-диск с программой в CD-ROM. Если на экране появилось окно запуска программы инсталляции — нажмите кнопку Установить. В ином случае просмотрите содержимое компакт-диска с помощью Проводника или другой программы для работы с файлами и запустите файл setup.exe, который находится в корневом каталоге или в папке помеченной как «Локальная установка» на инсталляционном диске.

Появится окно, изображенное на рис. 1.2.

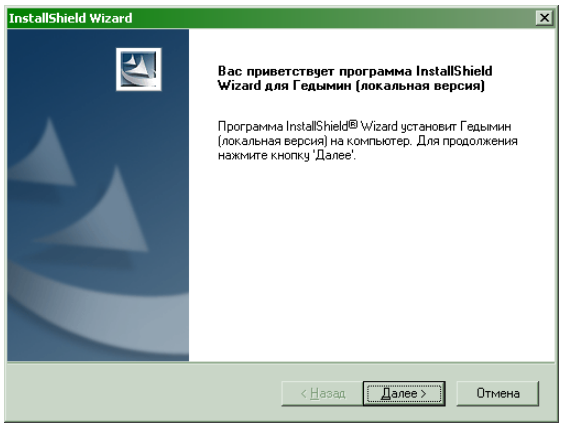


Рис. 1.2. Окно инсталяционной программы Гедымин

Нажмите кнопку Дальше.

Откроется окно (рис. 1.3), в котором нужно выбрать папку для помещения программных файлов и базы данных.

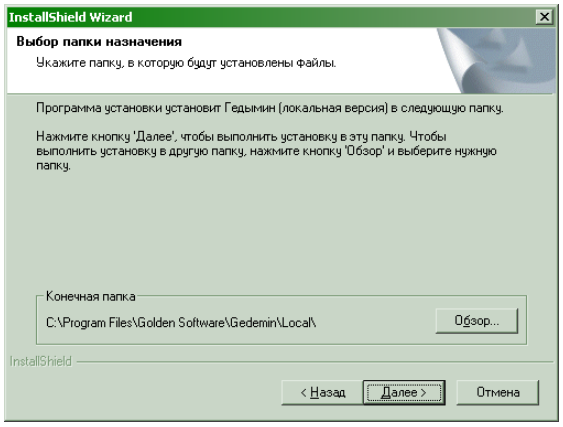


Рис. 1.3. Окно выбора папки для программных файлов и файла базы данных

Папка для установки программы и базы данных предлагается по умолчанию, но вы можете ее изменить. Нажмите кнопку Обзор — появится стандартный диалог для выбора папки. Выберите папку и нажмите Ок.

После выполнения этих действий нажмите кнопку Дальше.

Откроется окно, в котором можно выбрать вид установки: Обычная, Сокращенная или Выборочная (рис 1.4).

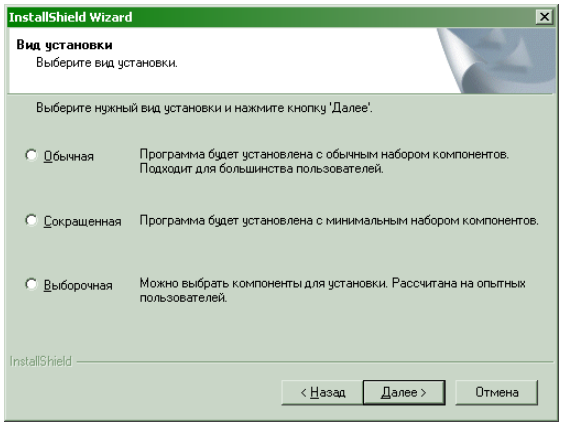


Рис. 1.4. Окно выбора вида установки

Если выбрана Обычная установка, то будут установлены все файлы. При сокращенной установке не будут инсталлироваться файл демонстрационной базы данных и файл справки по VBScript. И, наконец, выборочная установка позволяет указать какие файлы будут устанавливаться, а какие нет (рис 1.5).

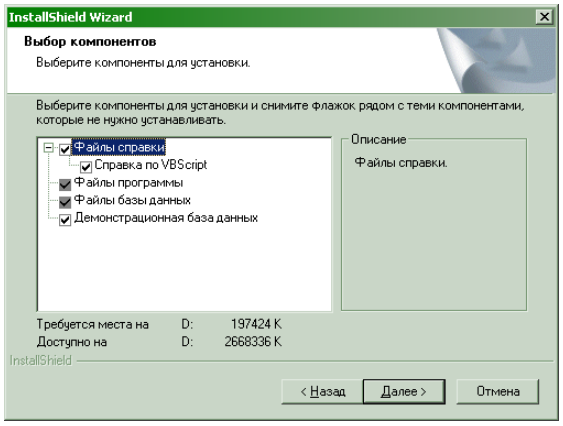


Рис. 1.5. Окно выбора файлов для установки

Отмечаем необходимые файлы и нажимаем кнопку Дальше.

После этого начнется процесс копирования файлов на жесткий диск по окончании которого на экран будет выведено сообщение (рис 1.6).

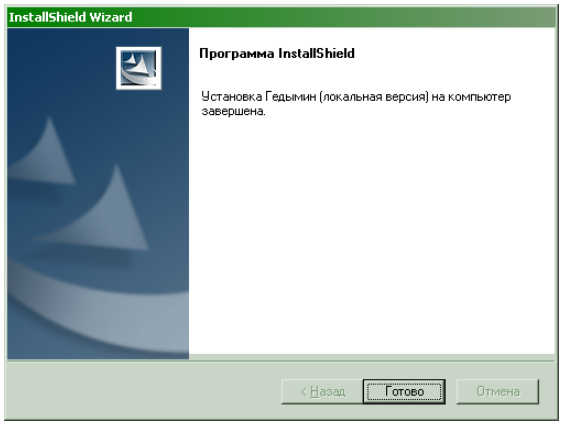
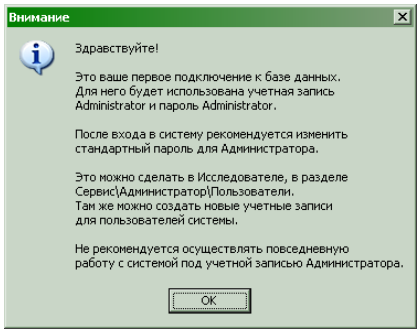


Рис. 1.6. Сообщение об успешном завершении установки

Запуск программы

Запустить Гедымин можно либо воспользовавшись соответствующим ярлыком, созданным инсталлятором на рабочем столе или через кнопку Старт, меню Программы, группа Golden Software Гедымин, пиктограммка Гедымин.

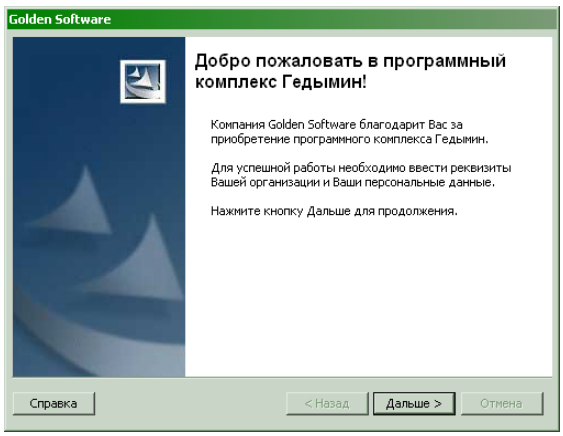
Если это первый запуск после установки на компьютере, то на экран будет выведено следующее информационное сообщение:



Из него видно, что для первого входа в систему автоматически будет выбрана учетная запись администратора. В дальнейшем рекомендуется создать свою (пользовательскую) учетную запись и использовать ее для входа в систему.

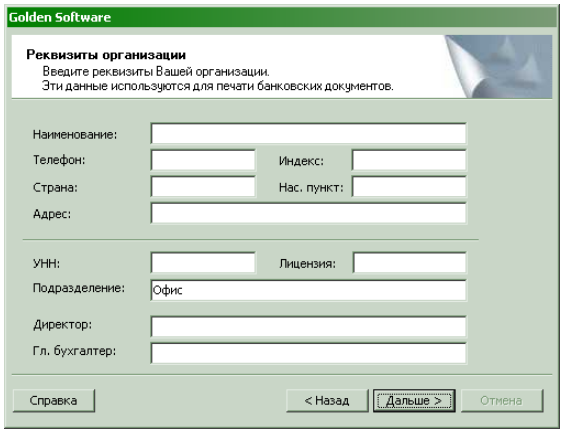
Заполнение начальных параметров

Для корректной работы системы необходимо заполнить начальные параметры. При первом запуске, на экран будет выведен мастер, облегчающий данную задачу.



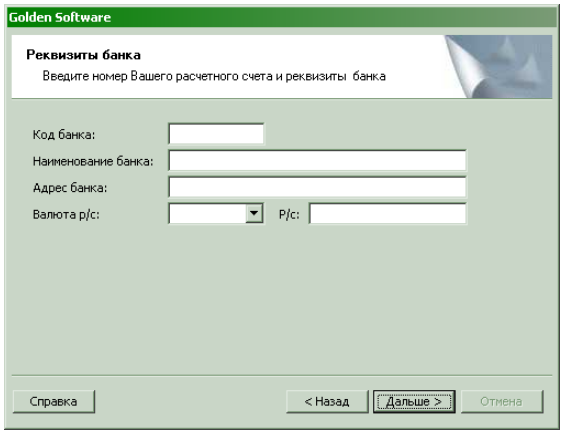
Нажимаем кнопку Дальше.

Откроется страница мастера для заполнения параметров организации.



Как минимум, необходимо заполнить поле Наименование. Рекомендуется заполнить так же и другие поля. По окончании ввода нажимаем кнопку Дальше.

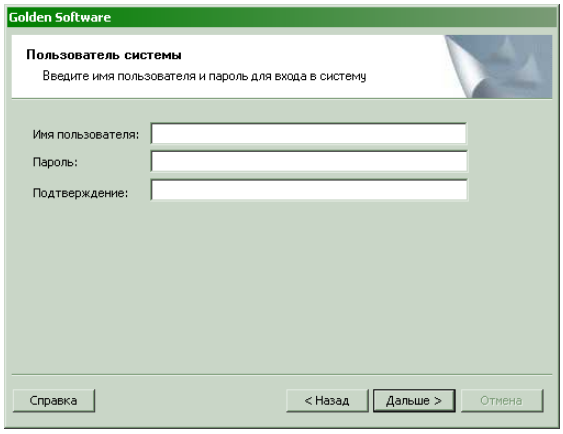
Откроется экран для заполнения банковских реквизитов организации.



Вводим код банка. Если такой код известен программе, то поля Наименование банка и Адрес будут заполнены автоматически[8]. Вводим расчетный счет его валюту. Нажимаем кнопку Дальше.

В Гедымин заложен алгоритм проверки корректности номера расчетного счета, применяемый национальным банком Республики Беларусь. Если вы используете Гедымин в другой стране — просто игнорируйте сообщение о некорректности рассчетного счета.

Откроется экран для ввода учетной записи и пароля для входа в систему.

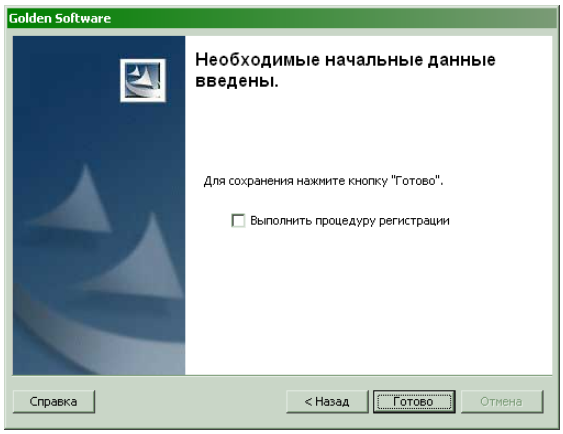


[8] База данных программы Гедымин содержит сведения о всех белорусских банках по состоянию на начало 2004 г.

Созданная учетная запись включается в группу Администраторы. При необходимости вы можете позже включить данную запись в другую группу или группы пользователей.

По окончании ввода нажимаем кнопку Дальше.

Откроется завершающий экран.

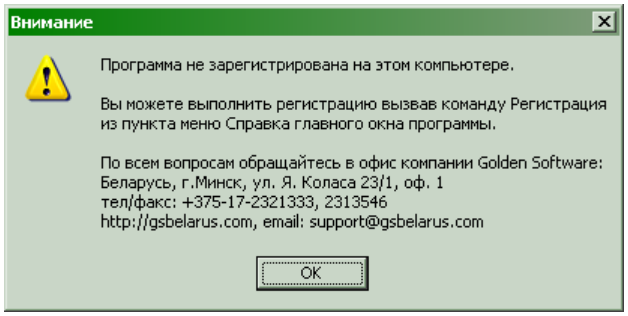


Если вся информация была введена корректно — нажимаем кнопку Готово.

Если был установлен флажок Выполнить процедуру регистрации, то откроется окно регистрации программы (привязки ее к данному компьютеру).

Регистрация программы

Гедымин защищен от несанкционированного копирования путем привязки к параметрам компьютера. При запуске незарегистрированной копии на экран выдается следующее предупреждение:



После нажатия на кнопку Ок вы можете продолжить работу с программой. Единственное ограничение с которым вы столкнетесь — это отсутствие возможности посылать отчеты на печать, хотя вы сможете просматривать их на экране.

Для выполнения привязки программы к компьютеру необходимо открыть меню «Справка» и вызвать команду «Регистрация…».

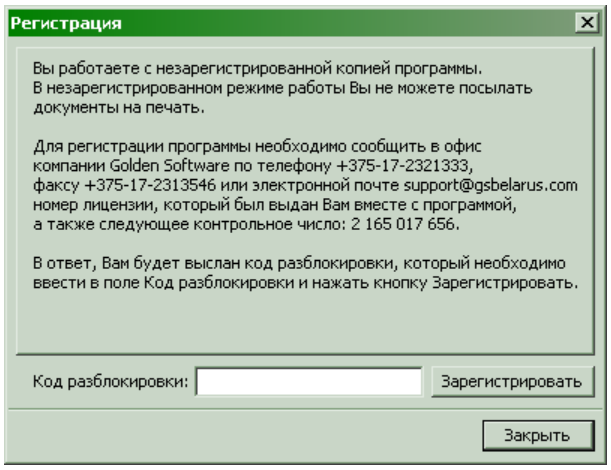


Рис. 2. Диалоговое окно "Регистрация"

Следуя инструкции, приведенной в окне, вам необходимо связаться с офисом компании Golden Software и сообщить номер своей лицензии и контрольное число, указанное в тексте сообщения. Если вы приобретали программу, то номер лицензии был передан вам вместе со всеми сопроводительными документами. Если же вы не приобретали программу, то сотрудник компании выдаст вам демонстрационную лицензию, ограниченную по времени действия. В ответ на сообщенную информацию вам будет выдан код разблокировки, который необходимо ввести в соответствующее поле внизу окна и нажать кнопку «Зарегистрировать». На экран будет выдано сообщение о результате выполнения операции. Либо программа была успешно зарегистрирована, либо вы допустили ошибку при вводе кода. В последнем случае процедуру регистрации следует повторить. Регистрацию так же придется повторить в случае, если конфигурация компьютера была изменена. Если вы просто переустановили операционную систему, то можно воспользоваться прежним кодом разблокировки. Поэтому рекомендуется сохранять его.

Удаление программы

Если Гедымин больше не нужен вам, вы сможете удалить его зайдя в раздел «Установка и удаление программ» на панели управления Windows. Найдите Гедымин в списке установленных программ и нажмите кнопку «Изменить/удалить».

Обратите внимание, что в целях безопасности программа удаления никогда не удаляет файл базы данных. Вы должны будете сделать это вручную.

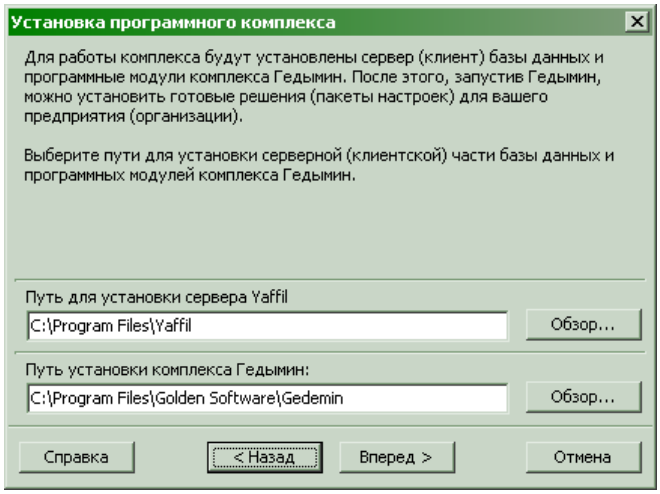
Сетевая установка

Установка многопользовательской сетевой версии Гедымина подразумевает:

1. Установку серверной части (сервер Interbase и файл базы данных) на компьютер сервер;
2. Установку клиентской части Гедымина на каждой рабочей станции.

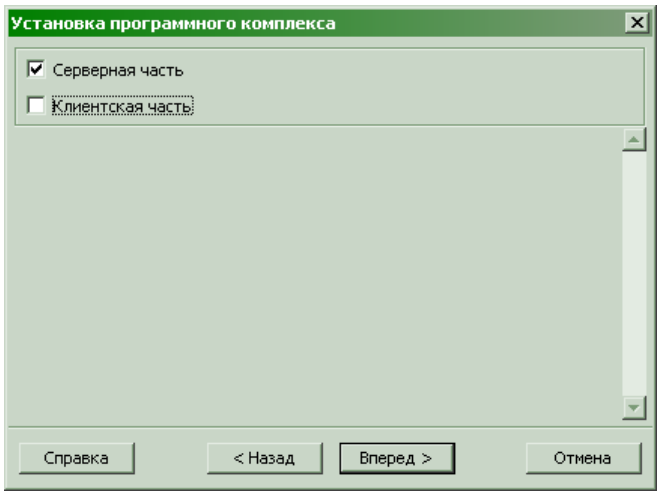
Установка серверной части

Запускаем инсталлятор сетевой версии платформы Гедымин. На экране возникнет диалоговое окно для ввода путей размещения файлов сервера Interbase и файла базы данных:



Вводим свои пути размещения или оставляем предложенные программой. Нажимаем кнопку «Вперед».

На экране появится окно выбора типа установки:



Отмечаем только флажок «Серверная часть» и нажимаем кнопку «Вперед».

Начнется процесс копирования файлов, ход которого отображается на экране. В процессе установки файла базы данных вам будет предложено выбрать папку назначения. Кроме этого вы сможете выбрать какие дополнительные базы данных следует установить. Конечно же, если дополнительные базы данных присутствуют на вашем установочном диске.

Если сервер Interbase устанавливается в первый раз, то вам будет предложено изменить пароль для учетной записи SYSDBA. Не принебрегайте этой возможностью. SYSDBA — это «супер пользователь» Interbase, который может сделать все что угодно с вашей базой данных. Его пароль по-умолчанию — masterkey — не является тайной и хорошо известен всем, в том числе и злоумышленникам, которые могут покуситься на вашу информацию.

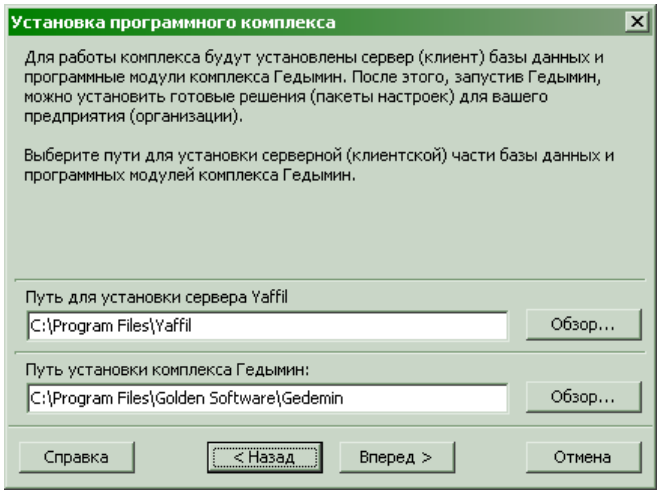
По завершении установки на экран будет выдано сообщение об успехе или информация о возникшей ошибке, если установку не удалось провести по какой-либо причине.

Подробная информация о ходе установки записывается в файл C:\Program Files\Golden Software\Gedemin\install.log. Файл может располагаться в другой папке, если вы изменяли путь для установки комплекса Гедымин, предложенный по умолчанию.

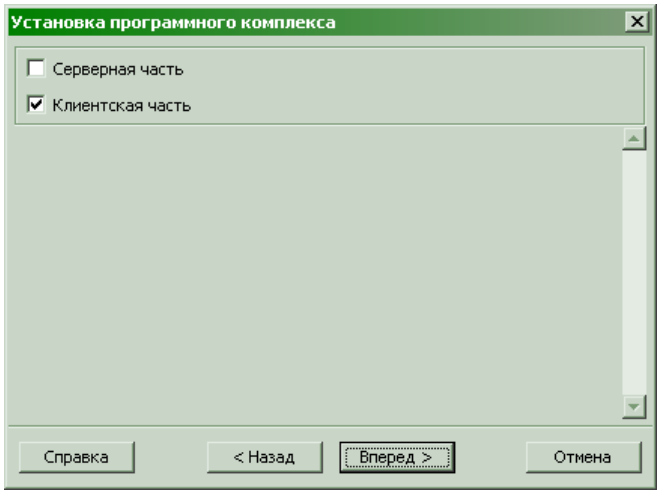
Установка клиентской части

После установки серверной части необходимо установить клиентскую часть Гедымина на каждую рабочую станцию в вашей сети.

Запускаем тот же инсталлятор на рабочем месте. Снова на экране возникнет окно ввода путей назначения:

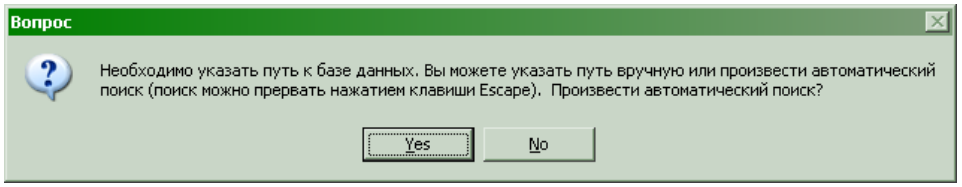


И окон выбора вида установки:

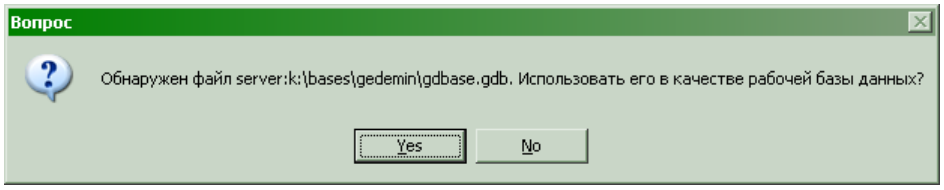


В этот раз отметим только флажок «Клиентская часть» и нажмем кнопку «Вперед».

Начнется процесс копирования файлов ход которого будет отражаться на экране. После того как необходимые файлы будут скопированы вам будет предложено установить подключение к базе данных.

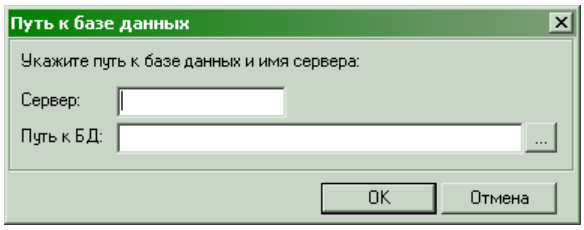


Если вы выберете режим автоматического поиска[9], то программа установки будет просматривать все компьютеры в вашей сети, пока не обнаружит тот, на котором выполняется сервер Interbase. Далее инсталлятор попытается установить соединение с базой данных. Если попытка прошла успешно, то на экране будет выведено сообщение следующего вида:



[9] Не рекомендуется производить автоматический поиск в больших сетях, поскольку построение списка компьютеров может занять весьма продолжительное время. В любом случае, вы можете прервать уже запущенный процесс нажав клавишу ESC.

В случае, если сервер не найден или вас не удовлетворяет предложенная база данных или вы изначально выбрали вариант ручной установки — на экран будет выведено окно, где вы сможеет указать имя сервера в сети и путь к файлу базы данных на сервере:



Заполните предложенные поля и нажмите Ок. Если введенная информация корректна — установка завершится успешно, о чем вы получите соответствующее уведомление.

Как и в случае с установкой серверной части лог всех выполненных действий вы найдете в файле C:\Program Files\Golden Software\Gedemin\install.log.

Установка под Windows XP SP2

Операционная система Windows XP со вторым установленным сервис паком по умолчанию защищена брандмауэром, который блокирует пересылку TCP/IP пакетов в результате чего клиентская часть Гедымина не может осуществить подключение к серверу базы данных. Для решения данной проблемы необходимо либо выключить брандмауэр (не рекомендуется), либо настроить его таким образом, чтобы он пропускал пакеты, адресованные на 3050-й[10] порт (рекомендуется).

Установка в ручном режиме

Оставим использование автоматического установщика, рассмотренного нами выше, неискушенным новичкам. Грамотный настройщик, который считает себя настоящим профессионалом, просто обязан знать всю подноготную процесса установки платформы Гедымин и уметь устанавливать ее в «ручном» режиме, имея только набор необходимых файлов.

Для того, чтобы развернуть Гедымин в сети необходимо:

1. На каждую рабочую станцию установить клиента Interbase;
2. На компьютер, выбранный в качестве сервера базы данных, установить серверную часть Interbase;
3. На компьютер, выбранный в качестве сервера базы данных, установить файл базы данных Гедымин;
4. На каждую рабочую станцию установить Гедымин.

Ниже, мы рассмотрим каждый шаг подробно, но сначала ответим на вопрос: где взять необходимые файлы? Их можно найти на установочном диске с сетевой версией платформы Гедымин в следующих подкаталогах:

Имя каталога	Содержит файлы
Database	Архивы баз данных. Список баз данных и их наименования можно найти в файле database.ini в главном установочном каталоге.
Gedemin	Файл gedemin.exe и динамическая библиотека midas.dll.
Gedemin\Help	Файлы справочной системы.
GUDF	Библиотека UDF функций компании Golden Software.
Microsoft	Динамическая библиотека msvcr70.dll и файлы для установки скриптов на разные версии операционной системы.
YAFiles	Файлы сервера Interbase.

[10] Порт номер 3050 используется сервером по-умолчанию и может быть изменен с помощью параметра командной строки –i. Более подробно, смотрите документацию по серверу Interbase.

Кроме этого можно воспользоваться интернетом. Последнюю версию сервера Interbase (Yaffil) можно найти по адресу http://yaffil.ibase.ru. Установочные пакеты подсистемы выполнения скриптов располагаются по адресу http://msdn.microsoft.com/scripting. И, конечно же, сайт http://gsbelarus.com содержит необходимые файлы для установки самой новейшей версии платформы Гедымин.

Установка клиента Interbase

Клиентская часть Interbase состоит всего из двух файлов: gds32.dll и msvcr70.dll. Опытные специалисты могут заявить, что абсолютный минимум — это библиотека gds32.dll, которую следует положить в один каталог с файлом gedemin.exe. Однако для 100%-ной гарантии правильной установки клиента необходимо все же копировать оба файла.

Файл msvcr70.dll — это одна из самых банальных динамических библиотек, которая почти всегда имеется в системе Windows. Обычно этот файл устанавливается в системный каталог Windows. Если файл с таким именем в системном каталоге уже существует, проверьте его версию. Не стоит перезаписывать более новый файл старой версией. Это может привести к неработоспособности системы.

Самым важным является динамическая библиотека gds32.dll, в которой сосредоточена вся основная функциональность, называемая нами «клиентом Interbase». Поэтому установке gds32.dll следует уделить особое внимание.

Прежде чем установить gds32.dll на компьютер, необходимо убедиться, что на компьютере нет другой копии этой динамической библиотеки. Для этого необходимо осуществить поиск этого файла в следующих каталогах: системном каталоге Windows (это Windows\System для 9x ОС и Windows\System32 для NT\2000\XP); в установочных каталогах Interbase 4.x, 5.x, 6.x; в установочном каталоге BDE, а также во всех каталогах, которые включены в переменную среды PATH.

Если будет найдена копия gds32.dll в одном з перечисленных каталогов, то необходимо выяснить ее версию и сравнить с версией gds32.dll, которую вы устанавливаете. Если устанавливаемая библиотека имеет версию новее, чем у существующего файла, то можно произвести замену старой версии на новую. Не рекомендуется заменять более новую версию старой, так как старая версия gds32.dll может неправильно взаимодействовать с более новой версией сервера.

Рекомендуется разместить билиотеку gds32.dll в системном каталоге Windows. Альтернативным решением может быть размещение билиотеки в одном каталоге с файлом gedemin.exe. В последнем случае можно вообще не искать существующие на компьютере файлы gds32.dll, так как при загрузке динамической библиотеки, операционная система в первую очередь просматривает каталог, где расположен выполняемый файл.

Установка протокола TCP/IP

Клиент Interbase подключается к серверу по протоколу TCP/IP поэтому необходимо проверить его наличие и, в случае если протокол TCP/IP не установлен, установить его. Обратите внимание, что правильно установленный и сконфигурированный протокол TCP/IP необходим даже в том случае, если и клиентская часть и сервер Interbase располагаются на одной физической машине.

Установка сервера Interbase

Для установки сервера Interbase необходимо создать каталог, где будут размещены файлы сервера. Как правило, такой каталог имеет имя c:\program files\yaffil, в дальнейшем мы будет ссылаться на этот каталог по имени <Interbase_root>. В каталоге <Interbase_root> необходимо создать два подкаталога: BIN и UDF. Далее следует скопировать следующие файлы:

Файл	Описание файла	Куда копировать
ibserver.exe	Основной исполняемый файл Interbase.	<Interbase_root>\Bin
yaconfig	Файл конфигурации Interbase.	<Interbase_root>
isc4.gdb	База данных пользователей Interbase.	<Interbase_root>
gds32.dll	Клиентская библиотека.	<Interbase_root>\Bin, системный

		каталог Windows.
msvcr70.dll	Динамическая библиотека.	Системный каталог Windows.
ib_util.dll	Динамическая библиотека, необходимая для работы с UDF функциями.	<Interbase_root>\Bin
gudf.dll	Билиотека UDF функций компании Golden Software.	<Interbase_root>\UDF

Обратите внимание, что файл gds32.dll копируется в два каталога: системный каталог Windows и <Interbase_root>\Bin. Это служит дополнительной гарантией того, что ibserver.exe при запуске обнаружит gds32.dll той же самой версии, что и у него самого.

В случае установки поверх уже существующего сервера ни в коем случае нельзя затирать существующую базу данных пользователей isc4.gdb.

Ключи в реестре для сервера Interbase

После копирования файлов необходимо создать следующие записи в реестре Windows:

Ключ	Значение
HKEY_LOCAL_MACHINE\SOFTWARE\Yaffil\Version	Номер версии сервера, например: WI-1.3.884 Yaffil SQL Server
HKEY_LOCAL_MACHINE\SOFTWARE\Yaffil\RootDirectory	Установочный каталог Interbase, например: c:\program files\yaffil\
HKEY_LOCAL_MACHINE\SOFTWARE\Yaffil\ServerDirectory	Каталог, из которого запускается ibserver.exe. Например: c:\program files\yaffil\bin\
HKEY_LOCAL_MACHINE\SOFTWARE\Yaffil\DefaultMode	-r

Параметр «DefaultMode= -r» означает, что Interbase-сервер будет запускаться в режиме сервиса NT/2000/XP. Если нужно запускать Interbase в режиме приложения, то необходимо указать ключ «–a».

Можно воспользоваться утилитой instreg.exe из поставка Interbase для создания указанных выше ключей в реестре Windows. Синтаксис вызова следующий:

```
instreg.exe install "c:\program files\yaffil"
```

Запуск Interbase-сервера

Interbase-сервер, функционирующий под управлением NT/2000/XP, может выполняться в двух режимах — в виде службы (service) и в виде приложения. На Windows 9x Interbase может использоваться только в режиме приложения. Давайте рассмотрим, как настроить и запустить наш сервер после установки. Чтобы запустить Interbase в режиме приложения, необходимо выполнить команду «ibserver.exe –a». Чтобы установить Interbase в качестве сервиса на NT\2000\XP, следует воспользоваться утилитой instsvc.exe, которая зарегистрирует Interbase-сервер в качестве службы. Как известно, служба может либо запускаться автоматически при запуске Windows, либо запускаться вручную. Для запуска сервиса в автоматическом режиме необходимо выполнить команду:

```
instsvc.exe install "c:\program files\yaffil" -auto
```

Установка базы данных

Файл базы данных должен быть расположен на сервере. Как правило, он располагается в каталоге c:\program files\golden software\gedemin\database\ и имеет имя gdbase.gdb. Если у вас есть архив (backup) базы данных, его необходимо распаковать, либо средствами Гедымина, либо с помощью утилиты командной строки gbak.exe. В последнем случае архив необходимо скопировать на сервер, например, в каталог базы данных, и выполнить следующую команду:

```
gbak.exe -r "c:\program files\golden software\gedemin\database\gdbase.bk"
"server:c:\program files\golden software\gedemin\database\gdbase.gdb" -user
SYSDBA -pas masterkey
```

Обратите внимание, что в приведенном примере использовано имя компьютера сервера «server», в вашем конкретном случае оно может отличаться, точно так же, как может отличаться имя файла архива базы данных и имя файла базы данных.

Установка платформы Гедымин

Для установки Гедымина на каждом рабочем месте необходимо создать каталог, как правило, c:\program files\golden software\gedemin, и скопировать в него два файла gedemin.exe и midas.dll.

Библиотеку midas.dll следует зарегистрировать. Для этого необходимо выполнить команду:

```
regsvr32.exe midas.dll
```

Утилита regsvr32.exe входит в стандартную поставку Windows.

Установка Microsoft Script

Гедымин использует VBScript для выполнения макросов. Установочные файлы находятся в каталоге Microsoft инсталяционного диска системы Гедымин. Их несколько, в зависимости от версии и языковых настроек используемой операционной системы.

Версия и языковые настройки операционной системы	Выполняемые команды
Windows 9x, язык английский	1. scr56en.exe /q 2. sct10en.exe /q
Windows 9x, язык русский	1. scr56ru.exe /q 2. sct10en.exe /q
Windows NT/2000/XP, язык английский	1. scripten.exe /q 2. sct10en.exe /q
Windows NT/2000/XP, язык русский	1. scriptru.exe /q 2. sct10en.exe /q

Подключение к базе данных

Осталось сделать последний шаг перед началом эксплуатации Гедымина — указать путь к файлу базы данных. Сделать это можно тремя способами:

- Вручную прописать необходимые ключи в реестре Windows;
- Запустить Гедымин и ввести необходимую информацию в окне регистрации базы данных;
- Настроить ярлык для запуска Гедымина, прописав путь к базе данных в параметрах командной строки.

Рассмотрим каждый из трех предложенных способов подробнее:

Создание ключей в реестре

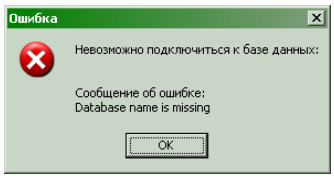
Наименование ключа	Значение
HKEY_LOCAL_MACHINE\SOFTWARE\Golden Software\Gedemin\ServerName	Сетевой путь к файлу базы данных. Например, server:c:\program files\golden software\gedemin\database\gdbase.gdb
HKEY_LOCAL_MACHINE\SOFTWARE\Golden	Сетевой путь к файлу базы данных.

Software\Gedemin\Client\CurrentVersion\ServerName	Например, server:c:\program files\golden software\gedemin\database\gdbase.gdb
HKEY_LOCAL_MACHINE\SOFTWARE\Golden Software\Gedemin\Client\CurrentVersion\Access\DatabaseName	Сетевой путь к файлу базы данных. Например, server:c:\program files\golden software\gedemin\database\gdbase.gdb
HKEY_LOCAL_MACHINE\SOFTWARE\Golden Software\Gedemin\Client\CurrentVersion\Access\ИмяБазы\Database	Сетевой путь к файлу базы данных. Например, server:c:\program files\golden software\gedemin\database\gdbase.gdb

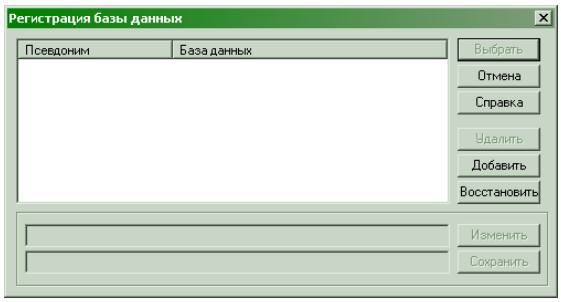
Обратите внимание, что в последнем ключе ИмяБазы — это имя базы данных, которое будет выводиться в выпадающем списке баз данных в окне ввода имени пользователя и пароля при запуске программы.

Использование окна «Регистрация базы данных»

Если просто переписать на компьютер файл gedemin.exe так, как было указано выше, и запустить его, не настраивая соответствующие ключи в реестре, то на экран будет выдано сообщение о том, что подключиться к базе данных невозможно:



После нажатия на кнопку Ок, на экране появится окно для регистрации базы данных:



Для регистрации базы данных необходимо нажать кнопку Добавить в левой части окна и ввести отображаемое имя базы данных (ее псевдоним) и полный сетевой путь к ней в двух полях внизу окна. После чего следует нажать кнопку Сохранить. База данных будет зарегистрирована и появится в списке. Для того, чтобы закрыть окно регистрации необходимо нажать кнопку Выбрать.

Настройка ярлыка для запуска Гедымина

Указать путь к базе данных можно в параметрах командной строки. Для того, чтобы не вводить их каждый раз следует создать ярлык на рабочем столе Windows и заполнить соответствующие поля. Например так:

Следующие параметры командной строки поддерживаются Гедымином:

Параметр	Описание
/sn <database>	Подключаться к указанной базе данных. Например, /sn “server:c:\program files\golden software\gedemin\database\gdbase.gdb”
/user <user_name>	Использовать указанное имя пользователя при подключении. Например, /user Luda
/password <password>	Использовать указанный пароль пользователя при подключении. Например, /password 123
/ns	Не выводить на экран заставку при запуске программы.
/lang	Установка языка пользовательского интерфейса программы. Примеры использования: /lang:by — белорусский язык; /lang:en — английский язык.
/langfile	Файл с переводом интерфейса. Если параметр не указан будет использоваться файл local.xml из того каталога, где находится файл gedemin.exe. Пример использования: /langfile:c:\myfile.xml
/langsave	По выходу из Гедымина сохраняет в файле с переводом пользовательского интерфейса все строки, которые не были переведены (отсутствовали в файле на момент загрузки программы).

Очевидно, что вместо настройки ярлыка можно создать BAT файл, содержащий единственную строку — вызов Гедымина с соответствующими параметрами командной строки.

Настройка параметров программы

Основные параметры Гедымина можно просмотреть или изменить, если вызвать команду Опции из выпадающего меню Сервис, которое расположено в главном меню. Диалоговое окно Опции имеет четыре закладки:

- Общие;

- Аудит;
- Политики;
- Архивное копирование;

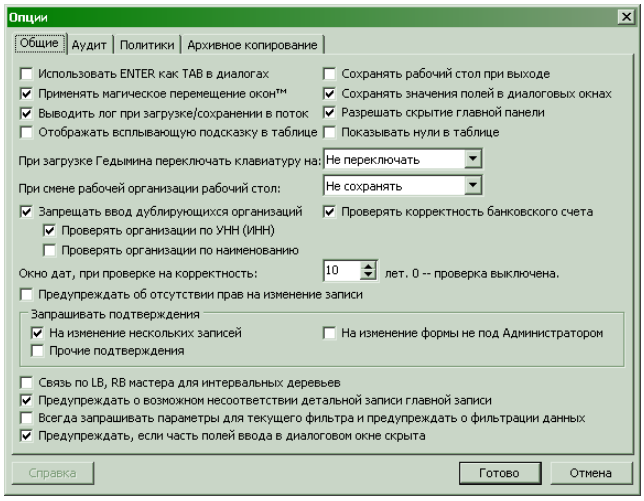


Рис. 3 Диалоговое окно Опции

В зависимости от того, входит ли текущий пользователь в группу Администраторы или нет некоторые закладки и опции системы могут быть недоступными для изменения.

Рассмотрим содержимое каждой закладки.

Общие

Использовать Enter как Tab в диалогах

Во многих старых ДОСовских программах клавиша Enter служила для перемещения к следующему полю при вводе данных в экранную форму. Поскольку, многие пользователи испытывают определенный дискомфорт при переходе со старого текстового интерфейса на графический интерфейс операционной системы Windows в Гедымине предусмотрен режим эмуляции ДОС, при котором курсор перемещается между полями ввода в диалоговых окнах не только с помощью клавиши табуляции, но и при нажатии на клавишу Enter.

Для того, чтобы подтвердить ввод и закрыть окно при включенном режиме эмуляции используя клавиатуру необходимо либо установить фокус на кнопку Готово (Ок) и нажать Enter, либо воспользоваться коомбинацией клавиш Ctrl+Enter. Обратите внимание, что если курсор находится в мемо поле, то нажатие клиавиши Enter приведет к переходу на новую строку, а не к перемещению на следующее поле в окне.

Опция устанавливается только для текущего пользователя системы.

Применять магическое перемещение окон

Режим магического перемещения окон позволяет быстро и красиво расположить окна программы на рабочем столе. Если режим включен, то:

- При двойном щелчке на заголовке исследователя он занимает место слева под главным окном программы, причем его высота устанавливается во всю доступную выстоту экрана;
- При двойном щелчке на заголовке формы просмотра, она занимает все доступное место на экране, ограниченное с двух сторон главным окном программы и окном исследователя;
- При изменении размеров формы просмотра, они будут автоматически выравниваться по другим формам, присутствующим на экране;
- При перемещении формы просмотра, ее положение будет автоматически выравниваться по другим формам, присутствующим на экране;

Если надо временно отключить режим магического перемещения, например, для того, чтобы быстро раскрыть окно на весь экран или переместить его произвольным образом, достаточно нажать и удерживать клавишу Ctrl во время выполнения соответствующих действий.

Опция устанавливается только для текущего пользователя системы.

Выводить лог при загрузке/сохранении в поток

При сохранении бизнес объекта на диске или загрузке его с диска в базу данных на экране отображается окно с информацией о ходе процесса. Если в нем нет особой нужды, просто снимите данную опцию.

Опция устанавливается только для текущего пользователя системы.

Отображать всплывающую подсказку в таблице

Если поле содержит данных больше, чем можно отобразить в клетке таблицы, то для того, чтобы просмотреть все данные необходимо навести указатель мыши на такую клетку и подождать около секунды. На экране появится всплывающая подсказка. Если она создает неудобства в работе, то ее можно отключить с помощью данной опции программы.

Опция устанавливается только для текущего пользователя системы.

Сохранять рабочий стол при выходе

Как следует из названия опции, если она установлена, то текущая конфигурация окон или, по другому, рабочий стол, будет запоминаться при выходе из программы и восстанавливаться при следующем входе в нее.

Опция устанавливается только для текущего пользователя системы.

Сохранять значения полей в диалоговых окнах

Включение данной опции позволяет облегчить и упростить ввод большого объема однородных данных. Значения, введенные пользователем при добавлении записи, будут запоминаться и подставляться в соответствующие поля при вводе следующей записи.

Опция устанавливается только для текущего пользователя системы.

Разрешать скрытие главной панели

Данная опция влияет на то, как будет вести себя программа, если в окне, отображающем данные, связанные как мастер-дитэйл, скрыть главную панель. Если опция установлена, то Гедымин запомнит состояние панели и при следующей загрузке она будет скрыта. Если нет — то при следующей загрузке программы главная панель будет открыта. Напомним, что скрытие главной панели позволяет отобразить в детальном списке все имеющиеся в базе данных записи, а не только те, которые относятся к конкретной записи в главном списке.

Опция устанавливается только для текущего пользователя системы.

Показывать нули в таблице

Если опция не установлена, то для числовых полей в таблицах не будут выводиться значения, если они равны нулю.

Опция устанавливается только для текущего пользователя системы.

При загрузке Гедымина переключать клавиатуру на

Данная опция позволяет выбрать язык ввода, на который будет переключаться операционная система при загрузке Гедымина.

Опция устанавливается только для текущего пользователя системы.

При смене рабочей организации рабочий стол

С помощью данной опции можно настроить поведение рабочего стола при смене активной рабочей организации: рабочий стол будет сохраняться, не сохраняться или пользователь будет запрашиваться о необходимом действии.

Опция устанавливается только для текущего пользователя системы.

Запрещать ввод дублирующихся организаций

Серьезной проблемой при эксплуатации системы является появление двух и более записей в базе данных, представляющих один и тот же объект. Например, один пользователь может ввести организацию как ООО «Рога и Копыта», а второй — «Рога и Копыта» ООО. Гедымин позволяет избежать появления дублирующихся записей путем проверки информации при вводе.

Опция устанавливается для всех пользователей системы.

Проверять организацию по УНН (ИНН)

Проверка будет осуществляться по коду УНН (ИНН, УНП) организации. Если в базе уже есть организация с кодом, как у вновь вводимой, то на экран будет выдано предупреждение. Пользователь может настоять на сохранении записи с дублирующимся кодом или вернуться в окно редактирования с тем, чтобы изменить код или вообще отказаться от ввода новой записи.

Опция устанавливается для всех пользователей системы.

Проверять организацию по наименованию

Аналогично проверке по коду УНН, только проверка будет осуществляться по наименованию организации.

Опция устанавливается для всех пользователей системы.

Проверять корректность банковского счета

Если проверка корректности номера банковского счета включена, то при попытке ввести некорректный номер счета пользователю будет выдано предупреждение.

В Беларуси применяются 13-ти значные банковские счета в номере которых первые 12 цифр содержат, собственно, номер счета, а последняя, тринадцатая, является контрольной, рассчитываемой особым образом на основании 12-ти предыдущих и трехзначного кода банка. Таким образом, вероятность того, что ошибочно введенный номер счета пройдет проверку на корректность не превышает 10%.

Очевидно, что данную проверку стоит отключить при эксплуатации программы за пределами Беларуси.

Опция устанавливается для всех пользователей системы.

Окно дат, при проверке на корректность

Часто, пользователи допускают ошибки при вводе дат, набирая не тот год, например, 2094 вместо 2004. Количество таких ошибок можно существенно снизить, если установить окно корректных дат.

Опция устанавливается для всех пользователей системы.

Предупреждать об отсутствии прав на изменение записи

Запрашивать подтверждения

На изменение нескольких записей

Гедымин позволяет вносить изменения в несколько записей одновременно. Для этого необходимо выделить нужные строки в таблице удерживая клавишу Ctrl и щелкая по ним левой кнопкой мыши или используя Shift и перемещаясь с помощью стрелок на клавиатуре и выбрать команду Изменить. Если данная опция включена, то при попытке открыть для редактирования более одной записи на экран будет выдано соответствующее предупреждение.

Опция устанавливается только для текущего пользователя системы.

На изменение формы не под администратором

В Гедымине можно изменить (настроить) любую экранную форму. Изменения, выполненные под учетной записью адмнистратора распространяются на всех пользователей, в то время, как изменения, сделанные под любой другой учетной записью будут действовать только для нее. Соответствующее

предупреждение будет выведено на экран, при попытке изменить экранную форму не под администратором, если рассматриваемая опция включена.

Опция устанавливается только для текущего пользователя системы.

Прочие подтверждения

Данная опция позволяет управлять выводом на экран всех прочих предупреждений.

Опция устанавливается только для текущего пользователя системы.

Связь по LB, RB мастера для интервальных деревьев

Как известно, в интервальных деревьях связь между родительской и дочерними записями можно установить по полям LB и RB (левой и правой границе интервала родительской записи). В этом случае будут отобраны все дочерние записи (включая дочерние дочерних, дочерние дочерних дочерних и т.д.), относящиеся к данной родительской. С технической точки зрения отбор может быть выполнен двумя способами: в детальный запрос передаются непосредственно значения левой и правой границ родительской записи и в детальный запрос передается идентификатор родительской записи, которая через JOIN по условию вхождения в интервал объединяется с дочерними записями. Очевидно, что первый способ приведет к более быстрому запросу на извлечение дочерних записей (на один JOIN меньше), зато второй способ гарантирует всегда правильное извлечение дочерних записей, даже если границы родительской записи изменились, например вследствие добавления новой записи другим пользователем системы, а датасет не был обновлен.

Если опция включена, то будет использоваться связь по непосредственным значениям границ интервала родительской записи. В противном случае, связь будет устанавливаться по идентификатору родительской записи.

Опция устанавливается для всех пользователей системы.

Предупреждать о возможном несоответствии детальной записи главной записи

Если опция активна и в детальный объект добавляется запись, которая несоответствует текущей записи в главном объекте, то на экран будет выдано предупреждение о том, что добавленная запись не будет видна в списке («исчезнет» из него) после обновления, перечитывания данных.

Опция устанавливается только для текущего пользователя системы.

Всегда запрашивать параметры для текущего фильтра и предупреждать о фильтрации данных

Если данная опция установлена, то при открытии формы просмотра, данные которой отфильтрованы с помощью фильтра с параметрами, на экран будет выведено окно для ввода этих параметров. Если фильтр не параметризованный, то на экран будет выведено предупреждение о том, что список отфильтрован.

Если опция не активна, то пользователь не будет предупрежден о примененной фильтрации и параметрам, если они требуются, будут присвоены предыдущие, сохраненные значения.

Данная опция была введена в программу, когда разработчики столкнулись с большим количеством обращений от пользователей вроде: «куда пропали мои данные?» или «я вчера ввел документ, а сегодня не вижу его в списке».

Напомним, что если к данным применена фильтрация, то пиктограмка «воронка» на панели инструментов меняет свой цвет с серого на красный и название фильтра выводится в строке состояния окна. Более подробная информация о примененном фильтре выводится во всплывающей подсказе, если навести указатель мыши на кнопку фильтрации на панели инструментов или на строку состояния.

Опция устанавливается только для текущего пользователя системы.

Предупреждать, если часть полей ввода в диалоговом окне скрыта

В сложных диалоговых окнах сверху может отображаться набор закладок с полями ввода, а под ними — таблица с данными. Размер верхней и нижней части можно изменять, перетягивая разделительную полосу вверх-вниз. Можно уменьшить размер верхней части так, что не все поля будут видны, что, в свою очередь, может привести к нежелательным ошибкам, при вводе данных пользователем. Если данная опция включена, то в таких случаях, при открытии диалогового окна, будет выдаваться

предупреждение и предложение пользователю автоматически скорректировать размеры верхней и нижней частей окна с тем, чтобы все поля были видны на экране.

Опция устанавливается только для текущего пользователя системы.

Аудит

Программная платформа, предназначенная для эксплуатации в промышленных условиях, просто обязана иметь функции аудита действий пользователя и регистрации изменений данных в базе. Сказанное выше в полной мере относится к Гедымину. Его возможности позволяют:

- Регистрировать действия пользователя, такие как:
  - вход в программу,
  - открытие диалогового окна для изменения записи,
  - выход из диалогового окна по нажатию кнопки Ок или Отмена,
  - удаление записи,
  - объединение записей,
  - построение отчета,
  - печать отчета,
  - и др.
- Регистрировать изменения данных в таблицах. Для регистрации изменений на каждую таблицу создаются по три триггера для отслеживания, соответственно, операций редактирования, добавления и удаления записей;
- Блокировать определенный период. Блокировка распространяется на следующие таблицы:
  - ac_entry — бухгалтерские проводки;
  - gd_document — документы;
  - inv_card — карточки складского учета;
  - inv_movement — складское движение.

Настройка аудита, регистрации изменений данных и периода блокировки осуществляется на закладке Аудит, диалогового окна Опции.

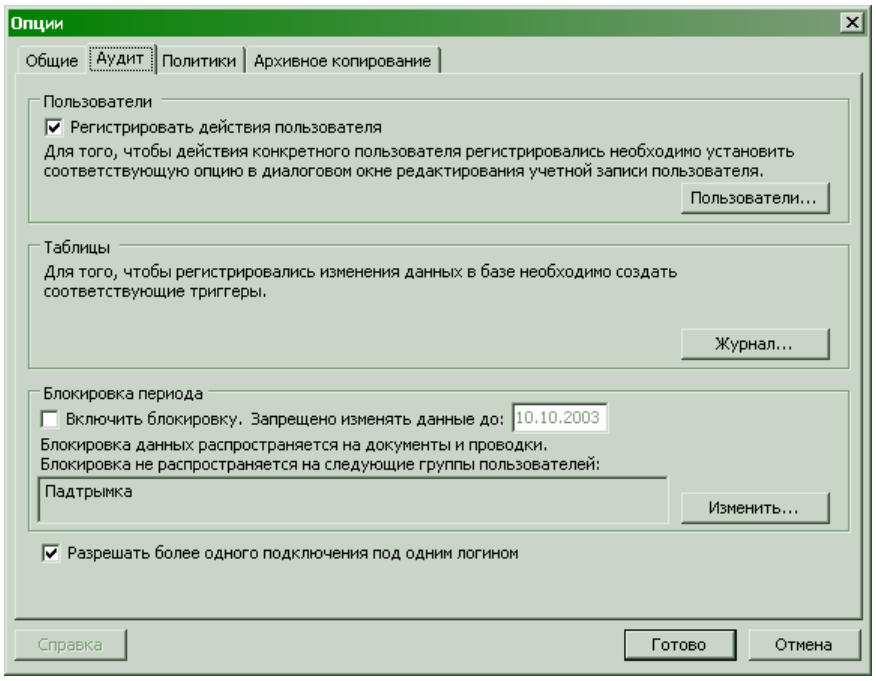


Рис. 4. Диалоговое окно "Опции". Закладка "Аудит".

Обратите внимание, что для регистрации действий пользователя необходимо выполнить две операции:

1. Установить флаг «Регистрировать действия пользователя»;
2. С помощью кнопки «Пользователи…» перейти в окно со списком пользователей системы и установить флаг «Регистрировать операции пользователя» для тех учетных записей, действия которых необходимо заносить в журнал.

Для регистрации изменений в базе данных необходимо с помощью кнопки «Журнал…» перейти в журнал событий и создать необходимые триггеры, воспользовавшись соответствующей командой на панели инструментов.

Создавать триггеры для регистрации изменений в базе данных можно только под учетной записью Администратор.

Для активизации блокировки периода необходимо установить флаг «Включить блокировку» и указать дату окончания периода блокировки. Все хозяйственные операции, проводки, складское движение, произведенные до указанной даты будут заблокированы. Изменить или удалить их будет невозможно.

Включать или выключать режим блокировки периода можно только под учетной записью Администратор.

Дополнительно, можно указать список групп пользователей, на которых не будет распространяться блокировка периода.

Внизу закладки находится флаг «Разрешать более одного подключения под одним логином». Если его снять, то пользователь не сможет несколько раз одновременно подключиться к Гедымину под одним и тем же логином. Применение данной опции позволяет снизить риск несанкционированного доступа, а также избежать путаницы, которая может возникнуть, когда несколько лиц входят в программу и работают в ней под одной и той же учетной записью.

Политики

Политики групповой безопасности позволяют разграничить доступ пользователей к тем или иным функциям системы.

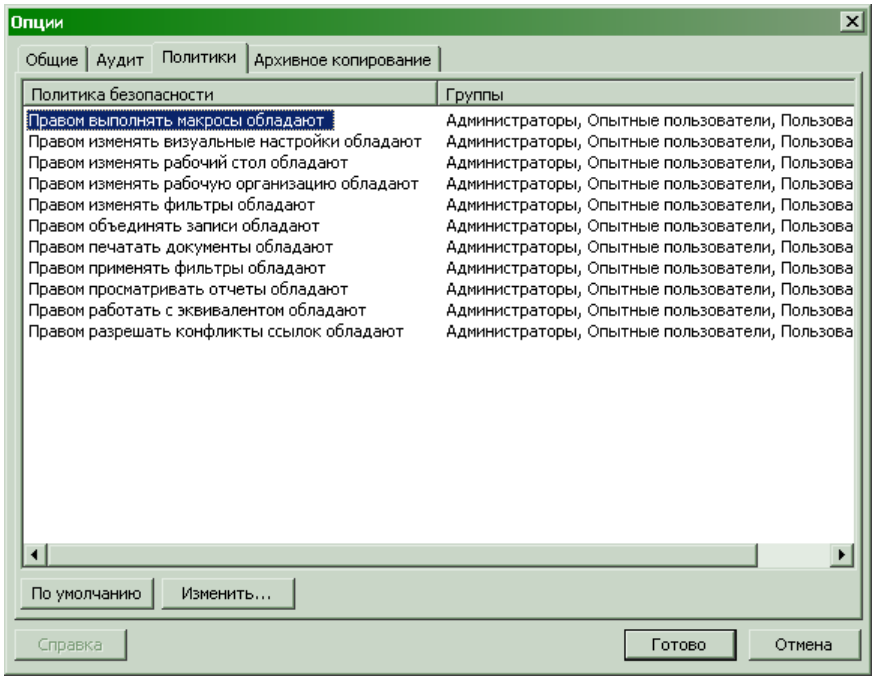


Рис. 5. Диалоговое окно "Опции". Закладка "Политики"

Для установки конкретной политики необходимо установить курсор на нее в списке и выбрать кнопку «Изменить…» внизу закладки. Откроется окно выбора со списком групп пользователей. Напротив наименований тех групп, которые мы хотим включить в данную политику необходимо установить галочки.

Кнопка «По умолчанию» позволяет вернуть выбранной политике значения, поставляемые в исходном состоянии системы.

Архивное копирование

Выполнять регулярное архивное копирование базы данных необходимо по двум причинам: во-первых, насколько бы ни был надежен сервер Interbase все равно возможно повреждение файла базы данных и, как следствие, потеря всей или части информации. Основной причиной повреждения базы, как правило является сбой электропитания или принудительное несанкционирование выключение сервера в момент, когда происходила запись данных на диск. Во-вторых, наличие архивной копии, позволит восстановить данные, утерянные или поврежденные в следствие ошибочных или злонамеренных действий пользователей.

Гедымин обладает функцией автоматического архивного копирования, настроить которую можно в окне Опции на закладке Архивное копирование.

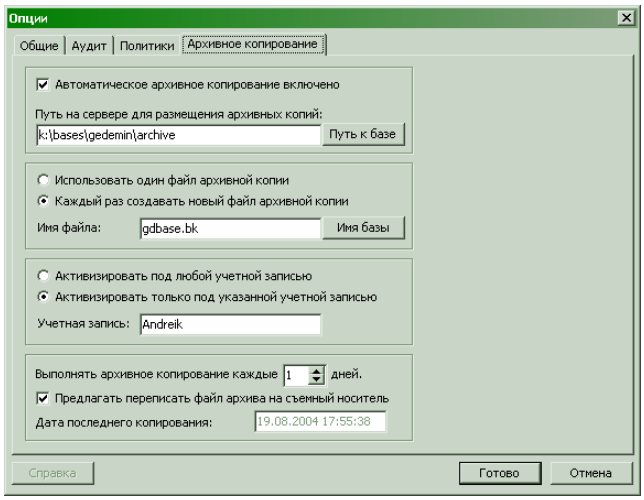


Рис. 6 Опции архивного копирования.

Для того, чтобы активизировать архивное копирование необходимо:

1. Установить флаг «Автоматическое архивное копирование включено»;
2. Указать путь к папке на сервере, где будут размещаться архивные файлы. Обратите внимание, что архивные файлы могут создаваться только на компьютере, где выполняется сервер Interbase, т.е. нельзя указать имя папки на другом сетевом компьютере. С помощью кнопки «Путь к базе» можно быстро ввести в поле имя папки, где располагается файл текущей базы данных;
3. Автоматическое архивное копирование может создавать каждый раз отдельный архивный файл или всегда использовать один и тот же файл. Очевидно, что использование одного файла существенно повышает вероятность потери данных, особенно в случае логических сбоев в алгоритмах программы или злонамеренных действий пользователей. Использование одного файла оправдано только в случае, когда на диске, где размещается архив, мало свободного места или если системный администратор регулярно копирует архивный файл на съемный носитель информации. Если выбрана опция «Каждый раз создавать новый файл архивной копии», то архивные файлы будут иметь имена вида <Имя файла>ГГГГММДД.<Расширение файла>, где имя файла и его расширения задаются в поле «Имя файла», а ГГГГММДД — это дата создания данного архива. Если нажать кнопку «Имя базы», то в поле «Имя файла» будет записано имя файла текущей базы данных с расширением BK.
4. Ввести имя учетной записи при загрузке под которой будет стартовать архивное копирование, либо настроить архивное копирование так, чтобы оно запускалось при входе в систему под любой учетной записью.
5. Указать периодичность (в днях) с которой будут создаваться архивные копии.
6. Активизировать или нет опцию «Предлагать переписать файл на съемный носитель», в зависимости от того, хотите ли вы, чтобы каждый раз после создания архива на экран выдавалось напоминание с информацией о том, где расположен только что созданный файл.

После того, как все поля заполнены надлежащим образом необходимо нажать кнопку Готово. Архивное копирование будет выполнено при очередном входе пользователя в программу.

Получение информации о системе

Получить информацию о версии используемого файла gedemin.exe, версии и расположении клиентской части Interbase, прочих параметрах платформы и системного окружения можно в окне «О системе», которое вызывается из пункта «Справка» главного меню. Данное диалоговое окно имеет следующие закладки:

- О программе;
- Файлы;
- Переменные среды;
- Подключение;
- База данных;

Если открыть окно «О системе», когда нет активного подключения к базе данных — часть закладок будет недоступна.

Рассмотрим подробнее содержимое каждой закладки:

О программе

Данная закладка содержит общую информацию о платформе, сведения об авторских правах, контактные реквизиты компании Golden Software и список разработчиков, принимавших участие в создании Гедымина.

Внизу окна находится кнопка «О системе», которая вызывает стандартную утилиту Windows System Information. С ее помощью можно получить подробные сведения об аппаратной части компьютера, установленном программном обеспечении и параметрах операционной системы.

Файлы

Закладка «Файлы» содержит сведения и клиенте и сервере Interbase, а также о версии и расположении файла gedemin.exe.

Библиотека GDS32.DLL

Расположение	Расположение клиента Interbase, который используется программой. Данная информация может быть весьма полезна при разрешении конфликтов, вызванных наличием на компьютере нескольких версий клиентской библиотеки. Напомним, что более старые клиентские библиотеки могут некорректно взаимодействовать с более новой версией сервера. Всегда рекомендуется использовать клиента той же версии, что и сервер.
Версия	Номер версии клиента Interbase.
Описание файла	Строка с описанием клиентской библиотеки. От сюда, в частности, можно выудить информацию о типе библиотеки. Например, встроенный сервер имеет подстроку “(EM)” в своем описании.

Сервер

Версия сервера	Информация о версии сервера.
Имя файла БД	Полное имя файла базы данных на сервере. Включает букву диска и полный путь.
Имя компьютера	Сетевое имя сервера.
ODS версия	Версия дисковой структуры базы данных. ODS — On Disk Structure.
Размер страницы	Размер страницы базы данных в байтах.
Принудительная запись	Состояние режима принудительной записи (Forced Writes). 1 — включен, 0

	— выключен. Режим принудительной записи снижает вероятность потери данных в случае аварийного выключения компьютера, где находится сервер базы данных. В случае, если режим включен, Interbase будет дожидаться подтверждения успешной записи на диск от операционной системы, если режим выключен — данные попадут в кэш и Windows будет сама решать когда записать их на винчестер. Обратная сторона медали — это снижение производительности при использовании принудительной записи.
Количество буферов	Количество страниц в кэше сервера для текущей базы данных.
Используемая память	Используемая память под кэш сервера Interbase для текущей базы данных. Размер выводится в байтах и рассчитывается как произведение размера страницы на количество буферов.

Гедымин

Имя файла	Имя текущего файла. Как правило, gedemin.exe.
Расположение	Расположение текущего выполняемого файла.
Версия файла	Информация о версии файла.

Переменные среды

На данной закладке находится информация о переменных среды (переменных окружения) Windows, которые могут влиять на работу Гедымина или клиента Interbase.

ISC_USER	Имя пользователя Interbase, используемое по умолчанию при вызове утилит командной строки.
ISC_PASSWORD	Пароль пользователя Interbase, используемый по умолчанию при вызове утилит командной строки.
ISC_PATH	Путь к базе данных на сервере, используемый по умолчанию утилитами командной строки.
TEMP	Размещение каталога временных файлов. Гедымин использует временные файлы для хранения кэша макросов.
TMP	Размещение каталога временных файлов. Гедымин использует временные файлы для хранения кэша макросов.
PATH	Изучение содержимого перенной PATH может помочь в тех случаях, когда на компьютере присутствуют несколько версий библиотеки gds32.dll и необходимо разобраться какая из них загружается программой и почему?

Подключение

На закладке Подключение раполагается информация о параметрах текущего подключения к базе данных и о параметрах командной строки (если они есть).

Учетная запись	Имя учетной записи, используемое для текущего подключения.
Контакт	Имя контакта, связанного с текущей учетной записью.
Пользователь ИБ	Имя пользователя Interbase, под которым осуществлено текущее подключение.
ИД учетной записи	Идентификатор текущей учетной записи, т.е. идентификатор записи в таблице gd_user.
ИД контакта	Идентификатор контакта, связанного с текущей учетной записью, т.е. идентификатор записи в таблице gd_contact.
Сессия	Номер текущей сессии. Целое число, которое хранится в генераторе GD_G_SESSION и увеличивается на 1 при каждом подключении к базе

	данных.
Дата и время покдлючения:	Дата и время подключения к базе данных.
Командная строка	Полная командная строка, использованная для запуска gedemin.exe. Содержит имя исполняемого файла и параметры, если они были указаны.

База данных

На данной закладке находится информация о файле базы данных.

Версия файла БД	Версия структуры текущего файла базы данных.
ИД файла БД	Идентификатор базы данных. Целое положительное число, хранится в генераторе GD_G_DBID. Обратите внимание, что идентификатор базы данных может измениться, например, после выполнения бакапа/восстановления базы данных средствами Гедымина.
Дата релиза БД	Дата релиза текущей структуры базы данных.
Комментарий	Строка комментария структуры текущей базы данных.
Параметры подключения	Параметры подключения к базе данных. Как правило, это: Lc_ctype=win1251 — кодовая таблица Windows для кирилических шрифтов; sql_role_name=ADMINISTRATOR — подключение под ролью Administrator;

Организация пользовательского интерфейса

Гедымин имеет многооконный пользовательский интерфейс. Однородная информация, такая как, например, список компаний, платежных документов или хозяйственных операций, располагается в отдельных окнах, называемых так же окнами (экранными формами) просмотра. Существует несколько базовых (абстрактных) типов окон просмотра, от которых наследуются окна для отображения объектов конкретного типа. Основными базовыми типами окон просмотра являются:

- простое окно с таблицей;
- окно с двумя таблицами для отображения данных, связанных с помощью связи мастер-дитэйл. Такие окна бывают в вертикальном или горизонтальном исполнении;
- окно с деревом и таблицей для отображения связанных данных.

Базовые типы окон просмотра служат для облегчения труда разработчика и унификации интерфейса пользователя, однако при необходимости разработчик может всегда создать свое окно что называется «с нуля» в соответствии со своим вкусом и требованиями конкретной задачи.

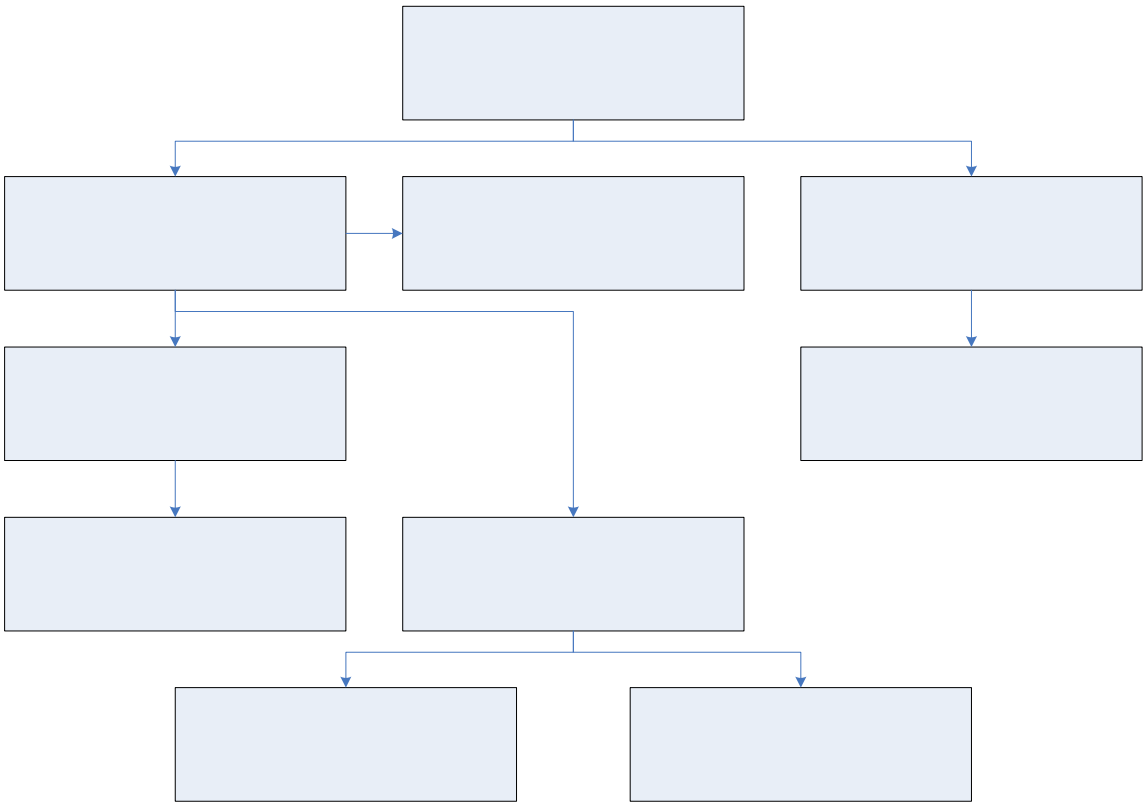


Рис. 7 Иерархия базовых классов окон просмотра.

Tgdc_frmG
Базовая форма пр

Пользовательский интерфейс Гедымина организован в стиле SDI (Single Document Interface). Его прообразом послужил пользовательский интерфейс Borland Delphi 5. Главное окно программы, как правило, располагается вверху экрана, занимает всю его ширину и имеет небольшую высоту. Окна просмотра располагаются обособленно от главного окна программы на рабочем столе Windows. Размер окон просмотра может быть изменен, при этом становится доступной часть рабочего стола Windows или окна других программ, располагающиеся под Гедымином. Такое устройство пользовательского интерфейса особенно удобно, если необходимо организовать одновременную работу с несколькими приложениями. Например, работать с Гедымином и иметь перед глазами данные, находящиеся в электронной таблице Excel.

Tgdc_frmMDH
Базовая форма просмотра для
отображения данных мастер-дитэйл,
горизонтальное расположение

Tgdc_frmMD
Базовая форма прос
отображения данных м
горизонтальное рас

Обычно, при работе с Гедымином, экран имеет следующий вид:

Tgdc_frmMDHGr
Базовая форма просмотра для

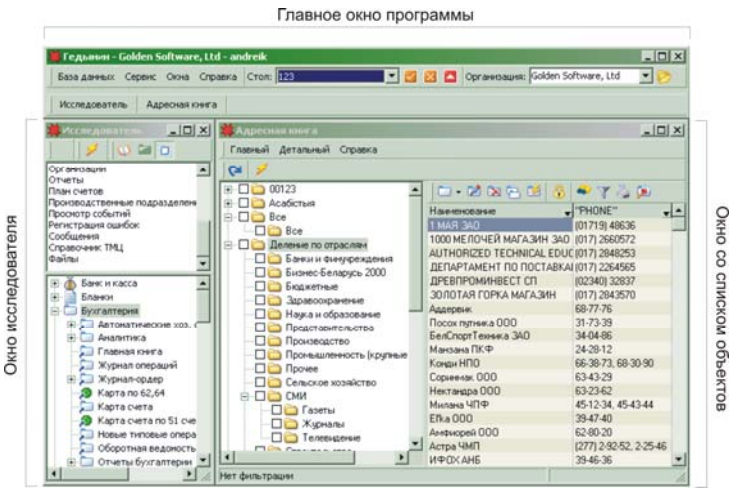


Рис. 8 Содержимое экрана при работе с Гедымином.

Обратимся к рисунку. На нем мы видим три открытых окна системы Гедымин:

- Главное окно программы;
- Окно Исследователя;
- Окно просмотра информации из адресной книги.

Опишем каждое из этих окон:

Главное окно программы

Главное окно программы предназначено для вызова основных команд, отображения информации о текущей рабочей организации и текущем пользователе системы, выбора и управления рабочими столами, выбора или изменения текущей рабочей организации, а также для отображения списка загруженных в настоящий момент времени окон просмотра.

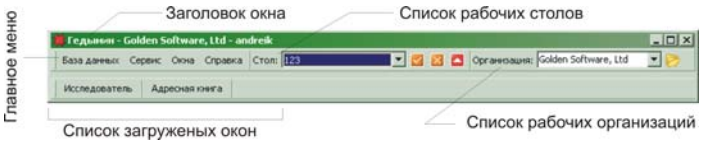


Рис. 9 Главное окно программы.

Опишем составляющие части главного окна программы:

Заголовок окна

Заголовок главного окна содержит наименование программы, наименование текущей рабочей организации, а также наименование учетной записи пользователя программы.

Главное меню

Главное меню программы содержит четыре выпадающих подменю:

- База данных;
- Сервис;
- Окна;
- Справка.

Ниже приводится список команд с описанием для каждого из подменю.

База данных

Команда	Описание
Подключенные пользователи	Выводит на экран окно со списком подключенных к базе данных

	пользователей. Команда доступна только для пользователя, входящего в группу Администраторы.
Подключиться к базе данных	Команда позволяет выполнить подключение к другой базе данных не перезапуская Гедымин. Перед подключением необходимо отключиться от текущей базы данных с помощью команды «Отключиться от базы данных». Команда доступна только для пользователя, входящего в группу Администраторы.
Отключиться от базы данных	Закрывает текущее соединение с базой данных. Команда доступна только для пользователя, входящего в группу Администраторы.
Подключиться в однопользовательском режиме	Позволяет подключиться к базе данных в однопользовательском режиме. Перед подключением необходимо отключиться от текущей базы данных с помощью команды «Отключиться от базы данных». Внимание: подключиться к базе данных в однопользовательском режиме можно только под учетной записью Administrator.
Вернуть базу данных в нормальный режим	Команда позволяет вернуть базу данных в многопользовательский режим после того, как она была переведена в однопользовательский. Перед выполнением команды необходимо сначала выполнить отключение от базы данных.
Архивное копирование	Открывает окно создания архивной копии базы данных. Команда доступна только для пользователя, входящего в группы Администраторы или Операторы архива.
Восстановление базы данных	Открывает окно восстановления базы данных из архивной копии. Команда доступна только для пользователя, входящего в группы Администраторы или Операторы архива.
Обновление статистики индексов	Обновляет статистику индексов[11]. Команда доступна только для пользователя, входящего в группу Администраторы.
Выход	Выход из программы. Обратите внимание, что если запущен процесс архивирования или восстановления базы данных из архива, то выйти из программы нельзя. Необходимо сначала дождаться завершения процесса.

Сервис

Команда	Описание
Редактор скрипт-объектов…	Открывает окно редактора скрипт объектов. Команда доступна только для пользователя, входящего в группу Администраторы.

[11] Статистика индекса — это число в диапазоне от 0 до 1, которое показывает как распределены значения данных в колонке, по которой создан индекс. Статистика используется оптимизатором при создании плана выполнения запроса. Сервер Interbase не обновляет статистику индекса при изменении, добавлении или удалении данных, поэтому рекомендуется периодически выполнять команду Обновление статистики индексов или выполнять архивное копирование и последующее восстановление базы данных. Неверные значения статистики индексов могут привести к генерации неоптимальных планов и, соответственно, к снижению производительности работы системы.

Редактор форм…	Открывает окно редактора экранных форм. Команда доступна только под учетной записью Administrator.
SQL редактор…	Открывает окно редактора SQL запросов. Команда доступна только для пользователя, входящего в группу Администраторы.
Просмотр шаблона документов…	Открывает окно тестирования шаблонов импорта текстовых документов. Команда доступна только для пользователя, входящего в группу Администраторы.
Установить пакеты настроек…	Открывает окно установки пакетов настроек. Команда доступна только для пользователя, входящего в группу Администраторы.
Опции…	Открывает окно с опциями системы.

Окна

Команда	Описание
Исследователь	Открывает окно Исследователя.
Список окон…	Открывает список загруженных в данный момент времени окон.
Очистить память	Удаляет из памяти все окна просмотра, которые загружены в данный момент но не видимы на экране (т.е. находятся в закрытом состоянии).
Сохранить стол…	Команда позволяет сохранить текущий рабочий стол.
Удалить стол	Команда позволяет выбрать из списка и удалить один из сохраненных ранее рабочих столов.

Справка

Команда	Описание
Начинаем работать	Справочная система по установке программы и началу работы с ней.
Руководство пользователя	Справочная система по использованию программы.
Руководство разработчика	Справочная система по разработке приложений на платформе Гедымин.
Руководство программиста	Справочная система, рассчитанная на программиста, который имеет доступ к исходным кодам платформы Гедымин.
Справка по VBScript	Справочная система по языку программирования VBScipt.
Справка по FastReport	Справочная система по системе построения отчетов FastReport.
www.gsbelarus.com	Команда открывает Internet Explorer и переходит на сайт компании Golden Software.
Регистрация	Регистрация программы на данном компьютере.
О системе	Вызов окна с полезной информацией о системе.

Список рабочих столов

Справа от главного меню располагается выпадающий список рабочих столов, а также кнопки для вызова команд сохранения текущего рабочего стола, удаления рабочего стола и очистки экрана. Последняя команда закрывает и удаляет из памяти все окна за исключением Исследователя системы.

Что такое рабочий стол?

Рабочим столом, в терминах платформы Гедымин, называется совокупность окон просмотра, расположенных определенным образом на экране. Рабочему столу можно дать имя и сохранить его в базе данных.

Разработчику в процессе работы могут понадобиться: редактор скрипт-объектов, редактор SQL и список таблиц в базе данных. Для того, чтобы не выполнять одни и те же действия каждый раз при входе в программу, можно один раз открыть три вышеупомянутых окна, расположить их на экране удобным образом и сохранить как рабочий стол под именем, например, «Разработка». При следующей загрузке Гедымина сохраненные окна будут открыты автоматически. Если разработчик, к тому же, работает и с бухгалтерскими функциями платформы, он может создать второй рабочий стол, назвав его «Бухгалтерия» и включив в него окна «Журнал-ордер», «Журнал хозяйственных операций» и «План счетов». В любой момент времени можно перейти от одного рабочего стола к другому, выбрав его из выпадающего списка на главном окне. Если, в последствии, необходимость в рабочем столе «Бухгалтерия» отпадет, то его можно удалить с помощью команды «Удаление рабочего стола».

По умолчанию, рабочий стол сохраняется только при вызове соответствующей команды, однако установив соответствующий параметр в опциях системы можно добиться автоматического сохранения текущего рабочего при выходе их программы. Там же можно определить, как будет вести себя рабочий стол при смене рабочей организации: сохраняться автоматически, не сохраняться или необходимое действие будет запрашиваться у пользователя.

Обратите внимание, что список рабочих столов свой не только для каждого пользователя, но и для каждого разрешения экрана.

Запрет на изменение рабочего стола

В ряде случаев требуется запретить пользователю самостоятельно изменять рабочий стол. Сделать это можно. Применив соответствующую политику безопасности на закладке «Политики» в окне «Опции», которое вызывается из пункта «Сервис» главного меню.

Метаданные

Список всех рабочих столов хранится в таблице GD_DESKTOP, которая имеет следующую структуру:

PK	FK	Поле Домен Тип данных	NN	Описание
●		ID DINTKEY INTEGER	●	Идентификатор записи.
	●	USERKEY DINTKEY INTEGER	●	Ссылка на таблицу GD_USER. Пользователь, к которому привязан рабочий стол.
		SCREENRES DINTEGER INTEGER		Разрешение экрана, под которое создан данный рабочий стол. Целое число, вычисляется как произведение разрешения экрана по вертикали на разрешение по горизонтали.
		NAME DNAME VARCHAR(60)	●	Наименование рабочего стола.
		SAVED DTIMESTAMP TIMESTAMP	●	Дата и время последнего сохранения рабочего стола.
		DTDATA DBLOB4096 BLOB		Данные рабочего стола.
		RESERVED DRESERVED		Зарезервировано.

		INTEGER		

Список рабочих организаций

На платформе Гедымин можно вести учет по нескольким организациям, которые называются рабочими организациями. Одна из них, является активной в данный момент и называется текущей рабочей организацией. Сменить текущую рабочую организацию можно воспользовавшись выпадающим списком рабочих организаций на главном окне. Справа от списка расположена кнопка, с помощью которой можно открыть окно просмотра рабочих организаций.

Список загруженных окон

Нижнюю часть главного окна занимает список загруженых в настоящий момент окон просмотра. Загруженное окно не обязательно видно на экране, оно может быть и скрыто. В этом случае для его отображения необходимо щелкнуть мышью по названию окна в списке загруженных окон.

Свернуть или закрыть?

Если щелкнуть правой кнопкой мыши на списке загруженных окон, то откроется контекстное меню с четырьмя командами:

- Свернуть форму;
- Свернуть все;
- Закрыть форму;
- Закрыть все;

Первые две команды позволяют свернуть, убрать с экрана конкретную форму или все видимые, открытые формы. Для того, чтобы снова отобразить форму на экране необходимо кликнуть по ее наименованию в Списке загруженных окон. Команды «Закрыть форму» и «Закрыть все» не просто убирают форму с экрана, но и удаляют ее из памяти компьютера, высвобождая тем самым ресурсы компьютера. После закрытия формы, ее наименование уже не будет высвечиваться в Списке загруженных окон и для повторного открытия необходимо будет прибегнуть к Исследователю.

В отличие от других программ Windows, в Гедымине при щелчке по пиктограммке закрытия окна (крестик на заголовке, в правом верхнем его углу) окно не закрывается, а сворачивается, т.е. остается в оперативной памяти компьютера. Так сделано, чтобы уменьшить нагрузку на сервер базы данных. Ведь, при открытии окна может выполняться длительный запрос, а пользователь может часто открывать-закрывать окна, например, при работе с несколькими документами. Если необходимо не свернуть, а именно закрыть окно (удалить его из памяти компьютера), то следует удерживать нажатой клавишу Shift в момент щелчка по пиктограммке закрытия.

Работаем из макросов

Как свернуть или закрыть форму из макросов.

Исследователь

Окно просмотра

Рассмотрим теперь из каких частей состоит окно просмотра. Для начала познакомимся с простым окном, содержащим одну таблицу с данными.

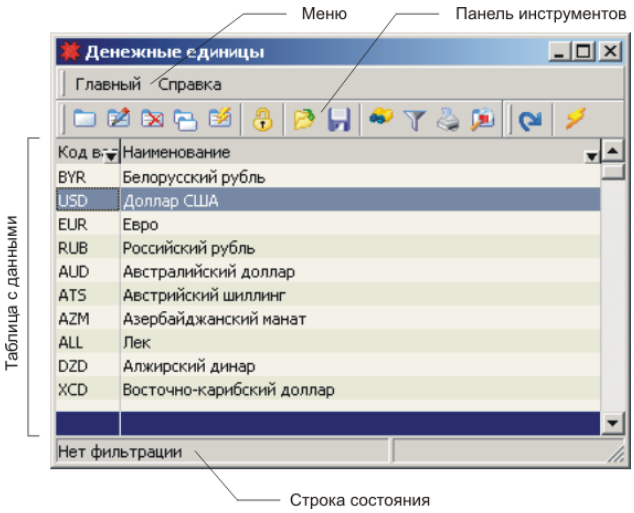


Рис. 10 Окно просмотра.

Основными составляющими частями являются:

- Меню;
- Панель инструментов;
- Таблица с данными;
- Строка состояния.

Рассмотрим их более подробно:

Меню

В простейшем случае меню содержит два выпадающих подменю:

- Главный — содержит команды для работы с данными в окне;
- Справка — содержит команду вызова справки.

Рассмотрим команды содержащиеся в этих выпадающих меню:

	Команды	Быстрый вызов	Описание
	Добавить…	Ins	Открывает диалоговое окно ввода новой записи.
	Изменить…	Ctrl+Enter, Enter	Открывает диалоговое окно для изменения данных выделенной записи или выделенных записей.
	Удалить	Ctrl+Del	Удаляет выделенную запись или записи.
	Дубликат	Ctrl+D	Создает копию текущей записи.
	Слияние…	Ctrl+R	Открывает окно слияния (объединения) двух записей.
	Загрузить из файла…		По выбору данной команды на экране откроется стандартное окно выбора файла. Выбрав файл с сохраненными ранее записями можно загрузить их в базу данных.
	Сохранить в файл…		Данная команжа позволяет сохранить выделенную запись или записи в указанном файле. После чего их можно будет загрузить в другую базу данных.
	Добавить в отмеченные		Добавляет выделенную запись или записи в отмеченные. Впоследствии, перейдя в режим «Только отмеченные» можно работать со списком отобранных ранее записей.
	Удалить из отмеченных		Убирает выделенную запись или записи из списка отмеченных.

	Сохранить		Команда позволяет сохранить изменения данных, если идет их редактирование непосредственно в таблице.
	Поиск	Ctrl+S	Открывает или закрывает панель поиска.
	Фильтр		Открывает меню работы с фильтрами.
	Печать	Ctrl+P	Открывает меню с отчетами (печатными формами) формы.
	Макросы	Ctrl+M	Открывает меню с макросами формы.
	Только отмеченные		Включает/выключает режим отображения только отмеченных записей.
	Загрузить настройки формы…		Позволяет скопировать визуальный настройки формы из провиля другого пользователя системы.

Панель инструментов

На панели инструментов располагаются команды для работы с данными в окне. Все команды, которые присутствуют на панели инструментов находятся так же и в меню «Главный». Мы уже рассмотрели их выше.

Таблица с данными

Пространство, ограниченное панелью инструментов сверху и строкой состояния снизу, называется Рабочей областью. Обычно здесь располагается таблица, хотя, при создании своих окон, разработчик волен разместить здесь любой управляющий элемент для отображения и редактирования данных.

По щелчку правой кнопки мыши над таблицей можно вывзвать контекстное меню, которое содержит основные команды по работе с данными, а так же команды поиска в таблице и настроки ее внешнего вида.

Строка состояния

В зависимости от того, применен к данным фильтр или нет в строке состояния выводится либо наименование примененного фильтра, либо надпись «Нет фильтрации». В первом случае, более подробную информацию о фильтре можно получить наведя указатель мыши на строку состояния и обождав пока не появится всплывающая подсказка.

Окно просмотра мастер-дитэйл

Выше мы рассмотрели простейший случай — окно просмотра с одной таблицей. Теперь рассмотрим окна, предназначенные для отображения двух датасетов, связанных как мастер-дитэйл.

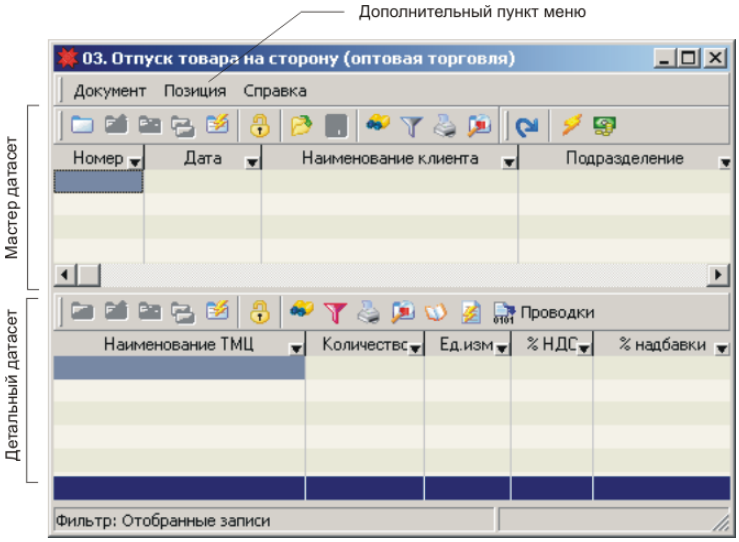


Рис. 11 Окно мастер-дитэйл. Горизонтальная раскладка.

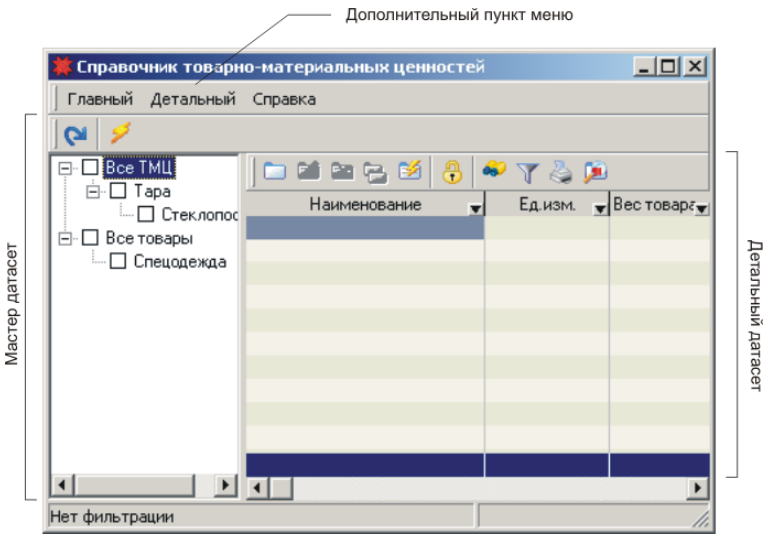


Рис. 12 Окно мастер-дитэйл с деревом и таблицей.

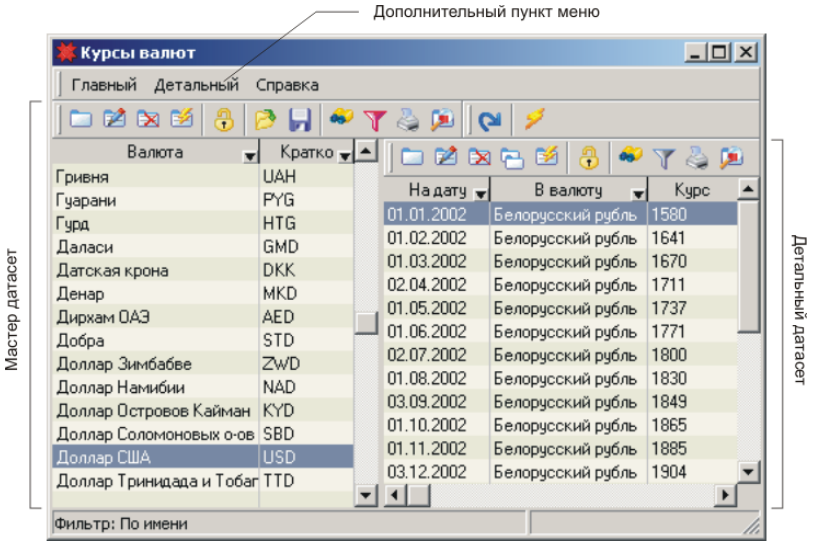


Рис. 13 Окно мастер-дитэйл. Вертикальная раскладка.

Как видно на представленных рисунках, окна мастер-дитэйл могут быть в горизонтальном и вертикальном исполнении. В первом случае главный датасет располагается над детальным, а во втором — слева от него. В зависимости от типа данных для отображения главного датасета может использоваться таблица или дерево.

Для работы с детальным датасетом в меню окна появился отдельный пункт, который называется «Детальный» или «Позиция». Кроме этого, детальный датасет имеет свою панель управления и свое контекстное меню, которое вызывается по нажатию на правую кнопку мыши. Поскольку команды «Добавить», «Изменить», «Удалить» и т.д. ведут себя аналогично командам для главного датасета мы не будем останавливаться здесь на их подробном описании.

Режим выбора

Операция выбора нескольких записей достаточно часто встречается в повседневной работе. Например, оператор выбирает товары при заполнении счет-фактуры, системный администратор выбирает группы пользователей, при настройке политик безопасности и т.д. Для выбора нескольких записей используется форма просмотра, открытая в режиме выбора.

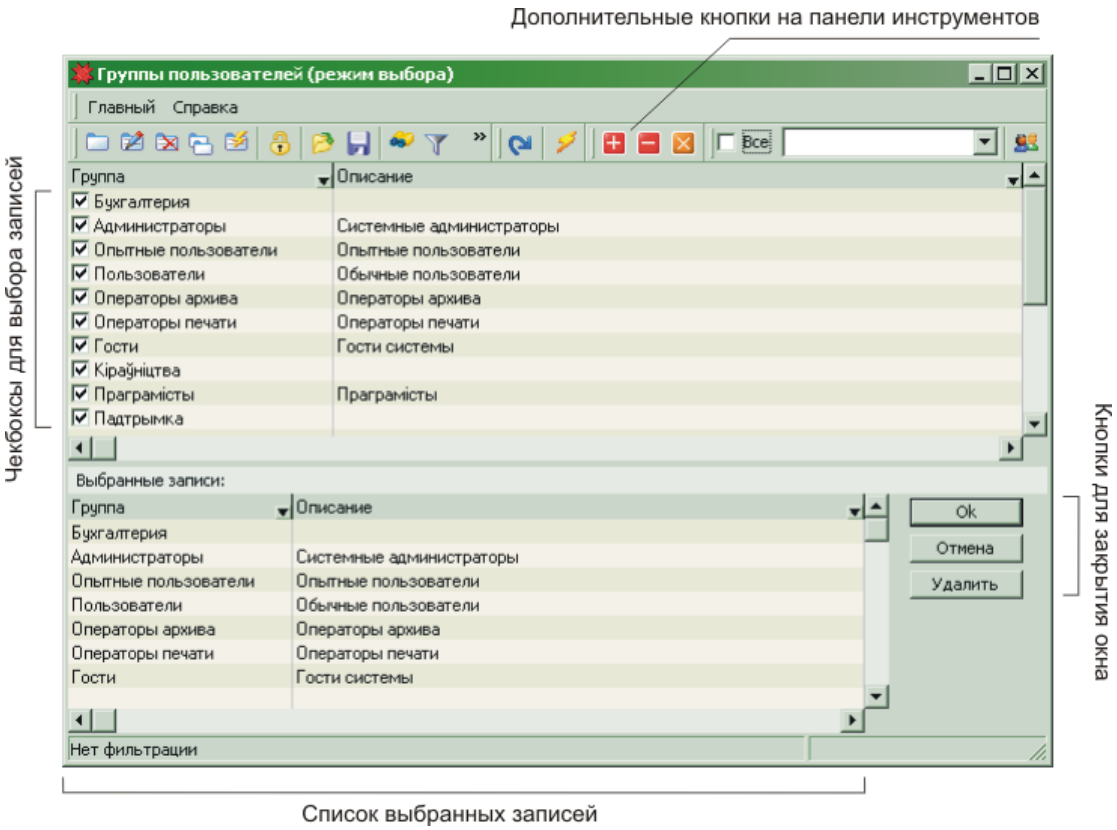


Рис. 14. Форма просмотра в режиме выбора.

Обратимся к рисунку. Выбор (пометка) записей производится с помощью чекбоксов, которые выводятся в крайней левой колонке таблицы. Дополнительные кнопки на панели инструментов, которые появляются в режиме выбора, позволяют:

- Пометить все записи в таблице;
- Снять пометку со всех записей в таблице;
- Инвертировать текущее выделение;

В нижней части окна находится панель на которой выводится список выбранных записей, а так же кнопки для подтверждения выбора и закрытия окна или для отмены и закрытия окна. Третья кнопка — «Удалить» — удаляет запись из списка выбранных, т.е. снимает с нее пометку.

Форма в режиме выбора открывается всегда в модальном режиме. Т.е. для продолжения работы с программой ее обязательно необходимо закрыть с помощью кнопок «Ок» или «Отмена». Закрытие формы с помощью пиктограммки в виде крестика в правом верхнем углу равносильно нажатию кнопки «Отмена».

Поиск данных

В базе данных могут находиться миллионы записей. Часто среди них надо отыскать одну-две, удовлетворяющие определенным условиям. Гедымин предоставляет пользователю пять способов поиска. Рассмотрим их более подробно, а также постараемся дать ответ на вопрос: Когда какой поиск лучше использовать?

Поиск по первым введенным символам

Данный вид поиска доступен в таблице и дереве. Если вы хотите найти определенное значение, установите курсор в нужную колонку и начинайте набирать первые символы значения. Если, такая запись есть в базе данных, — курсор перейдет на нее. Если несколько записей удовлетворяют введенным начальным символам — курсор перейдет на первую из них.

Поиск данных в колонке

Данный вид поиска позволяет найти запись по вхождению введенной подстроки в данные определенной колонки. Вызвать его можно либо из контекстного меню, либо используя сочетание клавиш Ctrl+F. Для осуществления поиска необходимо ввести искомую подстроку в появившемся диалоговом окне, установить параметры и нажать кнопку «Ок».

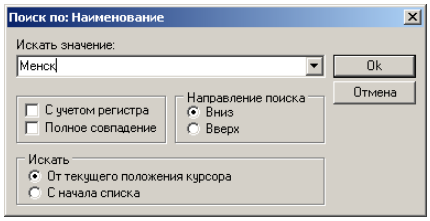


Рис. 15 Диалоговое окно поиска.

Если искомые данные найдены — курсор будет установлен на соответствующую запись. В противном случае на экран будет выдано сообщение о том, что поиск окончился безуспешно.

Для того, чтобы продолжить поиск, найти следующую запись, удовлетворяющую введенным критериям достаточно нажать клавишу F3 или выбрать команду «Найти следующее» из контекстного меню.

Данный вид поиска доступен как в таблице, так и в дереве. Причем, если в таюлице установлено расширенное отображение, то поиск будет осуществляться сразу по нескольким колонкам, данные которых отображаются в клетке таблицы, на которой установлен курсор. Если вы сомневаетесь по каким именно колонкам будет осуществляться поиск, — обратите внимание на заголовок диалогового окна поиска, там вы найдете интересующую вас информацию.

Автофильтр

Кто работал с электронными таблицами Microsoft Excel наверняка сразу поймет о чем идет речь. Автофильтр позволяет быстро отобрать данные на основании значений, уже имеющихся в колонке. Для установки автофильтра необходимо щелкнуть мышью по треугольничку в правом нижнем углу заголовка колонки.

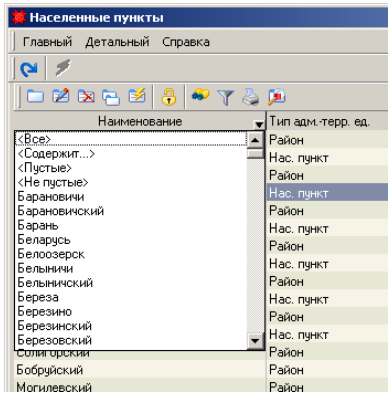


Рис. 16 Автофильтр.

На экране появится список значений, содержащихся в данной колонке. Выберите конкретное значение и на экране отобразятся только содержащие такое значение записи.

Если автофильтр установлен, то текст в заголовках колонок таблицы выводится сисним цветом. Кроме этого, синим цветом подсвечивается треугольничек в той колонке, по которой установлена автофильтрация.

Вверху списка располагаются четыре команды:

- <Все> — снимает автофильтр и выводит все записи в таблице;
- <Содержит> — позволяет отобрать только те записи, которые содержат введенную подстроку;
- <Пустые> — позволяет отобрать записи, которые не содержат значений в данной колонке;
- <Не пустые> — позволяет отобрать записи, которые содержат какое либо, не пустое, значение в данной колонке.

В целях экономии памяти автофильтр выводит в списке только первые несколько десятков уникальных значений из колонки.

Поиск с помощью панели поиска

Панель поиска открывается с помощью кнопки Поиск на панели инструментов или с использованием команды Поиск в меню окна.

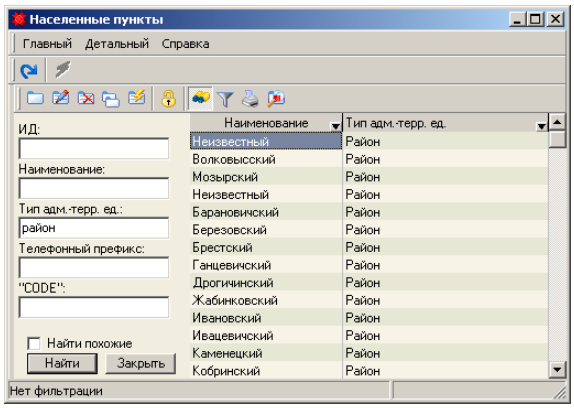


Рис. 17 Панель поиска.

На ней отображается список полей и две кнопки: «Найти» — выполняет поиск данных согласно введенным критериям и «Закрыть» — закрывает панель поиска, возвращает датасет в исходное состояние.

Поиск может осуществляться только по строковым полям и полям типа дата, дата и время. В зависимости от типа поля поиск осуществляется:

- Для строковых полей — по вхождению введенной подстроки;
- Для полей типа дата, дата и время — по точному соответствию введенного значения;

При вводе значений для поиска по строковым полям можно использовать шаблоны % и _. Первый заменяет произвольное (вкдючая ноль) количество любых символов, а второй — один любой символ в данной позиции. Если необходимо найти символ процента или подчеркивания непосредственно, наберите их, предворив символом процента. Например, %% — найдет все записи, содержащие % в указанном поле, а %_ — все записи, содержащие символ подчеркивания в указанном поле.

Фильтрация данных

Когда какой поиск использовать?

При таком разнообразии предоставляемых возможностей поиска возникает естественный вопрос: какой способ следует предпочесть в той или иной ситуации?

Прежде чем попытаться сформулировать ответ, стоит пояснить какие механизмы использует каждый из описанных выше видов поиска. Поиск по первым введенным символам, поиск по колонке и автофильтр — отрабатывают на клиенте, пробегаясь от первой до последней записи в наборе данных и сверяя каждую запись на соответствие введенному условию. Очевидно, что если датасет большой, содержит десятки колонок и десятки или сотни тысяч записей, то сканирование его приведет к:

1. загрузке сервера;
2. передаче большого объема данных по сети;
3. загрузке всех данных на клиентскую машину и, как следствие, потребление большого объема оперативной памяти для их размещения. Не говоря уже о том, что при попытке загрузить слишком большое количество данных оперативная память, доступная программе может исчерпаться и на экран будет выдано сообщение об ошибке;

Поиск с использованием фильтров или панели поиска приводит к добавлению условий отбора данных в SQL запрос и отсылке его на сервер. При этом минимизируется сетевой трафик и на клиентский компьютер пересылаются только те записи, которые отвечают заданным критериям отбора.

Очевидно, что выбор вида поиска в конкретной ситуации — это компромисс, между удобством и доступностью поиска по первым введенным символам или автофильтра и потреблением оперативной памяти и загрузкой сервера и сети. Если набор данных относительно не велик (на практике это означает не более нескольких тысяч записей) то можно смело прибегать к первым трем видам поиска. Если записей в датасете десятки тысяч и более, то следует использовать панель поиска или создать фильтр. Можно также порекомендовать использовать коомбинацию двух видов поиска. Например, на большой таблице настроить фильтр, который бы отбирал относительно небольшую часть записей и затем искать в них необходимые данные с помощью поиска по колонке, поиска по первым введенным символам или автофильтра.

Диалоговое окно

В предыдущих разделах мы ознакомились с формами просмотра, которые служат для отображения множества записей в таблицах или деревьях. Для отображения данных одной записи, их изменения, в Гедымине широко применяются диалоговые окна разных типов. Как и формы просмотра, типы диалоговых окон образуют свою иерархию:

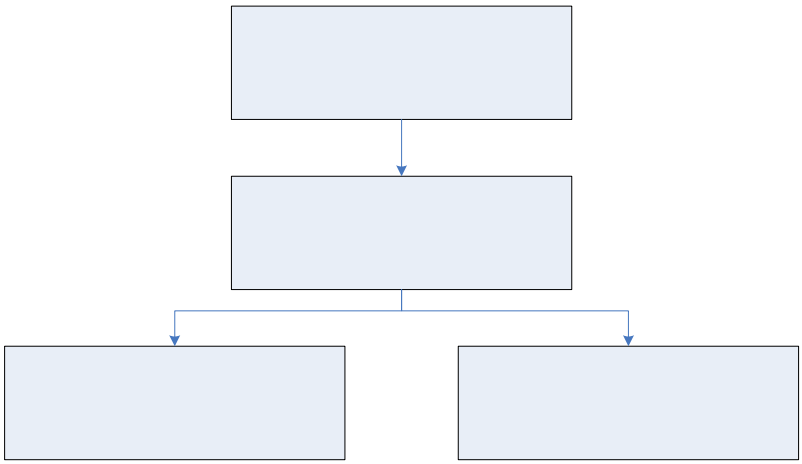


Рис. 18 Иерархия классов диалоговых окон.

Базовое диалоговое окно, базовое диалоговое окно с транзакцией

Каждое диалоговое окно имеет, как минимум, один или несколько управляющих элементов для отображения и редактирования данных, а так же пять кнопок:

- «Ок» — подтвердить ввод или изменение данных и закрыть окно;
- «Отмена» — отменить ввод или изменение данных и закрыть окно;
- «Меню» — открыть меню с дополнительными командами;
- «Новый» — сохранить текущую запись и приступить к вводу новой записи;
- «Справка» — открыть справку по текущему диалоговому окну;

Tgdc_dlgG
Базовое диалоговое окно.

Tgdc_dlgTR
Базовое диалоговое окно с транзакцией, для отображения данных связанных объектов.

Tgdc_dlgTRPC
Базовое диалоговое окно с транзакцией и набором закладок.

Tgd
Базовое диалогов
отображения доку
(заголовок докум
(деталь

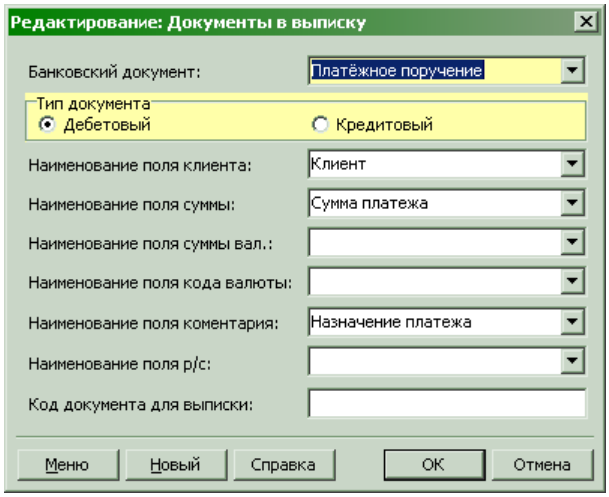


Рис. 19 Пример диалогового окна с транзакцией.

Базовое диалоговое окно с транзакцией отличается от простого базового диалогового окна наличием внутренней транзакции, которая стартует при открытии окна и комитится при любом его закрытии, либо по нажатию на кнопку «Ок», либо по «Отмене». Как правило, на этой транзакции работают выпадающие списки и вспомогательные бизнес-объекты, связанные с редактируемым объектом.

Меню диалогового окна

По кнопке «Меню» на экране открывается список дополнительных команд:

Команда	Клавиши быстрого вызова	Описание

Базовое диалоговое окно с закладками

Базовое диалоговое окно с таблицей

Элементы управления

Визуальные элементы управления используются для отображения данных на экране компьютера, организации их редактирования. Основными визуальными элементами в системе Гедымин являются:

- Таблица;
- Дерево;
- Выпадающий список;
- Выпадающий список Множество;
- Поле для отображения и ввода даты и времени;

- Калькулятор;

Расмотрим подробнее каждый из этих визуальных элементов:

Таблица

Таблица предназначена для отображения набора данных и бесспорно является центральным элементом пользовательского интерфейса любой программы, написанной на платформе Гедымин. В простейшем случае одна строка таблицы соответствует одной записи из набора и каждая колонка отображает данные своего поля. В более сложных случаях таблица может быть настроена так, чтобы одна запись занимала несколько строк и каждая клетка содержала данные нескольких полей. Такой случай приведен на Рис. 20 «Таблица». Обратите внимание на колонку «Покупатель/Телефон», в каждой клетке этой колонки выводится два значения: наименование покупателя и его контактный телефон.

Рис. 20 «Таблица»

Формат отображения данных может быть настроен пользователем. Таблица позволяет организовать условное форматирование, когда параметры визуального отображения конкретной записи определяются ее данными. Кроме, собственно, отображения данных, таблица позволяет выполнять следующие операции:

- Сортировку данных по определенной колонке в прямом и обратном порядке;
- Фильтрацию данных;
- Редактирование данных;
- Суммирование данных (строка Итого).

На Рис. 20 «Таблица» обозначены основные составляющие элементы таблицы. Рассмотрим их подробнее:

Индикатор состояния записи

Индикатор состояния записи находится в левой части таблицы и может принимать следующие значения:

- Сплошной треугольник показывает, что данная запись является текущей, т.е. на ней установлен курсор;
- Сплошной круг показывает, что данная запись входит в группу выделенных записей;

- Сплошной круг со знаком "больше" показывает, что запись входит в группу выделенных записей и одновременно является текущей записью;
- Если индикатор имеет форму курсора ввода текста, то это означает, что текущая запись находится в состоянии редактирования.

Как правило, индикатор состояния записи скрыт и отображается только в процессе редактирования данных непосредственно в таблице.

По умолчанию, выделенные записи обозначаются в таблице другим цветом фона и, возможно, другим шрифтом.

Заголовок колонки

Заголовок колонки предназначен для отображения наименования поля или полей, которые выводятся в данной колонке. В правом нижнем углу заголовка располагается кнопка автофильтра. Если данные таблицы упорядочены по содержимому колонки, то в заголовке отображается индикатор сортировки, а сама колонка выводится в более темных тонах. Красный треугольник в левом верхнем углу означает сортировку по убыванию, в левом нижнем — по возрастанию.

Полоса прокрутки

Полоса прокрутки предназначена для перемещения по набору данных с помощью мыши.

Строка Итого

Строка «Итого» предназначена для отображения сумм по числовым полям. Если в наборе данных больше записей, чем изначально отображается в таблице, то в строке «Итого», в числовых колонках будет выведена надпись «Рассчитать…». Для получения суммы необходимо щелкнуть указателем мыши по этой надписи. В настройках таблицы можно указать для каких полей следует подсчитывать итоговое значение, а для каких нет. Например, сумма по колонке «Количество» в накладной имеет смысл, а по колонкам «Цена» или «Процент НДС» — нет. Так же в настройках таблицы можно указать будет отображаться строка «Итого» на экране или нет.

Подножие таблицы

Подножие таблицы предназначено для вывода некоторой обобщающей информации о данных таблицы. по умолчанию здесь отображается количество записей в таблице, количество выделенных записей и номер текущей записи. Настройщик может предусмотреть (путем перекрытия соответствующего события) вывод иной информации в подножии таблицы. Например, на Рис. 20 «Таблица» в подножии таблицы выводится информация о сумме всех неоплаченных на данный момент времени счетов.

Контекстное меню

Контекстное меню вызывается по нажатию на правую кнопку мыши над записями таблицы и содержит следующие команды:

- «Мастер установок» — открывает окно мастера установок таблицы;
- «Скопировать ячейки в буфер» — копирует выделенные ячейки в буфер обмена;
- «Обновить данные» — перечитывает содержимое таблицы;
- «Найти…» — открывает диалоговое окно поиска данных в таблице;
- «Найти следующее» — повторяет предыдущий поиск.

Обратите внимание, что если таблица располагается на форме просмотра, то по правой кнопке мыши откроется контекстное меню формы, куда будут добавлены команды таблицы.

Если нажать правую кнопку мыши над заголовком колонки, то контекстное меню будет содержать две команды:

- «Скрыть колонку» — скрывает указанную колонку;
- «Заголовок колонки» — позволяет изменить заголовок колонки;

Перемещение по таблице

Сортировка данных

Автофильтрация данных

Настройка таблицы

Настройка внешнего вида таблицы осуществляется в диалоговом окне «Мастер установок» вызвать которое можно либо по нажатию на клавишу F10, находясь в таблице, либо выбрав соответствующий пункт из контекстного меню, вызываемого по правой клавише мыши.

Окно настройки таблицы содержит следующие закладки:

- Таблица;
- Колонки;
- Условия;
- Шаблоны;
- Запрос;

Рассмотрим содержимое этих закладок:

Таблица

На закладке «Таблица» находятся управляющие элементы, которые позволяют настраивать общие параметры внешнего вида таблицы. В левой части располагается группа кнопок, которая предназначена для установки начертания шрифта и цвета фона для: данных таблицы, выделенных записей, заголовков колонок. Справа вверху находится флаг, предназначенный для включения/выключения режима отображения полос. Если режим включен, то фоновый цвет четных и нечетных записей различается. Соответствующие цвета настраиваются с помощью кнопок, расположенных под флагом «Использовать». Ниже располагается пример оформления таблицы, согласно установленным параметрам. И, наконец, справа внизу располагаются четыре флага:

- «Горизонтальные линии таблицы» — если флаг установлен, то между записями будут выводиться сплошные линии;
- «Растягивать колонки по ширине таблицы» — если флаг установлен, то ширина колонок будет изменяться автоматически так, чтобы они занимали все доступное пространство. Если флаг снять, то часть колонок может быть не видна на экране и для их просмотра необходимо использовать полосу горизонтальной прокрутки внизу таблицы или перемещать курсор с помощью клавиш влево-вправо;
- «Показывать строку итого» — данный флаг определяет будет ли видна строка «Итого» на экране;
- «Показывать подножие таблицы» — данный флаг определяет будет ли видно подножие таблицы на экране.

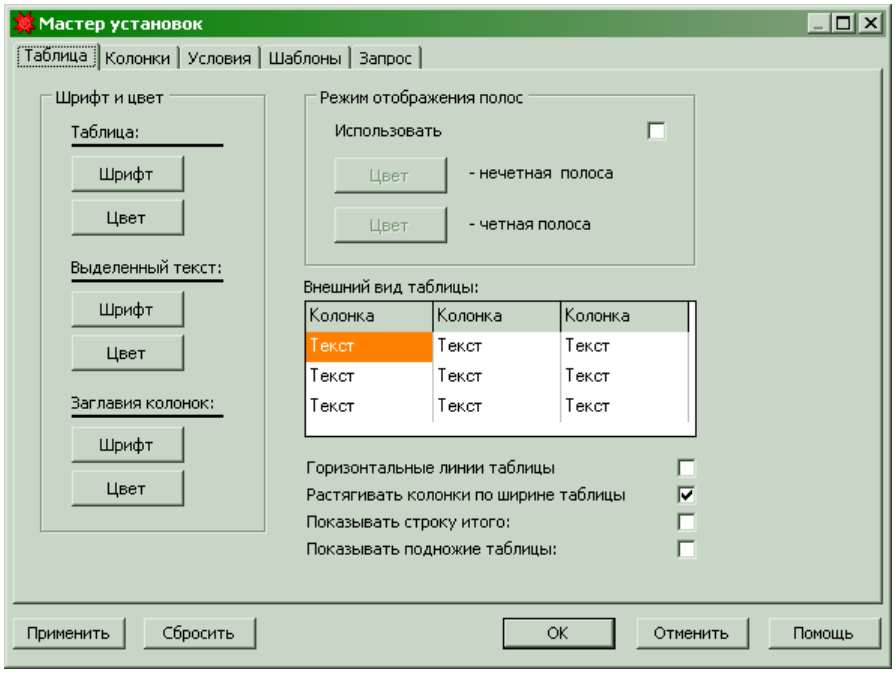


Рис. 21 Диалоговое окно настройки таблицы.

Колонки

На закладке «Колонки» производится настройка:

- внешнего вида колонок таблицы,
- визуального форматирования данных,
- расширенного отображения данных.

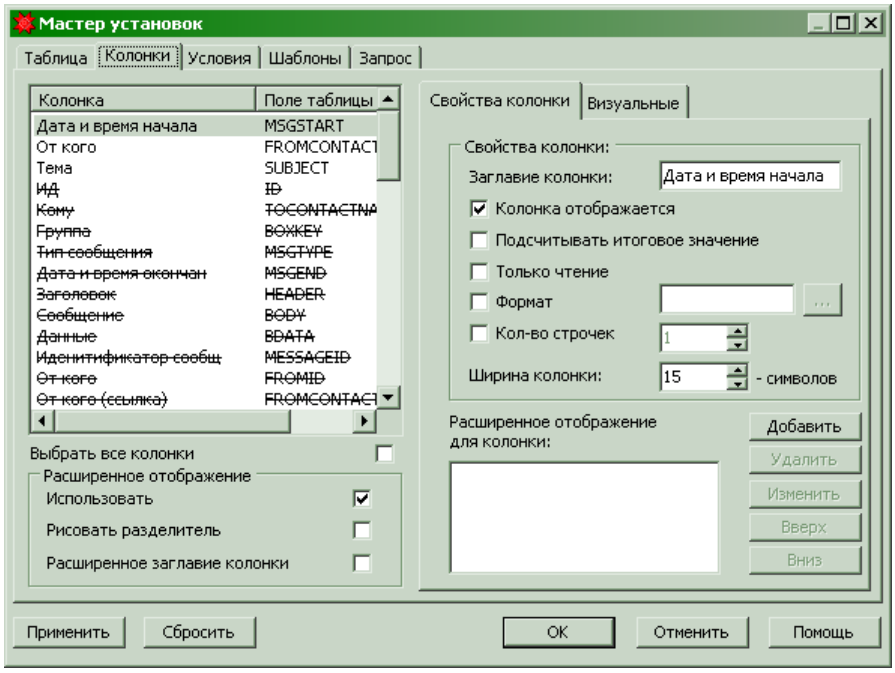


Рис. 22 Диалоговое окно настройки таблицы.

Для того, чтобы настроить параметры отображения конкретной колонки необходимо выбрать ее в списке в левой части закладки. Обратите внимание, что колонки, которые не отображаются в таблице (скрыты), выводятся в списке перечеркнутым шрифтом. Далее в правой части закладки можно установить:

- «Заглавие колонки» — строка, которая будет отображаться в заголовке колонки;
- «Колонка отображается» — с помощью этого флага можно установить будет отображаться колонка в таблице или нет;
- «Подсчитывать итоговое значение» — действует только для числовых полей. Если флаг установлен, то будет подсчитываться сумма по колонке;
- «Только чтение» — данные колонки нельзя будет изменять при переводе таблицы в режим редактирования;
- «Формат» — форматировать данные согласно заданному шаблону. Действует для числовых полей и полей типа дата, дата и время, время. Строку форматирования (маску) можно ввести вручную (см. соответствующее описание ниже) или воспользоваться кнопкой «…» спрва от поля ввода маски и выбрать один из предложенных вариантов.
- «Кол-во строчек» — задает количество строк, которое будет использовать для выводя значения каждой клетки указанной колонки.
- «Ширина колонки» — задает ширину колонки в символах. Обратите внимание, что если установлен режим «Растягивать колонки по ширине таблицы», то ширина колонок будет изменяться автоматически вне зависимости от установленного в данном поле значения.

На вкладке «Визуальные» можно установить такие параметры как:

- Шрифт, цвет фона и выравнивание для отображения данных в колонке;
- Шрифт, цвет фона и выравнивание заголовка колонки;

Можно настроить параметры визуального отображения сразу для нескольких колонок. Для этого необходимо выделить их в списке либо щелкая по ним мышью при нажатой клавише Ctrl, либо воспользовавшись флажком «Выбрать все колонки» внизу списка. В последнем случае будут выделены сразу все колонки в списке.

Расширенное отображение данных

При расширенном отображении данных в одной клетке таблицы могут отображаться значения сразу нескольких полей. Для настройки расширенного отображения необходимо выбрать колонку с первым полем, которое будет отображаться в клетке таблицы. Нажать кнопку «Добавить», расположенную в правой нижней части закладки, и выбрать дополнительные поля, которые будут отображаться в этой же колонке. Повторить эту операцию для всех колонок, для которых необходимо установить расширенное отображение. При необходимости, выбранные дополнительные поля можно удалять из списка или перемещать вверх-вниз. Соответствующие кнопки расположены под кнопкой «Добавить». Активизировать режим расширенного отображения, включив флаг «Использовать» в левой нижней части закладки. Дополнительно можно настроить такие параметры как: «Рисовать разделитель» — если флаг установлен, то поля в рамках одной клетки будут разделяться горизонтальными линиями. «Расширенное заглавие колонки» — если флаг установлен, то заголовок колонки будет так же выводиться в несколько строк, по числу полей, заданных в расширенном отображении.

Форматирование числовых значений

Маска, задающая формат вывода числовых значений может содержать следующие символы:

- 0 — Если форматируемое значение имеет цифру в той позиции, в которой в строке маски находится «0», то эта цифра копируется в результирующую строку. В противном случае символ «0» будет помещен в результирующую строку.
- # — Если форматируемое значение имеет цифру в той позиции, в которой маска содержит «#», то эта цифра копируется в результирующую строку. В противном случае результирующая строка не меняется.
- . — Десятичная точка. Первый символ «.» в строке маски определяет положение десятичной точки в результирующей строке. Все остальные символы «.», если они есть, игнорируются. То, какой именно символ будет использован в результирующей строке в качестве десятичного разделителя, определяется настройками операционной системы просмотреть и/или изменить которые можно в Панели управления Windows.

- , — Разделитель тысяч. Если маска содержит один или несколько символов «,», то результирующая строка будет содержать разделители тысяч между каждыми тремя цифрами слева от десятичной точки. Конкретное расположение символов «,» в маске и их количество не влияет на положение разделителей тысяч в результирующей строке. Важен сам факт их наличия. То, какой именно символ будет использован в результирующей строке в качестве разделителя тысяч, определяется настройками операционной системы просмотреть и/или изменить которые можно в Панели управления Windows.
- E+ — инженерная нотация. Если любая последовательность из «E+», «e+», «E-», «e-» содержится в маске то чсило будет отформатировано согласно инженерной нотации. Группа в количестве до 4 символов «0» следующая сразе за последовательностью символов «E+», «e+», «E-» или «e-» определяет минимальное количество цифр в экспоненте. Последовательности «E+», «e+» указывают на то, что знак «+» будет выводиться для положительных экспонент и знак «-» — для отрицательных. Последовательности «E-», «e-» указывают на то, что знак экспоненты будет указываться только для отрицательных значений.
- «xx»/'xx' — символы, заключенные в двойные или одинарные кавычки, копируются напрямую в результирующую строку из маски и не влияют на форматирование.
- ; — точка с запятой разделяет секции, содержащие строки форматирования для положительных, отрицательных и нулевых значений.

Положение самго левого символа «0» слева от десятичного знака и самого правого символа «0» справа от десятичного знака определяет количество цифр, которые всегда присутствуют в результирующей строке.

Форматируемое число всегда округляется до того количество десятичных знаков, которое задано с помощью символов «#» или «0» справа от десятичной точки. Если маска не содержит десятичной точки, то форматируемое число округляется до ближайшего целого числа.

Строка формата может содержать от одной до трех секций, разделенных символом «;», задающих формат положительных, отрицательных и нулевых значений. Если секция отрицательных чисел или секция нулевых значений пропущены (т.е. содержат пустую строку), то используется строка форматирования, заданная для положительных значений.

Ниже приводятся примеры форматирования числовых значений:

Строка форматирования/Значение	1234	-1234	0.5	0
	1234	-1234	0.5	0
0	1234	-1234	1	0
0.00	1234.00	-1234.00	0.50	0.00
#.##	1234	-1234	.5	
#,##0.00	1,234.00	-1,234.00	0.50	0.00
#,##0.00;(#,##0.00)	1,234.00	(1,234.00)	0.50	0.00
#,##0.00;;Zero	1,234.00	-1,234.00	0.50	Zero
0.000E+00	1.234E+03	-1.234E+03	5.000E-01	0.000E+00
#.###E-0	1.234E3	-1.234E3	5E-1	0E0

Форматирование значений типа дата и время

Маска, задающая форматирования значений типа дата и время может содержать следующие латинские символы:

- c — Отображает дату в соответствии с коротким форматом даты, установленным в операционной системе, и время в соответствии с длинным форматом времени.
- d — Отображает день ввиде числа без начального нуля (1-31).
- dd — Отображает день ввиде числа с начальным нулем (01-31).
- ddd — Отображает день ввиде аббривиатуры (Пн-Вс).

- dddd — Отображает полное наименование дня недели.
- ddddd — Отображает дату в соответствии с коротким форматом даты, установленным в операционной системе.
- dddddd — Отображает дату в соответствии с длинным форматом даты, установленным в операционной системе.
- m — Отображает месяц ввиде числа без начального нуля (1-12).
- mm — Отображает месяц ввиде числа с начальным нулем (01-12).
- mmm — Отображает месяц ввиде аббривиатуры (Янв-Дек).
- mmmm — Отображает наименование месяца (Январь-Декабрь).
- yy — Отображает год, как двухзначное число (00-99).
- yyyy — Отображает год, как четырех значное число (0000-9999).
- h — Отображает часы как число без начального нуля (0-23).
- hh — Отображает часы как число с начальным нулем (00-23).
- n — Отображает минуты как число без начального нуля (0-59).
- nn — Отображает минуты как число с начальным нулем (00-59).
- s — Отображает секунды как число без начального нуля (0-59).
- ss — Отображает секунды как число с начальным нулем (00-59).
- z — Отображает тысячные доли секунды как число без начальных нулей (0-999).
- zzz — Отображает тысячные доли секунды как число с начальными нулями (000-999).
- t — Отображает время согласно короткому формату отображения времени, установленному в операционной системе.
- tt — Отображает время согласно длинному формату отображения времени, установленному в операционной системе.
- / — Отображает символ-разделитель частей даты установленный в операционной системе.
- : — Отображает символ-разделитель часов, минут, секунд, установленный в операционной системе.
- «xx»/’xx’ — символы, заключенные в двойные или одинарные кавычки, копируются напрямую в результирующую строку из маски и не влияют на форматирование.

Символы, задающие формат могут быть заданы как в верхнем, так и внижнем регистре. На результат это не влияет.

Если задана пустая строка в качестве маски, то форматирование производится как будто был задан символ «c».

Условия

Условное форматирование — зависимость параметров визуального отображения записи от ее данных — позволяет существенно облегчить восприятие информации. Для каждой таблицы можно задать множество условий и соответствующих им параметров визуального отображения.

Список условий задается на закладке «Условия» «Мастера установок».

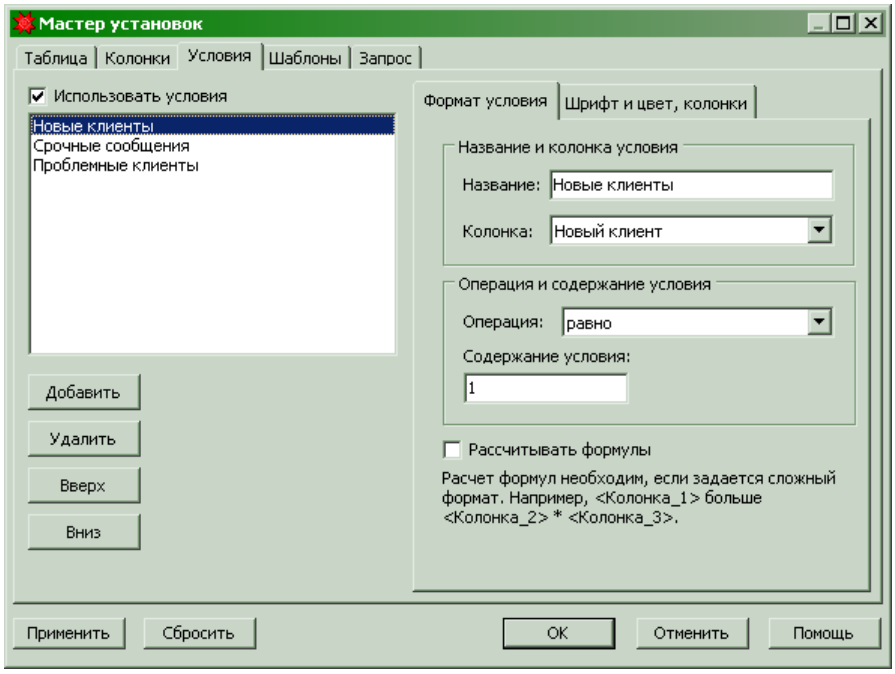


Рис. 23 Диалоговое окно настройки таблицы.

Для добавления нового условия необходимо:

1. Нажать кнопку «Добавить», расположенную под списком условий;
2. Ввести в поле «Название» произвольное наименование условия, под которым оно будет отображаться в списке;
3. Из выпадающего списка «Колонка» выбрать колонку, данные которой будут проверяться на соответствие/не соответствие условию;
4. Выбрать операцию сравнения из выпадающего списка «Операция».
5. Ввести значение с которым будет осуществляться сравнение. Можно вводить как непосредственные значения, так и формулы, в которых могут использоваться значения других колонок таблицы. В последнем случае надо установить флаг «Рассчитывать формулы» в правой нижней части закладки.
6. Перейти на закладку «Шрифт и цвет, колонки» и указать параметры визуального форматирования, которые будут применяться в случае, если заданное условие выполняется и к каким колонкам эти параметры будут применяться.

После того, как все условия будут добавлены в список, необходимо активизировать применение условий с помощью флага «Использовать условия» в левой верхней части закладки.

Обратите внимание, что условия применяются в том порядке, в котором они указаны в списке. Если несколько условий воздействуют на одни и теже клетки таблицы, то будет использоваться визуальное форматирование, определяемое последним условием.

Изменять порядок условий, удолять их из списка можно с помощью кнопок. расположенных в правой нижней части закладки.

Шаблоны

Закладка «Шаблоны» предназначена для выполнения следующих действий:

- Быстрой настройки внешнего вида таблицы с помощью выбора из списка одного из предустановленных шаблонов. Для выбора предустановленного шаблона необходимо установить на него курсор в списке и нажать кнопку «Установить шаблон»;
- Сохранения настроек таблицы в файле на диске и загрузки их из файла. Кнопки «Сохранить шаблон» и «Загрузить шаблон» соответственно;

- Установки первоначальных параметров колонок. Кнопка «По умолчанию». Первоначальные параметры: заголовок колонки, видимость, ширина, визуальное форматирование данных берутся из информации о таблицах базы данных, получить доступ к которой можно через «Исследователь», раздел «Сервис»—«Аттрибуты»—«Таблицы».

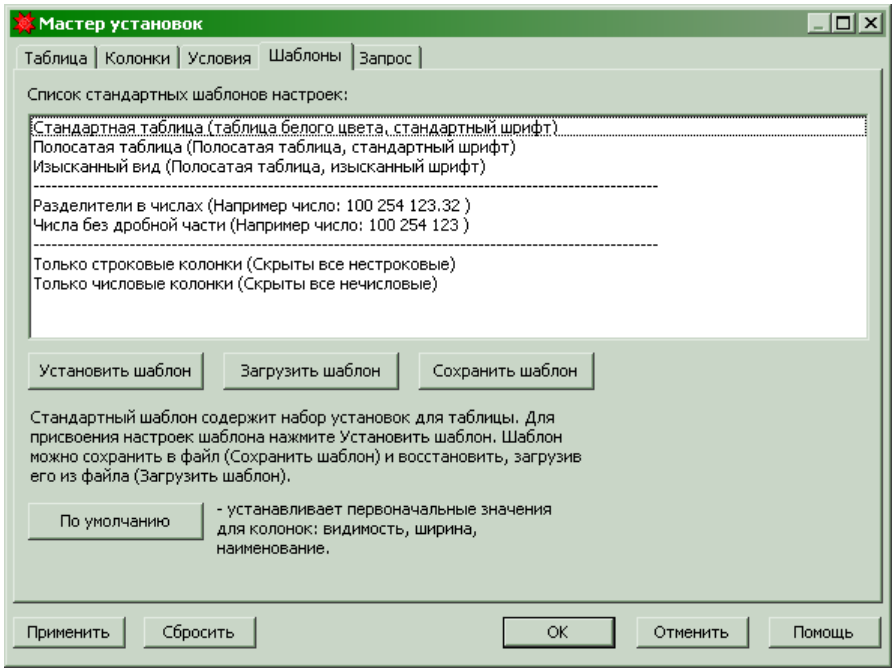


Рис. 24 Диалоговое окно настройки таблицы.

Запрос

На закладке «Запрос» отображается SQL запрос, который используется для извлечения набора данных отображаемого в таблице. Внизу запроса выводится план, который использует сервер при извлечении данных.

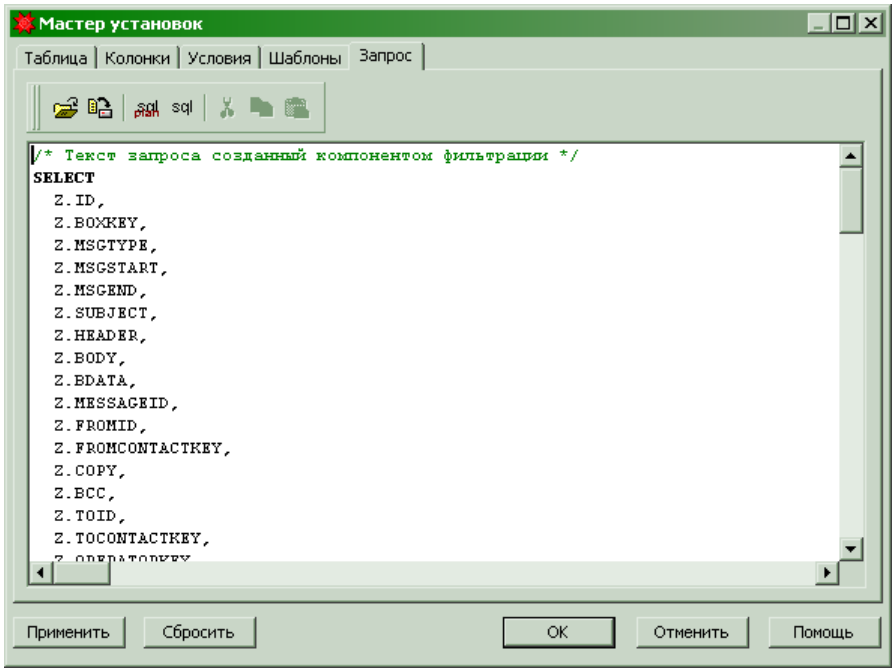


Рис. 25 Диалоговое окно настройки таблицы.

Сверху поля с текстом запроса находится панель инструментов. С помощью кнопок расположенных на ней можно сохранить текст запроса в файле или загрузить его из файла. Там же располагаются кнопки,

которые позволяют применить отредактированный запрос (с планом или без). Применить в данном контексте означает заменить исходный запрос новым.

Ограничение доступа к настройке таблицы

Настройка большой таблицы представляет собой достаточно трудоемкое занятие. В реальной практике часто требуется защитить таблицу от неквалифицированного пользователя, не дать ему возможности изменять ее настройки. Ограничить доступ пользователей можно через политики групповой безопасности. Соответствующая политика называется «Изменение визуальных настроек».

Копирование настроек таблицы

Если с программой работают десятки пользователей, то нет никакого смысла настраивать таблицудля каждого из них заново. Достаточно настроить ее под одной учетной записью и затем распространить на другие учетные записи. В Гедымине существует четыре способа копирования визуальных настроек от одного пользователя другому:

1. Скопировать настройки конкретной таблицы от одного пользователя другому можно следующим образом:
    a. Загрузиться под логином пользователя, настройки которого будут скопированы;
    b. Открыть окно, содержащее нужную таблицу;
    c. Перейти в таблицу и открыть «Мастер установок»;
    d. Перейти на закладку «Шаблоны»;
    e. Нажать кнопку «Сохранить шаблон» и ввести имя файла;
    f. Выйти из Гедымина и зайти под учетной записью пользователя которому необходимо скопировать настройки;
    g. Открыть окно, содержащее нужную таблицу;
    h. Перейти в таблицу и открыть «Мастер установок»;
    i. Перейти на закладку «Шаблоны»;
    j. Нажать кнопку «Загрузить шаблон» и выбрать ранее сохраненный файл.
2. Скопировать визуальные настройки формы от одного пользователя другому. В том числе, скопируются и настройки всех таблиц, находящихся на форме:
    a. Загрузиться под учетной записью пользователя которому необходимо скопировать настройки;
    b. Открыть нужную форму;
    c. В пункте «Главный» меню формы выбрать команду «Загрузить настройки формы»;
    d. Выбрать учетную запись, настройки которой будут скопированы;
3. Скопировать все визуальные настройки от одного пользователя другому:
    a. Открыть раздел «Сервис»→«Администратор» → «Пользователи» в «Исследователе»;
    b. Открыть учетную запись пользователя которому будем копировать настройки;
    c. Выбрать кнопку «Профиль» в правой части диалогового окна;
    d. Выбрать из выпадающего списка учтеную запись, все визуальные настройки которой будут скопированы данному пользователю;
    e. Закрыть окно изменения учтеной записи.
4. Скопировать данные непосредственно из хранилища одного пользователя в хранилище другого.

Дерево

Выпадающий список

Выпадающий список предназначен для подстановки некоторого объекта в поле записи, путем выбора его из списка или посика в базе данных по наименованию.

Выпадающий список состоит из следующих частей:

1. Поле ввода текста;
2. Копка вызова/скрытия списка;
3. Список объектов;
4. Панель инструментов;
5. Полоса прокрутки списка.

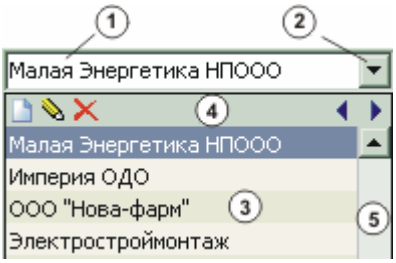


Рис. 26 Выпадающий список

Поле ввода текста

Поле ввода текста предназначено для ввода и отображения наименования объекта. При выборе объекта из базы данных путем поиска по наименованию в это поле необходимо ввести все наименование объекта или его часть. После этого необходимо нажать клавишу F3 или щелкнуть мышкой по кнопке вызова/скрытия списка. Будет осуществлен поиск объекта. Если в базе данных существует единственный объект, удовлетворяющий введенному наименованию, то он будет выбран и его наименование будет подставлено в поле ввода.

Если в базе данных существует несколько объектов, наименования которых удовлетворяют введенной строке, то на экране будет отображен список этих объектов. Если среди них находится искомый объект, то его необходимо выбрать с помощью мыши или используя клавиши управления курсором выделить его и нажать Enter.

Если список слишком велик — можно воспользоваться полосой прокрутки.

Кнопка вызова/скрытия списка

Если поле ввода пустое, то при нажатии на кнопку вызова списка на экран будет выведен список всех доступных объектов. Если в поле ввода введено наименование объекта или его часть, то при нажатии на кнопку вызова списка будет произведен поиск в базе данных и на экран будет выведен список найденных объектов. Если найден только один объект, то его наименование будет подставлено в поле ввода.

Если при открытом списке объектов нажать на кнопку вызова/скрытия списка, то он будет закрыт.

Список объектов

Список объектов выводится на экран когда пользователь нажимает на кнопку вызова списка или если был задан поиск объекта по наименованию и для введенной строки в базе данных существует несколько подходящих объектов.

Полоса прокрутки списка

Если список содержит большее количество объектов чем помещается на экране, то можно воспользоваться полосой прокрутки для пролистывания списка вверх или вниз.

Панель инструментов

Находясь в списке объектов можно вызвать команды:

- Создания нового объекта,

- Изменения выбранного объекта,
- Удаления выбранного объекта,
- Копирования выбранного объекта.

Соответствующие пиктограммки расположены на панели инструментов в верхней части выпадающего списка.

В правой части панели инструментов расположены стрелки изменения ширины выпадающего списка.

Горячие клавиши

Обычно, при работе с выпадающим списком (когда курсор находится в поле ввода) доступны горячие клавиши, приведенные ниже. Мы говорим обычно, потому что список доступных команд и, следовательно, активных горячих клавишь зависит как прав текущего пользователя, так и от конкретной настройки выпадающего списка, произведенной настройщиком.

- F1 — вызов справки;
- F2 — создание нового объекта;
- F3 — поиск;
- F4 — изменение выбранного объекта;
- Ctrl-R — объединение двух записей;
- F7 — точный поиск;
- F8 — удаление выбранного объекта;
- F9 — форма объекта;
- F11 — свойства объекта;
- F12 — переключение раскладки клавиатуры.

Вызов справки

Вызов справки по использованию выпадающего списка.

Создание нового объекта

При нажатии клавиши Ф2 на экран будет выведено диалоговое окно ввода нового объекта. После заполнения необходимых полей необходимо нажать Ок, после чего объект будет сохранен в базе, а его наименование подставлено в поле ввода.

В некоторых случаях, когда выпадающий список позволяет выбрать объекты разных типов, например: Компанию, Контакт, Банк и т.п. Перед окном ввода параметров объекта на экран будет выведен список доступных для создания типов объектов.

Поиск

При нажатии клавиши Ф3 будет выполнен поиск в базе данных объекта, наименование которого содержит введенную строку. Можно задать достаточно сложное условие поиска, если использовать специальные символы: "_" и "%". Символ "_" (подчеркивание) означает обязательное наличие любого символа в данной позиции. Символ "%" означает произвольную последовательность символов (включая пустую строку). Эти два символа аналогичны символам "?" и "*", которые можно использовать при поиске файлов в операционной системе.

Например, введя в строке ввода: "технолог%1" мы найдем в базе данных и "ООО Технология, филиал №1" и "Технологическое управление №1".

Изменение выбранного объекта

Если выбран существующий в базе объект, то по нажатию на клавишу Ф4 можно отредактировать его в отдельном окне.

Текущий ключ

Просмотреть идентификатор выбранного объекта можно воспользовавшись клавишей Ф5.

Объединение двух записей

По коомбинации клавиш Ctrl-R можно вызвать окно объединения двух записей.

Точный поиск

По клавише Ф7 можно выполнить точный поиск. Точный поиск выполняется гораздо быстрее, особенно если по полю, по которому осуществляется поиск объекта в базе данных создан индекс. Однако, при этом необходимо набрать полное имя объекта. Если будет введена только часть наименования объект не будет найден.

Удаление выбранного объекта

Выбранный объект можно удалить из базы данных нажав клавишу Ф8.

Форма объекта

Если необходимо просмотреть список объектов или осуществить поиск по сложному критерию, то необходимо открыть Форму объекта по нажатию клавиши Ф9. Выбрать/найти там нужный объект и закрыть форму.

Свойства объекта

По нажатию клавиши Ф11 можно открыть стандартное окно с просмотром свойств выбранного объекта. В частности, из этого окна можно узнать уникальный идентификатор текущей записи.

Переключение раскладки клавиатуры

Чрезвычайно удобная функция для тех, кто привык набирать текст не отрывая глаз от клавиатуры. Если вы набрали строку на белорусском языке, когда ваша клавиатура была переключена на английскую раскладку, вам не придется печатать все заново. Нажатие на F12 не только переключает раскладку, но и конвертирует набранный вами текст из одной раскладки в другую.

Выпадающий список Множество

Поле ввода даты

Калькулятор

Для ввода числовых данных применяется поле ввода с калькулятором. Данный управляющий элемент состоит из следующих частей:

- Поле ввода;
- Кнопка вызова калькулятора;
- Панель с кнопками (панель калькулятора).

Поле ввода

Поле ввода позволяет набрать числовое значение. В простейшем случае его можно использовать как обычное поле ввода, т.е. ввести числовое значение и по клавише табуляции перейти на следующее поле в окне. Если необходимо выполнить вычисление, то следует набрать значение первого операнда. Затем нажать требуемую арифметическую операцию, после чего на экране появится панель калькулятора. Затем следует ввести значение второго операнда и нажать клавишу Ввод или Равно. Панель калькулятора исчезнет с экрана, результат арифметической операции будет вычислен и подставлен в поле ввода.

Кнопка вызова калькулятора

Кнопка вызова калькулятора позволяет вывести на экран панель калькулятора с кнопками. После чего, можно с помощью мыши вводить числовые значения и выполнять над ними арифметические операции. Для того, чтобы закрыть панель калькулятора необходимо нажать кнопку «=» (равно).

Панель с кнопками

Панель с кнопками (панель калькулятора) предназначена для ввода цифр и выбора арифметических операций с помощью мыши. Панель калькулятора отображается на экране либо по нажатию на кнопку вызова калькулятора, либо при нажатии на клавишу Ввод (Enter) когда курсор находится в поле ввода, либо при нажатии на кнопки арифметических операций, когда курсор находится в поле ввода.

Приемы работы с калькулятором

Следует обратить внимание, что поле ввода с калькулятором ведет себя несколько по иному, чем обычное поле ввода. Так, если находясь в обычном поле ввода нажать клавишу Ввод, то сработает кнопка Ок (или Готово) диалогового окна. Если же курсор находится в поле ввода с калькулятором, то по нажатию на клавишу Ввод на экране откроется панель с кнопками. Для того, чтобы закрыть диалоговое окно необходимо сперва с помощью клавиши табуляции переместить курсор на следующее поле и только затем нажать клавишу Ввод.

Первое приложение на Гедымине

В этой главе представлен кратких обзор языка программирования, используемого в платформе Гедымин. После прочтения вы сможете написать свое первое Гедымин-приложение. Здесь мы рассмотрим только основные возможности языка, не останавливаясь на деталях. Также познакомимся с основным инструментарием разработчика, получим общее представление о среде исполнения программного кода. Конкретные свойства и возможности Гедымина подробно изучаются в последующих главах.

Первое приложение

За основу языка программирования в Гедымине был взят VBScript. Зная VBScript, вы не встретите трудностей при написании своих или расширении возможностей уже существующих модулей Гедымина. Если же вам еще не приходилось сталкиваться с программированием на скриптах, то вы можете довольно быстро наверстать упущенное.

VBScript (Visual Basic Scripting Edition) - это упрощенная версия Visual Basic. Не имея таких широких возможностей, как исходный Visual Basic, он, тем не менее является мощным и простым в использовании средством, которое может использоваться для написания небольших программ. Если вы уже знакомы с Visual Basic или Visual Basic for Applications, вы найдете, что работа в VBScript очень проста и необычайно эффективна. Не смущайтесь, если вы не работали с другими версиями Visual Basic. VBScript очень прост в изучении, даже для новичков.

По сложившейся традиции первая программа на изучаемом языке программирования должна выводить фразу «Hello, World!». Текст такой программы на VBScript-е будет выглядеть следующим образом:

```
sub HelloWorld
   MsgBox "Hello, World!"
end sub
```

Запустите Гедымин. Откройте Сервис | Редактор скрипт-объектов. В меню появившегося окна выберите Окна | Проводник. Проводник представляет собой вспомогательное окно с деревом доступных скрипт-объектов. Гедымин позволяет создавать следующие типы скрипт-объектов:

- Макросы (глобальные и локальные)
- Отчеты
- VB-классы
- Константы и переменные
- Методы и события
- Скрипт-функции
- Глобальные VB-объекты
- Проводки

Для создания нашей первой программы мы воспользуемся объектом Скрипт-функция. Для этого необходимо в проводнике выделить папку «Скрипт-функции» и выбрать в контекстном меню пункт «Добавить скрипт-функцию» (чтобы вызвать контекстное меню, нажмите на папке правую кнопку мыши или сочетание клавиш Shift-F10). На экране появилось окно для редактирования скрипт-функций. Оно состоит из нескольких закладок. На закладке «Свойства» в поле «Наименование функции» введите HelloWorld (как на Рис. 27 Рисунок). Затем перейдите на закладку «Скрипт» и наберите текст нашей первой программы (см. Рисунок 0.1). Чтобы сохранить наши изменения выберите в меню Редактора скрипт-объектов Скрипт | сохранить. Теперь в дереве скрипт-объектов в папке «Скрипт-функции» должен появиться наш скрипт.

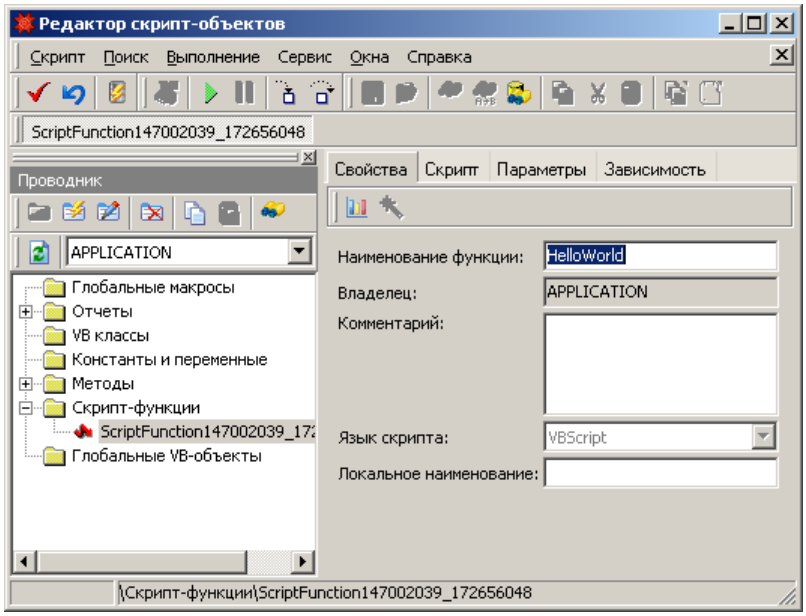


Рис. 27 Рисунок

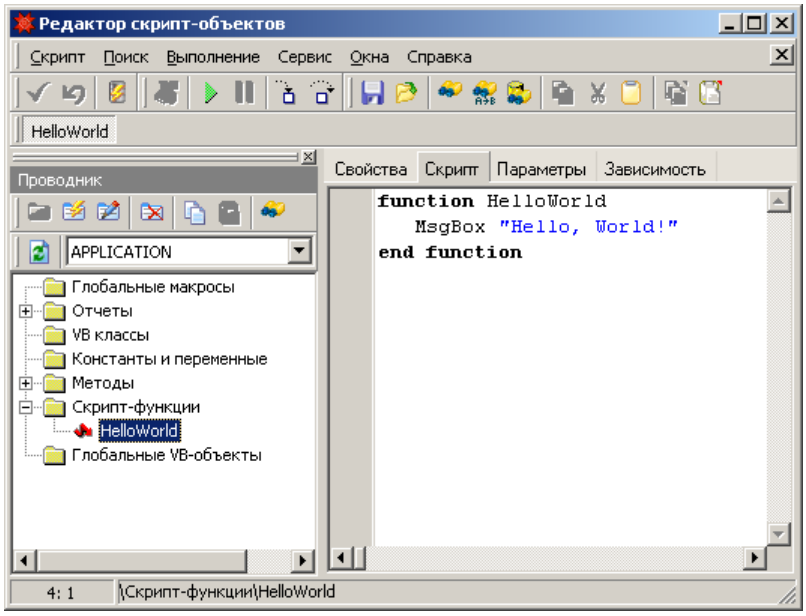


Рисунок 0.1

Осталось запустить наше приложение. Для этого выберите в меню Выполнение | Запустить. Если вы все сделали правильно, на экране у вас появится сообщение: Hello, World!

Итак, мы написали первую программу. Но как она работает?

Программы на VBScript-е строятся на основе скриптов. Скрипт – это код, который выполняется в приложении для некоторых целей. Скрипт может быть создан при помощи скриптового языка. Скриптовой язык позволяет интерпретировать код на стадии его выполнения. Скрипт может состоять из одной или нескольких процедур, описания классов, объявления переменных и констант. Поэтому не стоит отожествлять значение слова «скрипт» со словами «функция», «процедура». Процедуры - это относительно самостоятельные фрагменты программы, оформленные особым образом и обозначенные именами. Упоминание этих имен в тексте программы называется вызовами процедур. Процедуры помогают разбить программу на ряд независимых частей, что необходимо для экономии памяти. Каждая процедура существует в программе в единственном экземпляре, но обращаться к ней можно из разных точек программы. При вызове процедуры вступает в действие последовательность операторов, которые находятся внутри этой процедуры. Процедуре могут быть переданы некоторые параметры.

В VBScript существует два вида процедур: подпрограмма (Sub) и функция (Function). Подпрограмма (Sub) - это последовательность VBScript-операторов, обрамляемая операторами Sub и End Sub. Подпрограммы могут принимать параметры (константы, переменные или выражения, которые передаются при вызове процедуры), но не возвращают значений.

```
sub MyFirstProcedure
  ' тело процедуры
end sub
```

Функция (Function) - это последовательность операторов VBScript, обрамляемая операторами Function и End Function. Она похожа на подпрограмму, но отличается от последней тем, что может возвращать некоторое значение. Функция может принимать параметры (константы, переменные или выражения, передаваемые при вызове). Возвращаемый тип у Function - Variant.

```
function MyFirstProcedure
  ' тело процедуры
  ' возвращаем значение
  MyFirstProcedure = Value
end function
```

Передаваемые параметры могут передаваться либо как значение (обозначается ключевым словом ByVal перед названием параметра), либо как ссылка (обозначается ключевым словом ByRef перед названием параметра). Если параметр передается как значение (ByVal), то после выполнения процедуры его значение не изменится. При передаче параметра как ссылки по завершению выполнения процедуры мы получим новое значение параметра.

```
sub MyFirstProcedure (ByVal AValue, ByRef AReference)
  AValue = 5 'изменяем значение AValue
  AReference = 10 'изменяем значение AReference
end sub
```

В рассматриваемом примере мы изменили внутри процедуры оба параметра. Однако по выходу из процедуры мы получим изменившееся значение только для параметра AReference, т.к. он был передан по ссылке. Значение же параметра AValue останется не измененным.

Все скрипты выполняются при помощи встроенного в платформу Гедымин интерпретатора языка. Интерпретатор переводит компьютеру набранные вами команды непосредственно в момент их выполнения. Он построен на основе встроенного в Windows Microsoft Script Control. Script Control в свою очередь является посредником между интерпретатором Гедымина и Microsoft Windows Script Host. Microsoft Windows Script Host, в русских версиях Windows называемый Сервером Сценариев, представляет собой языконезависимый сервер (контроллер обработчиков сценариев) для 32-х разрядных операционных систем Microsoft Windows. В состав WSH включены обработчики сценариев (scripting engines) языков Visual Basic Script и Java Script. Интерпретатор Гедымина поддерживает скрипты, созданные при помощи Visual Basic Script. Вы можете рассмотреть этот процесс на Рисунок 0.2.

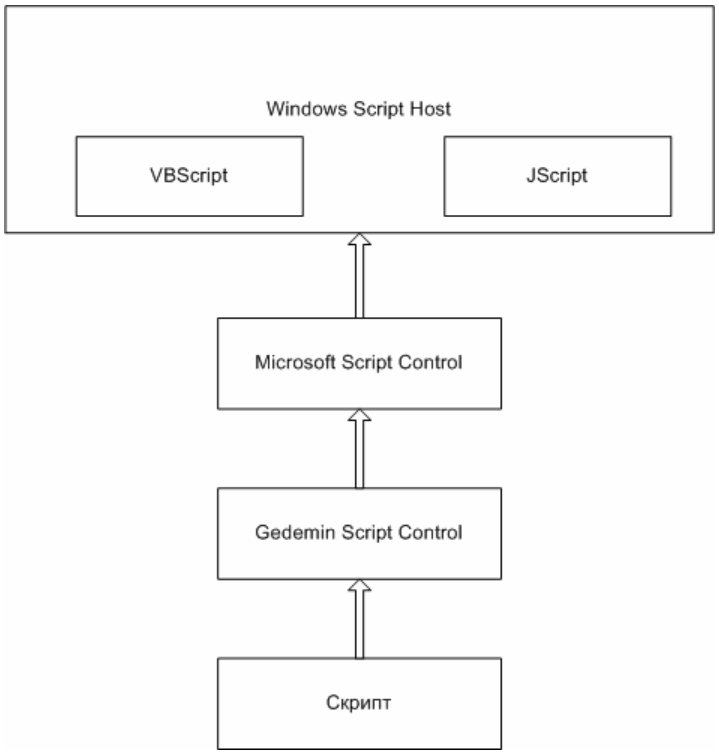


Рисунок 0.2

Текст скрипта загружается в Script Control, проверяется на корректность в соответствии с выбранным языком и выполняется. Как мы уже знаем, скрипт может состоять из нескольких функций. Поэтому мы должны указать Script Control-у, какую функцию запускать, так называемую главную функцию. Все остальные процедуры, присутствующие в скрипте, называются вспомогательными. Именно для этого и служит поле «Наименование функции» на закладке «Свойства» окна редактирования скрипт-функций.

Что такое скрипт-объект

Что же такое скрипт-объект?

Скрипт-объект – это объект, организующий определенным образом работу со скриптами: их создание, изменение, запуск. Как мы уже знаем, Гедымин поддерживает несколько видов скрипт-объектов. Рассмотрим их подробнее.

Скрипт-функция

Скрипт-функция – это наиболее простой случай скрипт-объектов. Она состоит из тела скрипта (собственно сам скрипт), а также из описания свойств функции и ее параметров. Хотя мы и используем в данном случае слово функция в единственном числе, тем не менее тело скрипта может содержать кроме главной функции (та функция, которая будет передана на выполнение), еще и вспомогательные (дополнительные процедуры, необходимые для выполнения главной функции).

Свойства скрипт-функции описывают:

- Название функции – название главной функции, которое будет передано интерпретатору при запуске скрипта на выполнение. Название функции должно быть уникально в рамках ее владельца, если это локальная функция, или по всему Гедымину, если это глобальная функция. По-умолчанию функции присваивается имя ScriptFunctionxxxx_xxxx, где xxxx_xxxx – РУИД функции.
- Владелец – указывает владельца, которому принадлежит функция. Владельцем может быть форма, класс или же собственно сама платформа. Здесь следует различать локальные и глобальные скрипт-функции. Владельцем глобальных функций является APPLICATION. Эти функции доступны из всех модулей Гедымина. Использование же локальных функций ограничено их владельцем. Указать владельца можно в выпадающем списке проводника скрипт-объектов (см Рисунок 0.3).
- Комментарий – содержит описание созданной скрипт-функции.

- Язык скрипта – указывает используемый скрипт-язык. В первой версии это свойство недоступно для изменения, т.к. поддерживается только VBScript. Однако в будущем планируется сделать поддержку также и Java Script.
- Локальное наименование – по сути дела представляет собой сокращенный комментарий.

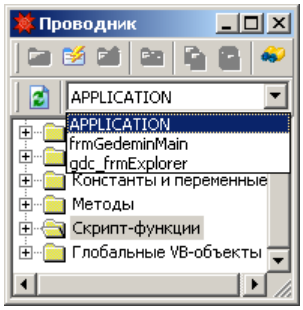


Рисунок 0.3

Некоторые значения, передаваемые при вызове функции и влияющие на ее выполнение, называются параметрами. Рассмотрим пример. Наше первое приложение выводит сообщение «Hello, World!». Однако в заголовке сообщения мы видим строку «VBScript». Расширим нашу функцию и передадим ей параметр, который будет указывать заголовок сообщения.

```
function HelloWorld(ByVal title)
   MsgBox "Hello, World!", vbOkOnly, title
end function
```

Вообще, VBScript работает с типом Variant и по умолчанию считает все переменные вариантными. Однако, если вы указали, что ваша процедура имеет входные параметры, то при вызове ее на выполнение вам будет предложено ввести значения для этих параметров в окне параметров. Изначально все параметры будут представлены целочисленным типом, и на окне параметров будет указан соответствующий визуальный элемент управления для ввода целых чисел. Для того, чтобы указать, что наш параметр является строкой, станьте на закладку «Параметры» окна редактирования функции. В поле наименование параметра укажите его локальное наименование, например «Заголовок», а в выпадающем списке выберите тип «Строка» (см Рисунок 0.).

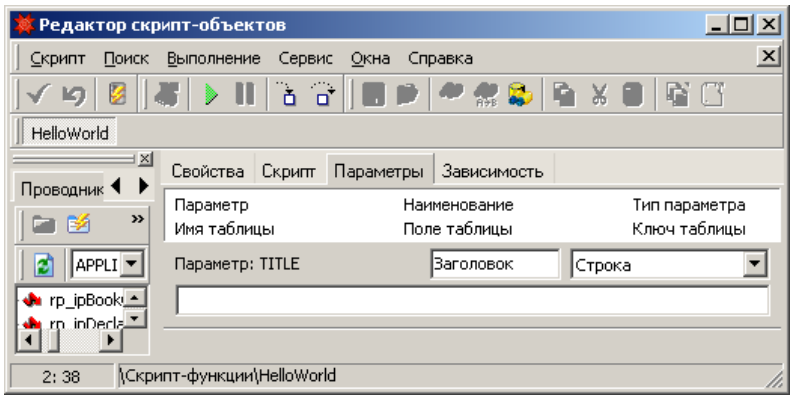


Рисунок 0.5

Теперь, при запуске функции, вам будет предложено ввести заголовок в визуальный элемент управления, поддерживающий ввод строк.

Закладка «Зависимость» указывает, от каких внешних скрипт-функций зависит редактируемая функция, и какие функции зависят от нее. Если взять за пример наше первое приложение, то наша скрипт-функция не зависит ни от кого и ни одна другая скрипт-функция не зависит от нашей.

Макрос

Макрос – подпрограмма, которую можно запустить из меню формы. По сути дела это скрипт-функция с некоторыми специфическими свойствами. Скрипт-функция в общем-то является базовым объектом для всех остальных скрипт-объектов Гедымина. Запустить ее на выполнение мы можем только из окна редактора. Макрос же позволяет настроить запуск скрипта.

Свойства макроса:

- Наименование макроса – наименование, с которым макрос будет отображаться в дереве скрипт-объектов или в меню формы. По-умолчанию равно наименованию функции.
- Горячая клавиша – сочетание клавиш, по нажатию которых будет вызван макрос. По-умолчанию не задано.
- Отображать в меню макросов формы – указывает, отображать ли макрос в меню формы-владельца. Если макрос является глобальным, то он будет отображаться на всех формах.

По умолчанию, каждый макрос создается с параметром OwnerForm. В описании типа этого параметра стоит «не запрашивается». Данный тип параметров используется для передачи скрипт-функции ее владельца. Владельцем является форма / объект, вызвавшие скрипт на выполнение. Если макрос является глобальным, то владелец не определен. Поэтому для глобальных макросов необходимо удалять первый параметр OwnerForm. Если вы оставили этот параметр, то перед каждым запуском глобального макроса вам будет выдаваться сообщение, что параметр OwnerForm не определен. При этом вам придется подтвердить запуск макроса.

Константы и переменные

Как уже ясно из названия, данный скрипт будет описывать константы и переменные. Этот скрипт-объект может быть только глобальным, т.е. его владельцем может быть только APPLICATION. Описанные константы и переменные будут доступны из всех модулей Гедымина. Поэтому их названия должны быть уникальны. При попытке добавить глобальную константу или глобальную переменную с уже существующим именем Script Control выдаст ошибку о переопределении переменной / константы и отключит все глобальные объекты.

Отличием от скрипт-функции является то, что тело скрипта не содержит заголовка функции и ее окончания. Текст состоит только из объявления переменных и констант. Например:

```
public GlobalVariable 'глобальная переменная
public const GlobalConst = 1 'глобальная константа
```

Глобальные переменные и константы становятся доступными из всех скриптов Гедымина сразу же после их сохранения.

Отчеты

Отчет – это наиболее сложный вид скрипт-объекта. Используется для формирования отчетов в системе Гедымин. Описание отчетов читайте в главе 10.

Методы и события

Методы и события – это скрипт-объекты, используемые для расширения функциональности классов форм и бизнес-объектов. Более подробное их описание читайте в главах 6 и 7.

VB-классы и VB-объекты

VBScript позволяет создавать классы и объекты.

Класс – это вид структуры, инкапсулирующей данные и функции в удобную упаковку. По сути, класс – это шаблон, по которому создаются экземпляры VB-объектов. Он определяет свойства объекта, его поведение. Однако в отличие от классов, используемых в объектно-ориентированных языках программирования, VB-классы не поддерживают наследования.

VB-классы и VB-объекты могут быть локальными и глобальными. Глобальный VB-объект доступен из дерева скрипт-объектов, создается автоматически при загрузке скриптов и доступен из любого модуля Гедымина. Локальный VB-объект вы можете создать прямо в скрипте. После выполнения скрипта он уничтожится.

Например, создадим локальный объект класса TCreator:

```
set Creator = new TCreator
```

Описание VB-классов и объектов читайте в приложении f (!!!!).

Проводка

Объект Проводка предназначен для создания бухгалтерских проводок. Более подробное описание читайте в главе 17.

Параметры скрипт-объектов

Мы уже успели немного познакомится с параметрами скрипт-функций при описании объектов. Теперь рассмотрим их более подробно.

Итак, параметр – это некоторое значение, передаваемое функции / процедуре при ее вызове и влияющее на ее выполнение. При запуске скрипт-функции Гедымин определяет, есть ли у нее запрашиваемые параметры. При наличии таковых пользователю будет предложено задать необходимые значения.

Не запрашиваемым параметром в Гедымине является OwnerForm с типом «Не запрашивается», который говорит системе, что данному скрипт-объекту необходимо передать владельца. Он используется только в макросах и отчетах и должен стоять первым в списке.

Запрашиваемые параметры могут иметь тип:

- Число целое
- Число дробное
- Дата
- Дата и время
- Время
- Строка
- Логический
- Ссылка на элемент
- Ссылка на множество

Для всех простых типов параметров необходимо задать их локализованное наименование, выбрать тип, и (не обязательно) указать комментарий, который будет выводится пользователю как подсказка.

По умолчанию, каждый параметр добавленный пользователем имеет целочисленный тип.

Более сложными типами являются ссылка на элемент и ссылка на множество.

Очень часто нам необходимо выбрать запись из какой-либо таблицы по текстовому полю, но вернуть не значение текстового поля, а ключ записи. Для этого используется тип параметра «Ссылка на элемент». Более сложный случай, когда вам необходимо вернуть массив ключей записей. Для этого используется тип «Ссылка на множество».

Кроме свойств, присущих простым типам параметров, для сложных типов добавляются:

- **Имя таблицы** – наименование таблицы или представления, из которого будет осуществляться выборка записей.
- **Поле таблицы** – поле, значение которого будет выводится в выпадающем списке. По этому значению пользователь будет выбирать запись.
- **Ключ таблицы** – поле, значение которого будет передано как параметр в скрипт. Чаще всего это идентификатор записей таблицы (первичный ключ).
- **Условие** – условие, по которому будут отфильтрованы записи в таблице. Если вы пишете условие с использованием синтаксиса VBScript, то вам необходимо выбрать в выпадающем списке язык VBScript. Для простых условий можно не указывать язык. Использование VBScript-а чаще всего необходимо при вставке в условие глобальных переменных Гедымина, глобальных функций, преобразовании типов и т.д.

И «ссылка на элемент» и «ссылка на множество» возвращают результат в виде массива. Обращение к элементам данного массива начинается с индекса, равного нулю.

Например, нам необходимо отобразить в списке все контакты, имеющие тип «Человек». Для этого мы должны выбрать из таблицы GD_CONTACT все записи, для которых значение поля CONTACTTYPE равно 2. В виде SQL-запроса это выглядело бы так:

```
SELECT id, name
FROM gd_contact
WHERE contacttype = 2
```

При настройке параметра «Ссылка на элемент», закладка «Параметры» будет выглядеть как на Рисунок 0.4

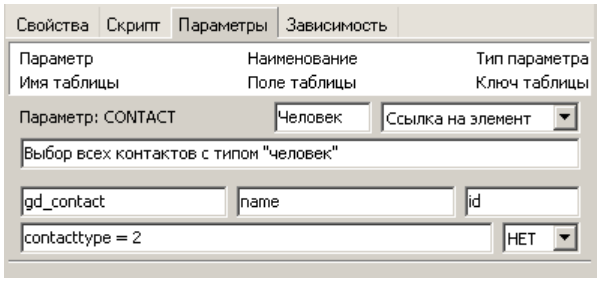


Рисунок 0.4

Пример скрипта, которому передается параметр типа «ссылка на элемент»:

```
option explicit
sub Choose_Contacts(ByVal Contact)
  Dim ContactId
  ContactId = Contact(0)
end sub
```

Здесь мы объявляем переменную ContactId и присваиваем ей переданный нам параметр типа «Ссылка на элемент». При запуске скрипта на экране появится окно вида (смотри Рисунок 0.5)

Рисунок 0.5

Если пользователь нажмет кнопку «Отмена», то запуск скрипта будет отменен. Кнопка «Ok» продолжает запуск скрипта, передавая ему указанные пользователем параметры. Если параметр не был указан, то передается значение, используемое по умолчанию для каждого конкретного типа. Значения, используемые по умолчанию вы можете посмотреть в Таблица 0.1.

Тип	**Значение по умолчанию**
Число целое	0
Число дробное	0
Дата	30.12.1899
Дата и время	30.12.1899 00:00:00
Время	00:00:00
Строка	Пустая строка
Логический	Ложь
Ссылка на элемент	Массив с одним элементом = -1
Ссылка на множество	Вернет ошибку. Требует обязательного указания параметра.

Таблица 0.1

Примечание. Произвольно задаваемые параметры можно использовать не во всех скрипт-объектах, но только в макросах, простых скрипт-функциях и отчетах.

Подключение внешних процедур. Инструкция Include

Итак, мы уже знаем, что скрипт может содержать несколько процедур. Одна из них является главной, остальные вспомогательными. Однако не всегда вспомогательные процедуры будут содержаться в том же скрипте, что и главная. Существует возможность подключить процедуры из другого скрипта. Для этого используется инструкция **Include**.

Синтаксис подключения внешней процедуры следующий:

```
'#Include <Имя процедуры>
```

Теперь вы можете вызывать подключенную процедуру из любого места скрипта. Если вы забудете подключить процедуру и попробуете к ней обратиться, то VBScript выдаст вам ошибку о несоответствии типа.

Подключать можно не все процедуры. Вы можете подключить только главные процедуры из глобальных скриптов или скриптов, имеющих того же владельца, что и владелец вашего скрипта. Подключенная процедура ищется по имени. При этом, если ваш скрипт локальный, т.е. имеющий владельцем форму или объект, то Script Control сначала будет искать процедуру с таким именем среди скриптов текущего владельца, и только потом среди глобальных скриптов.

Редактор скрипт-объектов

С некоторыми пунктами меню редактора скрипт-объектов мы уже успели познакомится, теперь рассмотрим его более подробно.

Меню Скрипт

Содержит следующие пункты для работы с телом скрипт-объекта:

- Сохранить **– *сохраняет изменения текущего скрипт-объекта в базу.***
- Сохранить все – сохраняет изменения всех открытых на редактирование скрипт-объектов.
- Откат изменений – отменяет все сделанные изменения для текущего скрипт-объекта.
- Сохранить скрипт в файл – сохраняет текст текущего скрипта в файл на диске.
- Загрузить скрипт из файла – загружает текст скрипта из выбранного файла. Загрузка/сохранение скрипта в файл доступны только, когда активна закладка «Скрипт» окна редактирования скрипта.
- Копировать – копирует выделенный текст из окна редактирования тела скрипта в буфер обмена.
- Вырезать – вырезает выделенный текст из окна редактирования тела скрипта в буфер обмена.
- Вставить – вставляет в тело скрипта скопированный ранее текст из буфера обмена.
- Копировать SQL – аналогична команде «Копировать», за исключением того, что при копировании убирает начальные и концевые двойные кавычки и знаки переноса строки.
- Вставить SQL – аналогична команде «Вставить», за исключением того, что при вставке добавляет начальные и концевые двойные кавычки и знаки переноса строки.
- Закрыть – закрывает текущий редактируемый скрипт-объекта.
- Закрыть все – закрывает все открытые на редактирование скрипт-объекты.
- Открыть – предлагает список из девяти последних измененных скрипт-объектов.

Меню Поиск

Содержит следующие пункты для организации поиска и замены:

- ***Поиск*** – выводит диалог для поиска заданного текста внутри тела текущего скрипта.
- ***Поиск в базе данных*** – выводит диалог для поиска скриптов в базе данных по заданным параметрам. Результат поиска отображается в окне «Сообщения».
- ***Замена*** – выводит диалог для замены искомого текста заданным. Действует только для текущего скрипта.

Меню Выполнение

Данное меню содержит команды по отладке и выполнению скрипт-функций. Содержит следующие пункты:

- ***Компилировать проект*** – проверяет все скрипт-объекты на уникальность имен. Результаты компиляции отображаются в окне «Сообщения».
- ***Построить отчет*** – запускает на построение текущий отчет. Если отчет не выбран, пункт меню недоступен.
- ***Запустить*** – выполняет текущую скрипт-функцию.
- ***Проверка синтаксиса*** – проверяет синтаксис текущего скрипта в соответствии с выбранным скрипт-языком.
- ***Пауза*** – приостанавливает выполнение скрипта.
- ***Шаг в*** – выполняет текущий оператор скрипта и останавливает скрипт перед выполнением следующего. Используется при включенном режиме отладки. Если к моменту обращения к этому пункту режим отладки не был запущен, то выполнение скрипта останавливается перед первым исполняемым оператором. Если текущий оператор содержит обращение к другому скрипту, то управление будет передано во внутрь этого скрипта.
- ***Шаг через*** – работает как и предыдущий пункт меню, за исключением того, что если текущий оператор содержит обращение к другому скрипту, то этот скрипт будет выполнен обычным образом и выполнение остановится перед оператором, следующим за обращением к скрипту. Используется при включенном режиме отладки.
- ***Перейти к курсору*** – начинает или продолжает режим отладки исполняемого скрипта. Скрипт будет выполняется до тех пор, пока не достигнет указанной строки. Затем выполнение будет приостановлено. Продолжить выполнение скрипта можно командами «Запустить», «Шаг в», «Шаг через», «Перейти к курсору». используется при включенном режиме отладки.
- ***Перейти на выполняемую строку*** – работает в режиме отладки скрипта. Возвращает курсор на выполняемую строку.
- ***Сброс программы*** – доступен в режиме отладки. Останавливает выполнение скрипта. Крайне нежелательно часто использовать, так как данная команда не высвобождает до конца ресурсы.
- ***Установить/снять точку останова*** – устанавливает/снимает точку прерывания исполнения скрипта. Скрипт будет выполняться до точки останова. Затем выполнение будет приостановлено. Используется при включенном режиме отладки.
- ***Вычислить*** – используется в режиме отладки для вычисления текущего значения выбранной переменной.
- ***Добавить в просмотр переменных*** – добавляет переменную скрипта в окно просмотра, что позволяет видеть ее значение по мере выполнения скрипта. Используется при включенном режиме отладки.
- ***Информация о типе*** – выводит информацию о типе выбранной переменной. Доступно в режиме отладки.

Меню Сервис

Содержит команды для отладки скриптов и настройки окна редактора скрипт-объектов. Пункты меню Сервис:

- ***Опции*** – выводит диалог «Опции» с настройками по отладке скриптов, отображению ошибок, фильтрации скрипт-объектов. Диалог «Опции» содержит следующие закладки:
    a. ***Общие***
        i. ***Автосохранение изменений при закрытии редактора*** – автоматически сохраняет внесенные изменения для всех открытых скриптов при закрытии окна «Редактор скрипт-объектов». Иначе при закрытии редактора выдается запрос пользователю о необходимости сохранения изменений. Установлена по умолчанию.
        ii. ***Автосохранение изменений перед запуском*** – автоматически сохраняет сделанные изменения при запуске редактируемого скрипта. Иначе перед запуском скрипта пользователю будет выведен запрос о необходимости сохранения изменений. Установлена по умолчанию.

iii. ***Сохранять положение курсора и закладки*** – сохраняет положение курсора и активную закладку при закрытии скрипт-объекта. При следующем редактировании скрипт-объекта, он откроется на той же закладке, курсор автоматически станет на сохраненную позицию. Установлена по умолчанию.

iv. ***Выводить имя класса + подтип*** - при установленной опции в Проводнике наименование классов выводится в формате имя класса + подтип. При не установленной опции для классов имеющих подтипы выводится только имя подтипа. По умолчанию установлена.

v. ***Предупреждение об обновлении дерева*** – если данная опция установлена то, перед обновлением дерева скрипт-объектов выдается предупреждение о необходимости закрытия редактируемых скрипт-объектов. Установлена по умолчанию.

vi. ***Восстанавливать рабочий стол*** – сохраняет информацию об открытых на редактирование скриптах. При последующей загрузке редактора скрипт-объектов, рабочий стол будет восстановлен. Установлена по умолчанию.

b. ***Отладка***

i. ***Использовать отладочную информацию*** – при установленной опции становится доступной вся функциональность встроенного отладчика: пошаговое исполнение, остановка на точках прерывания, вывод скрипта и выделение строки, в которой произошла ошибка и т.д. По умолчанию выключена.

ii. ***Сохранять время выполнения СФ в файл ScriptRunTime.log*** - при установленной опции происходит сохранение времени выполнения каждой функции. Установка данной опции полезна при оптимизации кода функций. Просмотреть время выполнения можно в окне Время выполнения либо в файле ScriptRuntime.log, находящегося в директории, содержащей gedemin.exe. При каждой загрузке Гедымина данная опция сбрасывается. Рекомендуется использовать только при отладке функциональности.

c. ***Ошибки***

i. ***Останавливать при возникновении ошибок*** – при возникновении ошибки Visual Basic-a и установленной опции открывается окно Редактор скрипт-объектов с открытой на редактирование функцией, содержащей ошибочный код. Строка, в которой произошла ошибка, подсвечивается. В окне Сообщения выводится информация об ошибке. Если опция сброшена, то пользователю выводится сообщение об ошибке. По умолчанию выключена. Рекомендуется использовать только при отладке функциональности.

ii. ***Останавливать при возникновении внутренних ошибок*** – аналогична предыдущей опции. Здесь необходимо различать внутренние ошибки и ошибки VBScript-a. Внутренняя ошибка – это ошибка, возникшая внутри COM-объекта. По умолчанию выключена. Рекомендуется использовать только при отладке функциональности.

iii. ***Сохранять информацию об ошибках в файл*** - при установленной опции информация об ошибках сохраняется в указанный файл. Параметр «ограничивать количество строк» ограничивает величину файла ошибок. При достижении максимальной строки, старая информация начинает последовательно стираться с начала, а новая дописывается в конец.

d. ***Фильтр скрипт-функций***

Данные опции позволяют установить скрипт-функции для объектов каких типов будут отображаться. Для каждого скрипт-объекта создаются скрипт-функции различных типов.

i. Выводить скрипт-функции созданные пользователем – отображает в папке скрипты объекта Скрипт-функция.

ii. Выводить скрипт-функции VB-классов – отображает в папке скрипты объекта VB-класс.

iii. Выводить скрипт-функции макросов – отображает в папке скрипты объекта Макрос.
iv. Выводить скрипт-функции отчетов – отображает в папке скрипты объекта Отчет.
v. Выводить скрипт-функции методов – отображает в папке скрипты объекта Метод..
vi. Выводить скрипт-функции событий – отображает в папке скрипты объекта Событие.
vii. Выводить скрипт-функции проводок – отображает в папке скрипты объекта Проводка.

e. ***Фильтр***
позволяет настраивать дерево проводника скрипт-объектов, фильтруя данные по определенным параметрам, таким как наименование скрипт-объектов различных групп, отображение только переопределенных методов и событий, отображение только отключенных методов и событий. Про переопределение и отключение методов и событий см в главах 6 и 7.

- ***Установки редактора*** – выводит диалог для настройки собственно окна редактора скриптов. Диалоговое окно содержит три закладки:

f. ***Общие***
Содержит опции редактора, такие как настройка табуляции, отката выбора схемы горячих клавиш. Группа **Опции редактора** содержит следующие подпункты:
i. Режим автоматического отступа – при переносе на следующую строку позиционирует курсор ровно под первым значащим символом предыдущей строки, добавляя в начало новой строки пробелы. Если эту опцию отключить, то при добавлении новой строки курсор будет устанавливаться в самый первый слева символ. По умолчанию включен.
ii. Режим вставки – переключает режимы вставки / замены при наборе текста. Аналогичен работе клавиши Ins. Указывает какой режим будет включен при открытии скрипта на редактирование. По умолчанию установлен.
iii. Интеллектуальная табуляция – включает «интеллектуальную» табуляцию, т.е. при нажатии на клавишу Tab курсор устанавливается в позицию равной позиции первого символа следующего слова в предыдущей строке. По умолчанию включен.
iv. Замена табуляции на пробелы – заменяет символ табуляции на пробелы. Количество пробелов равно количеству позиций, на которые сдвигается курсор при нажатии на Tab. По умолчанию выключен.
v. Групповой откат – при отмене последней операции отменяет всю группу повторяющихся команд. Например, добавление нескольких пустых строк подряд будет отменено при откате. Если же эту операцию отключить, откатится только последняя добавленная строка. По умолчанию включен.
vi. Курсор за пределами EOF – Дает возможность позиционировать курсор в любой части рабочей области редактора, или, проще говоря, за пределами набранного текста. По умолчанию отключен.
vii. Откат после сохранения – Сохраняет историю команд после сохранения скрипта. По умолчанию отключен. (!!! Не работает)
viii. Отслеживать пустые пробелы – Сохраняет пробелы в конце строки. По умолчанию включен. (!!! Не работает)
ix. Поиск текста от курсора – Помещает слово, на котором стоит курсов, в поле «искомый текст» при вызове команды поиск. Если эта опция отключена, то искомый текст необходимо вводить руками. По умолчанию включен.
x. Подсветка синтаксиса – включает подсветку синтаксиса. Параметры подсветки можно настроить на закладке «Цвета». По умолчанию включен.

Группа **граница и гуттер** содержит следующие подпункты:

xi. Показывать границу – показывает правую границу в редакторе скрипта. Ширину границы можно задать в поле «Граница». По умолчанию, ширина равна 80 символам. При начальных установках этот пункт отключен.

xii. Показывать гуттер - показывает гуттер. Гуттер – левая часть рабочей области редактора, имеющая серый фон, используется как «поле для заметок». Например, на нем устанавливается точка прерывания. Ширина гуттера по умолчанию равна 30 пикселям. При начальных установках этот пункт отключен.

Быстрая настройка редактора предназначена для быстрой конфигурации редактора (!!! Не работает). По умолчанию Default.

Размер буфера отката устанавливает количество команд, которые сохраняются в истории. По умолчанию, сохраняется 1024 команды. При достижении максимального количества, самые старые команды удаляются, новые добавляются в список. Внимание, ввод / удаление одного символа равны одной команде.

Позиция табуляции устанавливает на какое количество символов будет перемещаться курсор при нажатии на клавишу Tab. По умолчанию количество символов равно восьми. При включенном режиме интеллектуальной табуляции не работает.

g. Назначение клавиш

На данной закладке устанавливается схема работы горячих клавиш.

Всего доступно две схемы: Default (назначение клавиш как в ОС Windows) и Classic (назначение клавиш, как в классике Borland-a). По умолчанию используется схема Default. Вы можете назначить свое сочетание клавиш в каждой из схем посредством нажатия кнопки «Редактировать», или вернуть настройки по умолчанию посредством нажатия кнопки «Сбросить по умолчанию».

h. Цвета

Данная закладка предназначена для настройки подсветки синтаксиса. Для наилучшего восприятия кода программы рекомендуется выделять различными цветами и типами начертания шрифтов числа, строки, комментарии, зарезервированные слова. Настройка подсветки синтаксиса производится для каждого языка по отдельности. Выбрать язык вы можете в выпадающем списке (см Рисунок 0.6).

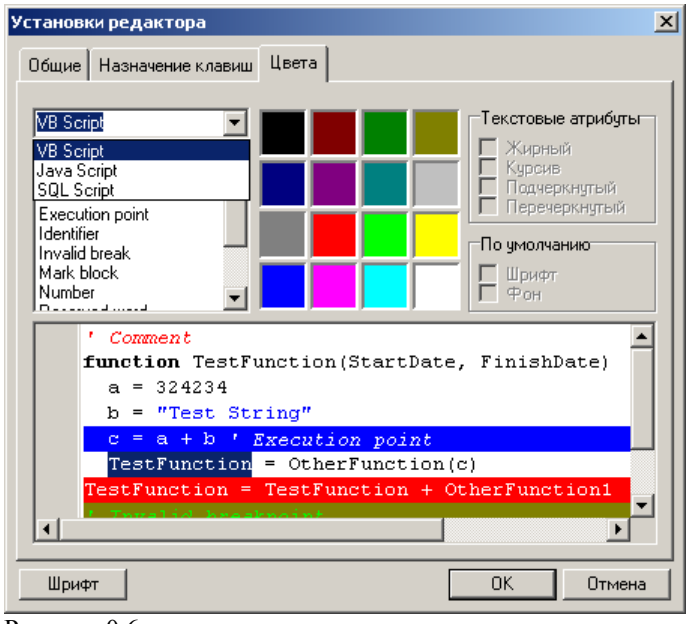


Рисунок 0.6

Также внизу окна есть кнопка «**Шрифт**», которая выводит диалог настройки шрифта редактора скриптов.

- ***SQL редактор*** – выводит окно для выполнения SQL-запросов. Окно поддерживает подсветку синтаксиса SQL (см закладку «Цвета» диалога «Установки редактора»).

Меню Окна

Содержит команды для отображения вспомогательных окон редактора скрипт-объектов. Пункты меню:

Проводник – выводит окно проводника скрипт-объектов. Мы уже успели с ним познакомится несколько раньше (см Рисунок 0.3).

Время выполнения – выводит окно (см. Рисунок 0.7) для отслеживания времени выполнения скрипт-функций. Используется при отладке для оптимизации времени, затрачиваемого на исполнение программного кода. Панель окна содержит три кнопки:

Сохранять время выполнения – аналогична опции «Сохранять время выполнения СФ в файл ScriptRunTime.log». Включает сохранение времени выполнения скрипта в файл. Во включенном состоянии выводит содержимое файла в окно. При этом в окне отображается:

ID – идентификатор скрипт-функции;

Runtime – время выполнения скрипт-функции в миллисекундах;

FunctionName – имя выполняемой скрипт-функции;

BeginTime – время начала выполнения скрипт-функции.

Не обновлять – останавливает обновление информации в окне. При этом информация продолжает записываться в файл. Работает только вместе с включенной опцией «Сохранять время выполнения».

Обновить – обновляет содержимое окна. Выводит информацию о функциях, которые выполнялись в текущую секунду. Работает вместе с включенной опцией «Не обновлять».

Контекстное меню окна позволяет открыть на редактирование функцию, информация о которой отображается в окне, очистить список.

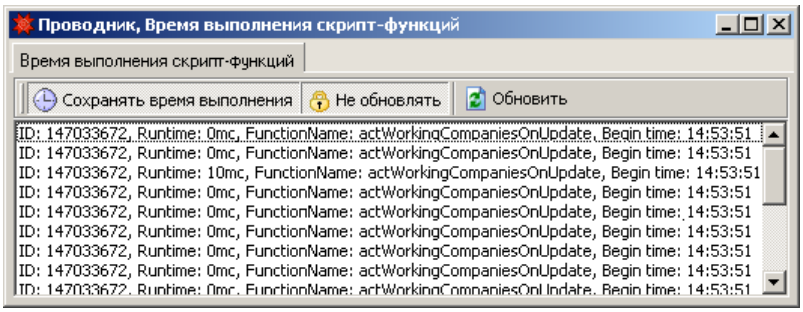


Рисунок 0.7

- ***Инспектор классов*** – выводит вспомогательное окно с иерархией всех классов, доступных в системе Гедымин, глобальных объектов Гедымин, объектов VBScript-а. На первой закладке окна «Классы» представлено дерево классов и объектов. Выпадающий список предназначен для поиска класса / объекта по полному наименованию. Закладка «Поиск» предназначена для поиска по частичному наименованию классов / объектов, их методов и свойств.
- ***Сообщения*** – выводит информационное окно. В окне «Сообщения» отображается информация об ошибках, о поиске, предупреждения, возникшие в процессе компиляции объекта и др.
- ***Стек вызовов*** – выводит окно, в котором в режиме отладки отображается история вызовов функций. В нем вы можете посмотреть последовательность вызова функций и параметры, переданные каждой функции. Верхняя запись списка окна «Стек вызовов» указывает на последнюю вызванную функцию. Ниже находится список функций, вызванных до неё. Самая нижняя запись указывает на функцию, вызванную первой.
- ***Список переменных*** – выводит окно, в котором в режиме отладки вы можете видеть текущие значения заданных вами выражений. Если в точке выполнения одна из переменных в выражении не определена, значение выражения также становится неопределенным.

- ***Точки остановки*** – выводит окно со списком всех точек прерывания, заданных текущим пользователем. В окне отображается идентификатор скрипт-функции, наименование скрипта, условие и количество проходов для каждой точки остановки.

Меню Справка

Содержит команды для вывода справки по таким разделам, как VBScript, Fast Report (используется для создания отчетов), Руководство пользователя и программиста.

Хранение скрипт-объектов в базе. Кэширование

В этом разделе дается краткий обзор структуры для хранения скрипт-объетов в базе данных и рассказывается о кэшировании скриптов на локальном компьютере. Поэтому, если вы почувствовали зевоту, то можете смело пропустить данный раздел.

В основе всех скрипт-объектов лежит скрипт-функция. Скрипт-функции хранятся в таблице GD_FUNCTION. Поле MODULE данной таблицы содержит тип функции. Может принимать следующие значения:

Значение поля MODULE	**Объект, которому принадлежит функция**
CONST	Константы и переменные
ENTRY	Проводка
EVENTS	Событие
GLOBALOBJECT	Глобальный VB-объект
MACROS	Макрос
METHOD	Метод
REPORTMAIN	Главная функция отчета
REPORTPARAM	Функция параметров отчета
REPORTEVENT	Функция событий отчета
UNKNOWN	Скрипт-функция
VBCLASSES	VB-класс

Таблица GD_FUNCTION кроме непосредственного описания функции хранит также ее параметры, права доступа, дату и время последнего редактирования функции.

Для сложных объектов, таких как События, Методы, Отчеты существуют вспомогательные таблицы:

Скрипт-объект	**Дополнительные таблицы**
Событие, метод	Все таблицы с префиксом EVT_
Отчет	Все таблицы с префиксом RP_

Больше всего времени обычно уходит на считываение скриптов из базы и последующую загрузку их в скрипт-контрол. Для ускорения работы системы Гедымин используется кэширование скриптов на локальный диск, т.е. сохранение всех скриптов в определенной структуре в файл. При последующей загрузке Гедымина проверяется, изменялись ли скрипты в базе, и если версии скриптов в базе и в локальном кэше совпадают, то скрипты грузятся из файла. В противном случае файл уничтожается, и загрузка происходит из базы. Версионность скриптов отслеживается при помощи триггеров на таблице GD_FUNCTION. Последняя версия скриптов хранится в генераторе GD_G_FUNCTIONCH. Считывание скриптов из файла происходит быстрее, чем выполнение запроса к базе данных. Однако, здесь есть свои минусы. Кэширование не позволяет в процессе работы увидеть изменения, внесенные в скрипты другим пользователем. Вы увидите изменения только после переподключения к базе. Кроме того, при редактировании некоторых скриптов вами, их изменения загрузятся в Script Control только после последующей загрузке Гедымина.

Общие правила оформления кода

Эта глава содержит рекомендации по оформлению кода, а также раскрывает несколько важных моментов по работе с оперативной памятью.

Что такое хороший код

Понятие хорошего кода не является жестко определенным. По этому вопросу до сих пор ведутся дискуссии в программистских кругах. На данный момент негласно используется несколько принципов, которым должен отвечать «хороший» код:

- **Использование определенного стиля написания кода**. Стиль программирования – это написание кода, согласно некоторым правилам, которые описывают визуальное форматирование, именование переменных, типов, процедур, функций, классов и других категорий. Использование определенного стиля упрощает чтение текста программ, помогает при визуальном тестировании кода на выявление ошибок. Кроме того, код становится более доступным для других программистов, что немаловажно. Принцип использования стиля применяется не только в программировании, но и других областях. Возьмем, к примеру, любое печатное издание. Если каждую главу оформить по-разному, то затрудняется восприятие информации. И, напротив, при однородном оформлении читатель уже знает, на что следует обратить внимание, а что можно пропустить.
- **Модульность.** Разбиение на модули достаточно важный принцип в программировании. Например, вам для различных целей нужна одна и та же функция. Вместо того чтобы дублировать код и копировать необходимую функцию в каждый модуль, лучше выделить более общие и часто используемые функции и константы в отдельный модуль и подключать его по мере надобности. Подобный подход уменьшает количество возможных ошибок, т.к. достаточно большой процент их возникает при копировании кусков кода из одного модуля в другой. Выделенный в отдельный модуль функции достаточно один раз отладить и затем их можно использовать в любом месте программы.
- **Расширяемость и адаптируемость.** Это наиболее важный и востребованный принцип создания программного средства. Предположим, вы написали программу, которая печатает платежные документы, соответствующие принятым в Республике Беларусь стандартам. Однако на территории Российской федерации ваша программа уже будет не востребована, т.к. у них используются совсем другие требования по оформлению платежных документов. Но вы можете адаптировать ваш код под новые условия и продать программу в другой стране. Расширяемость позволяет достаточно легко нарастить функционал. Обычно с принципом расширяемости довольно тесно связан принцип наследования. Возможность достаточно легкой адаптации программы под новые условия и расширяемости ее функциональности – это признак профессионализма.
- **Инкапсуляция.** Данный принцип уже захватывает основы объектно-ориентированного программирования. Если говорить просто, то некоторая служебная информация (функции, переменные) должна быть доступна только в определенном месте во избежание установления некорректных значений, что может привести к ошибке. Использование принципов функционального программирования не позволяло пользователям назначать уровни доступа к отдельным переменным. Однако объектно-ориентированное программирование решило данную проблему. Теперь вы можете описать класс, который будет выполнять определенные функции, скрывая от пользователя промежуточные вычисления и не давая доступ к внутренним переменным.
- **Использование комментариев.** Данный пункт достаточно близко граничит со стилем оформления кода. Грамотный комментарий позволяет упростить понимание вашей логики. Каждый человек по-своему уникален. Его решение конкретной задачи может быть не понятно другому человеку без сопутствующих объяснений. Комментарии важны не только для сторонних программистов, т.к. часто сам создатель не может вспомнить, что именно он хотел реализовать в данном методе.
- **Использование принципов наследования.** Предположим, вам нужно создать несколько форм со стандартным поведением, а различаться они будут в некоторых мелочах. Можно конечно создать каждую форму отдельно и продублировать повторяющийся код. Однако намного правильнее создать базовую форму, на которой описать стандартную функциональность, а затем от нее наследовать остальные формы.

На наследуемых формах можно добавить необходимые дополнительные методы или перекрыть существующие. Принцип наследования также относится к объектно-ориентированному программированию.

- **Своевременное высвобождение ресурсов.** Несоблюдение данного принципа является не просто дурным стилем программирования, но и существенно влияет на работоспособность программного продукта. Одним из ресурсов, о которых идет речь, является память. Если вовремя не высвобождать ее, то на каком-то этапе ваш программный продукт может вызвать сбой в системе, ее зависание и даже перезагрузку компьютера. Это не говоря уже о том, что поглощение памяти существенно снижает скорость работы всех программ. Еще одним важным ресурсом является процессор, или степень его загрузки. Поэтому для весьма продолжительных и часто повторяющихся операций логично применять кэширование на локальный компьютер. Например, вам необходимо проверять права пользователей на доступ к различным объектам. Функция, которая возвращает права для конкретного пользователя, делает запрос к базе, затем вычисляет выражение с использованием полученных значений. Права могут проверяться достаточно часто, а их изменение происходит очень редко. Поэтому логично вычислить при первом обращении и сохранить в некой структуре. Еще один ресурс, на котором мы заострим внимание – это дисковое пространство. Грамотное написание кода позволяет минимизировать размер вашего программного продукта. Кроме того, создавая временные файлы при работе вашей программы, вы должны обеспечить их своевременное удаление.
- **Обработка ошибок.** Данный принцип тесно связан с принципом о высвобождении ресурсов. Как обеспечить, чтобы занимаемый вами ресурс освободился, если у вас возникла ошибка? Грамотная обработка ошибок позволяет и высвобождать ресурсы, и не допускать сохранения некорректных значений в базе данных. Кроме того, при возникновении определенной ошибки вы можете выполнить определенную операцию, если вам это необходимо.
- **Предварительное объявление переменных.** Данный принцип актуален только для языков, которые позволяют работать с необъявленными переменными. Языки высокого уровня, такие как Pascal, C++, требуют обязательного объявления переменных. Макроязыки, например, VbScript позволяют использовать переменную без ее предварительного объявления. С одной стороны это удобно: нужна переменная – сразу ей присваиваешь значение и работаешь. Однако в данном случае затрудняется отладка приложения. Ошиблись вы на одну букву, и язык решил, что это новая переменная, инициализировал ее неким значением и пустил в дело. Только функция уже будет работать не правильно.
- **Типизация переменных.** Данный принцип также актуален не для всех языков. Некоторые языки программирования просто не поддерживают нетипизированных переменных. И, напротив, существуют языки которые работают только с нетипизированными переменными. Чем плохо использование переменных с вариантным типом? При использовании подобных переменных язык неявно приводит их к подходящему по смыслу типу. Всякие неявные операции уже таят в себе лазейку для ошибки.

Визуальное форматирование

Как уже было сказано выше, визуальное форматирование не является обязательным условием при написании программного кода, однако существенно упрощает чтение текста программ. Четких правил по форматированию кода нет, однако наиболее часто используются следующие:

- Открывающая и закрывающая конструкции должны быть на одном уровне. Например:

```
function Test(ByVal DataSet)
  DataSet.First
  do while not DataSet.Eof
    ' Здесь операции, проводимые над записями датасета
    . . .
    DataSet.Next
```

```
  loop
end function
```

В данном примере есть две пары открывающих и закрывающих конструкций: function и end function, do while и loop

- Каждый новый вложенный уровень текста программы должен иметь отступ на два пробела от предыдущего. Строки одного уровня вложенности должны иметь одинаковый отступ.
- Каждую операцию необходимо писать с новой строки.
- Если текст операции слишком длинный и не может целиком отображаться на экране, то его необходимо корректно перенести на следующую строку. Причем первый перенос увеличивает основной отступ на два пробела. Все последующие переносы пишутся на одном уровне с первым.
- Если текст процедуры слишком громоздкий, то желательно разграничивать его функциональные блоки пустыми строками.
- Для простоты восприятия выражения необходимо разделять его операторы и операнды пробелами.
- При написании перечисления через запятую между запятой и последующим членом перечисления должен быть пробел.
- В сложных выражениях со множеством различных операторов желательно использовать скобки для определения приоритетности выполнения операций.
- При написании sql-запросов желательно придерживаться следующих правил:
  - каждый новый блок должен начинаться с новой строки;
  - вложенные блоки должны начинаться с новой строки с отступом от левого края верхнего уровня на 2 позиции;
  - первый перенос содержимого блока на новую строку должен увеличить отступ на 2 позиции, все последующие переносы пишутся под первым.

Например:

```
SELECT g.id as goodkey, g.name as goodname,
  v.id as valuekey, v.name as valuename
FROM
  gd_good g
  JOIN gd_value v ON v.id = g.valuekey
WHERE
  g.disabled = 0
```

Венгерская нотация, использование префиксов в наименованиях объектов

Венгерская нотация – это правила написания имен переменных. Смысл этих правил в том, чтобы по имени переменной можно было определить ее тип и предназначение. Почему она называется венгерской? Создателем данной нотации является программист из Microsoft по имени Чарльз Симонэ родом из Венгрии, в честь которой нотация и получила название. На данный момент Венгерская нотация достаточно распространена в мире. Microsoft довольно активно использует ее уже больше десяти лет. Однако нотация – это всего лишь стиль программирования. Использовать ее или нет – это полностью надлежит решить вам.

В чем же ее смысл?

Когда возникает нужда ввести в программу новый идентификатор, хороший программист начинает думать о следующих его свойствах:

- ***Содержательность*** – чаще всего ваш код придется использовать другим программистам. Желательно давать такие имена переменным, которые были бы понятны остальным, а также кратко характеризовали саму переменную.
- ***Мнемоничность*** – имя должно легко запоминаться.
- ***Стиль*** – при написании программы желательно придерживаться определенного стиля. Это влияет на информативность текста программ. Проще говоря, нужно давать сходные имена сходным переменным.
- ***Скорость выбора*** – придерживаясь определенной схемы именования переменных, вам не придется затрачивать много времени на придумывание имен.

В основу Венгерской нотации была положена идея именования переменных по типу. Создатель данных правил исходил из следующего:

- Имена будут содержательными и мнемоническими, т.к. по типу переменной легко вспомнить ее назначение и наоборот.
- Соблюдение некоторых правил именования определяет стиль программирования.
- На придумывание имен не тратится время – они конструируются механически.
- Становится более легким визуальное тестирование кода, т.к. из имени переменной легко узнать ее тип, а для каждого типа определены свои операции.

Итак, на основе предложенного подхода были сформулированы следующие правила по выбору имен переменных:

1. Переменные именуются согласно типу, за которым может следовать квалификатор. Их рекомендуется разделять посредством использования заглавных букв.
2. Квалификаторы служат различению переменных одного типа, существующих в общем контексте. Таким контекстом может служить вся система, модуль, процедура или структура данных, в зависимости от того чему принадлежит наша переменная. Если для конкретного случая существует "стандартный" квалификатор, то его и следует использовать. В противном случае выбор остается за вами. Квалификаторы должны содержать информативную часть с точки зрения назначения переменной. Если сам квалификатор состоит из сокращения нескольких слов, то каждое новое слово рекомендуется начинать с заглавной буквы.
3. Простые типы следует обозначать коротко. Это относится к переменным, используемым для промежуточных вычислений, например для переменной цикла, которая используется как счетчик. Кроме того следует ограничить длину имени идентификатора, т.к. очень длинные идентификаторы тяжело не то что запомнить, но и прочесть. Т.е. слишком длинное наименование противоречит принципу мнемоничности.
4. Имена сложных или производных типов следует конструировать из обозначений составляющих типов. К стандартным случаям таких типов относятся указатели, массивы, классы.

Оригинал документа «Венгерская нотация» расположен по адресу:
http://msdn.microsoft.com/library/techart/hunganotat.htm.

И в заключение рассмотрим пример. Данной процедуре в качестве параметра передается список строк. Она пробегается по списку и последовательно выводит его содержимое на экран.

```
option explicit


'SL – TStringList. Название переменной состоит
'из первых букв слов, из которых сформировано название типа
sub ShowListContents(ByVal SL)


 ' iCount – целочисленная переменная для
 ' считывания количества записей в SL
  Dim iCount
```

```
    iCount = SL.Count

  'i – переменная цикла
    Dim i
    for i = 0 to iCount – 1
      MsgBox SL(i)
    next
  end sub
```

В системе Гедымин широко используется соглашение об использовании префиксов в наименованиях: **каждый объект, имеющий отношение к определенной подсистеме, начинается с префикса, характеризующего данную подсистему**. В данном случае под подсистемой понимают функциональный блок, реализующий работу в определенной области, а под объектом – любой скрипт-объект, пользовательские формы, объекты базы данных. Соотношение между префиксами и подсистемами можно увидеть в таблице 4.1.

Префикс	Подсистема
acc	Бухгалтерские настройки
bn	Банковские настройки
fa	Основные средства
gs	Общие для все настроек объекты
inv	Складские настройки
ip	Индивидуальный предприниматель
MatInv	Настройки, относящиеся к материальному складу
mn	Меню
OptInv	Настройки, относящиеся к оптовому складу
ren	Аренда
RetailInv	Настройки, относящиеся к розничному складу
wc	Спецодежда
wg	Зарплата, отдел кадров

Таблица 0.1

Предварительное объявление переменных

VbScript позволяет работать с необъявленными предварительно переменными. При вводе новой переменной, ей присваивается значение по умолчанию, исходя из контекста. Например:

```
Quantity = 10
Summa = Quantity * Price ’Summa = 0
```

Здесь мы вводим переменную Quantity и инициализируем ее значением 10, затем вводим переменную Summa и инициализируем ее выражением. Однако в выражении присутствует не объявленная ранее переменная Price. Исходя из контекста, она создается как переменная с численным типом и инициализируется нулем. Поэтому мы получим Summa, равную нулю.

Скрипт-контрол в данном случае не обратит никакого внимания на необъявленную ранее переменную, а просто создаст ее и инициализирует. Подобное поведение интерпретатора языка может привести к многим ошибкам. Ошиблись вы при написании имени переменной, и он просто тихо создаст новую переменную и выполнит скрипт. Чтобы не допустить подобной ситуации, в самом начале скрипта необходимо вставить инструкцию Option Explicit.

Вы уже создавали скрипт-объекты и, наверное, замечали что Option Explicit вставляется автоматически. Использование данной инструкции не позволяет вводить в выражения необъявленные переменные. Это значит, что при обнаружении подобной переменной интерпретатор выдаст сообщение об ошибке и остановит выполнение скрипта. Чтобы объявить переменную необходимо использовать инструкцию

```
Dim <имя переменной>
```

Применяя эти правила, рассмотренный ранее пример будет выглядеть так:

```
Option Explicit
. . .
Dim Quantity, Price, Summa
Quantity = 10
Price = 15
Summa = Quantity * Price 'Summa = 150
```

Разбиение на модули

Что такое модуль?

В классическом программировании используется следующие понятие: модуль (блок) - самостоятельная программная единица, ресурсы которой могут быть использованы другими программами. В Гедымине понятие модуля несколько расширено. Ранее уже шла речь о подсистемах Гедымина, и в данном случае можно сказать, что в основе существующей функциональности, созданной на Гедымине, лежит принцип модульности. Чтобы продолжить данную тему, нам придется столкнуться с понятием «настройка». Настройка на Гедымине – это объединенные по области использования объекты (скрипт-объекты, объекты БД, формы, пользовательский интерфейс). Загрузка настройки на базу означает модификацию структуры базы данных, создание необходимых скриптов, включение обработчиков событий, создание пользовательских форм, настройку интерфейса пользователя, загрузку определенных данных. Более подробно ознакомится с настройками можно в главе 14.

Каждая настройка по сути дела является своеобразным модулем, т.к. она создает законченный по функциональности блок. При этом ресурсы одной настройки могут быть использованы другими.

Зачем вообще деление на модули? Куда как проще все объединить в один блок. Здесь мы сталкиваемся с принципом экономии ресурсов, в частности, дискового пространства. Если вам необходимы только платежные документы, то зачем загружать базу лишними объектами ненужных вам аренды, склада, и др. Чем больше объектов вы создаете на базе, тем больше места она занимает. Чем больше объем базы, тем медленнее работают запросы к ней. Также стоит упомянуть степень простоты работы с базой данных: вполне естественно, что чем меньше объектов создано, тем легче разобраться и в структуре БД, и в существующей функциональности.

Что же делать, если вам не хватает той функциональности, которую предлагает каждый отдельный модуль? Существует два варианта решения данной проблемы:

- дополнение существующего модуля своей функциональностью;
- создание нового модуля с установлением зависимости от существующего.

Первый пункт решения проблемы возможно более простой, чем второй. Однако здесь мы можем столкнуться с трудностями при обновлении модуля. Например, мы дополнили некий модуль своей функциональностью. Первоначальный разработчик модуля со своей стороны также обновил его. Теперь, чтобы загрузить новую версию, нам необходимо полностью повторить все операции по дополнению существующей функциональности нашими объектами, либо отказаться от использования новой версии. Второй пункт в данном случае представляет более гибкое решение проблемы. При загрузке более новой версии первоначального модуля нам нужно всего лишь проверить, не изменились ли вызовы функций или объекты, которые мы расширяли в своем модуле.

Как это все реализуется в системе Гедымин, читайте в главе 14.

Высвобождение ресурсов, использование TCreator. Понятие объекта

Выше мы уже касались вопроса о том, как важно освобождать неиспользуемые более ресурсы. В данной главе мы более подробно рассмотрим высвобождение такого ресурса, как оперативная память.

Любая переменная, объявленная в программном коде, требует резервирования памяти. Размер занимаемой памяти зависит от типа переменной. для переменных простых типов непосредственного управления памятью не требуется: когда мы объявляем переменную определенного типа или инициализируем ее конкретным значением (для языков, поддерживающих нетипизированные переменные), то система автоматически выделяет под нее необходимую память. Как только переменная перестает использоваться, память освобождается. Примерами таких типов являются целочисленные и дробные типы, дата и время, символьные типы и т.д. Немного сложнее обстоит ситуация с объектными переменными.

Итак, что же такое объект? Объект – это совокупность переменных состояния и связанных с ними методов (операций). Эти методы определяют, как объект взаимодействует с окружающим миром.

Примером объекта является таблица (реализация класса TDataSet). Данный объект не только хранит информацию о названии таблицы, о полях, о записях, но также предоставляет пользователю удобный инструментарий для навигации по набору данных, для изменения данных, отображения их свойств.

Класс – это, проще говоря, тип объекта. В отличие от простых типов, класс полностью описывает поведение объекта, возможности работы с ним.

Чтобы создать объект недостаточно просто объявить, экземпляром какого класса он является. Необходимо вызвать конструктор класса. Конструктор – это специальный метод, который создает экземпляр класса. Именно на него возложена роль выделения необходимой памяти и первоначальной инициализации объекта. Для уничтожения объекта нужно вызвать деструктор класса, который корректно освободит занимаемую память.

Во многих объектно-ориентированных языках программирования все операции по управлению памятью возложены на программиста. С одной стороны, это хорошо, т.к. явное управление памятью позволяет осуществлять более жесткий контроль. С другой стороны, это может привести к утечке памяти. Примерами таких языков являются Object Pascal, C ++. Существуют однако и такие языки программирования, которые освободили программистов от необходимости очищать память. Такие языки имеют встроенный сборщик мусора, который отслеживает неиспользуемые объекты и освобождает занимаемую ими память. Примерами таких языков являются Java, VBScript.

Т.к. Гедымин использует интерпретатор VBScript-а, то остановимся подробнее на этом языке, а также рассмотрим некоторые особенности, характерные только для данной платформы.

Синтаксис создания экземпляра класса в VBScript следующий:

```
set NewObject = new ClassName
```

Обратите внимание на два ключевых слова: **set** – говорит интерпретатору о том, что следующая за ним переменная будет объектом, **new** – вызывает конструктор класса ClassName. Забегая вперед, скажу, что класс, экземпляр которого вы будете создавать, обязательно должен быть зарегистрирован описан ранее при помощи синтаксиса VBScript и загружен в скрипт-контрол, иначе интерпретатор его просто не найдет и выдаст сообщение об ошибке.

VBScript является языком, который сам управляет освобождением памяти. исключением являются объекты, реализующие экземпляры внутренних классов Гедымина. Эти объекты создаются дизайнером Гедымина и нуждаются в принудительном уничтожении. Как обойти данную проблему, используя классические средства VBScript? Выход напрашивается сам собой: необходимо создать обычный VB-объект, который будет содержать ссылки на объекты Гедымина, и в деструкторе класса этого VB-объекта прописать уничтожение объектов Гедымина. Тогда по завершению выполнения скрипта этот VB-объект автоматически уничтожится и уничтожит все содержащиеся в нем ссылки.

Чтобы не изобретать каждый раз велосипед, в настройку «Общие данные» был вставлен глобальный VB-класс TCreator. Загрузив данную настройку на базу, вы получите доступ к этому классу из любого скрипта.

Экземпляр класса TCreator создается как обычный VB-объект, используя оператор new. Класс обладает следующими методами:

- ***GetObject(Params, ClassName, Name)*** – создает новый экземпляр класса ClassName Гедымина с именем Name. Params – это параметры, которые передаются в конструктор класса. Чаще всего первый параметр будет nil, а последний – пустая строка. Здесь нужно обратить внимание на два обстоятельства: во-первых, конструкторы классов Гедымина могут иметь входные параметры, в отличие от конструкторов VB-классов; во-вторых метод GetObject можно использовать только для создания объектов Гедымина.
- ***DestroyObject(Object)*** – уничтожает указанный объект и убирает ссылку на него из объекта класса TCreator.
- ***DestroyAllObjects*** – уничтожает все объекты, ссылки на которые имеются в объекте класса TCreator.

Следует заметить, что методы TCreator, вызывающие явное уничтожение объектов, практически никогда не используются.

И в заключение рассмотрим пример.

```
'Включаем отслеживание необъявленных переменных
Option Explicit
'Создаем макрос, который будет последовательно
'выводить все товары
sub mcr_ShowAllGoods
  'Объявляем переменные
  Dim Creator, gdcGood
  'Создаем VB-объект класса TCreator
  set Creator = new TCreator
  'Создаем объект Гедымина класса TgdcGood
  'TgdcGood – дает доступ к таблице gd_good, которая
  'хранит информацию о товарах
  set gdcGood = Creator.GetObject(nil, "TgdcGood", "")
  'Открываем объект для доступа к записям
  gdcGood.Open
  do while not gdcGood.Eof
    'В цикле пробегаемся по всем записям gdcGood
    'и отображаем последовательно название товаров
    MsgBox gdcGood.FieldByName("name").AsString
    gdcGood.Next
  loop
end sub
```

По завершению выполнения данной процедуры VBScript удалит объект Creator. Удаление объекта вызовет деструктор класса TCreator. В деструкторе проверяется, есть ли ссылки на внешние объекты. В данном случае мы имеем ссылку на gdcGood, который и будет уничтожен деструктором объекта Creator.

Забегая вперед, можно сказать, что подобный принцип уничтожения связанных между собой объектов используется классом TComponent. В деструкторе объекта класса TComponent проверяется, какие компоненты были созданы с указанием в качестве владельца данного объекта. Т.к. владелец уничтожается, то логично уничтожить и все порожденные от него объекты.

Подробнее про дизайнер и создание объектов читайте в главе 6.

Обработка ошибок

Создавая некий программный продукт невозможно полностью избежать ошибок. Ошибки можно разделить на:

- синтаксические;
- логические;
- ошибки, вызванные нарушением доступа;
- ошибки, вызванные определенными действиями пользователя.

Синтаксические ошибки отловить легче всего: любой интерпретатор или компилятор просто не пропустит строку, в которой находится синтаксическая ошибка.

Логические ошибки – это ошибки, которые допускает программист при описании определенного алгоритма. Их отловить тяжелее. Они могут не выдавать никаких предупреждений, но влиять на результат выполнения, делая его некорректным. Поэтому в процесс создания программного обеспечения обязательно входит этап тестирования.

Ошибки, вызванные нарушением доступа – это самые грубые ошибки. Они возникают, когда ваша программа пытается получить доступ к чужой области памяти. Чаще всего это происходит, когда объект уже уничтожен, но программа пытается вызвать какой-либо его метод.

Ошибки, вызванные определенными действиями пользователя – это заложенные специально в программный код ошибки. Такие ошибки называют исключениями. Зачем нужны исключения? Затем, что некоторые действия пользователя могут быть неправомерными, и, чтобы не допустить сохранения некорректной информации, программа генерирует ошибку, и откатывает все действия пользователя. Таким образом обеспечивается целостность данных. В отличие от первых трех типов ошибок, данный тип обычно подвергается дополнительной обработке. Т.е. в зависимости от кода возникшей ошибки программа может выполнить те или иные действия.

Возникновение ошибок третьего и четвертого типа прерывает выполнение процедуры / функции и возвращает выполнение программы в ту точку, из которой эта процедура / функция была вызвана. Таким образом компилятор будет подниматься все выше и выше, пока не достигнет уровня, где исключение будет обработано и выведено на экран.

Однако не все исключительные ситуации должны приводить к столь фатальному результату. Например, вы не знаете какого типа пришла к вам переменная. Возможно в ней хранится дата, а возможно строка. Проверить это можно, используя явное приведение типов. Если при попытке преобразовать вашу переменную к типу дата не возникло ошибки, значит она действительно хранит дату. Как же сделать так, чтобы при возникновении исключительной ситуации программа не прерывала свое выполнение, но указала какую операцию выполнить в данном случае?

Здесь мы вплотную подходим к обработке ошибок.

В системе Гедымин существует две возможности обработать возникшую ошибку. Первая возможность реализуется средствами VBScript. VBScript предоставляет программисту конструкцию

```
On Error Resume Next
```

```
On Error Goto 0
```

On Error Resume Next включает подавление ошибок. Это значит, что при возникновении ошибки программа не прервет свое выполнение, но продолжит его просто перейдя к следующей операции. При этом во внутренний объект Err будет передана вся информация о возникшей ошибке.

On Error Goto 0 выключает подавление ошибок и сбрасывает объект Err.

Оба выражения действуют только для скрипта, внутри которого они объявлены, и не распространяются на вложенные процедуры и функции.

Объект Err содержит информацию о последней ошибке, возникающей в процессе выполнения программы. Если вы включили подавление ошибок, то с помощью объекта Err вы можете отслеживать, возникла ли какая-нибудь ошибка.

Свойства объекта Err:

- ***Description*** – хранит описание возникшей ошибки в текстовом виде.
- ***HelpContext*** – хранит идентификатор, указывающий на раздел в файле помощи.

- ***HelpFile*** – хранит полный путь к файлу помощи.
- ***Number*** – возвращает код последней возникшей ошибки. Если Number = 0, то значит к текущему моменту выполнение программы шло без ошибок.
- ***Source*** – хранит оригинальное название объекта, вызвавшего ошибку.

Методы объекта Err:

- ***Clear*** – очищает объект Err.
- ***Raise(number, source, description, helpfile, helpcontext)*** – генерирует ошибку с заданными параметрами. Названия параметров соответствуют свойствам объекта Err.

В качестве примера рассмотрим случай с определением типа переменной.

```
Option Explicit
sub mcr_IsItDate(ByVal AVar)
  On Error Resume Next

  Dim ADate
  ADate = CDate(AVar)

  if Err.Number = 0 then
    MsgBox CStr(ADate) + " является датой"
  else
    MsgBox CStr(AVar) + " не является датой"
  end if

  On Error Goto 0
end sub
```

Отметим, что в конце процедуры не обязательно писать выражение On Error Goto 0, т.к. действие конструкции On Error Resume Next прекращается по завершению процедуры.

Однако, чтобы грамотно работать с конструкцией On Error Goto 0, вам придется в критических точках проверять состояние объекта Err. Иногда это удобно, но чаще всего нет. Нередко требуется выполнить какие-нибудь действия при возникновении исключения и прервать выполнение текущей процедуры. Наиболее распространенными операциями, которые должны выполнится в результате ошибки, являются откат транзакции и удаление лишних объектов.

Как заставить программу перед прерыванием функции выполнить определенные действия? Для этого используется конструкция Гедымина, не присутствующая в классическом варианте VbScript (все, что указано в квадратных скобках является необязательным):

```
Except ExcName(argument1 [argumentList])
  [операторы]
[End Except]
```

Оператор **Except** вызывает процедуру обработки ошибки с наименованием ExcName в случае возникновения исключительной ситуации. Или, проще говоря, если после объявления оператора except и до объявления end except у вас произошла ошибка, то перед прерыванием выполнения функции / процедуры выполнится скрипт с именем ExcName. Скрипт обработки исключительной ситуации обязательно должен иметь хотя бы один входной параметр (argument1).

Замечания

- Функция ExcName вызывается с параметрами переданы ей в момент инициализации, т.е., если переменные, переданные функции ExcName, изменились в дальнейшем коде, то, в случае вызова ExcName, в ней будут использаваться переменные со значениями, которые в них были в момент инициализации ExcName. Инициализация обработчика ошибок происходит при считывании оператора Except.
- Каждое следующее объявление оператора Except отменяет действие предыдущего.
- End Except завершает действие всех вышеобъявленных операторов Except данного скрипта.
- Функцию обработки ошибки можно описать, как в том-же скрипт-объекте после основной функции, так и включить по Include
- Инструкция Except не поддерживается в функциях, используемых для построения отчета.

Описание обработки ошибок будет не полным, если не затронуть вопрос о том, как сгенерировать исключение самому. При рассмотрении объекта Err вы наверняка обратили внимание на метод Raise, который генерирует ошибку с переданными параметрами. Однако по такой ошибке тяжело будет проследить историю ее возникновения (см стек вызовов). В системе Гедымин используется глобальный объект Exception, который позволяет генерировать исключения различных типов.

Синтаксис создания исключения следующий:

```
Exception.Raise <Имя класса>, <Сообщение>
```

Гедымин позволяет генерировать исключения трех типов:

- Exception – в принципе, это наиболее общий случай, используемый в 99 процентах. Исключение данного типа останавливает выполнение скрипта и выводит пользователю указанное сообщение.
- EDivByZero – используется для генерации исключения при делении на 0.
- EAbort – это наиболее интересный тип исключений. Он прерывает выполнение программы, но пользователю ничего не говорит. Поэтому параметр «сообщение» может быть любой строкой. Используется обычно для обрыва некоторых действий. Например, вам нужно запретить добавлять новую запись. Тогда на событие BeforeInsert вы пишите исключение с типом EAbort. Пользователь не увидит, что возникло исключение, но и добавить новую запись не сможет.

Например, перед сохранением значений, которые ввел пользователь, вы обнаруживаете, что он забыл указать номер документа. Сохранять такой документ некорректно. Поэтому необходимо выдать сообщение пользователю и прервать сохранение документа. Забегая вперед, скажу, что такое исключение лучше создавать в методе DoBeforePost или событии BeforePost бизнес-объекта.

```
. . .
if gdcDocument.FieldByName("number").IsNull then
  Exception.Raise "Exception", "Укажите номер документа!"
end if
. . .
```

Кроме того, существует исключение как объект базы данных. Данный тип исключения (SQL-исключения) используется для описания бизнес-логики, используя средства баз данных. Ознакомится более подробно с объектом базы данных «Исключение» можно в главе 8.

Приведение типов

У результата каждого выражения имеется определенный тип. Он задается типом компонентов в выражении и семантикой операторов. Например, если арифметический оператор применяется к операндам целого типа, то результат будет целочисленный. Если же в этом выражении встречаются операнды вещественного типа, то результат будет вещественный.

Приведение типа – это указание конкретного типа переменной. При этом могут осуществляться преобразования от одного типа к другому. Например, преобразование целого числа в строку, даты в строку и т.д.

Существует явное и неявное преобразование типа. При явном приведении типов программист при помощи специальных функций указывает, что он хочет преобразовать переменную к конкретному типу. Неявное преобразование осуществляется самим компилятором / интерпретатором языка при вычислении каких-либо выражений.

VBScript является языком, который поддерживает нетипизированные переменные. Это означает, что тип переменной устанавливается в момент ее обработки в зависимости от ее значения и проводимых над ней операций. Узнать, какой конкретно тип сопоставлен переменной в какой-либо момент выполнения программы, можно, используя функцию VarType.

```
VarType(<переменная>)
```

Значения, которые может вернуть данная функция приведены в таблице 4.2.

Константа	**Значение**	**Описание**
vbEmpty	0	Пусто (не инициализировано)
vbNull	1	Null (не содержит каких либо данных)
vbInteger	2	Целый тип
vbLong	3	Длинный целый тип
vbSingle	4	Вещественный тип с плавающей точкой обычной точности
vbDouble	5	Вещественный тип с плавающей точкой с двойной точностью
vbCurrency	6	Денежный тип
vbDate	7	Дата
vbString	8	Строка
vbObject	9	COM-объект
vbError	10	Ошибка
vbBoolean	11	Логический тип
vbVariant	12	Вариантный тип (используется только с вариантными масивами)
vbDataObject	13	Объект доступа к данным
vbByte	17	Байт
vbArray	8192	Массив

Таблица 0.2

Однако неявное приведение типа может сработать не всегда. Например, если вам необходимо объединить строку и целое число посредством оператора «+», то целое число придется явно привести к строковому типу. Например:

```
MsgBox "Количество подключенных пользователей: " + _
  CStr(iUserCount)
```

Правда, в данном случае можно избежать явного преобразования типа, если использовать вместо оператора «+», оператор «&». Данный оператор не просто сцепляет строки, но и попутно приводит к строковому типу все не строковые операнды выражения.

Так зачем же нужно явное приведение типов? В случае работы с Гедымином явное преобразование типов чаще всего используется для обработки данных, введенных пользователем или для более понятного представления данных. Не секрет, что дата хранится в виде значения вещественного типа. Чтобы вывести дату в нормальном виде с учетом текущих региональных установок используется функция CDate. Кроме того, явное преобразование типов в VBScript позволяет конвертировать дробные числа в целые с их округлением. Функции, преобразования типов, приведены в таблице 4.3.

Функция	Описание
CBool	Выводит логический результат в виде слов True (истина) и False (ложь)
CByte	Преобразует число к типу байт. Если число было дробным, то оно округляется, если число превышает допустимые значения (0..255), то возникнет ошибка.
CCurr	Преобразует число к денежному типу. При этом дробная часть округляется до 4-х знаков после запятой.
CDate	Преобразует выражение к типу дата. При этом выражение может содержать дату как в числовом представлении, так и в строковом виде, соответствующим текущим региональным установкам.
CDbl	Преобразует выражение к вещественному типу с двойной точностью.
CLng	Преобразует выражение к длинному целому типу.
CInt	Преобразует выражение к целому типу.
CSng	Преобразует выражение к вещественному типу с одинарной точностью.
CStr	Преобразует выражение к строковому типу.

Таблица 0.3

Наиболее интересной эта тема является при рассмотрении SQL-запросов. В отличие от VBScript SQL, используемый в Interbase, является жестко типизированным языком. А потому, программисту приходится использовать явное преобразование типов. Для явного преобразования в SQL существует следующая конструкция:

cast(<поле|значение> **as** <тип>).

Например, выведем все товары с заголовком. Первое поле используется только для сортировки:

```
SELECT 1, cast(NULL as integer),
  cast('Все товары' as varchar(60))
FROM rdb$database
UNION
SELECT 2, id, name
FROM gd_good
ORDER BY 1, 3
```

Второе приложение. Пишем Тетрис.

Проводя собеседования с кандидатами на вакансии программистов в нашей компании мне часто приходится сталкиваться с такой реакцией: узнав о том, что в качестве основного средства разработки используется Гедымин, человек делает недовольное выражение лица и начинает выяснять почему мы не пишем на C++, Java или хотя бы на Delphi. В его понимании есть «настоящие» языки программирования и, как бы это выразиться по-мягче, «ненастоящие», годные лишь для того, чтобы создавать простенькие программки печати накладных или учета товаров на складе. Наиболее ретивые сразу заявляют, что иметь что-то общее с «низшими» языками для них слишком оскорбительно и их профессиональная гордость просто не позволит им этого. Конечно, настоящий профессионал никогда не позволит себе подобных высказываний. Ведь он знает, что важно не то, на чем написана задача, а как она написана. Тем не менее, после подобного общения всегда остается определенный горький осадок и желание защитить ни в чем не повинный Гедымин, доказать что он ничем не хуже остальных средств разработки. Можно было бы просто сослаться на те задачи, которые уже созданы на нашей платформе, задачи состоящие из сотен тысяч строк кода и эксплуатирующиеся в реальных, что называется, боевых условиях на сотнях больших и малых предприятий. Можно было бы так же упомянуть, что разработчики, использующие Гедымин, часто выходили победителями в заочных дуэлях с конкурирующими компаниями, которые использовали такие известные языки как Delphi, Java, C#. Но, сразу предвижу возражение скептика: опять вы, мол, про склад и бухгалтерию. Скучно. Вот если бы вы могли доказать, что Гедымин способен решать не только экономические задачи. Удивить нас чем нибудь интересненьким. Вот тогда было бы другое дело. Что ж, отложим разработку складских, бухгалтерских задач до следующих глав и покажем на сколько гибки и обширны возможности системы программирования Гедымин. Создадим… игрушку. Конечно, на четвертый Дум мы не посягнем, но логическая головоломка вроде тетриса нам вполне по силам.

Правила игры

Врядли найдется взрослый человек ни разу в жизни не сталкивавшийся с этой забавной динамичной головоломкой. Правила классического тетриса можно описать следующим образом: фигурки, состоящие из четырех, прилегающих друг к другу, квадратных сегментов, падают сверху вниз в прямоугольном стакане. В стадии падения фигуркой можно управлять, перемещая ее влево-вправо или вращая. Падающая фигурка останавливается достигнув дна стакана или лежащих на дне фигурок. При этом, если образовался один или несколько полностью заполненных квадратными сегментами рядов, то они сгорают (исчезают) и все вышележащие фигурки сдвигаются вниз. Цель игры: продержаться максимальное время и заработать максимальное количество очков, управляя падающими фигурками и размещая их на дне стакана как можно плотнее. Игра считается оконченной, если стакан заполнен настолько, что в нем нет места для размещения очередной фигурки. В процессе игры очки могут начисляться за каждую фигурку, каждый сгоревший ряд и т.п. Скорость падения фигурок может увеличиваться при наборе определенного количества очков или по истечении заданного временного интервала.

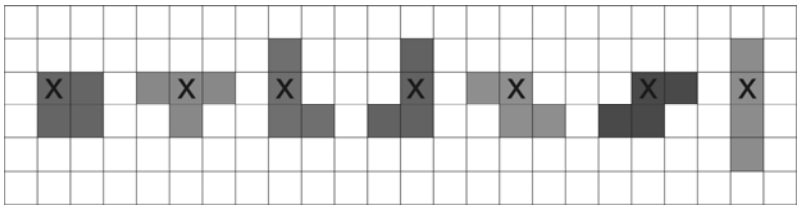


Рис. 28 «Фигуры классического тетриса».

Проектирование тетриса

Согласно методике объектно-ориентированного проектирования рассмотрим задачу и выделим самостоятельные сущности, которые опишем с помощью классов. Очевидно, что такими сущностями в нашем случае являются:

- Фигура;
- Стакан;
- Тетрис;

Фигура

В тетрисе каждая фигура состоит из четырех, прилегающих друг к другу, сегментов. Всего возможно семь различных фигур, отображенных на Рис. 28 «Фигуры классического тетриса». Фигуру можно поворачивать на 90° за один раз, т.е. возможно четыре угла поворота: 0°, 90°, 180°, 270°. Поворот присходит вокруг оси, перпендикулярной плоскости фигурки и проходящей через центр сегмента, помеченного на рисунке крестиком (в дальнейшем мы будем называть этот сегмент «Центральным», хотя он и не соответствует геометрическому центру фигуры). Фигура «Квадрат» не поворачивается вообще, а фигура «Прямоугольник» может принимать только два положения: вертикальное и горизонтальное, т.е. для нее поворот возможет только на 0° и 90°.

Положение фигуры в стакане будет задаваться декартовыми координатами ее так называемого центрального сегмента (на рисуне помечен крестиком). «Так называемого» — потому что это сегмент не всегда совпадает с геометрическим центром фигуры. Центр системы координат мы расположим в левом верхнем углу стакана, а вертикальную и горизонтальные оси направим вниз и вправо соответственно. Угол поворота фигуры будем задавать целочисленным коэффициентом в диапазоне от 0 до 3 включительно. Для того, чтобы получить реальный угол поворота надо данный коэффициент умножить на 90°. Вид фигуры будет также задаваться целым числом в диапазоне от 1 до 7 включительно.

Осталось придумать как задавать конфигурацию фигуры в нашем двумерном пространстве. Координаты фигуры соответствуют координатам центрального сегмента и не меняются при вращении фигуры. Т.е. вращение происходит вокруг этого центрального сегмента. Очевидно, что для описания фигуры достаточно задать координаты трех оставшихся сегментов относительно центрального сегмента. двумерном пространстве нам надо два числа, для того, чтобы задать координаты объекта. Итого, получим три пары чисел для каждого угла поворота. Всего 24 числа, учитывая, что возможных углов поворота у нас четыре. Эти числа мы разместим в трехмерном массиве, где первый индекс — это угол поворота, второй — номер сегмента, и третий — указывает какая координата хранится в данной ячейке — 0 соответствует координате по оси X, 1 — по оси Y. Данный массив будет заполняться при создании фигуры, в зависимости от ее типа.

Предусмотрим методы для выполнения следующих действий над объектом Фигура:

- Поворот на 90° по часовой стрелке. Данный метод должен инкрементировать на единицу коэффициент, определяющий угол поворота. Если, при этом, значение коэффициента превысило 3, то устанавливать его в 0;
- Перемещение относительно текущего положения на заданное расстояние по вертикали и горизонтали;
- Сохранить текущее положение фигуры, т.е. сохранить ее координаты и угол поворота;
- Вернуться к ранее сохраненному положению фигуры;
- Отобразить фигуру на экране (задается канва, координаты верхнего левого угла стакана, в пикселах, картинка, которая содержит прорисованные изображения сегментов);

На этом проектирование класса Фигура можно считать законченным и можно приступать к его кодированию. Но перед этим мы сделаем небольшое отступление и познакомимся с тем как в системе Гедымин создаются и используются новые классы.

Синтаксис VB класса

VBScript позволяет создавать новые классы, которые в дальнейшем мы будем называть VB классами. Вообще говоря, полноценными классами в понимании объектно-ориентированного программирования они не являются, поскольку не поддерживают наследование и, соответственно, полиморфизм. Так что из трех китов, на которых базируется объектно-ориентированная парадигма остается только инкапсуляция — возможность объединять в рамках одной сущности данные и методы.

Определение класса осуществляется с помощью следующей конструкции:

```
Class name
   statements
End Class
```

где name — это имя класса, a statements — это одно или несколько определений переменных, свойств, процедур или функций, называемых так же членами класса. Обратите внимание, что в отличие от Delphi,

где код определения класса содержит только объявления процедур и функций, в VB классе тексты членов прописываются прямо в тексте класса.

Члены класса могут быть объявлены как Private или Public. Первые видны только внутри кода данного класса, вторые же доступны как для внутреннего кода, так и снаружи. Если переменная или функция (процедура) не содержат явного определения Public или Private, то они считаются общедоступными. Процедуры или функции объявленные как Public внутри блока класса становятся методами данного класса. Переменные, объявленные как общедоступные, становятся свойствами класса наравне со свойствами объявленными непосредственно с помощью конструкций Property Get, Property Let, Property Set.

Создание и уничтожение экземпляра VB класса

Создание экземпляра VB класса осуществляется с помощью оператора New.

```
Dim X
Set X = New classname
```

Уничтожение ранее созданного экземпляра происходит автоматически по завершении блока кода, где была объявлена соответсвующая переменная и при условии, что на нее нет внешних ссылок. Если необходимо уничтожить экземпляр вручную, то необходимо присвоить значение Nothing переменной.

```
' объявление переменной и создание экземпляра класса
Dim X
Set X = New classname

...
' использование экземпляра класса
...

' уничтожение экземпляра класса
Set X = Notning

...
```

События Initialize и Terminate

Событие Initialize происходит при создании экземпляра класса, а Terminate — при его уничтожении. Разработчик может определить свои обработчики данных событий. Ниже приведен пример использования данных событий:

```
Class TestClass

   ' Определение обработчика события Initialize.
   Private Sub Class_Initialize
     MsgBox("TestClass started")
   End Sub

   ' Определение обработчика события Terminate.
   Private Sub Class_Terminate
      MsgBox("TestClass terminated")
   End Sub

End Class

' Создание экземпляра класса TestClass.
Set X = New TestClass

' Уничтожение экземпляра.
Set X = Nothing
```

Класс TCreator

Грамотно написанный программный код должен очень бережно относиться к системным ресурсам. Объекты необходимо создавать непосредственно перед их использованием и удалять как только они перестанут быть нужными. Причем, высвобождение ресурсов должно гарантироваться не только при нормальном выполнении кода, но и при возникновении исключительных ситуаций, ошибок. В Delphi для гарантированного высвобождения ресурсов существует конструкция try…finally…end. Казалось бы, что в VBScript нет необходимости особо беспокоится о высвобождении ресурсов, ведь объекты автоматически удаляются при завершении того блока кода, в котором они были созданы и использовались. Однако при работе с объектами Гедымина существует определенный нюанс. Дело в том, что при создании объекта Гедымина с помощью вызова метода Designer.CreateObject одновременно создаются два объекта. Один — это, собственно, объект указанного класса и второй — это COM объект-оболочка поддерживающий интерфейс IDispatch. Объект-оболочка необходим для того, чтобы с объектом Гедымина можно было работать в VBScript. При завершении блока кода объект-оболочка будет автоматически удален, а вот объект Гедымина останется в памяти. Для его удаления необходимо принудительно вызвать метод DestroyObject. Можно, конечно, на каждый созданный объект, где-нибудь в конце процедуры или функции разместить вызов деструктора, однако гарантировать их удаление не удастся — при возникновении ошибки объекты останутся не удаленными. Тут на помощь приходит TCreator — VB класс широко используемый в настройках на платформе Гедымин. Его основное предназначение — помнить список созданных объектов. При удалении экземпляра TCreator вызывается обработчик события Terminate, который автоматически удаляет все созданные с помощью этого объекта объекты.

Использование TCreator выглядит следующим образом:

```
Sub MySub

  ' Создание экземпляра класса TCreator
  Dim Creator
  Set Creator = New TCreator

  ...

  ' Создание объекта Гедымина с помощью Creator
  Dim Obj
  Set Obj = Creator.GetObject(<Owner>, <ClassName>, <ComponentName>)

  ' Использование объекта Гедымина
  ...

  ' Объект Гедымина удалится автоматически вместе
  ' с Creator, который будет удален по завершении процедуры

End Sub
```

VB класс TCreator входит в настройку «Общие данные». Ниже приводится его текст:

```
'Класс служит для создания объектов методом GetObject.
'Класс гарантирует освобождение объекта по завершению скрипта.
'При использьзовании GetObject освобождение объектов непосредственно в
скрипт-функции не требуется,
'оно происходит автоматически при завершению скрипт-функции.

Class TCreator
  Private FCount
  Private FObjectArray()

  Public Sub DestroyAllObjects
    for I = UBound(FObjectArray) to LBound(FObjectArray) step -1
      if VarType(FObjectArray(I)) = vbObject then
        FObjectArray(I).DestroyObject
        FObjectArray(I) = Empty
      end if
```

```
    next
    FCount = -1
  End Sub

  Public Function GetObject(Params, ClassName, Name)
    FCount = FCount + 1
    if FCount > UBound(FObjectArray) then
      ReDim Preserve FObjectArray((UBound(FObjectArray) + 1) * 2 - 1)
    end if
    set FObjectArray(FCount) =_
      Designer.CreateObject(Params, ClassName, Name)
    set GetObject = FObjectArray(FCount)
  End Function

  'Используется, в случае необходимости,
  'для уничтожения объктов созданных Креатором
  Public Sub DestroyObject(Object)
    for I = UBound(FObjectArray) to LBound(FObjectArray) step -1
      if VarType(FObjectArray(I)) = vbObject then
        if Addr(FObjectArray(I)) = Addr(Object) then
          FObjectArray(I) = Empty
          Object.DestroyObject
          exit sub
        end if
      end if
    next

    call Exception.Raise("Exception",_
      "В списке не найден переданный объект.")
  End Sub

  Private Sub Class_Initialize
    ReDim FObjectArray(7)
    FCount = -1
  End Sub

  Private Sub Class_Terminate
    DestroyAllObjects
  End Sub
End Class
```

Создание VB классов в Гедымине

Для того, чтобы создать VB класс в Гедымине необходимо открыть окно редактора скрипт-объектов, открыть Проводник, установить курсор на раздел VB классы и нажать правую кнопку мыши.

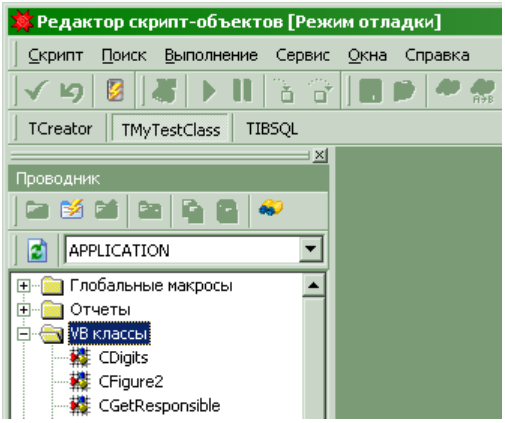


Рис. 29 Раздел VB классы.

В появившмся меню необходимо выбрать команду «Добавить VB класс» в результате чего будет создан новый VB класс, окно с кодом которого откроется справа, на рабочей области редактора скрипт-объектов.

Данное окно содержит три вкладки:

- Свойства;
- Скрипт;
- Зависимость;

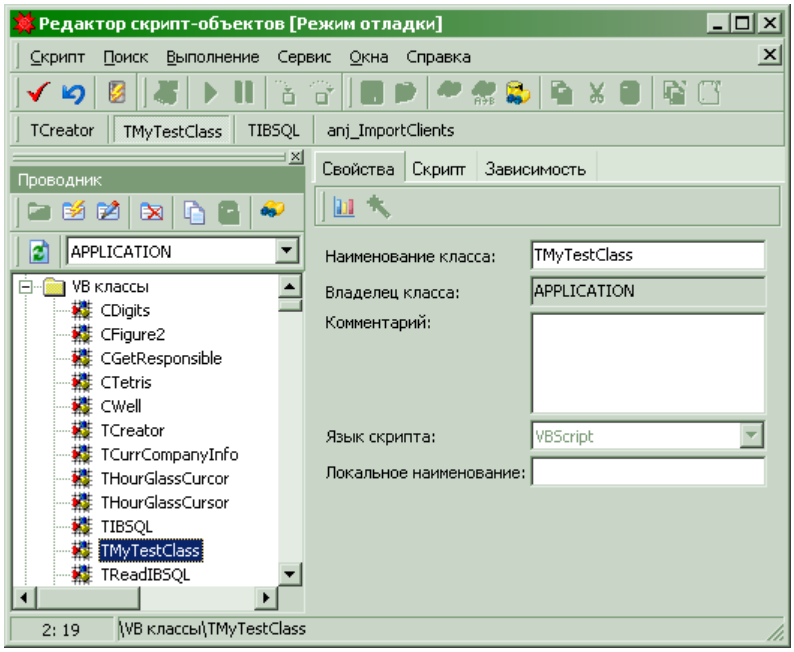


Рис. 30 VB класс. Вкладка Свойства.

На вкладке «Свойства» указывается имя класса, имя модуля, которому он принадлежит, произвольный комментарий и локализованное наименование.

При создании нового класса система сама присвоит ему имя. Что-нибудь вроде: «TVBClass154664603_266640537». Присмотревшись, можно заметить что в сгенерированном имени присутствует префикс «TVBClass» и РУИД класса в базе данных. Если имя, предложенное системой, покажется вам неудобочитаемым (скорее всего, так и будут), то класс можно переименовать. Сделать это необходимо в двух местах: в поле «Наименование класса» на вкладке «Свойства» и, непосредственно, в исходном коде определения класса на вкладке «Скрипт».

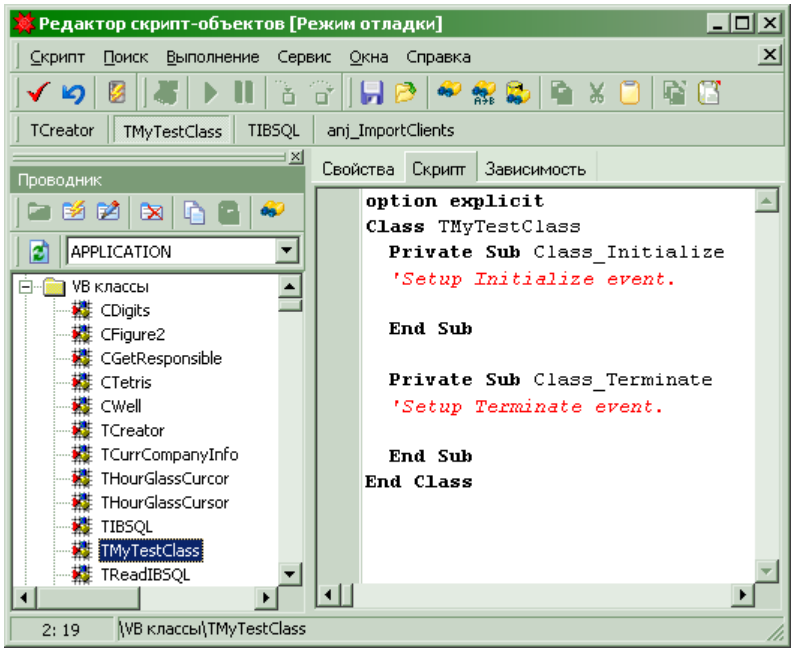


Рис. 31 VB класс. Вкладка Скрипт.

На вкладке «Скрипт» располагается, собственно, исходный код класса. При создании нового класса система автоматически сгенерирует пустые обработчики событий создания и уничтожения экземпляра класса.

Не забудьте, что при изменении имени класса, точно такое же имя следует указать и на вкладке «Свойства».

Третья вкладка, «Зависимость», показывает какие скрипт-функции зависят от данного класса. т.е. включают его по директиве #include, и, в свою очередь, от каких скрипт-функций зависит данный класс.

Окно тетриса

Покончив с классами, создадим экранную форму (окно) для тетриса. Условно, формы, создаваемые в Гедымине можно разделить на две части: формы, связанные с бизнес объектом и произвольные экранные формы. Первые, создаются автоматически при создании нового бизнес объекта. В дальнейшем разработчик может изменять их. Как работать с экранными формами, связанными с бизнес объектами, будет подробно рассказано в следующих главах настоящего руководства. Вторые, произвольные формы, создаются разработчиком в Редакторе форм, который вызывается из меню Сервис главного окна.

Редактор форм

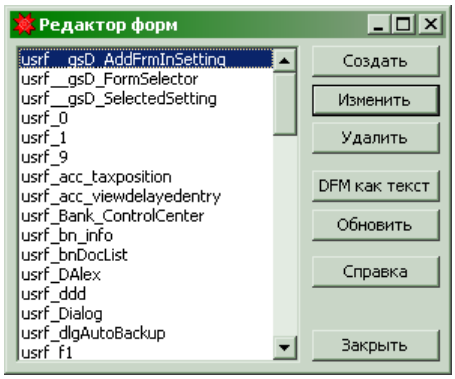


Рис. 32 Редактор форм.

Окно «Редактора форм» содержит список экранных форм, а так же кнопки для вызова команд:

- Создать — вызывает мастер создания новой формы;
- Изменить — открывает выбранную в списке форму для изменения;

- Удалить — удаляет выбранную в списке экранную форму;
- DFM как текст — открывает DFM формы,
- Обновить — перечитывает список форм[12],
- Справка — вызавает подсказку по работе с данным окном,
- Закрыть — закрывает окно.

Создание новой формы

С помощью кнопки «Создать» вызовем мастер создания экранной формы.

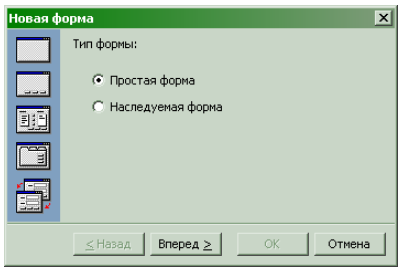


Рис. 33 «Мастер создания экранной формы»

Как видно на рисунке, сперва мы должны решить: будем ли мы создавать нашу форму «с нуля» или наследуем ее от одного из предопределенных в Гедымине классов.

В нашем случае нам нужна абсолютно чистая форма, поэтому выбираем «Простая форма» и нажимаем кнопку «Вперед».

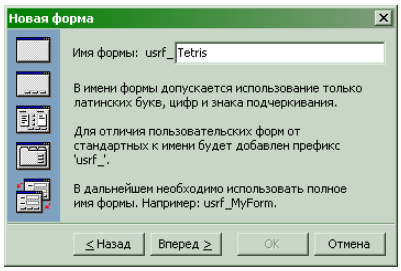


Рис. 34 Ввод имени экранной формы.

Введем имя класса формы. Обратите внимание, что все формы имеют префикс “usrf_”, изменить который нельзя.

Нажмем кнопку «Вперед» и затем «Ок» для подтверждения. Новая форма будет создана и открыта в дизайнере.

Дизайнер форм

Дизайнер экранных форм системы Гедымин во многом копирует дизайнер среды программирования Delphi.

Когда форма находится в режиме редактирования на экране присутствуют следующие окна:

- Палитра компонентов;
- Инспектор объектов;

Палитра компонентов

Если палитра компонентов закрыта, ее можно вызвать на экран воспользовавшись клавишей F10.

---

[12] Поскольку окно «Редактора форм» не модальное список форм может измениться, например, в результате загрузки настройки или выполнения макроса, в этом случае нужно перечитать его из базы данных.

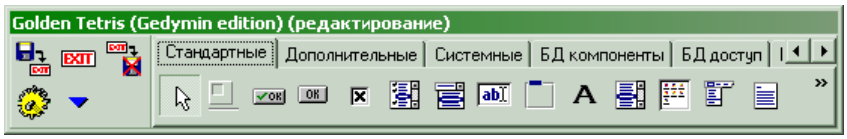


Рис. 35 Палитра компонентов.

Как видно на рисунке, окно палитры компонентов разбито на две части. В левой части, в верхнем ряду находятся:

- Команда выхода из режима редактирования и сохранения внесенных изменений;
- Команда выхода из режима редактирования без сохранения изменений;
- Команда выхода из режима редактирования и сброс всех сделанных изменений. Возвращение формы в ее первоначальное состояние. Будьте осторожны! Используя данную команду вы рискуете удалить все изменения, в том числе и те, которые вам необходимы. Не забывайте периодически создавать архивную копию базы данных.

Ниже находится кнопка вызова списка имен классов всех доступных компонентов. Используя данный список можно выбрать нужный компонент и поместить его на форму.

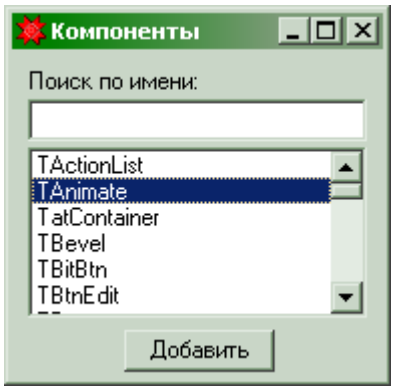


Рис. 36 Список компонентов.

Рядом с кнопкой вызова списка компонентов находится кнопка, открывающая дополнительный набор команд для работы с группой компонентов:

- Выравнивание положения компонентов внутри группы;
- Выравнивание размеров компонентов внутри группы;
- Установка порядка табуляции компонентов на форме.

В правой части палитры компонентов находятся вкладки с пиктограммками компонентов.

Стандартные

На данной вкладке находятся основные компоненты для постороения пользовательского интерфейса: кнопки, списки, закладки, меню, панели, радиокнопки, чекбоксы и т.п.

Дополнительные

На данной вкладке находятся дополнительные компоненты для построения пользовательского интерфейса: управляющие элементы календарь и калькулятор, поле для ввода даты, процентная строка и т.п.

Системные

На данной вкладке находятся такие компоненты как: список действий (ActionList), список картинок, таймер и другие.

БД Компоненты

На вкладке «БД Компоненты» находятся визуальные компоненты, предназначенные для отображения информации из базы данных.

БД Доступ

На данной вкладке находятся компоненты TClientDataSet и TDataSource.

Interbase

На вкладке находятся компоненты дял доступа к базе данных на сервере Interbase/Firebird.

Диалоги

Вкладка содержит компоненты для работа с часто используемыми диалоговыми окнами: сохранения и открытия файла, выбора и настройки принтера, поиска и т.д.

SynEdit

Вкладка содержит четыре компонента для работы с текстовым редактором поддерживающим подсветку синтаксиса.

GDC

На вкладке “GDC” находятся все доступные в системе бизнес-объекты.

По правой кнопке мыши можно вызвать контекстное меню, которое содержит единственную команду: расположить Палитру компонетов поверх всех окон.

Инспектор объектов

Инспектор объектов позволяет просматривать и изменять значения свойств выбранного на редактируемой форме компонента, а также свойств самой формы.

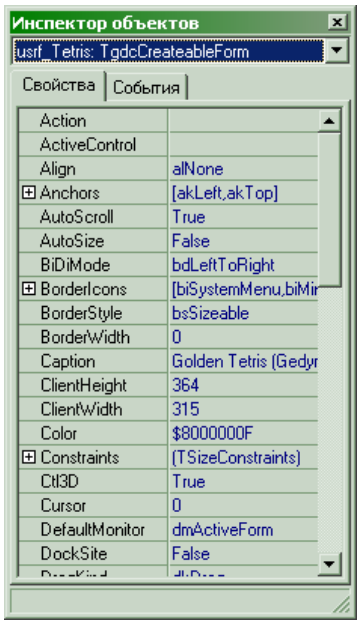


Рис. 37 Инспектор объектов.

Вверху Инспектора объектов находится выпадающий список, содержащий все компоненты находящиеся на форме. Ниже находятся две вкладки: на одной перечислены все свойства выбранного компонента, а на другой — все присущие этому компоненту события.

Если Инспектор объектов скрыт, вывести его на экран можно с помощью нажатия клавиши F11.

По правой кнопке мыши можно вызвать контекстное меню, которое содержит единственную команду: расположить Инспектор объектов поверх всех окон.

Создание новой наследованной формы

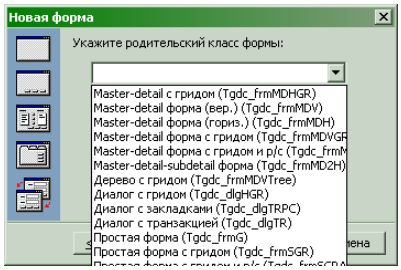


Рис. 38 Выбор базового класса окна.

Конструируем окно тетриса

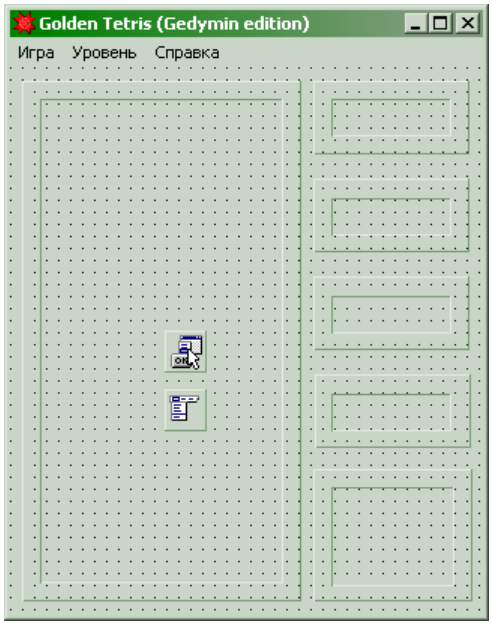


Рис. 39 Окно тетриса.

Исходный код

Класс CFigure2

```
Class CFigure2
  ' горизонтальная координата фигуры, т.е. координата
  ' ее центрального сегмента
  Private FX

  ' вертикальная координата фигуры, т.е. координата
  ' ее центрального сегмента
  Private FY

  ' конфигурация фигуры задается указанием координат
  ' трех ее сегментов относительно центрального сегмента
  ' для каждого из возможных углов поворота
  ' в массиве FPieces первый индекс -- это угол,
  ' второй -- номер сегмента и третий указывает по какой
  ' оси задается координата, 0 -- по X, 1 -- по Y
  Private FPieces(3, 2, 1)

  ' текущий угол поворота фигуры
  Private FAngle
```

```
' тип фигуры. Целое число от 1 до 7
Private FType

' три поля используются для сохранения и последующего
' восстановления положения фигуры
Private FOldX
Private FOldY
Private FOldAngle

' конструктор
Private Sub Class_Initialize
  CreateFigure NULL
End Sub

' данный метод инициализирует внутренний поля, т.е.
' фактически создает фигуру. Входящий параметр -- это
' другая фигура. Если он указан, то будет создана
' ее копия, если передан NULL, то тип фигуры будет
' определен случайным образом
Public Sub CreateFigure(ByRef APeekFigure)

  FX = 4
  FY = 0
  FAngle = 0

  if IsNull(APeekFigure) then
    FType = Int(7 * Rnd + 1)
  else
    FType = APeekFigure.FigureType
  end if

  select case FType
    ' "палка"
    case 1
      FPieces(0, 0, 0) = -1
      FPieces(0, 0, 1) = 0
      FPieces(0, 1, 0) = 1
      FPieces(0, 1, 1) = 0
      FPieces(0, 2, 0) = 2
      FPieces(0, 2, 1) = 0

      FPieces(1, 0, 0) = 0
      FPieces(1, 0, 1) = -1
      FPieces(1, 1, 0) = 0
      FPieces(1, 1, 1) = 1
      FPieces(1, 2, 0) = 0
      FPieces(1, 2, 1) = 2

      FPieces(2, 0, 0) = -1
      FPieces(2, 0, 1) = 0
      FPieces(2, 1, 0) = 1
      FPieces(2, 1, 1) = 0
      FPieces(2, 2, 0) = 2
      FPieces(2, 2, 1) = 0

      FPieces(3, 0, 0) = 0
      FPieces(3, 0, 1) = -1
      FPieces(3, 1, 0) = 0
      FPieces(3, 1, 1) = 1
      FPieces(3, 2, 0) = 0
```

```
        FPieces(3, 2, 1) = 2

      ' літара T
      case 2
        FPieces(0, 0, 0) = -1
        FPieces(0, 0, 1) = 0
        FPieces(0, 1, 0) = 1
        FPieces(0, 1, 1) = 0
        FPieces(0, 2, 0) = 0
        FPieces(0, 2, 1) = 1

        FPieces(1, 0, 0) = 0
        FPieces(1, 0, 1) = -1
        FPieces(1, 1, 0) = 0
        FPieces(1, 1, 1) = 1
        FPieces(1, 2, 0) = 1
        FPieces(1, 2, 1) = 0

        FPieces(2, 0, 0) = -1
        FPieces(2, 0, 1) = 0
        FPieces(2, 1, 0) = 1
        FPieces(2, 1, 1) = 0
        FPieces(2, 2, 0) = 0
        FPieces(2, 2, 1) = -1

        FPieces(3, 0, 0) = 0
        FPieces(3, 0, 1) = -1
        FPieces(3, 1, 0) = 0
        FPieces(3, 1, 1) = 1
        FPieces(3, 2, 0) = -1
        FPieces(3, 2, 1) = 0

      ' літара G
      case 3
        FPieces(0, 0, 0) = -1
        FPieces(0, 0, 1) = 0
        FPieces(0, 1, 0) = 1
        FPieces(0, 1, 1) = 0
        FPieces(0, 2, 0) = 1
        FPieces(0, 2, 1) = 1

        FPieces(1, 0, 0) = 0
        FPieces(1, 0, 1) = -1
        FPieces(1, 1, 0) = 1
        FPieces(1, 1, 1) = -1
        FPieces(1, 2, 0) = 0
        FPieces(1, 2, 1) = 1

        FPieces(2, 0, 0) = -1
        FPieces(2, 0, 1) = 0
        FPieces(2, 1, 0) = -1
        FPieces(2, 1, 1) = -1
        FPieces(2, 2, 0) = 1
        FPieces(2, 2, 1) = 0

        FPieces(3, 0, 0) = 0
        FPieces(3, 0, 1) = -1
        FPieces(3, 1, 0) = 0
        FPieces(3, 1, 1) = 1
        FPieces(3, 2, 0) = -1
        FPieces(3, 2, 1) = 1
```

```
    ' літара G2
    case 4
      FPieces(0, 0, 0) = -1
      FPieces(0, 0, 1) = 0
      FPieces(0, 1, 0) = 1
      FPieces(0, 1, 1) = 0
      FPieces(0, 2, 0) = -1
      FPieces(0, 2, 1) = 1

      FPieces(1, 0, 0) = 0
      FPieces(1, 0, 1) = -1
      FPieces(1, 1, 0) = 1
      FPieces(1, 1, 1) = 1
      FPieces(1, 2, 0) = 0
      FPieces(1, 2, 1) = 1

      FPieces(2, 0, 0) = -1
      FPieces(2, 0, 1) = 0
      FPieces(2, 1, 0) = 1
      FPieces(2, 1, 1) = -1
      FPieces(2, 2, 0) = 1
      FPieces(2, 2, 1) = 0

      FPieces(3, 0, 0) = 0
      FPieces(3, 0, 1) = -1
      FPieces(3, 1, 0) = 0
      FPieces(3, 1, 1) = 1
      FPieces(3, 2, 0) = -1
      FPieces(3, 2, 1) = -1

    ' літара Z
    case 5
      FPieces(0, 0, 0) = 1
      FPieces(0, 0, 1) = 0
      FPieces(0, 1, 0) = -1
      FPieces(0, 1, 1) = 1
      FPieces(0, 2, 0) = 0
      FPieces(0, 2, 1) = 1

      FPieces(1, 0, 0) = 0
      FPieces(1, 0, 1) = -1
      FPieces(1, 1, 0) = 1
      FPieces(1, 1, 1) = 0
      FPieces(1, 2, 0) = 1
      FPieces(1, 2, 1) = 1

      FPieces(2, 0, 0) = -1
      FPieces(2, 0, 1) = 0
      FPieces(2, 1, 0) = 1
      FPieces(2, 1, 1) = -1
      FPieces(2, 2, 0) = 0
      FPieces(2, 2, 1) = -1

      FPieces(3, 0, 0) = 0
      FPieces(3, 0, 1) = 1
      FPieces(3, 1, 0) = -1
      FPieces(3, 1, 1) = 0
      FPieces(3, 2, 0) = -1
      FPieces(3, 2, 1) = -1
```

```
      ' літара Z2
      case 6
        FPieces(0, 0, 0) = -1
        FPieces(0, 0, 1) = 0
        FPieces(0, 1, 0) = 0
        FPieces(0, 1, 1) = 1
        FPieces(0, 2, 0) = 1
        FPieces(0, 2, 1) = 1

        FPieces(1, 0, 0) = 0
        FPieces(1, 0, 1) = 1
        FPieces(1, 1, 0) = 1
        FPieces(1, 1, 1) = 0
        FPieces(1, 2, 0) = 1
        FPieces(1, 2, 1) = -1

        FPieces(2, 0, 0) = -1
        FPieces(2, 0, 1) = -1
        FPieces(2, 1, 0) = 0
        FPieces(2, 1, 1) = -1
        FPieces(2, 2, 0) = 1
        FPieces(2, 2, 1) = 0

        FPieces(3, 0, 0) = 0
        FPieces(3, 0, 1) = -1
        FPieces(3, 1, 0) = -1
        FPieces(3, 1, 1) = 0
        FPieces(3, 2, 0) = -1
        FPieces(3, 2, 1) = 1

      ' square
      case 7
        FPieces(0, 0, 0) = 1
        FPieces(0, 0, 1) = 0
        FPieces(0, 1, 0) = 1
        FPieces(0, 1, 1) = 1
        FPieces(0, 2, 0) = 0
        FPieces(0, 2, 1) = 1

        FPieces(1, 0, 0) = 1
        FPieces(1, 0, 1) = 0
        FPieces(1, 1, 0) = 1
        FPieces(1, 1, 1) = 1
        FPieces(1, 2, 0) = 0
        FPieces(1, 2, 1) = 1

        FPieces(2, 0, 0) = 1
        FPieces(2, 0, 1) = 0
        FPieces(2, 1, 0) = 1
        FPieces(2, 1, 1) = 1
        FPieces(2, 2, 0) = 0
        FPieces(2, 2, 1) = 1

        FPieces(3, 0, 0) = 1
        FPieces(3, 0, 1) = 0
        FPieces(3, 1, 0) = 1
        FPieces(3, 1, 1) = 1
        FPieces(3, 2, 0) = 0
        FPieces(3, 2, 1) = 1

    end select
```

```
End Sub

' поворот фигуры на 90 градусов по часовой стрелке
Public Sub Rotate
  FAngle = FAngle + 1
  if FAngle = 4 then
    FAngle = 0
  end if
End Sub

' перемещение фигуры на DX, DY относительно текущего положения
Public Sub Move(DX, DY)
  FX = FX + DX
  FY = FY + DY
End Sub

' сохранение координат и угла поворота фигуры
Public Sub Save
  FOldX = FX
  FOldY = FY
  FOldAngle = FAngle
End Sub

' восстановление ранее сохраненных координат и угла поворота
Public Sub Restore
  FX = FOldX
  FY = FOldY
  FAngle = FOldAngle
End Sub

' отрисовывает фигуру на переданной канве
Public Sub DrawFigure(ByRef ACanvas, ByVal AX, ByVal AY,_
    ByRef ABmp)
  if FType = 0 then
    ' фигура еще не была проинициализирована
    exit sub
  end if

  Dim XX, YY, I

  XX = AX + FX * 16
  YY = AY + FY * 16
  ACanvas.CopyRect XX, YY, XX + 16, YY + 16, ABmp.Canvas,_
    FigureType * 16, 0, FigureType * 16 + 16, 16

  for I = 0 to 2
    XX = AX + PieceCoord(I, 0) * 16
    YY = AY + PieceCoord(I, 1) * 16
    ACanvas.CopyRect XX, YY, XX + 16, YY + 16, ABmp.Canvas,_
      FigureType * 16, 0, FigureType * 16 + 16, 16
  next
End Sub

Property Get X
  X = FX
End Property

Property Get Y
  Y = FY
End Property
```

```
  Property Get PieceCoord(APiece, ACoord)
    PieceCoord = FPieces(FAngle, APiece, ACoord)
    if ACoord = 0 then
      PieceCoord = PieceCoord + FX
    else
      PieceCoord = PieceCoord + FY
    end if
  End Property

  Property Get FigureType
    FigureType = FType
  End Property

End Class
```

Класс CWell

```
Class CWell
  ' калодеж. памеры 10x20. кожная клетка -- цэлы лік.
  ' 0  -- клетка пустая
  ' >0 -- клетка занятая. лік паказвае колер клеткі.

  Private FTetris
  Private WellArr

  Private Sub Class_Initialize
  ' Setup Initialize event.
    Dim Arr(9, 19)
    WellArr = Arr
    ResetWell
  End Sub

  Private Sub Class_Terminate
    Set FTetris = Nothing
  End Sub

  ' resets well to empty state
  Public Sub ResetWell
    for I = 0 to 9
      for J = 0 to 19
        WellArr(I, J) = 0
      next
    next
  End Sub

  Public Function GetXY(X, Y)
    if (X >= 0) and (X < 10) and (Y >= 0) and (Y < 20) then
      GetXY = WellArr(X, Y)
    else
      GetXY = -1
    end if
  End Function

  Public Sub PutFigure(AFigure)
    WellArr(AFigure.X, AFigure.Y) = AFigure.FigureType
    for I = 0 to 2
      WellArr(AFigure.PieceCoord(I, 0), AFigure.PieceCoord(I, 1)) =
AFigure.FigureType
    next
  End Sub
```

```
  Public Sub DisposeRows(R)
    for K = LBound(R) to UBound(R) - 1
      for I = R(K) - 1 to 0 step -1
        for J = 0 to 9
          WellArr(J, I + 1) = WellArr(J, I)
          WellArr(J, 0) = 0
        next
      next
    next
  End Sub

  Public Sub DrawWell(ByRef ACanvas, AX, AY, ByRef ABmp)
    Dim I, J, XX, YY

    for I = 0 to 9
      for J = 0 to 19
        XX = AX + I * 16
        YY = AY + J * 16
        ACanvas.CopyRect XX, YY, XX + 16, YY + 16, ABmp.Canvas, GetXY(I, J)
* 16, 0, GetXY(I, J) * 16 + 16, 16
      next
    next

    if FTetris.Paused then
      for I = 0 to 10 * 8
        for J = 0 to 20 * 8
          XX = AX + I * 2
          YY = AY + J * 2
          ACanvas.MoveTo XX, YY
          ACanvas.LineTo XX + 1, YY
        next
      next
    end if
  End Sub

  Public Property Set Tetris(Value)
    Set FTetris = Value
  End Property

  Public Property Get Tetris
    Set Tetris = FTetris
  End Property

End Class
```

Класс CTetris

```
Public Const sUninitialized = 0
Public Const sPaused = 1
Public Const sInProgress = 2
Public Const sGameOver = 3

class CTetris
  Public F
'  Private F
  Public Well
  Private Figure
  Private PeekFigure
  Private Bmp
  Private GameOver
```

```
  Private FPaused
  Private FState

  Private DPoints
  Private DLevel
  Private DFigures
  Private DLines

  Private Sub Class_Initialize
  ' Setup Initialize event.
    set F = GedeminApplication.FindComponent("MyTetrisForm")
    if not Assigned(F) then
      set F = Designer.CreateForm(Application, "usrf_Tetris",
"MyTetrisForm")
    end if
    set Well = New CWEll
    set Figure = New CFigure2
    set PeekFigure = New CFigure2

    set DPoints = New CDigits
    DPoints.SetW F
    Set Bvl = F.FindComponent("usr_Bevel_Andreik1")
    DPoints.X = Bvl.Left + 1
    DPoints.Y = Bvl.Top + 1

    set DLevel = New CDigits
    DLevel.SetW F
    Set Bvl = F.FindComponent("usr_Bevel_Andreik4")
    DLevel.X = Bvl.Left + 1
    DLevel.Y = Bvl.Top + 1

    set DFigures = New CDigits
    DFigures.SetW F
    Set Bvl = F.FindComponent("usr_Bevel_Andreik7")
    DFigures.X = Bvl.Left + 1
    DFigures.Y = Bvl.Top + 1

    set DLines = New CDigits
    DLines.SetW F
    Set Bvl = F.FindComponent("usr_Bevel_Andreik8")
    DLines.X = Bvl.Left + 1
    DLines.Y = Bvl.Top + 1

    set Bmp = Designer.CreateObject(NULL, "TBitmap", "")
    set TmpStream = Designer.CreateObject(NULL, "TMemoryStream", "")
    set TmpData = Designer.CreateObject(NULL, "TgdcAttrUserDefined", "")
    TmpData.SubType = "USR$TETR_DATA"
    TmpData.SubSet = "All"
    TmpData.Open
    TmpData.FieldByName("USR$DATA").SaveToStream TmpStream
    TmpStream.Position = 0
    Bmp.LoadFromStream TmpStream
    TmpData.Close
    Designer.DestroyObject(TmpData)
    Designer.DestroyObject(TmpStream)

    GameOver = True
  End Sub

  Private Sub Class_Terminate
  ' Setup Terminate event.
```

```
    set Well = Nothing
    set Figure = Nothing
    set PeekFigure = Nothing
    set DPoints = Nothing

    Designer.DestroyObject(F)
    Designer.DestroyObject(Bmp)
  End Sub

  Public Sub BindObjects(Value)
    Set Well.Tetris = Value
  End Sub

  Public Sub ShowWindow
    F.Show
    do
      Application.ProcessMessages
    loop while F.Visible
  End Sub

  Public Sub Play
    Randomize

    GameOver = False
    FPaused = False

    do while not GameOver

      ResetKeys
      Well.ResetWell

      DPoints.Value = 0
      DLevel.Value = 0
      DFigures.Value = 1
      DLines.Value = 0

      PeekFigure.CreateFigure Null
      Figure.CreateFigure Null

      if F.Visible then
        F.Invalidate
      else
        F.Show
      end if

      Application.ProcessMessages

      T = 0

      do while F.Visible and (not GameOver)

        do while not F.Active and F.Visible
          Application.ProcessMessages
          WinAPI.Sleep(200)
          ResetKeys
        loop

        if (WinAPI.GetAsyncKeyState(VK_PAUSE) and 1) <> 0 then
          FPaused = not FPaused
          F.Invalidate
        end if
```

```
    do while FPaused
      Application.ProcessMessages
      WinAPI.Sleep(200)

      if (WinAPI.GetAsyncKeyState(VK_PAUSE) and 1) <> 0 then
        FPaused = not FPaused
        F.Invalidate
      end if

      if not F.Visible then
        FPaused = False
      end if

      ResetKeys
    loop

    if (WinAPI.GetAsyncKeyState(VK_F2) and 1) <> 0 then
      Exit Do
    end if

    X = 0
    R = 0

    if T > ((9 - DLevel.Value) * 60 + 40) then
      Y = 1
      T = 0
    else
      Y = 0

      if (WinAPI.GetAsyncKeyState(VK_F8) and 1) <> 0 then
        if DLevel.Value < 9 then
          DLevel.Value = DLevel.Value + 1
        end if
      end if

      if (WinAPI.GetAsyncKeyState(VK_LEFT) and 1) <> 0 then
        X = X - 1
      end if

      if (WinAPI.GetAsyncKeyState(VK_UP) and 1) <> 0 then
        R = 1
      end if

      if (WinAPI.GetAsyncKeyState(VK_RIGHT) and 1) <> 0 then
        X = X + 1
      end if

      if (WinAPI.GetAsyncKeyState(VK_DOWN) and 1) <> 0 then
        Delay = 20
        do
          call Move(0, 1, 0)
          WinAPI.Sleep Delay
          Delay = Delay - 5
          if Delay < 0 then
            Delay = 0
          end if
        loop until Figure.Y = 0
        T = 0
      end if
    end if
```

```
        call Move(X, Y, R)

        Application.ProcessMessages
        WinAPI.Sleep(40)

        T = T + 40
      loop

      GameOver = not F.Visible

    loop

    if F.Visible then
      set FPlayerName = Designer.CreateForm(F, "usrf_TetrisPlayerName")
      if (FPLayerName.ShowModal = 1) then
        set FStat = Designer.CreateObject(NULL, "TgdcAttrUserDefined", "")
        FStat.SubType = "USR$TETR_STAT"
        FStat.SubSet = "ByID"
        FStat.Open
        FStat.Insert
        FStat.FieldByName("USR$DURATION").AsInteger = 1
        FStat.FieldByName("USR$LEVEL").AsInteger = 1
        FStat.FieldByName("USR$FIGURES").AsInteger = 1
        FStat.FieldByName("USR$POINTS").AsInteger = 1
        FStat.FieldByName("USR$LINES").AsInteger = 1
        FStat.FieldByName("USR$PLAYERKEY").AsInteger =
scrPublicVariables.Value("TetrisPlayerKey")
        FStat.FieldByName("USR$DATE").AsDateTime = Now
        FStat.Post
        FStat.Close
        Designer.DestroyObject(FStat)
      end if

      Designer.DestroyObject(FPlayerName)
    end if

  End Sub

  Private Sub ResetKeys
    WinAPI.GetAsyncKeyState(VK_DOWN)
    WinAPI.GetAsyncKeyState(VK_UP)
    WinAPI.GetAsyncKeyState(VK_LEFT)
    WinAPI.GetAsyncKeyState(VK_RIGHT)
    WinAPI.GetAsyncKeyState(VK_F2)
    WinAPI.GetAsyncKeyState(VK_F8)
  End Sub

  Public Sub Draw(Wnd)
    DPoints.Draw
    DLevel.Draw
    DLines.Draw
    DFigures.Draw
    DrawWell
  End Sub

  Private Sub DrawWell
    Dim I, J, XX, YY
    Set Bvl = F.FindComponent("usr_Bevel_Andreik2")

    Call Well.DrawWell(F.Canvas, Bvl.Left + 1, Bvl.Top + 1, Bmp)
```

```
    Call Figure.DrawFigure(F.Canvas, Bvl.Left + 1, Bvl.Top + 1, Bmp)

    Set Bvl = F.FindComponent("usr_Bevel_Andreik3")
    for I = 0 to 4
      XX = Bvl.Left + 1 + I * SqSize
      for J = 0 to 3
        YY = Bvl.Top + 1 + J * SqSize
        F.Canvas.CopyRect XX, YY, XX + SqSize, YY + SqSize, Bmp.Canvas, 0,
0, SqSize, SqSize
      next
    next

    Call PeekFigure.DrawFigure(F.Canvas, Bvl.Left + 1 - SqSize * 2, Bvl.Top
+ 1 + SqSize, Bmp)
  End Sub

  Private Sub Move(AX, AY, AR)
    Dim OldA(3, 1), NewA(3, 1)
    Set Bvl = F.FindComponent("usr_Bevel_Andreik2")

    Figure.Save

    OldA(0, 0) = Figure.X
    OldA(0, 1) = Figure.Y
    for I = 0 to 2
      OldA(I + 1, 0) = Figure.PieceCoord(I, 0)
      OldA(I + 1, 1) = Figure.PieceCoord(I, 1)
    next

    Call Figure.Move(AX, AY)

    if AR <> 0 then
      Figure.Rotate
    end if

    NewA(0, 0) = Figure.X
    NewA(0, 1) = Figure.Y
    for I = 0 to 2
      NewA(I + 1, 0) = Figure.PieceCoord(I, 0)
      NewA(I + 1, 1) = Figure.PieceCoord(I, 1)
    next

    Flag = Well.GetXY(Figure.X, Figure.Y) = 0
    for I = 0 to 2
      Flag = Flag and (Well.GetXY(Figure.PieceCoord(I, 0),
Figure.PieceCoord(I, 1)) = 0)
    next

    if Flag then
      for I = 0 to 3
        for J = 0 to 3
          if (OldA(J, 0) = NewA(I, 0)) and (OldA(J, 1) = NewA(I, 1)) then
            OldA(J, 0) = -1
            OldA(J, 1) = -1
            NewA(I, 0) = -1
            NewA(I, 1) = -1
            Exit For
          end if
        next
      next
```

```
      for I = 0 to 3
        if NewA(I, 0) <> -1 then
          XX = Bvl.Left + 1 + NewA(I, 0) * 16
          YY = Bvl.Top + 1 + NewA(I, 1) * 16
          F.Canvas.CopyRect XX, YY, XX + 16, YY + 16, Bmp.Canvas,
Figure.FigureType * 16, 0, Figure.FigureType * 16 + 16, 16
        end if

        if OldA(I, 0) <> -1 then
          XX = Bvl.Left + 1 + OldA(I, 0) * 16
          YY = Bvl.Top + 1 + OldA(I, 1) * 16
          F.Canvas.CopyRect XX, YY, XX + 16, YY + 16, Bmp.Canvas, 0 * 16,
0, 0 * 16 + 16, 16
        end if
      next
    else
      Figure.Restore
      if (AY = 1) and (AX = 0) then
        Well.PutFigure(Figure)
        BurnRows
        Figure.CreateFigure PeekFigure

        DFigures.Value = DFigures.Value + 1

        if (Well.GetXY(Figure.X, Figure.Y) <> 0)_
          or (Well.GetXY(Figure.PieceCoord(0, 0), Figure.PieceCoord(0, 1))
<> 0)_
          or (Well.GetXY(Figure.PieceCoord(1, 0), Figure.PieceCoord(1, 1))
<> 0)_
          or (Well.GetXY(Figure.PieceCoord(2, 0), Figure.PieceCoord(2, 1))
<> 0) then

          GameOver = True
          BurnWell
        else
          DPoints.Value = DPoints.Value + Figure.FigureType

          if DPoints > (DLevel + 1) * 1000 and DLevel < 9 then
            DLevel.Value = DLevel + 1
          end if

          PeekFigure.CreateFigure NULL
          DrawWell
        end if
      end if
    end if

  End Sub

  Public Sub BurnRows
    ReDim R(0)
    for I = 0 to 19
      Flag = True
      for J = 0 to 9
        if Well.GetXY(J, I) = 0 then
          Flag = False
          Exit For
        end if
      next

      if Flag then
```

```
        ReDim Preserve R(UBound(R) + 1)
        R(UBound(R) - 1) = I
      end if
    next

    if UBound(R) > 0 then
      Set Bvl = F.FindComponent("usr_Bevel_Andreik2")
      for K = 8 to 10
        for I = 0 to UBound(R) - 1
          YY = Bvl.Top + 1 + R(I) * 16
          for J = 0 to 9
            XX = Bvl.Left + 1 + J * 16
            F.Canvas.CopyRect XX, YY, XX + 16, YY + 16, Bmp.Canvas, K * 16,
0, K * 16 + 16, 16
          next

          DPoints.Value = DPoints.Value + 11
        next

        DPoints.Draw
        WinAPI.Sleep(200)
      next

      DLines.Value = DLines.Value + UBound(R)
    end if

    Well.DisposeRows(R)
  End Sub

  Public Sub BurnWell
    Dim Bvl, K, I, J, Q, XX, YY
    Set Bvl = F.FindComponent("usr_Bevel_Andreik2")

    for I = 0 to 19
      for K = 8 to 10
        for J = 0 to I
          YY = Bvl.Top + 1 + J * SqSize
          for Q = 0 to 9
            XX = Bvl.Left + 1 + Q * SqSize
            if Well.GetXY(Q, J) <> 0 then
              F.Canvas.CopyRect XX, YY, XX + SqSize, YY + SqSize,
Bmp.Canvas, K * 16, 0, K * 16 + 16, 16

              DPoints.Value = DPoints.Value + 1
            end if
          next
        next
        Call WinAPI.Sleep(100)
      next
    next

  End Sub

  Public Sub NextLevel
    if DLevel < 9 then
      DLevel.Value = DLevel + 1
    end if
  End Sub

  Public Property Get Paused
    Paused = FPaused
```

```
  End Property

End Class
```

Заключение

Контрольные вопросы

- Для чего предназначен VB класс TCreator?
- Чем отличается переменная VB класса, объявленная как Public от Private?

Бизнес-объект

Эта глава расскажет о том, что такое бизнес-объект, как он работает, как создавать свои или расширять уже существующие бизнес-объекты. Также здесь мы познакомимся с такими понятиями, как РУИД, идентификатор, и научимся переносить данные между базами.

Что такое бизнес-объект

Бизнес-объект – это представление активных структур (единиц, понятий, атомов) бизнеса, которое обязательно включает в себя имя, определение, атрибуты, поведение, взаимосвязи, правила, политику и ограничения. Бизнес-объект может представлять собой, например, персону, место, событие, бизнес-процесс и т.д. Имя используется для идентификации бизнес-объекта в системе, а потому должно быть уникальным. Определение декларирует значения и цели, присвоенные бизнес-объекту. Атрибуты описывают свойства бизнес-объекта, и наконец, поведение обуславливает деятельность бизнес-объекта в тех или иных условиях, в частности при отображении бизнес-объекта, задает последовательность операций, изменение атрибутов и т.д.

Бизнес-объект в системе Гедымин представляет собой развитие TIBCustomDataSet (набора данных, работающего с БД Интербейза). Кроме стандартных функций, поддерживаемых датасетом, он предоставляет удобный вызов диалоговых окон, формы просмотра, реализует защищенный режим работы при создании, изменении и удалении записи, позволяет грамотно обрабатывать одновременный доступ к записям, а также организует перенос данных между базами. Бизнес-объект предоставляет пользователю выборку записей, отвечающих заданным условиям.

Базовым классом, реализующим бизнес-объект, является TgdcBase. Все остальные классы, наследованные от него характеризуются спецификой работы с определенными данными. Например, бизнес-объект для работы с контактами – TgdcBaseContact, для работы с документами – TgdcDocument, всю специфику по работе с товарами взял на себя TgdcGood. Описание специфики бизнес-объектов можно посмотреть в таблице 6.1.

Название класса бизнес-объекта	Описание
TgdcBase	Базовый бизнес-класс, реализующий основу для все остальных бизнес-классов.
TgdcAccount	Бизнес-класс для работы с банковскими счетами
TgdcAttrUserDefined	Бизнес-класс предоставляющий пользователю возможность создавать «с нуля» бизнес-объекты на основе пользовательских таблиц
TgdcBankStatement	Бизнес-класс для работы с выпиской.
TgdcBankCatalogue	Бизнес-класс для работы с картотекой.
TgdcBaseMessage	Бизнес-класс для работы с сообщениями в системе Гедымин. В данном случае подразумеваются сообщения, создаваемые пользователями для передачи друг-другу некой служебной информации, работа с электронной почтой.
TgdcConst, TgdcConstValue	Этих два бизнес-класса работают в паре. Они позволяют создавать некоторые константы, хранящие значения с градацией по времени, активной организации и т.д.
TgdcCurr, TgdcCurrate	Данные бизнес-классы предоставляют пользователю интерфейс для работы с курсами валют. Кроме того они хранят описание валюты, ее название в различных падежах, что используется при выводе отчетов.
TgdcDocument	Базовый класс для работы со всеми документами в системе Гедымин. От него наследовано несколько классов, предоставляющих специфичные возможности для обработки документов в различных областях.
TgdcInvBaseDocument	Предоставляет возможности для создания и обработки складских документов.
TgdcInvBasePriceList	Бизнес-класс для работы с прайс-листами.
TgdcUserBaseDocument	Базовый класс для создания пользовательских документов.
TgdcDocumentType	Позволяет создавать новые типы документов. Это могут быть

	пользовательские, складские документы, различные прайс-листы.
TgdcGood, TgdcGoodGroup	Предоставляют доступ к товарам и товарным группам.
TgdcFunction, TgdcEvent, TgdcMacros, TgdcReport	Предоставляют доступ к скрипт-объектам. Работа с этими классами напрямую подразумевает очень глубокие знания работы Гедымина в целом.
TgdcSetting, TgdcSettingPos, TgdcSettingStorage	Эти классы определяют работу настроек системы Гедымин. Понятия «настройка» мы уже касались в главе 4. Более подробно о настройках можно узнать в главе 14.
TgdcTax	Базовый класс для работы с налогами.
TgdcUser, TgdcUserGroup, TgdcUserStorage	Данные базовые классы дают доступ к зарегистрированным пользователям Гедымина, их группам и хранилищу, в котором содержаться специфические установки каждого пользователя.
TgdcValue	Базовый класс для работы с единицами измерения.

Таблица 0.1

В данной таблице предоставлены не все классы, доступные в системе Гедымин. Более подробное их описание с иерархией и расшифровкой доступных методов и свойств смотрите в приложении а (!!!!!).

Чтобы закончить наше краткое ознакомление с бизнес-объектом необходимо затронуть также понятие подтипа. Как мы уже знаем, каждый объект является реализацией некого класса. Классы могут наследоваться от других классов. При наследовании новый класс получает доступ ко всем открытым свойствам и методам его предка. Понятие подтипа было введено в Гедымин из-за невозможности реализации прямого наследования при создании своих классов. За счет создания нового подтипа у существующего класса вы получаете возможность пользоваться всеми его доступными свойствами и методами, изменять его поведение, добавлять свои свойства и методы. Теперь достаточно создать бизнес-объект с указанием класса и подтипа. Более подробно читайте об этом ниже в главах о создании своих бизнес-объектов и модифицировании уже существующих.

Что такое идентификатор объекта. Понятие РУИД-а

Что же такое идентификатор и с чем его едят?

Идентификатор – это совокупность символов (буквы, цифры), обозначающая, или именующая объект, например, программу, вычислительную систему, массив, структуру данных или их фрагменты, в языках программирования. Всякий идентификатор является уникальным внутри использующей его структуры. Т.е. проще говоря, по идентификатору можно определить о каком объекте идет речь.

Применительно к системе Гедымин идентификатор объекта в принципе является идентификатором записи бизнес-объекта. Идентификатор записи – это ключ записи, соответствующий значению ключевого поля в таблице, на основе которой создан бизнес-объект. По принятым положениям в Гедымине поддерживается уникальность идентификаторов внутри отдельной базы.

Одним из ограничений бизнес-объекта является то, что он поддерживает только целочисленные простые идентификаторы, т.е. в главной таблице, на основании которой строится запрос, может быть только одно ключевое поле типа Integer. Каждый раз при добавлении записи генерируется новый идентификатор и подставляется в ключевое поле. Последнее сгенерированное значение хранится в генераторе GD_G_UNIQUE. Значение данного генератора не рекомендуется устанавливать вручную, т.к. этим можно нарушить уникальность идентификаторов во всей базе.

Любые вновь созданные пользователем или системой объекты получают идентификатор в диапазоне 147000000 … 2147483647. Все идентификаторы менее 147000000 являются служебными и зарезервированы системой Гедымин для внутреннего пользования. В любом случае корректный идентификатор будет больше нуля. Если какие либо функции, которые должны возвращать идентификатор записи, возвращают значение меньше или равно нулю, то это значит, что бизнес-объект пустой. Доступ к идентификатору текущей записи бизнес-объекта можно получить, используя свойство ID или напрямую обратившись к ключевому полю. Например,

```
gdcGood.ID
```

или

```
gdcGood.FieldByName(gdcGood.GetKeyField(gdcGood.SubType)). _
  AsInteger
```

Во втором случае, мы по подтипу объекта (SubType) берем название его ключевого поля (GetKeyField) и возвращаем его значение. Первый способ обращения к идентификатору записи бизнес-объекта намного короче.

Зачем требуется уникальность идентификаторов именно в пределах базы? В большинстве случаев можно ограничиться уникальностью только внутри таблицы. Здесь мы подходим к понятию РУИД-а. РУИД (RUID – record unique identifier) – уникальный идентификатор записи между базами Гедымина. Используется для синхронизации как структуры, так и содержимого баз.

Затрагивая эту тему, необходимо упомянуть идентификатор базы, который хранится в генераторе GD_G_DBID и создается при первом подключении к базе. Однако если вы делаете копию с базы, желательно обнулить генератор GD_G_DBID, чтобы Гедымин сгенерировал для копии новый идентификатор. Уникальность идентификаторов баз – это необходимое условие для корректного взаимодействия БД.

РУИД формируется как:

<Идентификатор записи>_<Идентификатор базы>

Т.е. значение РУИД-а – это строка из двух идентификаторов, разделенных пробелом. За счет уникальности идентификаторов баз и уникальности идентификаторов записей в пределах каждой отдельной базы мы получаем уникальность РУИД-а между базами Гедымина. Что касается служебных идентификаторов (со значениями менее 147000000), то для них идентификатор базы всегда будет 17. Это необходимо для того чтобы РУИД-ы служебных записей были одинаковы между базами.

Например, идентификатор базы равен 1235792, идентификатор администратора – 650002, идентификатор нового клиента – 147000001. Для администратора РУИД будет равен 650002_17, а для нового клиента – 147000001_1235792.

Итак, мы по идентификатору вновь созданной записи можем узнать РУИД. Как узнать идентификатор по РУИД-у? Вовсе не факт, что первое числовое значение перед знаком подчеркивания в РУИД-е является идентификатором (далее ИД) записи в текущей базе. Оно скорее является значением ключевого поля в базе с ИД, равным числовому значению после знака подчеркивания. При переносе данных между базами РУИД-ы сохраняются, а идентификаторы изменяются, т.к. перенос данных – это по сути дела создание новых записей в базе. А как мы помним, при создании новой записи ей присвоится следующий по счету идентификатор из генератора GD_G_UNIQUE.

Чтобы узнать идентификатор по РУИД-у достаточно выполнить маленький запрос к таблице GD_RUID.

```
SELECT id
FROM gd_ruid
WHERE
  xid = :xid AND dbid = :dbid
```

В данный запрос необходимо подставить параметры, которые и являются составляющими РУИД-а. XID – это идентификатор объекта в исходной базе (т.е. в базе, где он изначально был создан), DBID – идентификатор самой исходной базы.

Структуру GD_RUID можно увидеть в таблице 6.2

ПК	Поле	Тип	Домен	УИ	Описание
x	Id	INTEGER	dintkey	x	Уникальный идентификатор записи в базе. Входит в первичный ключ таблицы.
	Xid	INTEGER	dintkey	x	Уникальный идентификатор записи в базе, где эта запись была создана. Входит в уникальный индекс GD_X_RUID_XID
	Dbid	INTEGER	dintkey	x	Уникальный идентификатор базы, в которой была создана запись. Входит в уникальный индекс GD_X_RUID_XID.
	modified	TIMESTAMP			Дата создания/изменения РУИД-а
	editorkey	INTEGER	dforeignkey		Кто создал/изменил РУИД. Ссылка на таблицу GD_CONTACT.

Таблица 0.2

Изменение содержимого таблицы GD_RUID напрямую крайне нежелательно. Ее данные используются при синхронизации различных БД. Гедымин сам добавляет, изменяет или удаляет записи в GD_RUID, если это необходимо.

Для работы с РУИД-ами в глобальном объекте **gdcBaseManager** существует несколько функций, которые позволяют узнать РУИД по идентификатору и наоборот, не выполняя лишних запросов к базе. Данный объект создается при загрузке программы и доступен из любого скрипт-объекта.

Функции gdcBaseManager для работы с РУИД-ами:

- **GetIDByRUID(XID: Integer; DBID: Integer): Integer** – возвращает идентификатор объекта по РУИД-у. РУИД передается как первоначальный идентификатор объекта (XID) и идентификатор базы, в которой был создан объект (DBID) двумя отдельными целочисленными параметрами.
- **GetIDByRUIDString(RUIDStr: String): Integer** – возвращает идентификатор объекта по РУИД-у. РУИД передается как строка (RUIDStr).
- **GetRUIDByID(ID: Integer; XID: Integer; DBID: Integer)** – возвращает РУИД двумя отдельными параметрами (XID, DBID) по переданному идентификатору(ID).
- **GetRUIDStringByID(ID: Integer): String** – возвращает РУИД строкой по переданному идентификатору (ID).

Данные функции работают на транзакции для чтения. Это означает, что если вы добавляете какие-либо объекты на одной транзакции, то пока вы ее не закроете, вы не сможете прочесть их РУИД-ы. Поэтому существуют аналогичные функции, в которых последним параметром передается открытая транзакция. Передавая необходимую вам транзакцию вы сможете узнать текущее значение РУИД-ов даже если транзакция еще не подтверждена. Однако, если транзакция будет откачена, то откатится и создание объектов, а следовательно и созданные РУИД-ы. Ниже приведен список функций gdcBaseManager для работы с РУИД-ами на конкретной транзакции.

- **GetIDByRUIDTr(XID: Integer; DBID: Integer; ibtr: TIBTRansaction): Integer**
- **GetIDByRUIDStringTr(RUIDStr: String; ibtr: TIBTransaction): Integer**
- **GetRUIDByIDTr(ID: Integer; XID: Integer; DBID: Integer; ibtr: TIBTransaction)**
- **GetRUIDStringByIDTr(ID: Integer; ibtr: TIBTransaction): String**

Зачем вообще нужны идентификаторы? Набор из более чем восьми цифр достаточно тяжело читается и не несет практически никакой информации о записи. Куда как понятнее характеризовать запись текстовым полем, которое будет хранить некое смысловое значение. Здесь есть несколько «но»:

- при использовании текстового поля в качестве идентификатора необходимо заботиться о его уникальности;
- индекс, создаваемый на текстовое поле может быть достаточно велик, что существенно может сказаться на размере базы;
- если рассматривать этот вопрос применительно к Гедымину, то, как было уже сказано выше, платформа поддерживает только простые целочисленные ключи.

Например, вы создали организацию, ссылки на которую есть в банковской выписке, платежных документах, проводках. Затем вы заметили, что в наименовании организации ошибка. Достаточно исправить название в справочнике клиентов, в остальных документах оно автоматически станет отображаться с последней правкой, т.к. идентификатор вы не меняли. Однако имеется и другая сторона медали. Например, организацию переименовали. Т.е. по сути возникло новое юридическое лицо, а старое перестало функционировать. В данном случае нельзя просто исправить название, т.к. во всех документах, которые были выписаны на предыдущее юридическое лицо, станет отображаться новое название. Нужно создать новую запись с новым идентификатором, а старой установить признак «не активна» (в Гедымине для этого служит поле DISABLED).

Работа бизнес-объекта с данными. Транзакции

Итак, наш бизнес-объект – это прежде всего некий набор данных. Однако писать самим запрос на выборку этих данных из базы вам скорее всего не придется. Мы уже сталкивались с понятием главной таблицы бизнес-объекта, ключевым полем и т.д. Рассмотрим сейчас эти понятия более подробно.

В каждом запросе, на основании которого мы получаем данные из базы в бизнес-объект, существует некая главная таблица, которая хранит идентификаторы записей и основную информацию. Все остальные таблицы (или представления), входящие в запрос являются дополнительными. Например, все сведения о товарах хранятся в таблице GD_GOOD. Однако некоторые поля этой таблицы являются

внешними ссылками на другие таблицы. Поле, хранящее информацию о единице измерения товара, является ссылкой на таблицу GD_VALUE. Чтобы вывести единицу измерения в текстовом представлении, а не в виде целочисленного ключа, нам необходимо в запрос к главной таблице GD_GOOD присоединить таблицу GD_VALUE. Т.е.

```
SELECT z.name as goodname, v.name as valuename
FROM gd_good z
  LEFT JOIN gd_value v ON v.id = z.valuekey
```

Некоторые дополнительные таблицы, входящие в запрос, играют не только информативную роль, но и роль ограничителей. Так, например, в таблице GD_DOCUMENT регистрируются все документы. Для каждого документа существует еще своя дополнительная таблица, которая хранит специфичную для типа документа информацию. Присоединение такой таблицы по JOIN будет ограничивать выборку из GD_DOCUMENT только ее данными.

Главную таблицу еще называют базовой. Все операции, производимые над бизнес-объектом (добавление, редактирование, удаление), производятся сначала над базовой таблицей и только затем, в случае необходимости, над дополнительными. В бизнес-объекте существует несколько методов, используя которые вы можете узнать на основании какой таблицы он создан:

- **GetListTable(const SubType: string): string** – возвращает название базовой таблицы бизнес-объекта, исходя из переданного подтипа объекта.
- **GetKeyField(const SubType: string): string** – возвращает название ключевого поля объекта.
- **GetListField(const SubType: string): string** – возвращает название поля для отображения. Это поле используется по умолчанию в выпадающих списках, а также при обработке запроса внутренним SQL-парсером.
- **SubType: string** – возвращает подтип бизнес-объекта. Для стандартных бизнес-объектов данное свойство всегда будет возвращать пустую строку. Для бизнес-объектов, созданных пользователем Гедымина, SubType обязательно будет заполнен.

Для работы с базой данный бизнес-объект использует пять видов запросов:

- SelectSQL – запрос на выборку данных из БД.
- InsertSQL – запрос для вставки новой записи в **базовую** таблицу.
- ModifySQL – запрос для модификации записи в **базовой** таблице.
- DeleteSQL – запрос для удаления записи из **базовой** таблицы.
- RefreshSQL – аналогичен SelectSQL, за исключением того, что возвращает только одну запись с заданным ключом. Используется для обновления только что созданной или отредактированной записи. Вызывается после сохранения изменений.

Как мы видим, запросы на изменение работают только с базовой таблицей. Для сложных бизнес-объектов, которые изменяют записи в нескольких таблицах за одну транзакцию выполняются дополнительные запросы.

Все виды запросов формируются автоматически, исходя из базовой таблицы и подтипа объекта. Таблицы в запросе на выборку обязательно имеют алиас – уникальный в пределах запроса псевдоним. У базовой таблицы алиас всегда будет «z». Кроме того, для всех пользовательских полей-ссылок в запрос на выборку внутренним парсером (разборщиком запросов) добавляются дополнительные таблицы, используя связку по LEFT JOIN, а в SELECT-часть добавляются поля для отображения из этих таблиц. Узнать какие таблицы вытянуты парсером не сложно: их алиас формируется как алиас таблицы, имеющей поле ссылку, плюс знак подчеркивания, плюс название поля ссылки. Поля для отображения, добавленные парсером, также имеют псевдонимы, которые формируются как алиас таблицы, которой они принадлежат, плюс знак подчеркивания, плюс непосредственно название самого поля. Как задать основное и дополнительные поля отображения для таблицы, или отменить вывод полей в конкретных объектах читайте в главе 8.

Управлять ссылками в SQL-запросе можно не только через настройки объектов базы данных. В каждом бизнес-объекте есть внутренний настройщик парсера SQL-запросов. Получить к нему доступ можно, используя свойство SQLSetup. Данный настройщик парсера используется чаще всего для указания, какие ссылки при обработке необходимо проигнорировать. Излишнее загромождение запроса

дополнительными таблицами и полями ведет к замедлению его работы. Пример использования SQLSetup вы можете увидеть в главе 6.6 «Основные методы и свойства».

Т.к. выше мы коснулись понятия пользовательских объектов базы данных, дадим ему определение. Пользовательский объект БД – это объект, созданный средствами Гедымина. Наименование любого пользовательского объекта обязательно имеет префикс USR$. Более подробную информацию на эту тему можно получить в главе 8.

Если вас не устраивает автоматически сформированный запрос, вы можете перекрыть соответствующий метод и подкорректировать запрос вручную. О перекрытии методов смотрите ниже.

Итак, наш бизнес-объект имеет пять основных запросов для работы с базой данных. При открытии объекта (метод Open) на сервере выполняется запрос на выборку и на клиентскую машину пересылается некоторое количество записей, которые помещаются в буфер бизнес-объекта. При необходимости записи докачиваются с сервера на клиента. Количество записей, по умолчанию передаваемое сервером, устанавливается сабсетом (или подмножеством) объекта при помощи свойства BufferChunks (определяет размер активного буфера). Подмножество объекта (свойство SubSet класса TgdcBase) задает условия ограничения на выборку записей. Существует несколько стандартных для всех бизнес-объектов сабсетов:

- **All** – сабсет, устанавливаемый по умолчанию. Означает, что на выборку не наложено никаких дополнительных условий. Количество подкачиваемых записей при этом сабсете равно 200.
- **ByID** – при данном подмножестве в выборке может быть только одна запись, соответствующая конкретному значению ключа, или же, если запись с таким ключом не найдена, бизнес-объект откроется пустым. Если при данном сабсете количество записей больше одной, то запрос на выборку некорректен.
- **ByName** – данное подмножество добавляет в WHERE-часть условие, которое ограничивает выборку по заданному значению поля-отображения (GetListField). Количество возвращаемых в сервера записей в этом случае равно 10.
- **OnlySelected** – данное подмножество работает в паре со свойством бизнес-объекта SelectedID. SelectedID представляет собой сортированный динамический массив целочисленных ключей. Он хранит отмеченные пользователем записи. При установке сабсета OnlySelected в выборке будут только те записи, ключи которых попали в свойство SelectedID текущего бизнес-объекта. При выборе данного подмножества, как и при выборе других подмножеств за исключением трех выше перечисленных, количество подкачиваемых с сервера записей равно 100.

В наследниках класса TgdcBase существуют и другие подмножества. Многие из них могут дополнять друг друга, некоторые же являются взаимоисключающими. Например, с сабсетом ByID нецелесообразно использовать другие сабсеты.

Когда пользователь изменяет данные, то он изначально работает с буфером бизнес-объекта. При подтверждении изменений на сервер посылается соответствующий запрос. Клиент получает с сервера сообщение об успешном выполнении запроса и при необходимости обновляет данные в буфере, для чего и используется RefreshSQL. Если операция по изменению данных по каким-то причинам на сервере не прошла, то все изменения в буфере бизнес-объекта откатываются.

В данном месте изложения мы подошли к понятию транзакции. Бизнес-объект использует два вида транзакций: транзакцию на чтение, которая открыта всегда, и транзакцию на изменение, которая открывается только при необходимости изменений данных.

Транзакция (transaction – в пер. с английского – операция) – это механизм, позволяющий объединять различные действия в логические блоки и обеспечить возможность принимать решения об успешности действий всего блока операций в целом. Логические блоки операций осуществляют перевод базы данных из одного целостного состояния в другое. Механизм транзакций служит для обеспечения изоляции изменений, совершенных операциями в контексте одной транзакции, от операций в других транзакциях. Завершение транзакции означает ее закрытие с подтверждением сделанных изменений или их откатом.

В Гедымине поддерживаются следующие уровни изоляции транзакций:

- Транзакция, которая видит записи в том состоянии, в котором они были на момент ее старта. Такая транзакция создается по-умолчанию. Она не видит изменений, сделанных другими пользователями. Подобная транзакция используется в отчетах.
- Транзакция, которая видит подтвержденные изменения других пользователей. Бизнес-объект использует транзакции данного типа.

Транзакция указывает бизнес-объекту к какой базе подключаться и как работать с данными. Параметры транзакции указывают степень видимости изменений, сделанных другими пользователями на других транзакциях. Как мы уже сказали, бизнес-объект работает с двумя транзакциями. Транзакция на чтение (свойство ReadTransaction класса TgdcBase) позволяет бизнес-объекту подключаться к базе данных и считывать из нее данные. При этом она должна быть настроена таким образом, чтобы видеть подтвержденные изменения других пользователей. По умолчанию каждому бизнес-объекту присваивается транзакция на чтение из свойства ReadTransaction объекта gdcBaseManager. Т.е. по умолчанию все бизнес-объекты системы Гедымин используют одну транзакцию на чтение. ReadTransaction должна содержать следующие параметры:

- read – разрешает только операции чтения.
- read_committed – возможность видеть подтвержденные изменения других пользователей. Без этого параметра транзакция будет видеть записи в том состоянии, в котором они были на момент ее старта.
- rec_version – используется вместе с read_committed. Позволяет читать записи, имеющие неподтвержденные версии. Версия записи – это копия записи, которая создается при попытке ее изменить. Эта версия целиком принадлежит транзакции, и все операции в рамках этой транзакции будут производить изменения над версией записи, а не над исходным оригиналом. Данный параметр означает, что при чтении какой-либо записи просто считывается последняя подтвержденная версия записи.
- nowait – при возникновении конфликта немедленно возникает ошибка.

Транзакция на изменение (свойство Transaction класса TgdcBase) у каждого объекта своя, если настройщиком не указано иначе. Она создается при создании объекта. Данная транзакция стартует только при необходимости выполнить какие-нибудь изменения. После выполнения изменений, транзакция закрывается. Транзакция на изменение должна иметь следующие параметры:

- rec_version
- read_committed
- nowait

Каждая транзакция работает как бы со своим срезом базы данных. Чем больше транзакций открыто, тем больше служебной информации обрабатывать и хранить приходится серверу. Увеличение нагрузки на сервер естественно сказывается на его производительности. Поэтому необходимо вовремя закрывать уже отработавшие транзакции. При аварийном завершении программы, все транзакции откатываются.

Чаще всего вам не придется самим отслеживать работу транзакций, т.к. большую часть работы на себя берет бизнес-объект. Однако существуют случаи, при которых необходимо вмешиваться в работу транзакций. Одним из таких случаем является организация связки master-detail. Подробнее о master-detail читайте ниже.

Работа бизнес-объекта с визуальными формами

Как мы знаем, бизнес-объект не просто набор данных. Он обладает некоторой функциональностью, которая позволяет пользователю достаточно просто организовать визуальную работу с объектом. В данной подглаве мы рассмотрим связь бизнес-объекта с формами.

Форма просмотра

У каждого бизнес-объекта существует своя форма просмотра. Форма просмотра представляет собой немодальное окно, в котором отображается список записей, соответствующих условиям выборки, меню для работы с объектом и панель элементов управления (смотри рисунок 6.1). Базовым классом для всех форм просмотра является класс **Tgdc_frmG**.

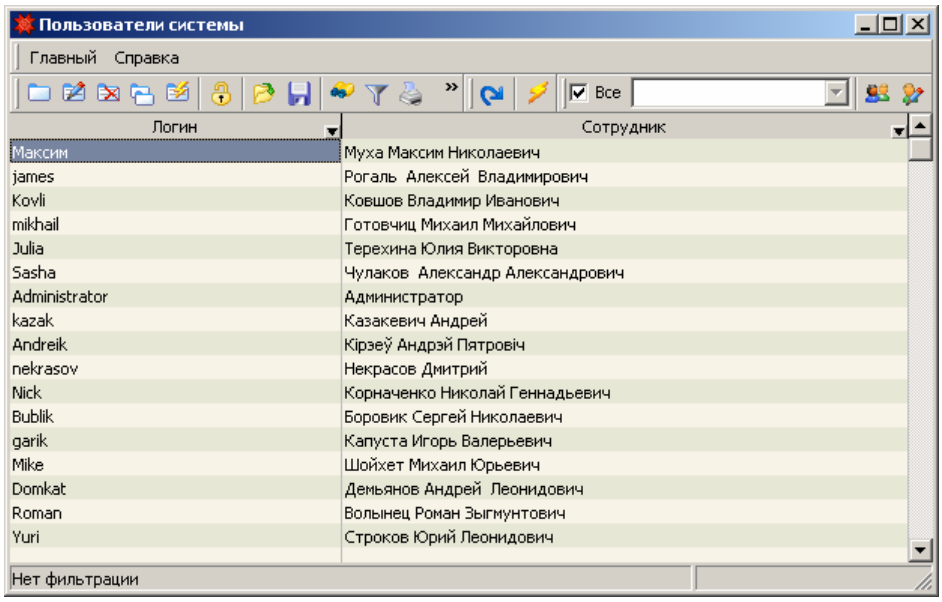


Рисунок 0.1

Между формой просмотра и бизнес-объектом существует двусторонняя связь: через бизнес-объект мы можем узнать название формы или создать ее, через форму мы можем получить доступ к бизнес-объекту. Следующие методы бизнес-объекта позволяют напрямую (из скриптов) работать с формой просмотра:

- **CreateViewForm**(AnOwner: TComponent; AClassName: String; ASubType: String): TForm – создает форму просмотра с переданным классом (AClassName) и сабтайпом (ASubType). Владельцем формы является AnOwner. Если вы не хотите указывать конкретного владельца, то первым параметром будет nil. CreateViewForm проверяет существует ли переданный класс, если нет, то создается форма с классом, используемым по умолчанию для объекта, вызвавшего метод. Т.е. если вам нужно создать форму просмотра для текущего объекта, то параметр AClassName может быть пустой строкой. Если же вам необходимо создать просмотровую форму для другого объекта, то вы можете просто передать в CreateViewForm необходимый вам класс формы и ее подтип. Кроме того данный метод проверяет, была ли уже создана необходимая форма. Если такая форма действительно уже создана, то метод просто вернет на нее ссылку. В обратном случае, он сначала создаст форму, а затем вернет на нее ссылку. Т.е. данный метод не позволяет создавать несколько форм одного класса и подтипа.
- **CreateViewFormNewInstance**(AnOwner: TComponent; AClassName: String; ASubType: String; ANewInstance: Boolean): TForm – данный метод аналогичен CreateViewForm, за исключением того, что он позволяет создавать несколько форм одного класса и сабтайпа. Для этого необходимо параметр ANewInstance установить в истину. Если этот параметр будет установлен в ложь, то метод будет работать точно так же, как и CreateViewForm. При ANewInstance равным True, метод в любом случае создаст форму указанного класса и подтипа и вернет на нее ссылку.
- **GetViewFormClassName**(ASubType: string): string – возвращает название класса формы просмотра для текущего объекта по указанному сабтайпу (ASubType).

Рассмотрим несколько примеров создания форм просмотра из макросов:

```
'Создаем бизнес-объект класса «Люди»
set gdcContact = Designer.CreateObject(nil, "TgdcContact", "")
'Создаем форму просмотра, используемую по умолчанию
'для класса «Люди»
'Если форма уже создана, то CreateViewForm просто вернет на
'нее ссылку
set frmContact = gdcContact.CreateViewForm(nil, "", "")
'Создаем форму просмотра товаров
'Метод CreateViewFormNewInstance сначала создаст форму,
'и затем вернет на нее указатель
```

```
set frmGood = gdcContact.CreateViewFormNewInstance(nil, _
  "Tgdc_frmMainGood", "")
```

Если нам необходимо вызывать форму просмотра из исследователя, то при редактировании ветки исследователя достаточно указать имя бизнес-класса и подтип. Смотри рисунок 6.2

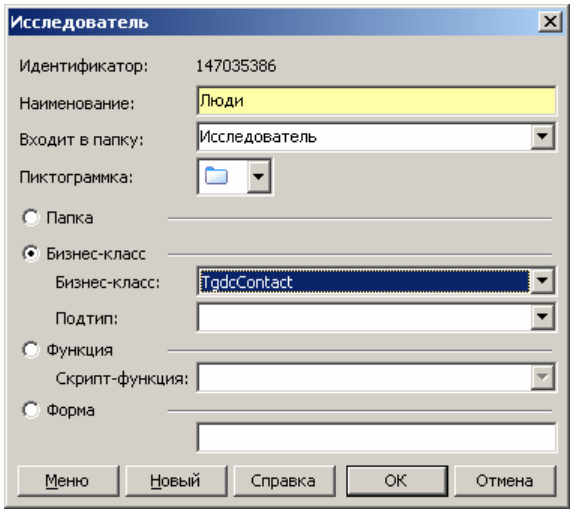


Рисунок 0.2

Формы просмотра можно поделить на простые и сложные. Простая форма предоставляет пользователю возможность просматривать один бизнес-объект. Сложные формы отображают связи между несколькими бизнес-объектами. Например, окно для просмотра выписки предоставляет пользователю две взаимосвязанных таблицы: таблицу шапки выписки и таблицу позиций.

Получить доступ к бизнес-объектам, лежащим на форме просмотра можно через свойства формы, такие как:

- gdcObject – дает доступ к главному бизнес-объекту на форме.
- gdcDetailObject – дает доступ к детальному (т.е. зависимому от некоторых свойств главного) бизнес-объекту на форме. Данное свойство доступно только на сложных формах просмотра. Базовым классом для таких форм является класс Tgdc_frmMDH.

Форма для выбора записей

Форма для выбора записей представляет собой разновидность формы просмотра. В принципе она и является формой просмотра бизнес-объекта, только настроенной определенным образом. На форме в режиме выбора становятся доступными панель элементов управления с кнопками «Выбрать все отображенные записи», «Удалить из выбранных все отображенные записи», «Удалить из выбранных все записи»; панель с гридом для выбранных записей и кнопками Ok, Отмена, Удалить; а также в первой колонке грида или в дереве (в зависимости от вида формы и бизнес-объекта, из которого будет идти выбор записей) становятся доступны элементы для выбора записей (checkbox - чекбокс). При выборе записи чекбокс становится отмеченным, а сама запись отображается также в нижнем гриде. Чтобы убрать запись из выбранных, нужно отщелкнуть элемент для выбора записей, или позиционировать курсор на необходимую запись в таблице для выбора на нижней панели и нажать кнопку «Удалить». Форму в режиме выбора можно увидеть на рис 6.3.

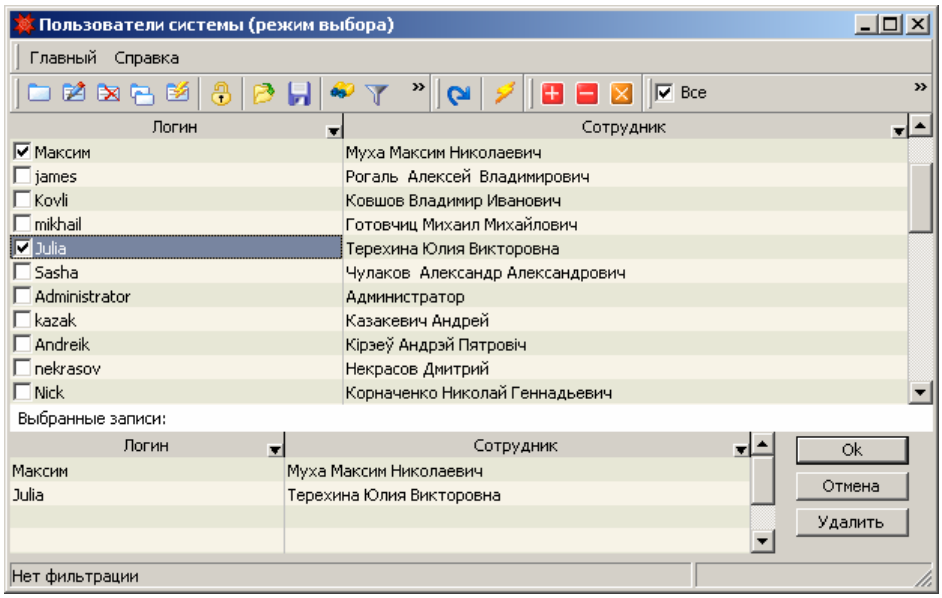


Рисунок 0.3

Чтобы вызвать форму просмотра в режиме выбора записей, можно воспользоваться следующими методами бизнес-объекта:

- ChooseItems
- ChooseItemsSelf
- ChooseOrderItems
- ChooseOrderItemsSelf

Метод **ChooseItems** используется для вывода формы выбора бизнес-объекта с заданным классом и подтипом. Если пользователь подтвердил выбор записей, то метод вернет Истина, в противном случае (при отмене операции выбора) – Ложь. Синтаксис вызова метода следующий:

```
ChooseItems(Cl: String; KeyArray: TgdKeyArray;
  DoProcess: Boolean; ChooseComponentName: String;
  ChooseSubset: String; ChooseSubType: String): Boolean;
```

Параметры:

Cl - наименование класса объекта, записи которого мы будем выбирать.

KeyArray - список ключей выбранных записей. Если мы передаем непустой список, то на форме выбора записи с переданными идентификаторами будут отмечены "птичками". При закрытии формы с подтверждением выбора KeyArray заполняется ключами вновь выбранных записей.

DoProcess - не используется. Осталось для совместимости с предыдущими версиями. Должно быть False.

ChooseComponentName - наименование компонента на форме из которого будет идти выбор записей. Если передана пустая строка, то выбор будет идти из датасета, класс и подтип которого наиболее подобны переданным классу и подтипу. Может быть полезен на формах с несколькими датасетами.

ChooseSubset - сабсет(ы), который подставится объекту для выбора записей.

ChooseSubType - сабтайп объекта, записи которого мы будем выбирать.

Метод **ChooseItemsSelf** используется для вывода формы выбора бизнес-объекта вызвавшего метод. При этом отмеченные записи попадают в свойство SelectedID объекта. Если перед вызовом SelectedID был не пустой, то на форме выбора будут отмечены записи, идентификаторы которых попали в SelectedID. Если пользователь подтвердил выбор записей, то метод вернет Истина, в противном случае (при отмене операции выбора) – Ложь. Синтаксис вызова метода следующий:

```
ChooseItemsSelf(DoProcess: Boolean; ChooseComponentName: String;
  ChooseSubset: String): Boolean;
```

Параметры:

DoProcess - не используется. Осталось для совместимости с предыдущими версиями. Должно быть False.

ChooseComponentName - наименование компонента на форме из которого будет идти выбор записей. Если передана пустая строка, то выбор будет идти из бизнес-объекта, класс и подтип которого наиболее подобны классу и подтипу объекта, вызвавшего метод. Может быть полезен на формах с несколькими бизнес-объектами. Однако здесь необходимо уточнить, что если указать наименование объекта, класс и подтип которого не совместимы с классом и подтипом бизнес-объекта, вызвавшего метод, то можно получить некорректную последовательность идентификаторов в свойстве SelectedID по выполнению операции выбора. Говоря о некорректной последовательности идентификаторов, мы имеем ввиду ключи, для которых не существует записей в бизнес-объекте, вызвавшем метод.

ChooseSubSet - сабсет(ы), который подставится объекту для выбора записей.

Методы ChooseItems и ChooseItemsSelf возвращают упорядоченные последовательности идентификаторов. Однако иногда полезно вернуть последовательность выбранных записей в том порядке, в котором их отметил пользователь. Для этого используются немного более расширенные методы ChooseOrderItems и ChooseOrderItemsSelf.

Метод **ChooseOrderItems** аналогичен методу ChooseItems, за исключением того, что его список параметров несколько расширен. Синтаксис вызова следующий:

```
ChooseOrderItems(Cl: String; KeyArray: TgdKeyArray;
  AChosenIDInOrder: VarArray; ChooseComponentName: String;
  ChooseSubset: String; ChooseSubType: String;
  ChooseExtraConditions: String): Boolean;
```

Параметры:

AChosenIDInOrder – массив ключей выбранных записей именно в том порядке, в котором их отметил пользователь. Заметим, что в этом варианте команды остался параметр KeyArray, который работает точно также как и в команде ChooseItems.

ChooseExtraConditions – дополнительные условия, накладываемые на бизнес-объект, из которого будет идти выбор. Как мы знаем, ограничить выборку можно используя свойство бизнес-объекта SubSet. Однако иногда требуется условие не объявленное через подмножества выборки бизнес объекта.

Метод **ChooseOrderItemsSelf** аналогичен методу ChooseItemsSelf. Его список параметров также расширен. Синтаксис вызова следующий:

```
ChooseOrderItems(AChosenIDInOrder: VarArray;
  ChooseComponentName: String;
  ChooseSubset: String; ChooseSubType: String;
  ChooseExtraConditions: String): Boolean;
```

Не используйте методы ChooseItemsSelf и ChooseOrderItemsSelf для выбора записей из объектов класс и подтип которых не совместим с классом и подтипом бизнес-объекта вызвавшего метод.

Диалоговое окно

На форме просмотра все записи представлены неким списком. И хотя на ней присутствует режим, включив который можно редактировать записи, все же это не всегда удобно и наглядно. Для наглядности модификации данных существует диалоговое окно. Диалоговое окно представляет собой форму, работающую в модальном режиме. Базовым классом для всех диалоговых окон является класс Tgdc_dlgG. Одной из особенностей диалоговых окон является то, что бизнес-объект, запись которого редактируется, не лежит на диалоге. Бизнес-объект передается диалоговому окну извне, после его (окна) создания. Т.е. в момент создания диалога, еще не известно с каким бизнес-объектом он будет работать. Это становится известно в момент инициализации диалога (метод SetupDialog класса Tgdc_dlgG).

Между диалоговым окном и бизнес-объектом, вызвавшем его, также существует двусторонняя связь. В бизнес-объекте есть несколько методов для работы с диалогами:

- **CreateDialogForm** – создает форму диалога, заданную для класса по умолчанию.
- **CreateDialog**(ADlgClassName: string) – переводит бизнес-объект в состояние вставки и выводит на экран модальное окно для редактирования содержимого объекта. Параметр ADlgClassName содержит наименование класса окна для редактирования. Если передана пустая строка, используется окно, заданное для текущего объекта по умолчанию (см. свойство CreateDialogForm). В свойство BaseState бизнес-объекта добавляется флаг sDialog

(sSubDialog - для вложенного диалога). Если в системе Гедымин включена опция "Сохранять значения полей в диалоговых окнах", то полям объекта присваиваются значения предыдущего ввода информации. Затем у диалоговое окно инициализируется (вызывается метод Setup, который передает на диалог ссылку на бизнес-объект, вызвавший метод, и настраивает окно для дальнейшей работы) и выводится на экран. После закрытия диалога, если пользователь подтвердил введенные значения, и в опциях системы установлен флаг "Сохранять значения полей в диалоговых окнах", в пользовательском хранилище сохраняется информация о последнем вводе записи.

- **CreateDialogSubType**(AClassName: string; ASubType: string) – добавляет запись заданного класса и подтипа и выводит окно для ее редактирования. Используется чаще всего при добавлении новых записей в базовый бизнес-объект. AClassName - наименование класса, запись типа которого будем добавлена. Этот класс должен быть наследником класса бизнес-объекта, вызвавшего метод. ASubType - подтип класса, запись типа которого будет добавлена. CreateDialogSubType создает объект заданного класса и подтипа и вызывает у объекта CreateDialog(""). Затем, если пользователь подтвердил сделанные изменения, происходит синхронизация содержимого созданного объекта и объекта, вызвавшего метод.
- **CopyDialog** – данный метод копирует текущую запись и выводит диалог для ее редактирования, используя метод EditDialog. Возвращает Истина если копирование произошло без ошибок и пользователь подтвердил его. В обратном случае возвращает Ложь.
- **EditDialog**(ADlgClassName: string): Boolean - используется для вывода диалога для редактирования текущей записи. Параметр ADlgClassName хранит наименование класса диалога для редактирования записи. По умолчанию передается пустая строка, что означает использование диалога, заданного изначально для текущего объекта. Если передана строка, равная "TGDC_DLGOBJECTPROPERTIES", то вместо диалога редактирования выводится диалог, отражающий свойства текущей записи. При необходимости бизнес-объект переводится в состояние редактирования. После создания диалог инициализируется (вызывается метод Setup) и выводится на экран. Возвращает Истина, если пользователь сохранил изменения, Ложь в случае отмены.

Диалоги, как и формы просмотра бывают простые и сложные. Простые диалоги предназначены для редактирования одного объекта, сложные – для редактирования нескольких объектов. При этом среди сложных диалогов можно выделить диалоги со связью мастер-дитейл, диалоги с объектами-множествами. Чтобы получить доступ к бизнес-объекту с диалога можно воспользоваться одним из следующих свойств:

- gdcObject – дает доступ к главному бизнес-объекту на диалоге. Здесь нельзя путать понятия главного объекта на форме просмотра и главного объекта на диалоге, т.к. главный объект на диалоговом окне может быть детальным на форме просмотра.
- gdcDetailObject – дает доступ к детальному (т.е. зависимому от некоторых свойств главного) бизнес-объекту на диалоге. Данное свойство доступно только на сложных диалогах. Базовым классом для таких диалогов является класс Tgdc_dlgHGR.

Следует еще остановится на транзакциях при работе с диалоговыми окнами. Чаще всего на простых диалоговых окнах транзакция на изменение стартует только в момент сохранения изменений в базу, т.е. при закрытии окна. На сложных диалоговых окнах транзакция должна быть открыта на протяжении всей работы с окном, т.к. нам приходится изменять сразу несколько зависимых друг от друга объектов. При закрытии окна в зависимости от подтверждения пользователем внесенных изменений или их отмены транзакция подтверждается или откатывается.

Окно Свойства Объекта

Любой бизнес-объект может обратиться к окну «Свойства объекта». Из макроса его можно вызвать через метод бизнес-объекта EditDialog("Tgdc_dlgObjectProperties"), оно доступно и с любой формы просмотра.

Окно «Свойства объекта» предназначено для просмотра основных полей объекта, назначения/изменения прав доступа, просмотра SQL запроса, а также просмотра и возможного изменения значений всех полей объекта.

Окно имеет три закладки:

- Общие – предоставляет основную информацию об объекте: его класс, подтип, подмножество, идентификатор, РУИД, кем и когда был создан объект, кто последний его менял (эта информация приводится только для объектов, для которых отслеживается создание и изменение под разными пользователями, т.е. в главной таблице присутствуют поля editorkey и creatorkey), отображаются права доступа. Для деревьев выводится идентификатор родителя, а для интервальных деревьев – правая и левая границы. Пример закладки «Общие» можно увидеть на рисунке 6.4

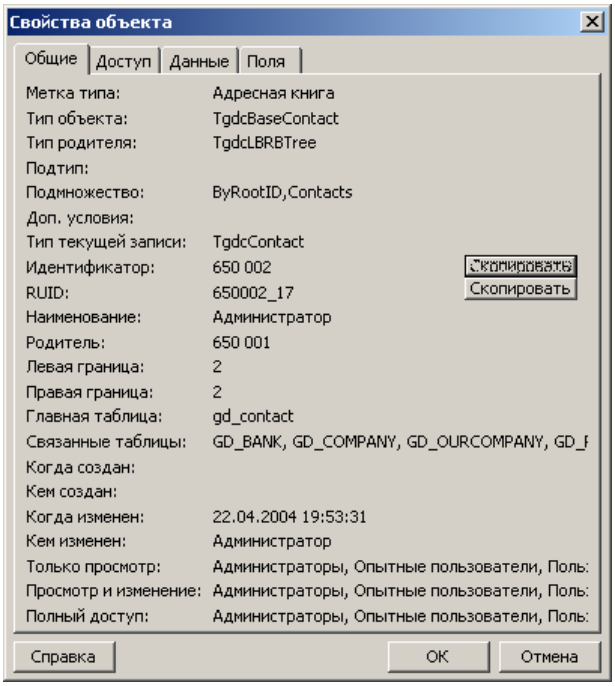


Рисунок 0.4

- Доступ – предназначена для установки прав пользователей к текущей записи (не следует путать с правами доступа на уровне класса). Данная закладка доступна только если в главной таблице присутствуют поля AVIEW, ACHAG, AFULL, которые хранят соответственно доступ к записи на просмотр, изменение, полный доступ. Права устанавливаются для групп пользователей. См. рисунок 6.5

- Данные – отображает данные текущей записи с возможностью их изменения. Кроме того, отображается сколько записей считано в буфер бизнес-объекта и сколько под них выделено оперативной памяти. Внизу закладки располагается кнопка, по которой можно посмотреть запрос бизнес-объекта на выборку данных, и выпадающий список с перечислением параметров и их текущих значений, если таковые имеются в запросе. См. рис 6.6

Окно «Свойства объекта» выводится также в том случае, если для текущего бизнес-объекта не предусмотрена (или не описана) операция добавления/редактирования записей.

Подробнее о свойствах форм можно узнать в главе 7.

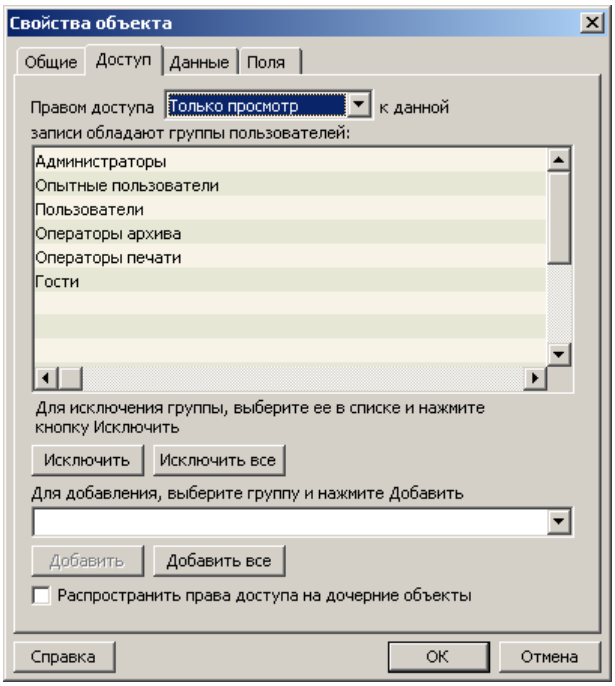


Рисунок 0.5

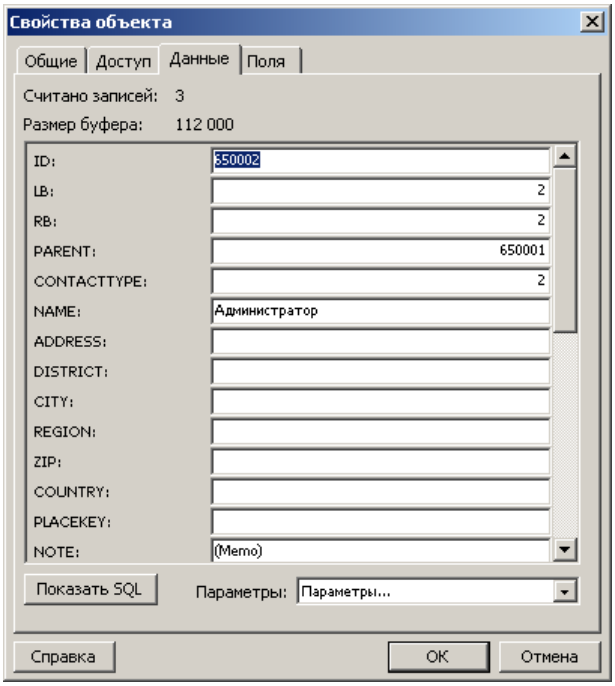


Рисунок 0.6

Связь master-detail. Множества.

Одна из основных причин использования реляционных баз данных (таких как Interbase, MSSQL, Oracle и т.д.) заключается в том, что формат базы данных прост для понимания и с ним легко обращаться. Кроме того, реляционная база данных позволяет в полной мере использовать механизмы по обеспечению целостности данных. Одним из таких механизмов является использование внешних ключей.

Внешний ключ используется для поддержания ссылочной целостности в базе данных. Он контролирует корректность ссылок, а также позволяет управлять этими ссылками. При этом связанные внешним ключом таблицы можно разделить на главную и подчиненную или таблицу-мастер и таблицу-деталь. В принципе, внешний ключ означает, что поле, не являющееся ключевым в одной таблице, является

ключом в другой, т.к. создать ссылку можно только на первичный ключ или уникальный индекс другой таблицы.

Между таблицами может быть три вида отношений:

- Один-к-одному
- Многие-к-одному
- Многие-ко-многим

Таблицы с отношением **один-к-одному** достаточно редки, т.к. в этом случае вся информация может быть включена в одну таблицу. Однако иногда такая связь имеет смысл. Т.к. мы можем использовать вспомогательную таблицу для хранения некоторых специфичных свойств, которые сопоставляются не всем записям главной таблицы. В Гедымине существует механизм для создания таких таблиц. Смотри в главе 7 описание объекта «Таблица со ссылкой».

Отношение **многие-ко-многим** часто реализуется при помощи промежуточной таблицы, которая хранит внешние ключи, связанные с каждой из основных таблиц. В Гедымине этот механизм реализуется при помощи множеств. Данный тип связи может быть полезен при организации множественной зависимости между двумя таблицами. Например, существует справочник единиц измерения и справочник товаров. Одному товару может соответствовать несколько единиц измерения, и в то же время одной единице измерения соответствуют несколько товаров.

Отношение **многие-к-одному** самое распространенное по использованию отношение между двумя таблицами. Данный механизм связи означает, что одной записи главной таблицы может соответствовать несколько записей детальной таблицы. Например, практически у каждого документа есть шапка и позиции. Вполне логично шапку документа хранить в одной таблице, а позиции в другой. Между этими двумя таблицами устанавливается связь, которая не позволяет создавать позиции, если не указана шапка.

Master-detail

Говоря об отношении master-detail в Гедымине мы чаще всего имеем ввиду отношения типа многие-к-одному. И как уже было сказано выше, мастер-таблица – это таблица, хранящая главные записи, а детальная таблица – это таблица хранящая детализацию записей главной таблицы. Т.к. наш бизнес-объект – это, по сути, набор данных, то вполне целесообразно использовать такие понятия, как мастер-объект и детальный объект.

Как же организуется связь master-detail в Гедымине?

Если рассмотреть эту ситуацию в отрыве от какой-либо платформы и базы данных, то вывод записей детального объекта, соответствующих мастеру, означает фильтрацию или отбор детальных записей по конкретному значению внешнего ключа. Т.е., например, нам нужно вывести все расчетные счета конкретной организации с идентификатором 650010. Для этого мы бы могли выполнить запрос вида:

```
SELECT *
FROM gd_companyaccount
WHERE
  companykey = 650010
```

В данный запрос значение ключа компании подставлено сразу в текст. Если нам нужно будет найти все счета для другой компании, то нам придется переприсвоить полностью весь текст запроса. Изменение текста запроса каждый раз, когда нам нужно изменить только значение какого-либо параметра, нецелесообразно. Т.к. компоненты для работы с Interbase (далее IBX-компоненты) отслеживают, что запрос был изменен, и каждый раз проводят все операции по подготовке запроса на выполнение сначала. В данном случае лучше использовать параметризированный запрос. В такой запрос значение параметра подставляется извне, сам текст запроса не изменяется, а, следовательно, IBX-компоненты подготавливают такой запрос на выполнение только один раз, что ведет к снижению времени работы запроса.

```
SELECT *
FROM gd_companyaccount
WHERE
  companykey = :companykey
```

Запрос детального объекта также содержит параметр, в который будет подставляться значение ключа мастера. Теперь поговорим о том, как добавить в запрос на выборку дополнительное условие с параметром и задать, какое значение в этот параметр должно передаваться.

Итак, мы уже знаем, что основные запросы бизнес-объекта генерируются автоматически, исходя из названий главной таблицы (метод GetListTable) и ключевого поля (метод GetKeyField). Также мы знаем, что для добавления некоторых условий в запрос можно использовать свойство бизнес-объекта SubSet. В принципе у всех бизнес-объектов существует некий стандартный набор подмножеств, задающих условия. Этот набор можно расширить при помощи перекрытия определенных методов. Как это сделать мы рассмотрим несколько ниже. А сейчас, на основе уже приведенного примера, организуем связь master-detail между двумя объектами gdcCompany и gdcAccount.

Если нам нужно организовать механизм master-detail в скрипте, то он будет выглядеть подобным образом:

```
  'Объявляем необходимые переменные
  Dim gdcCompany, gdcAccount, MasterSource, Creator
  'Создаем вспомогательный VB-объект
  'для управления созданием/уничтожением объектов Гедымина
  set Creator = new TCreator
  'Создаем бизнес-объект Компания
  set gdcCompany = Creator.GetObject(nil, "TgdcCompany", "")
  'Создаем "источник данных"
  set MasterSource = Creator.GetObject(nil, "TDataSource", "")
  'Указываем, из какого датасета мы будем тянуть данные
  MasterSource.DataSet = gdcCompany
  'Создаем бизнес-класс "Расчетный счет"
  set gdcAccount = Creator.GetObject(nil, "TgdcAccount", "")
  'Указываем, что нас интересует подмножество "по компании"
  gdcAccount.SubSet = "ByCompany"
  'Указываем по какому источнику данных будет идти фильтрация
  gdcAccount.MasterSource = MasterSource
  'Указываем название поля мастера, по которому будет
  'идти фильтрация
  gdcAccount.MasterField = _
    gdcCompany.GetKeyField(gdcCompany.SubType)
  'Указываем название параметра в запросе детального
  'объекта, в которое будет подставляться значение из
  'поля мастера
  gdcAccount.DetailField = "companykey"
```

Здесь нам встречается понятие источника данных. Остановимся на нем поподробнее. Источник данных предоставляет данные из указанного датасета другому датасету или визуальным компонентам. Использование такого промежуточного объекта, как источник данных, обусловлено тем, что он может предоставлять данные из любых датасетов, т.е. мы можем получать данные из базы даже не интересуясь, при помощи каких компонентов они оттуда считаны. Все визуальные компоненты, работающие с базой данных, подключаются к ней через источник данных.

Источник данных является реализацией класса TDataSource. Для указания, из какого датасета он будет получать данные, используется свойство DataSet.

При организации связи master-detail нам необходимо своевременно получать значение ключа мастера. Для этого мы указываем детальному объекту источник данных, который предоставляет нам доступ к мастеру. Для указания источника данных мастера используется свойство бизнес-объекта MasterSource.

Далее нам нужно добавить в запрос условие для фильтрации данных по ключу мастера. Для этого мы используем специфичный SubSet (описание стандартных подмножеств для различных бизнес-объектов смотрите в приложении а!!!!!). В нашем случае SubSet, равный «ByCompany», добавляет в запрос условие

z.companykey = :companykey

В итоге запрос на выборку для расчетных счетов выглядит следующим образом:

```
SELECT
  z.id,
```

```
  z.bankkey,
  t.name as joinaccounttype,
  z.companykey,
  z.disabled,
  z.currkey,
  z.account,
  z.accounttypekey,
  z.payername,
  c.name as bankname,
  b.bankcode,
  b.bankmfo,
  b.swift
FROM
  gd_companyaccount z
    LEFT JOIN gd_bank b ON z.bankkey  =  b.bankkey
    LEFT JOIN gd_contact c ON z.bankkey  =  c.id
    LEFT JOIN gd_compacctype t ON z.accounttypekey  =  t.id
WHERE
  z.companykey  =  :companykey
```

Теперь нам нужно указать, как называется ключевое поле мастера. Для этого в детальном объекте используется свойство MasterField. Также мы указываем **название параметра**, в который необходимо подставлять значение ключевого поля мастера. Для этого в детальном объекте используется свойство DetailField. Осталось открыть наши объекты. Теперь при движении по записям мастер-объекта записи в детальном объекте будут каждый раз отфильтровываться по новому ключу. Здесь важно учесть следующий момент: при фильтрации детального объекта по значению ключа мастера каждый раз выполняется запрос к базе. Поэтому, если вы изменяете запрос через соответствующие методы бизнес-объекта, постарайтесь оптимизировать время его выполнения.

Если вам необходимо связать объекты, лежащие на форме, то достаточно заполнить перечисленные выше свойства в инспекторе объектов. На нашем примере это будет выглядеть так:

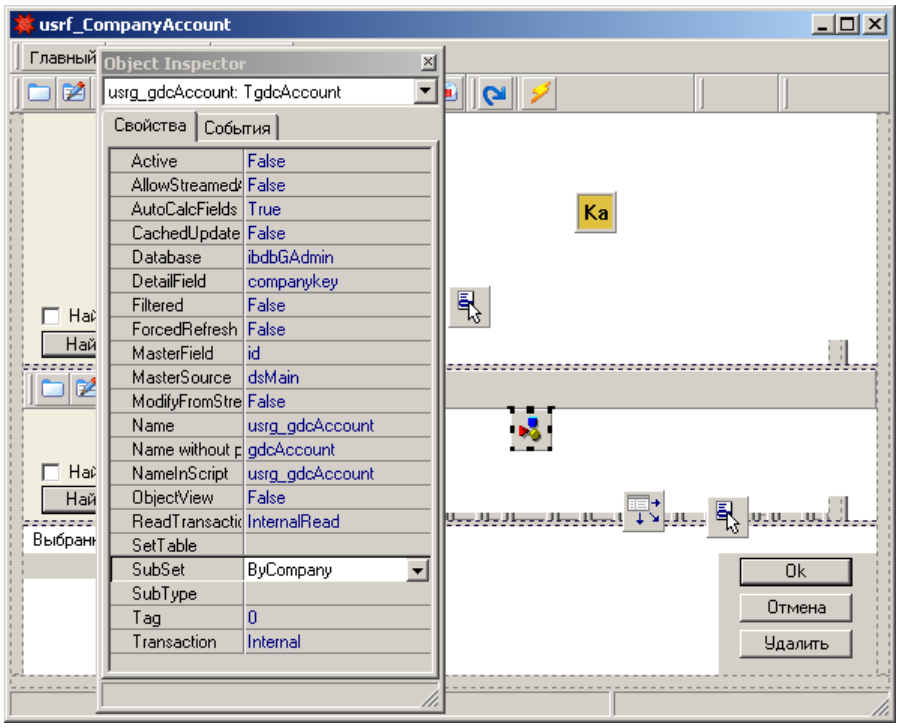


Рисунок 0.7

Также стоит остановиться на транзакциях. Как мы уже знаем, у бизнес-объекта есть две транзакции: на чтение и на запись. Транзакция на чтение предназначена для считывания записей и открыта, пока идет работа с бизнес-объектом. Транзакция на изменение обычно открывается только в момент сохранения изменений в базу. В случае связи master-detail транзакция на изменение может быть открыта значительно дольше.

Рассмотрим простейший пример. Мы добавили новую запись мастера и хотим добавить также записи детального объекта на одной транзакции. Чтобы добавить новую запись, детальному объекту необходимо видеть мастер-объект. А, следовательно, мы или должны закрыть транзакцию, на которой шло добавление записи в мастер-объект, или считать содержимое мастера на той же транзакции, на которой была добавлена запись. В Гедымине эта проблема решается через переприсвоение транзакций детального объекта в режиме редактирования связки master-detail. Т.е. при вызове диалога редактирования всем детальным объектам присваивается и на чтение и на запись транзакция на изменение (свойство Transaction) мастер-объекта. Данное присвоение осуществляется в методе SetupDialog диалогового окна. По окончанию редактирования детальным объектам возвращаются транзакции, которые были до начала редактирования.

Забегая вперед, скажем, что при создании сложных документов связь master-detail настраивается автоматически.

Множества

Как мы уже знаем, множества служат для организации отношения многие-ко-многим. В Гедымине множества реализуются при помощи промежуточных таблиц со сложным первичным ключом (т.е. в первичный ключ входит два или более полей). Бизнес-объект же поддерживает работу только с простыми первичными ключами. Каким же образом работать со множеством? Работа со множеством реализуется при помощи своеобразной связи мастер-дитейл в определенном режиме работы. Рассмотрим это на примере.

Итак, нам необходимо отобразить все возможные единицы измерения для товаров. Базовой таблицей для товаров является GD_GOOD, базовой таблицей для единиц измерения – GD_VALUE. Связкой между этими двумя таблицами является таблица GD_GOODVALUE. Если мы просто к таблице GD_VALUE присоединим при помощи JOIN GD_GOODVALUE, то в нашей выборке данных может не оказаться ни одного поля с уникальным значением. Почему? Потому что на одну единицу измерения может приходиться более чем один товар. Итак, мы получаем запрос вида (для вывода результатов присоединим также таблицу GD_GOOD):

```
SELECT
  z.id as valuekey,
  z.name as valuename,
  g.id as goodkey,
  g.name as goodname
FROM
  gd_value z
  JOIN gd_goodvalue gd$st ON gd$st.valuekey = z.id
  JOIN gd_good g ON g.id = gd$st.goodkey
```

При этом результаты могут быть представлены следующей таблицей:

valuekey (идентификатор единицы измерения)	**valuename (наименование единицы измерения)**	**goodkey (идентификатор товара)**	**goodname (наименование товара)**
147100001	килограмм	147200001	Бензин
147100002	литр	147200001	Бензин
147100001	килограмм	147200002	Молоко
147100002	литр	147200002	Молоко
147100003	бутылка	147200002	Молоко

Как мы видим, в результирующем наборе данных нет ни одного уникального поля. Идентификаторы товара и единицы измерения уникальны только внутри своих таблиц. При объединении этих таблиц в запросе через промежуточную таблицу уникальность можно наблюдать только по нескольким полям (в данном случае уникальной является связка полей goodkey и valuekey).

Как вывести результаты подобного запроса при помощи бизнес-объектов?

Итак, у нас есть бизнес-объект Товар (класс TgdcGood) и бизнес-объект Единица измерения (класс TgdcValue). Для таблицы-связки (GD_GOODVALUE) бизнес-класса не существует. При просмотре

товара мы бы хотели видеть все доступные для него единицы измерения. Для этого нам нужно отфильтровать единицы измерения по товару через таблицу-связку (также называется таблица-множество). В бизнес-объекте есть свойство **SetTable**, которое указывает название таблицы-множества. Если таблица указана, то бизнес-объект работает в режиме обработки множества. Т.е. при операциях изменения/добавления/удаления обрабатываются только записи таблицы связки. Основные записи самого объекта играют чисто информационную роль. Это значит, что в режиме работы со множеством вы не сможете создать новую единицу измерения, но сможете указать новую связь между уже существующей единицей измерения и товаром или изменить признаки уже существующей связки (в нашем примере это может быть поле коэффициента перевода из основной единицы измерения товара в текущую).

Свойство SetTable работает в паре со свойством MasterSource. Если вы указываете таблицу-множество, то у вас обязательно должен быть указан мастер-объект, по которому будут фильтроваться записи множества. В нашем примере мастер-объектом является Товар. При указании SetTable в бизнес-объекте автоматически заполняются свойства MasterField и DetailField, а в запрос добавляется условие для фильтрации. Запрос на выборку объекта Единицы измерения будет выглядеть следующим образом:

```
SELECT
  z.id,
  z.name,
  z.description,
  z.goodkey,
  z.ispack,
  gd$st.goodkey as s$goodkey,
  gd$st.valuekey as s$valuekey,
  gd$st.scale as s$scale,
  gd$st.discount as s$discount,
  gd$st.decdigit as s$decdigit
FROM
  gd_value z
  JOIN
    gd_goodvalue gd$st
  ON
    gd$st.goodkey = :master_record_id
      AND
    gd$st.valuekey = z.id
```

Как мы уже знаем, алиас базовой таблицы всегда будет «z». Алиас для таблицы множества в любом запросе будет «gd$st», при этом псевдонимы полей таблицы-множества будут формироваться как префикс «s$» + реальное наименование поля. Только поля таблицы-связки в данном объекте будут редактируемыми, остальные поля (без префикса «s$») будут иметь флаг «Только для чтения».

Как добавить таблицу-множество средствами Гедымина мы узнаем несколько позднее. Однако, забегая вперед, могу сказать, что при добавлении новой таблицы связки, она будет автоматически настраивать дополнительный бизнес-объект для работы со множествами и отображать его на отдельной закладке диалога редактирования основного объекта. В рассмотренном выше примере на диалоге редактирования объекта Товар появится закладка «Единицы измерения».

Основные методы и свойства

В данной подглаве будут рассмотрены основные методы и свойства, присущие всем бизнес-объектам.

Платформа Гедымин была написана на Delphi, а потому в описании методов и свойств используется синтаксис языка Object Pascal. Также при рассмотрении иерархии наследования бизнес-класса TgdcBase будут использованы классы Delphi.

Иерархия наследования базового бизнес-класса TgdcBase представлена на рисунке 6.8. Принцип наследования позволяет наследникам использовать методы и свойства родительских классов. Потому в описании свойств и методов бизнес-объекта вы можете встретить свойства и методы не столько самого бизнес-объекта, а сколько его родителя.

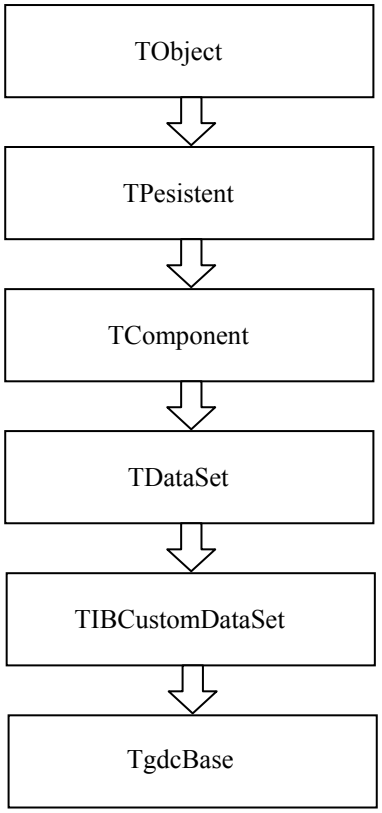


Рисунок 0.8

Свойства бизнес-объектов

- **Active**: Boolean – указывает, открыт объект или нет. Доступно на чтение и на запись. При присвоении значения, равного Истине, открывает объект, при значении, равным Ложь, закрывает объект.
- **BaseState**: String – хранит флаги состояний бизнес-объекта в текстовом виде через пробел. Может хранить следующие флаги:
- **sView** – владельцем бизнес-объекта является форма просмотра. Если объект был создан из макроса без указания принадлежности к форме просмотра, то данного флага не будет.
- **sDialog** – бизнес-объект вызвал диалоговое окно для редактирования/добавления записи.
- **sSubDialog** – бизнес-объект вызвал вложенное диалоговое окно для редактирования/добавления записи. Под понятием вложенное диалоговое окно будем подразумевать диалог, вызванные с другой диалоговой формы.
- **sSyncControls** – указывает, что при редактировании бизнес-объекта при помощи диалогового окна был вызван метод диалога SyncControls. Данный метод используется для синхронизации состояния визуальных элементов в соответствии с введенными данными. Флаг будет установлен ровно на протяжении работы SyncControls.
- **sLoadFromStream** – указывает, что данные объекта загружаются из потока. Загрузка и сохранение данных в поток позволяет переносить данные с одной базы на другую. Про работу с потоками читайте ниже.
- **sMultiple** – указывает, что идет обработка сразу нескольких записей бизнес-объекта. Например, удаление нескольких выделенных записей или их редактирование.

Пример содержимого свойства BaseState для бизнес-объекта, который лежит на форме просмотра, и вызвал диалог для редактирования:

" sView sDialog "

Свойство BaseState доступно только для чтения.

- **Bof**: Boolean – указывает, что курсор стоит на первой записи бизнес-объекта. Устанавливается в Истину при открытии бизнес-объекта, вызове метода First. Только для чтения
- **Bookmark**: String – определяет текущий маркер в датасете. Вы можете установить или считать маркер. При считывании маркера возвращается текущее положение курсора в датасете, при установке, курсор переходит на запись по маркеру.
- **CanChangeRights**: Boolean – возвращает для текущего пользователя права на изменение прав доступа к записи. Только для чтения.
- **CanCreate**: Boolean – возвращает, имеет ли право текущий пользователь на создание новой записи. Только для чтения.
- **CanDelete**: Boolean – возвращает, имеет ли право текущий пользователь на удаление записи. Только для чтения.
- **CanEdit**: Boolean – возвращает, имеет ли право текущий пользователь на изменение записи. Только для чтения.
- **CanPrint**: Boolean – возвращает, имеет ли право текущий пользователь на печать данных. Только для чтения.
- **CanView**: Boolean – возвращает, имеет ли право текущий пользователь на просмотр данных. Только для чтения.
- **CreationDate**: Date – возвращает дату создания записи, если в базовой таблице есть поле CREATIONDATE. Если данное поле отсутствует, то возвращается значение свойства EditionDate. Только для чтения.
- **CreatorKey**: Integer – возвращает идентификатор контактного лица, создавшего запись, если в базовой таблице есть поле CREATORKEY. Если данное поле отсутствует, то возвращается значение свойства EditorKey. Только для чтения.
- **CreatorName**: String – возвращает наименование контактного лица, создавшего запись, используя идентификатор, возвращаемый свойством CreatorKey. Только для чтения.
- **Database**: TIBDatabase – база данных, к которой подключен бизнес-объект. Доступно для чтения и для записи.
- **DeleteSQL**: TStrings – запрос на удаление записи из базовой таблицы. Формируется автоматически. Может быть перекрыт в методе CustomDelete. Только для чтения. Данное свойство доступно и для записи, но при изменении подмножества объекта все SQL-запросы будут перестроены.
- **DetailField**: String – название параметра в SQL-запросе, через который будет идти связка с мастер-объектом (читайте про связь master-detail). Доступно для чтения и для записи.
- **DetailLinks**(Index: Integer): TgdcBase – список детальных объектов. Обращение к детальному объекту идет через индекс Index (порядковый номер в списке). Только для чтения.
- **DetailLinksCount**: Integer – количество детальных объектов. Только для чтения.
- **DSModified**: Boolean – флаг, указывающий, что содержимое текущей записи бизнес-объекта было изменено. Только для чтения.
- **EditionDate**: Date – дата последнего изменения записи. Если в базовой таблице присутствует поле EDITIONDATE, то значение берется из этого поля, иначе значение берется из поля MODIFIED таблицы GD_RUID, если в ней содержится РУИД текущей записи. Иначе возвращается текущая дата. Только для чтения.
- **EditorKey**: Integer – идентификатор контакта, который последний изменял запись. Если в базовой таблице присутствует поле EDITORKEY, то значение берется из этого поля, иначе значение берется из поля EDITORKEY таблицы GD_RUID, если в ней содержится РУИД

текущей записи. Иначе возвращается ключ контакта текущего пользователя. Только для чтения.

- **EditorName**: String – возвращает наименование контактного лица, изменявшего запись, используя идентификатор, возвращаемый свойством EditorKey. Только для чтения.
- **ExtraConditions**: TStrings – список дополнительных условий, добавляемых в SELECT-запрос бизнес-объекта. Как мы уже знаем, некоторые условия добавляются в запрос через подмножества (свойство Subset). Однако предусмотреть все ситуации через подмножества нереально. Обновление данного списка вызывает переформирование всех запросов бизнес-объекта. При этом если до изменения списка бизнес-объект был открыт, то после переформирования запросов он открывается. Поэтому, если вам необходимо добавить несколько условий, сначала закройте бизнес-объект, а после добавления условий, откройте. Иначе на каждое добавление условия бизнес-объект будет закрываться-открываться.

Например, в запрос бизнес-объекта, представляющего товары, необходимо добавить условие, которое будет выбирать товар по штрих-коду.

```
Dim gdcGood, Creator
Set Creator = new TCreator
Set gdcGood = Creator.GetObject(nil, "TgdcGood", "")
call gdcGood.ExtraConditions.Add("z.barcode = :barcode")
gdcGood.ParamByName("barcode").AsString = "154002545"
gdcGood.Open
```

В результате SELECT-запрос для бизнес-объекта Товар будет выглядеть следующим образом:

```
SELECT
  z.id, z.groupkey, z.name, z.alias, z.shortname,
  z.description, z.barcode, z.valuekey, z.tnvdkey,
  z.isassembly, z.reserved, z.discipline, z.disabled,
  z.editiondate, z.editorkey, z.afull, z.achag, z.aview,
  t.name as tnvd, v.name as valuename
FROM
  gd_goodgroup gg
  JOIN gd_good z ON z.groupkey  =  gg.id
  LEFT JOIN gd_value v ON v.id  =  z.valuekey
  LEFT JOIN gd_tnvd t ON t.id  =  z.tnvdkey
WHERE
  z.barcode  =  :barcode
```

- **FieldCount**: Integer – количество полей, возвращаемых бизнес-объектом. Только для чтения.
- **Fields**: TFields – список полей бизнес-объекта. Только для чтения. Обращение к каждому полю возможно через его порядковый номер в списке (совпадает со следованием полей в запросе).
- **FieldsCallDoChange**: TStringList – список полей, при изменении которых вызывается метод DoFieldChange. Если список пустой, то метод DoFieldChange вызывается при изменении любого поля объекта, что значительно может снизить скорость обработки данных. Не изменяйте это свойство просто в режиме работы с бизнес-объектом! Правильнее всего будет задать его значение один раз после создания полей объекта (в методе CreateFields).

Например, в для бизнес-объекта Товар будем обрабатывать только изменение поля штрих-код. Для этого перекроем метод CreateFields.

```
option explicit
sub TgdcGoodCreateFields(Self)
  call Inherited(Self, "CreateFields", Array(Self))
  call Self.FieldsCallDoChange.Add("barcode")
end sub
```

В случае перекрытия методов параметр Self означает объект, метод которого перекрывается. В нашем примере это будет экземпляр класса TgdcGood (т.е. бизнес-объект Товар).

- **ID**: Integer – идентификатор текущей записи бизнес-объекта. Доступно для чтения и записи. При присваивании данного свойства в режиме просмотра бизнес-объект пытается позиционировать курсор на запись с данным идентификатором. Если запись с подобным идентификатором не найдена, то возможно два варианта поведения: если бизнес-объект имеет подмножество ByID, то он просто переоткроется и будет пустым, иначе пользователю будет выдано исключение с сообщением о некорректном идентификаторе. При присваивании данного свойства в режиме редактирования бизнес-объект попытается изменить идентификатор текущей записи. **Не меняйте идентификатор записи вручную!**
- **InsertSQL**: TStrings – запрос на вставку записи в базовую таблицу. Формируется автоматически. Может быть перекрыт в методе CustomInsert. Только для чтения. Данное свойство доступно и для записи, но при изменении подмножества объекта все SQL-запросы будут перестроены.
- **MasterField**: String – название ключевого поля мастер-объекта. Используется только при организации связи master-detail или при установке режима множества. Доступно для чтения и записи.
- **MasterSource**: TDataSource – источник записей мастер-объекта. Доступно для чтения и записи.
- **ModifySQL**: TStrings – запрос на изменение записи в базовой таблице. Формируется автоматически. Может быть перекрыт в методе CustomModify. Только для чтения. Данное свойство доступно и для записи, но при изменении подмножества объекта все SQL-запросы будут перестроены.
- **Name**: String – название компоненты, представляющей текущий бизнес-объект. Доступно для чтения и записи.
- **Objects**(Name: String): IDispatch – список объектных переменных, связанных с текущим бизнес-объектом. Обращение к переменной идет по ее наименованию. Добавить переменную в список можно используя метод AddObjectItem. Доступно для чтения и записи.
- **Owner**: TComponent – владелец бизнес-объекта. Только для чтения.
- **OwnerForm**: TForm – форма-владелец бизнес-объекта (т.е. форма, на которой лежит бизнес-объект). Только для чтения.
- **Params**: TParams – список параметров, присутствующих в SELECT-запросе бизнес-объекта. Обращение к параметрам идет по их порядковому номеру.
- **ParentForm**: TWinControl – аналогично OwnerForm.
- **Plan**: String – возвращает план выполнения SELECT-запроса. Только для чтения.
- **QueryFilter**: TQueryFilterGDC – возвращает ссылку на объект фильтрации для текущего бизнес-объекта. Объект фильтрации автоматически применяет к бизнес-объекту последний использованный фильтр, если свойство QueryFiltered установлено в Истину. Только для чтения. Про фильтры читайте в главе 9.
- **QueryFiltered**: Boolean – указывает, устанавливать ли после открытия бизнес-объекта последний использовавшийся фильтр. Доступно для чтения и записи.
- **ReadTransaction**: TIBTransaction – транзакция, используемая бизнес-объектом для чтения данных из базы. Для всех бизнес-объектов по умолчанию присваивается одна транзакция на чтение (она доступна через свойство глобального объекта gdcBaseManager.ReadTransaction). Доступно для чтения и записи.
- **RecordCount**: Integer – количество записей, считанных из базы. Для того чтобы получить реальное количество записей, которое может быть представлено через SELECT-запрос бизнес-объекта, вам необходимо сначала спозиционировать курсор на последнюю запись (метод Last), что повлечет за собой передачу всех записей на клиента. Если вы просто открыли бизнес-объект внутри макроса (метод Open), то данное свойство для непустого бизнес-объекта всегда будет возвращать единицу, для пустого ноль. Только для чтения.
- **RecordSize**: Integer – возвращает размер записи бизнес-объекта. Только для чтения.

- **RefreshMaster**: Boolean – если установлено в Истину, то после сохранения записи обновляет также запись в мастер-объекте (если такой имеется). По умолчанию отключено. Доступно для чтения и записи.
- **RefreshSQL**: TStrings – запрос для обновления записи в бизнес-объекте после сохранения изменений. Формируется автоматически. Может быть перекрыт для класса в методах GetSelectClause, GetFromClause, GetGroupClause, или для конкретного объекта, используя события OnGetSelectClause, OnGetFromClause, OnGetGroupClause. Только для чтения. Данное свойство доступно и для записи, но при изменении подмножества объекта все SQL-запросы будут перестроены.
- **SelectedID**: TgdKeyArray – сортированный список ключей отмеченных записей.
- **SelectSQL**: TStrings – запрос для выборки записей бизнес-объекта. Формируется автоматически. Может быть перекрыт для класса в методах GetSelectClause, GetFromClause, GetGroupClause, GetOrderClause, GetWhereClause, или для конкретного объекта, используя события OnGetSelectClause, OnGetFromClause, OnGetGroupClause, OnGetOrderClause, OnGetWhereClause. Только для чтения. Данное свойство доступно и для записи, но при изменении подмножества объекта все SQL-запросы будут перестроены.
- **SetTable**: String – название таблицы-множества. Используется только в режиме множеств.
- **SQLSetup**: TatSQLSetup – объект, который разбирает заданные пользователем (или сформированные автоматически) запросы, добавляет в них поля атрибуты, таблицы, на которые ссылаются атрибуты-ссылки, поля для отображения и расширенного отображения этих таблиц (читайте в главе 8 про таблицы). Используя данный объект можно заблокировать добавление полей-атрибутов для конкретной таблицы, входящей в первоначальный запрос.

Для каждого бизнс-объекта при необходимости формируется список «Игнорирований» (свойство SQLSetup.Ignores). Причем, игнорироваться (т.е. поля-атрибуты этой таблицы не добавляются в SELECT-часть) может как таблица целиком, так и только определенные атрибуты этой таблицы. По умолчанию при добавлении нового объекта Игнорирования пропускаются все поля-атрибуты (свойство объекта TatIgnore.IgnoryType = 0). Однако вы можете указать, что хотите игнорировать только атрибуты ссылки (свойство объекта TatIgnore.IgnoryType = 1). Таблицы, которые вы хотите проигнорировать нужно задать до открытия бизнес-объекта (например, в методе GetFromClause).

Например, для бизнес-объекта Подразделение проигнорируем все поля атрибуты для таблицы GD_CONTACT с алиасом CLB.

```
option explicit
function TgdcDepartmentGetFromClause(Self, ARefresh)
  TgdcDepartmentGetFromClause = _
    Inherited(Self, "GetFromClause", Array(Self, ARefresh))
  Dim Ig
  set Ig = Self.SQLSetup.Ignores.Add
  Ig.AliasName = "CLB"
end function
```

- **State**: Integer – состояние бизнес-объекта. Может принимать следующие значения: Не активный = 0, Режим просмотра = 1, Режим редактирования = 2, Режим вставки = 3, 4 (не используется), При вычислении значений калькулируемых полей = 5. Только для чтения.
- **SubSet**: String – хранит подмножества, установленные для бизнес-объекта. Если данное свойство пустое, то значит бизнес-объект вернет все записи. Бизнес-объект может использовать совокупность подмножеств (задается через запятую). Про сабсеты читайте в главе 6.3. Доступно для чтения и записи.
- **SubSetCount**: Integer – количество подмножеств, заданных для текущего бизнес-объекта. Только для чтения.
- **SubSets**(Index: Integer): String – обращение к списку заданных сабсетов по индексу. Только для чтения.

- **SubType**: String – текущий подтип бизнес-объекта. Может быть присвоено только один раз. При переприсвоении подтипа пользователь получит ошибку. Доступно для чтения и записи.
- **Transaction**: TIBTransaction – транзакция на изменение. Про транзакции читайте в главе 6.3.
- **Variables**(Name: String): Variant – список простых переменных, связанных с ткущим бизнес-объектом. Обращение к переменной идет по ее наименованию. Добавить переменную в список можно используя метод AddVariableItem. Доступно для чтения и записи.

Методы бизнес-объектов

- **AddObjectItem**(Name: String) – добавляет в список новую объектную переменную с именем Name, инициализируя ее nil.
- **AddSubSet**(ASubSet: String) – добавляет подмножество с именем ASubSet в список активных подмножеств бизнес-объекта. При этом добавляемое подмножество должно быть заявлено ранее, т.е. должно пройти проверку в методе CheckSubSet. Иначе AddSubSet кинет ошибку, что подмножество ASubSet не определено для данного класса.
- **AddToSelectedArray**(ASelectedID: TgdKeyArray) – добавляет элементы из сортированного списка ASelectedID в список отмеченных записей (свойство SelectedID) бизнес-объекта.
- **AddToSelectedBookmark**(BL: TBookmarkList) – добавляет идентификаторы выделенных в гриде записей в список отмеченных записей (свойство SelectedID) бизнес-объекта.
- **AddToSelectedID**(ID: Integer) – добавляет идентификатор ID в список отмеченных записей (свойство SelectedID) бизнес-объекта.
- **AddVariableItem**(Name: String) – добавляет в список новую переменную простого типа с именем Name, инициализируя ее пустым значением.
- **Append** – устанавливает курсор на последнюю запись бизнес-объекта и добавляет новую запись. Не рекомендуется использовать в бизнес-объектах, возвращающих большие объемы данных, т.к. установка курсора на последнюю запись означает перекачку записей с сервера на клиента.
- **BeforeDestruction**(Self: TgdcBase) – вызывается перед уничтожением бизнес-объекта. Self – бизнес-объект, вызвавший метод. Доступен только для перекрытия.
- **Cancel** – отменяет последнюю операцию изменения. Может быть применимо, если бизнес-объект находится в состоянии вставки новой записи или редактирования уже существующей (смотрите свойство State).
- **CheckBrowseMode** – проверяет, находится ли бизнес-объект в состоянии просмотра записей (State = 1). Если нет, то переводит объект в это состояние. Это значит, что если вы редактировали запись и после этого вызвали метод CheckBrowseMode, то запись будет сохранена.
- **CheckCurrentRecord** – проверяет, открыт ли бизнес-объект и содержит ли он записи. Если нет, то кидает исключение.
- **CheckSubSet**(ASubSetStr: String): Boolean – перекрываемый метод. Проверяет, заявлено ли подмножество ASubSetStr для текущего класса. Все подмножества, которые вы создали сами, необходимо прописать в данном методе. Метод CheckSubSet используется при изменении активного подмножества бизнес-объекта.
- **CheckTheSameStatement**(Self: TgdcBase): String – возвращает запрос, который используется для определения записи бизнес-объекта по некоторым уникальным признакам. Этот необходимо при загрузке данных из потока для определения существует ли загружаемая запись в базе. Self – бизнес-объект, вызвавший метод, который в данный момент загружается из потока. Доступен только для перекрытия.
- **ChooseItems** – используется для вывода формы выбора бизнес-объекта с заданным классом и подтипом. Если пользователь подтвердил выбор записей, то метод вернет Истина, в противном случае (при отмене операции выбора) – Ложь. Подробнее читайте в пункте 6.4.2

- **ChooseItemsSelf** - используется для вывода формы выбора бизнес-объекта вызвавшего метод. При этом отмеченные записи попадают в свойство SelectedID объекта. Если перед вызовом SelectedID был не пустой, то на форме выбора будут отмечены записи, идентификаторы которых попали в SelectedID. Если пользователь подтвердил выбор записей, то метод вернет Истина, в противном случае (при отмене операции выбора) – Ложь. Подробнее читайте в пункте 6.4.2
- **ChooseOrderItems** – аналогичен методу ChooseItems, за исключением того, что возвращает отмеченные записи в том порядке, в котором они были отмечены. Подробнее читайте в пункте 6.4.2
- **ChooseOrderItemsSelf** – аналогичен методу ChooseItemsSelf, за исключением того, что возвращает отмеченные записи в том порядке, в котором они были отмечены. Подробнее читайте в пункте 6.4.2
- **ClassName**: String – возвращает название класса бизнес-объекта.
- **ClassParent**: String – возвращает название класса, от которого наследован класс бизнес-объекта.
- **ClearFields** – очищает содержимое полей текущей записи бизнес-объекта. При этом бизнес-объект должен быть в состоянии вставки или редактирования записи (State = 2 или 3).
- **ClearSubSets** – удаляет все активные подмножества бизнес-объекта. Данный метод делает активным единственное подмножество All, оповещая таким образом бизнес-объект, что нужно отобразить все записи. Если перед вызовом метода бизнес-объект был открыт, то после отчистки активных подмножеств, он переоткроется.
- **Close** – закрывает бизнес-объект.
- **CloseOpen** – переоткрывает объект с сохранением текущей позиции курсора. Нередко подобная операция необходима, чтобы синхронизировать содержимое бизнес-объекта с базой.
- **ControlsDisabled**: Boolean – указывает, что визуальные элементы управления, связанные с бизнес-объектом, не обновляются при изменении содержимого бизнес-объекта. Отключить обновление визуальных элементов управления можно, используя метод DisableControls, включить – EnableControls.
- **Copy**(AFields: String; AValues: Variant; ACopyDetail: Boolean; APost: Boolean): Boolean – создает копию с текущей записи бизнес-объекта и устанвливает на нее курсор. При этом параметр ***AFields*** означает список полей, разделенных точкой с запятой, которым необходимо передать значения, отличные от значений этих полей в копируемой записи; параметр ***AValues*** - список значений, которые необходимо присвоить полям из параметра AFields; ***ACopyDetail*** - при значении истина, производит копирование детальных объектов; ***APost*** - указывает, сохранить скопированную запись или оставить ее в состоянии вставки.
- **CopyDialog** – копирует текущую запись и выводит диалог для ее редактирования. Подробнее читайте в пункте 6.4.3. Является перекрываемым методом.
- **CreateDialog**(ADlgClassName: String): Boolean – переводит бизнес-объект в состояние вставки и выводит на экран модальное окно для редактирования содержимого объекта. Параметр ADlgClassName содержит наименование класса окна для редактирования. Если передана пустая строка, используется окно, заданное для текущего объекта по умолчанию (см. свойство CreateDialogForm). Подробнее читайте в пункте 6.4.3. Является перекрываемым методом.
- **CreateDialogForm**: TCreateableForm – создает форму диалога, заданную для класса по умолчанию. Является перекрываемым методом.
- **CreateDialogSubType**(AClassName: String; ASubType: String): Boolean – добавляет запись заданного класса и подтипа и выводит окно для ее редактирования. Используется чаще всего при добавлении новых записей в базовый бизнес-объект. Подробнее читайте в пункте 6.4.3
- **CreateFields**(Self: TgdcBase) – вызывается после создания полей бизнес-объекта. Поля создаются в момент первого открытия бизнес-объекта. Self – бизнес-объект, вызвавший метод. Доступен только для перекрытия.

- **CreateNext** – сохраняет текущую запись и вставляет новую. При этом, если редактирование шло через диалоговое окно и у вас включена опция сохранять значения полей в диалоговых окнах, то новой вставленной записи будут присвоены значения полей предыдущей записи.
- **CreateViewForm**(AnOwner: TComponent; AClassName: String; ASubType: String): TForm – создает форму просмотра с переданным классом (AClassName) и сабтайпом (ASubType). Владельцем формы является AnOwner. Данный метод проверяет, была ли уже создана необходимая форма. Если такая форма действительно уже создана, то метод просто вернет на нее указатель. В обратном случае, он сначала создаст форму, а затем вернет на нее указатель. Подробнее читайте в пункте 6.4.1
- **CreateViewFormNewInstance**(AnOwner: TComponent; AClassName: String; ASubType: String; ANewInstance: Boolean): TForm – данный метод аналогичен CreateViewForm, за исключением того, что он позволяет создавать несколько форм одного класса и сабтайпа. Для этого необходимо параметр ANewInstance установить в истину. Подробнее читайте в пункте 6.4.1
- **CustomDelete**(Self: TgdcBase; Buff: Pointer) – посылает запрос на удаление записи бизнес-объекта на сервер. Self – бизнес-объект, вызвавший метод, Buff – ссылка на область памяти, которая хранит удаляемую запись. Доступен только для перекрытия. Перекрывается обычно для выполнения дополнительных запросов. Транзакция на изменение в данном методе открыта.
- **CustomInsert**(Self: TgdcBase; Buff: Pointer) – посылает запрос на вставку новой записи бизнес-объекта на сервер. Self – бизнес-объект, вызвавший метод, Buff – ссылка на область памяти, которая хранит добавляемую запись. Доступен только для перекрытия. Перекрывается обычно для выполнения дополнительных запросов. Транзакция на изменение в данном методе открыта.
- **CustomModify**(Self: TgdcBase; Buff: Pointer) – посылает запрос на изменение записи бизнес-объекта на сервер. Self – бизнес-объект, вызвавший метод, Buff – ссылка на область памяти, которая хранит изменяемую запись. Доступен только для перекрытия. Перекрывается обычно для выполнения дополнительных запросов. Транзакция на изменение в данном методе открыта.
- **Delete** – удаляет текущую запись бизнес-объекта. Если бизнес-объект неактивен или пустой генерирует исключение.
- **DeleteMultiple**(BL: TBookmarkList): Boolean – удаляет отмеченные в гриде (параметр BL) записи бизнес-объекта. Перед удалением запрашивает у пользователя подтверждение. Если BL не задан или не отмечено ни одной записи пытается удалить текущую запись (т.е. запись на которой стоит курсор). Если на удаляемую запись имеются ссылки и бизнес-объект лежит на форме, то выводит окно с перечислением записей, которые зависят от удаляемой.
- **DeleteSubSet**(Index: Integer) – удаляет из списка активное подмножество по индексу Index.
- **DestroyObject** – уничтожает бизнес-объект (высвобождает занимаемую им память).
- **DisableControls** – отключает обновление визуальных элементов управления, связанных с бизнес-объектом. Данный метод полезен, если вы производите много операций обновления над бизнес-объектом, а отобразить хотите лишь конечный результат. Однако не желательно использовать данный метод, если у ваш бизнес-объект участвует в связке master-detail.
- **DoAfterCancel**(Self: TgdcBase) – вызывается после отмены изменений. Self – бизнес-объект, вызвавший метод. Доступен только для перекрытия.
- **DoAfterCustomProcess**(Self: TgdcBase; Buff: Pointer; Process: Integer)
- **DoAfterDelete**(Self: TgdcBase) – вызывается после удаления записи. Self – бизнес-объект, вызвавший метод. Доступен только для перекрытия.
- **DoAfterEdit**(Self: TgdcBase) – вызывается после перевода бизнес-объекта в состояние редактирования. Self – бизнес-объект, вызвавший метод. Доступен только для перекрытия.
- **DoAfterExtraChanged**(Self: TgdcBase; Sender: TStrings) – вызывается после изменения списка дополнительных условий (свойство объекта ExtraConditions). Self – бизнес-объект, вызвавший метод, Sender – изменяемый список условий. Доступен только для перекрытия.

- **DoAfterInsert**(Self: TgdcBase) – вызывается после перевода бизнес-объекта в состояние втсавки. Self – бизнес-объект, вызвавший метод. Доступен только для перекрытия.
- **DoAfterOpen**(Self: TgdcBase) – вызывается после открытия бизнес-объекта. Self – бизнес-объект, вызвавший метод. Доступен только для перекрытия.
- **DoAfterPost**(Self: TgdcBase) – вызывается после сохранения изменений. Self – бизнес-объект, вызвавший метод. Доступен только для перекрытия.
- **DoAfterShowDialog**(Self: TgdcBase, DlgForm: TCreateableForm; IsOk: Boolean) – запускается после закрытия диалогового окна редактирования бизнес-объекта. Является перекрываемым методом. Параметр Self - бизнес-объект, вызвавший метод, DlgForm – диалоговое окно, IsOk – говорит о том, как было закрыто окно (с сохранением изменений или с их отменой).
- **DoAfterTransactionEnd**(Self: TgdcBase; Field: TIBBase) – вызывается после завершения транзакции на изменение. Транзакция может быть завершена как успешно, так и не успешно. Self – бизнес-объект, вызвавший метод, Field – база данных, с которой работает бизнес-объект. Доступен только для перекрытия.
- **DoBeforeClose**(Self: TgdcBase) – вызывается перед закрытием бизнес-объекта. Self – бизнес-объект, вызвавший метод. Доступен только для перекрытия.
- **DoBeforeDelete**(Self: TgdcBase) – вызывается перед удалением записи. Self – бизнес-объект, вызвавший метод. Доступен только для перекрытия.
- **DoBeforeEdit**(Self: TgdcBase) – вызывается перед переводом бизнес-объекта в состояние редактирования. Self – бизнес-объект, вызвавший метод. Доступен только для перекрытия.
- **DoBeforeInsert**(Self: TgdcBase) – вызывается перед переводом бизнес-объекта в состояние вставки. Self – бизнес-объект, вызвавший метод. Доступен только для перекрытия.
- **DoBeforeOpen**(Self: TgdcBase) – вызывается перед открытием бизнес-объекта. Self – бизнес-объект, вызвавший метод. Доступен только для перекрытия.
- **DoBeforePost**(Self: TgdcBase) – вызывается перед сохранением изменений. Self – бизнес-объект, вызвавший метод. Доступен только для перекрытия.
- **DoBeforeShowDialog**(Self: TgdcBase, DlgForm: TCreateableForm) – запускается перед появлением диалогового окна редактирования бизнес-объекта. Является перекрываемым методом. Параметр Self – бизнес-объект, вызвавший метод, DlgForm – диалоговое окно.
- **DoFieldChange**(Self: TgdcBase; Field: TField) – вызывается при изменении содержимого полей бизнес-объекта. Список полей, на изменение которых выполняется метод DoFieldChange, задается свойством FieldsCallDoChange. Self – бизнес-объект, вызвавший метод, Field – изменяемое поле. Доступен только для перекрытия. Перекрытие этого метода следует использовать только в случае задания постоянных взаимозависимостей между полями.
- **DoOnFilterChanged**(Self: TgdcBase; Sender: TQueryFilterGDC; AnCurrentFilter: Integer) – вызывается при изменении фильтра. Self – бизнес-объект, вызвавший метод, Sender – объект фильтрации, связанный с бизнес-объектом, AnCurrentFilter – идентификатор устанавливаемого фильтра. Если фильтр снимается, то AnCurrentFilter будет равен нулю. Доступен только для перекрытия.
- **DoOnReportClick**(Self: TgdcBase; Sender: TMenuItem) – вызывается после закрытия построенного отчета. Self – бизнес-объект, вызвавший метод, Sender – пункт меню, по выбору которого был построен отчет. Доступен только для перекрытия.
- **DoOnReportListClick**(Self: TgdcBase; Sender: TMenuItem) – вызывается при выборе любого пункта меню отчетов. Self – бизнес-объект, вызвавший метод, Sender – выбранный пункт меню. Доступен только для перекрытия.
- **Edit** – переводит бизнес-объект в состояние редактирования текущей записи или вставки новой (если бизнес-объект пустой). Если бизнес-объект был неактивным, то генерирует исключение. Если до вызова метода бизнес-объект уже был в состоянии редактирования или вставки, то сначала будет вызван метод Post, а только затем Edit.

- **EditDialog**(ADlgClassName: String): Boolean – используется для вывода диалога для редактирования текущей записи. Параметр ADlgClassName хранит наименование класса диалога для редактирования записи. По умолчанию передается пустая строка, что означает использование диалога, заданного изначально для текущего объекта. Является перекрываемым методом. Подробнее читаете в пункте 6.4.3.
- **EditMultiple**(BL: TbookmarkList; ADlgClassName: String): Boolean - одновременное редактирование некоторого множества записей. Параметр BL означает список отмеченных записей для редактирования, ADlgClassName - наименование класса диалоговой формы. Если передана пустая строка, то берется диалог, заданный по умолчанию. Если BL = nil, на редактирование пойдет текущая запись, иначе – текущая. После редактирования записи (при закрытии диалога) внесенные изменения накладываются на оставшиеся выделенные записи. Редактирование нескольких записей одновременно накладывает некоторые ограничения: редактировать можно только записи одного класса и подтипа. При пустом бизнес-объекте будет сгенерировано исключение.
- **EnableControls** – делает визуальные элементы управления, связанные с бизнес-объектом, обновляемыми.
- **FetchAll** – выбирает все записи на клиента.
- **FieldByName**(FieldName: String): TField – возвращает поле бизнес-объекта по его наименованию FieldName. Если такого поля нет, то генерирует исключение.
- **FieldNameByAliasName**(AnAliasName: String): String - для заданного алиаса AnAliasName возвращает физическое имя поля в таблице.
- **FindField**(FieldName: String): TField – ищет поле бизнес-объекта по его наименованию FieldName. Если поле не найдено, возвращает nil.
- **FindFieldByRelation**(ARelationName: String; AFieldName: String) – находит первое поле с именем AFieldName таблицы ARelationName, входящей в запрос. Внимание: в запросе может быть несколько одинаковых таблиц. Вернет только первое найденное поле.
- **First** – устанавливает курсор на первую запись бизнес-объекта. Если объект неактивен, сгенерирует исключение. Если перед вызовом метода какая-либо запись изменялась, то как и все методы, перемещающие курсор, First сохранит изменявшуюся запись бизнес-объекта.
- **GetCurrRecordClass**(AgdcClassName: String; ASubType: String) – возвращает через параметры AgdcClassName и ASubType класс и подтип текущей записи бизнес-объекта. Как мы уже знаем, можно работать с базовыми бизнес-объектами, которые отображают записи всех наследников. Примером тому может служить адресная книга, в которой одним списком представлены организации, контакты и т.д. Данный метод используется для уточнения класса и подтипа текущей записи для того, чтобы корректно обработать ее изменения, вызвать диалоги непосредственно относящиеся к возвращаемым им классу и подтипу. При этом класс и подтип должны быть «зашиты» в самой записи в виде некого признака. Например, в той же адресной книге (таблица GD_CONTACT) есть поле Тип контакта (CONTACTTYPE), которое и репрезентирует класс записи.
- **GetCurrRecordSubType**(Self: TgdcBase): String – используется в методе GetCurrRecordClass для определения пользовательского подтипа. Self – бизнес-объект, вызвавший метод. Доступен только для перекрытия. Перекрывать данный метод следует только в том случае, если вы создавали новые подтипы для стандартных классов (например, новый подтип товара). Для всех пользовательских классов подтип определяется автоматически. Определение подтипа должно быть «зашито» в записи бизнес-объекта в виде некого признака. Возвращает подтип в виде строки.
- **GetDialogDefaultsFields**(Self: TgdcBase): String – возвращает список полей, значения которых сохраняются и автоматически подставляются в поля в диалоговом окне при вводе новой записи. Имена полей разделены точкой с запятой. В конце не пустого списка точка с запятой. Пустой список вернет пустую строку. Доступен только для перекрытия.
- **GetDisplayName**(ASubType: String): String – возвращает локализованное название бизнес-объекта по его подтипу ASubType.

- **GetDistinctColumnValues**(AFieldName: String; S: TStrings; DoSort: Boolean) - метод заполняет переданный список S уникальными значениями из заданного поля AFieldName базовой таблицы. Параметр DoSort означает сортировать список или нет.
- **GetFieldList**(List: TList; FieldName: String) – возвращает список List указанных полей FieldName. FieldName содержит названия полей через точку с запятой. List будет содержать найденные поля –объекты (класс TField).
- **GetFieldNames**(List: TStrings) – возвращает список (List) наименований всех полей бизнес-объекта.
- **GetFieldValueForID**(AnID: Integer; AFieldName: String): Variant – возвращает значение поля AFieldName для записи с идентификатором AnID. При этом, если поля с таким названием не существует, будет сгенерировано исключение. Если нет записи с заданным идентификатором, то вернет пустое значение.
- **GetFromClause**(Self: TgdcBase; ARefresh: Boolean): String – возвращает FROM-часть SELECT-запроса бизнес-объекта. Self – бизнес-объект, вызвавший метод, ARefresh – указывает, что вызвало метод: формирование запроса на выборку или запроса на обновление записи. Доступен только для перекрытия.
- **GetGroupClause**(Self: TgdcBase): String – возвращает GROUP BY-часть SELECT-запроса бизнес-объекта. Self – бизнес-объект, вызвавший метод. Доступен только для перекрытия.
- **GetKeyField**(ASubType: String): String – возвращает название ключевого поля бизнес-объекта для заданного подтипа. Подробнее читайте в главе 6.3
- **GetListField**(ASubType: String): String – возвращает отображаемое поле бизнес-объекта для заданного подтипа. Подробнее читайте в главе 6.3
- **GetListFieldExtended**(ASubType: String): String – возвращает список дополнительных отображаемых полей бизнес объекта для заданного подтипа. Подробнее читайте в главе 6.3
- **GetListNameByID**(AnID: Integer): String – возвращает значение поля отображения по идентификатору записи AnID. Аналогичен методу GetFieldValueForID.
- **GetListTable**(ASubType: String): String – возвращает название базовой таблицы бизнес-объекта по подтипу. Подробнее читайте в главе 6.3
- **GetListTableAlias**: String – возвращает алиас базовой таблицы бизнес-объекта. Подробнее читайте в главе 6.3
- **GetNextID**(Increment: Boolean): Integer – возвращает следующий (если Increment равен Истине) или последний сгенерированный ключ. Генерация ключей происходит при помощи генератора GD_G_UNIQUE. Подробнее читайте в главе 6.2
- **GetNotCopyField**: String – возвращает строку, в которой через запятую перечислены поля, не использующиеся при копировании записей (метод Copy). Является перекрываемым методом. Чаще всего в копировании не нуждаются поля, значения которым присваиваются на _DoOnNewRecord, например, идентификатор записи.
- **GetOrderClause**(Self: TgdcBase): String – возвращает ORDER BY-часть SELECT-запроса бизнес-объекта. Self – бизнес-объект, вызвавший метод. Доступен только для перекрытия.
- **GetRestrictCondition**(ATableName: String; ASubType: String): String – возвращает условия, использующиеся для фильтрации записей главной таблицы бизнес-объекта ATableName с подтипом ASubType. Применяется при подстановке класса бизнес-объекта в свойство gdClassName выпадающего списка.
- **GetSelectClause**(Self: TgdcBase): String – возвращает SELECT-часть SELECT-запроса бизнес-объекта. Self – бизнес-объект, вызвавший метод. Доступен только для перекрытия.
- **GetSubSetList**: String – возвращает список допустимых для бизнес-объекта подмножеств в строке через точку с запятой. В этом списке не присутствуют подмножества, созданные настройщиком платформы Гедымин!

- **GetSubTypeList**(SubTypeList: TStrings): Boolean – возвращает список допустимых для бизнес-объекта подтипов (SubTypeList) в формате Локализованное имя подтипа = Подтип. Возвращает истину, если есть подтипы, Ложь – в обратном случае.
- **GetViewFormClassName**(ASubType: String): String - возвращает название класса формы просмотра для текущего объекта по указанному сабтайпу (ASubType). Подробнее читайте в пункте 6.4.1
- **GetWhereClause**(Self: TgdcBase): String – возвращает WHERE-часть SELECT-запроса бизнес-объекта. Self – бизнес-объект, вызвавший метод. Доступен только для перекрытия.
- **HasSubSet**(ASubSet: String): Boolean – возвращает Истину, если для бизнес-объекта задано указанное подмножество ASubSet (т.е. ASubSet является активным).
- **InheritsFrom**(AClassName: String): Boolean – возвращает Истину, если объект является наследником класса AClassName.
- **Insert** – переводит объект в состояние вставки. Если объект закрыт, генерирует исключение.
- **IsEmpty**: Boolean – возвращает Истину, если объект пустой (RecordCount = 0).
- **Last** – передвигает курсор на последнюю запись бизнес-объекта. Если объект закрыт, генерирует исключение.
- **LoadFromFile**(AFileName: String) – загружает данные бизнес-объекта из файла AFileName. Данный файл должен быть сформирован средствами Гедымина. (Читайте про сохранение объектов в поток).
- **Locate**(KeyFields: String; KeyValues: Variant; Options: String): Boolean – позиционирует курсор на запись с указанными параметрами. KeyFields – перечисление названий полей через запятую, KeyValues – массив с искомыми значениями полей KeyFields, Options – опции поиска через запятую. Options может содержать следующие опции: CASEINSENSITIVE – поиск без учета регистра, PARTIALKEY – поиск по частичному значению, т.е. сравнивается не слово целиком, а начало содержимого поля с указанным значением. Использование Locate вытягивает записи на клиента, пока не найдет необходимую. При удачном завершении поиска возвращает Истину и устанавливает курсор на найденную запись. Если искомая запись не найдена, то курсор не изменяет положения, Locate возвращает Ложь.

Например, необходимо найти товар, начинающийся со слова «Колбаса» со ссылкой на единицу измерения, равной 147027774.

```
Dim gdcGood, Creator
Set Creator = new TCreator
Set gdcGood = Creator.GetObject(nil, "TgdcGood", "")
gdcGood.Open
MsgBox gdcGood.Locate("name;valuekey", _
  Array("колбаса", 147027774), "CASEINSENSITIVE,PARTIALKEY")
```

- **MoveBy**(Distance: Integer): Integer – перемещает курсор на количество записей Distance. При этом, если параметр Distance отрицательный, то курсор возвращается на указанное количество записей, если положительный, то курсор перемещается вперед на указанное количество записей. Возвращает реальное количество записей, на которое был перемещен курсор.
- **Next** – перемещает курсор на следующую запись. Если курсор стоял на последней записи, то устанавливает свойство EOF в Истину. если бизнес-объект не активен, генерирует исключение.
- **ObjectExists**(Name: String): Boolean – возвращает Истина, если в списке объектных переменных, связанных с текущим бизнес-объектом, присутствует переменная с именем Name, Ложь – в обратном случае.
- **Open** – открывает бизнес-объект.

- **ParamByName**(Idx: String): TIBXSQLVAR – возвращает параметр из SELECT-запроса бизнес-объекта по его имени Idx. Если параметр с таким наименованием не найден, то генерирует исключение.
- **PopupFilterMenu**(X: Integer; Y: Integer) – выводит меню работы с фильтрами в точке экрана с координатами X и Y. Если X и Y равны -1, то меню будет выведено в точке положения указателя мыши.
- **PopupReportMenu**(X: Integer; Y: Integer) – выводит меню работы с отчетами в точке экрана с координатами X и Y. Если X и Y равны -1, то меню будет выведено в точке положения указателя мыши.
- **Post** – сохраняет изменения текущей записи бизнес-объекта и переводит объект в состояние просмотра записей. При этом бизнес-объект должен быть активен и находится в состоянии вставки или изменения записи. Иначе метод Post сгенерирует исключение.
- **Prepare** – подготавливает запросы бизнес-объекта к выполнению. Данный метод вызывается каждый раз после изменения запроса перед его выполнением или перед присваиванием значений параметров.
- **Prior** – перемещает курсор на предыдущую запись бизнес-объекта. Если курсор находился на первой записи, то устанавливает свойство BOF в Истину. Если бизнес-объект не активен, генерирует исключение.
- **Reduction**(BL: TBookmarkList): Boolean – вызывает диалог для слияния записей бизнес-объекта. Параметр BL – хранит список маркеров отмеченных записей, которые должны сливаться.
- **Refresh** – обновляет данные текущей записи на клиентской части, выполняя запрос RefreshSQL.
- **RelationByAliasName**(AnAliasName: String): String – возвращает реальное наименование таблицы по ее алиасу в запросе.
- **RemoveFromSelectedArray**(BL: TBookmarkList) – удаляет идентификаторы выделенных в гриде записей из списка отмеченных записей (свойство SelectedID) бизнес-объекта.
- **RemoveFromSelectedID**(ID: Integer) – удаляет идентификатор ID из списка отмеченных записей (свойство SelectedID) бизнес-объекта
- **RemoveSubSet**(ASubSet: String) – удаляет подмножество с именем ASubSet из списка активных подмножеств бизнес-объекта.
- **SaveToFile**(AFileName: String) – сохраняет в файл с именем AFileName текущую запись бизнес-объекта.
- **SetExclude**(ReOpen: Boolean) – исключает текущую запись из множества. Используется только в режиме работы множества. Параметр ReOpen указывает переоткрыть ли бизнес-объект по завершению операции.
- **SetInclude**(AnID: Integer) – добавляет запись с идентификатором AnID в множество. Используется только в режиме работы множества.
- **SetRefreshSQLOn**(SetOn: Boolean) – устанавливает (SetOn = Истина) или сбрасывает (SetOn = ложь) запрос ResfreshSQL. Если запрос RefreshSQL не задан, то после выполнения операций изменения записи не синхронизируются с базой. Отключение запроса обновления может быть необходимо для увеличения скорости выполнения большого количества операций изменения над бизнес-объектом.
- **ShowFieldInGrid**(AField: TField): Boolean – возвращает Истину, если поле является видимым в гриде, Ложь – в обратном случае.
- **VariableExists**(Name: String): Boolean – возвращает Истина, если в списке переменных простых типов, связанных с текущим бизнес-объектом, присутствует переменная с именем Name, Ложь – в обратном случае.

- **_DoOnNewRecord**(Self: TgdcBase) – вызывается при вставке новой записи. Устанваливает значения по умолчанию полям бизнес-объекта. Self – бизнес-объект, вызвавший метод. Доступен только для перекрытия.

События бизнес-объектов

- **AfterCancel**(Dataset: TgdcBase) – вызывается после отмены изменений. DataSet – бизнес-объект, вызвавший событие.
- **AfterClose**(Dataset: TgdcBase) – вызывается после закрытия бизнес-объекта. DataSet – бизнес-объект, вызвавший событие.
- **AfterDelete**(Dataset: TgdcBase) – вызывается после удаления записи. DataSet – бизнес-объект, вызвавший событие.
- **AfterEdit**(Dataset: TgdcBase) – вызывается после перевода бизнес-объекта в состояние редактирования. DataSet – бизнес-объект, вызвавший событие.
- **AfterInsert**(Dataset: TgdcBase) – вызывается после перевода бизнес-объекта в состояние вставки новой записи. DataSet – бизнес-объект, вызвавший событие.
- **AfterOpen**(Dataset: TgdcBase) – вызывается после открытия бизнес-объекта. DataSet – бизнес-объект, вызвавший событие.
- **AfterPost**(Dataset: TgdcBase) – вызывается после сохранения изменений. DataSet – бизнес-объект, вызвавший событие.
- **AfterRefresh**(Dataset: TgdcBase) – вызывается после обновления записи. DataSet – бизнес-объект, вызвавший событие.
- **AfterScroll**(Dataset: TgdcBase) – вызывается после перемещения курсора внутри бизнес-объекта. DataSet – бизнес-объект, вызвавший событие.
- **AfterShowDialog**(Sender: TgdcBase; DlgForm: TForm; IsOk: Boolean) – вызывается после закрытия диалогового окна редактирования бизнес-объекта. Sender – бизнес-объект, вызвавший событие, DlgForm – диалоговое окно, IsOk – говорит о том, как было закрыто окно (с сохранением изменений или с их отменой).
- **BeforeClose**(Dataset: TgdcBase) – вызывается перед открытием бизнес-объекта. DataSet – бизнес-объект, вызвавший событие.BeforeDelete(Dataset: TgdcBase)
- **BeforeEdit**(Dataset: TgdcBase) – вызывается перед переводом бизнес-объекта в состояние редактирования. DataSet – бизнес-объект, вызвавший событие.
- **BeforeInsert**(Dataset: TgdcBase) – вызывается перед переводом бизнес-объекта в состояние вставки новой записи. DataSet – бизнес-объект, вызвавший событие.
- **BeforeOpen**(Dataset: TgdcBase) – вызывается перед открытием бизнес-объекта. DataSet – бизнес-объект, вызвавший событие.
- **BeforePost**(Dataset: TgdcBase) – вызывается перед сохранением изменений. DataSet – бизнес-объект, вызвавший событие.
- **BeforeScroll**(Dataset: TgdcBase) – вызывается перед перемещением указателя внутри бизнес-объекта. DataSet – бизнес-объект, вызвавший событие.
- **BeforeShowDialog**(Sender: TgdcBase; DlgForm: TForm) – вызывается перед появлением диалогового окна редактирования бизнес-объекта. Self – бизнес-объект, вызвавший метод, DlgForm – диалоговое окно.
- **OnCalcFields**(Dataset: TgdcBase) – вызывается при вычислении значения вычисляемых полей. DataSet- бизнес-объект, вызвавший событие.
- **OnDeleteError**(Dataset: TgdcBase; E: Exception; Action: Integer) – вызывается при возникновении ошибки при удалении записи. DataSet – бизнес-объект, вызвавший событие, E – исключение, Action – действие. Action может принимать следующие значения: 0 = прерывание операции и вывод ошибки на экран, 1 = прерывание операции без вывода ошибки, 2 = повтор операции.

- **OnEditError**(Dataset: TgdcBase; E: Exception; Action: Integer) – вызывается при возникновении ошибки при попытке изменения записи. DataSet – бизнес-объект, вызвавший событие, E – исключение, Action – действие. Action может принимать следующие значения: 0 = прерывание операции и вывод ошибки на экран, 1 = прерывание операции без вывода ошибки, 2 = повтор операции.
- **OnFilterChanged**(Sender: TgdcBase) – вызывается при изменении фильтра. Sender – бизнес-объект, вызвавший событие.
- **OnGetFromClause**(Sender: TgdcBase; Clause: String) – вызывается при считывании FROM-части SELECT-запроса бизнес-объекта. Sender – бизнес-объект, вызвавший событие, Clause – сформированная методом GetFromClause FROM-часть. Изменение запроса возможно через дополнение параметра Clause: Clause.Value = Clause.Value + <дополнение запроса>.
- **OnGetGroupClause**(Sender: TgdcBase; Clause: String) – вызывается при считывании GROUP BY-части SELECT-запроса бизнес-объекта. Sender – бизнес-объект, вызвавший событие, Clause – сформированная методом GetGroupClause GROUP BY-часть. Изменение запроса возможно через дополнение параметра Clause: Clause.Value = Clause.Value + <дополнение запроса>.
- **OnGetOrderClause**(Sender: TgdcBase; Clause: String) – вызывается при считывании ORDER BY-части SELECT-запроса бизнес-объекта. Sender – бизнес-объект, вызвавший событие, Clause – сформированная методом GetOrderClause ORDER BY-часть. Изменение запроса возможно через дополнение параметра Clause: Clause.Value = Clause.Value + <дополнение запроса>.
- **OnGetSelectClause**(Sender: TgdcBase; Clause: String) – вызывается при считывании SELECT-части SELECT-запроса бизнес-объекта. Sender – бизнес-объект, вызвавший событие, Clause – сформированная методом GetSelectClause SELECT-часть. Изменение запроса возможно через дополнение параметра Clause: Clause.Value = Clause.Value + <дополнение запроса>.
- **OnGetWhereClause**(Sender: TgdcBase; Clause: String) – вызывается при считывании WHERE-части SELECT-запроса бизнес-объекта. Sender – бизнес-объект, вызвавший событие, Clause – сформированная методом GetWhereClause WHERE-часть. Изменение запроса возможно через дополнение параметра Clause: Clause.Value = Clause.Value + <дополнение запроса>.
- **OnNewRecord**(Dataset: TgdcBase) – вызывается при добавлении новой записи. DataSet – бизнес-объект, вызвавший событие.
- **OnPostError**(Dataset: TgdcBase; E: Exception; Action: Integer) – вызывается при возникновении ошибки при попытке сохранения изменений. DataSet – бизнес-объект, вызвавший событие, E – исключение, Action – действие. Action может принимать следующие значения: 0 = прерывание операции и вывод ошибки на экран, 1 = прерывание операции без вывода ошибки, 2 = повтор операции.

Создание связанных с бизнес-объектом переменных

Итак, у нас есть бизнес-объект, обладающий некоторыми свойствами. Однако вам в run-time режиме могут понадобится дополнительные переменные, хранящие промежуточную информацию о бизнес-объекте. Существуют специальные свойства и методы, создающие связанные с бизнес-объектом переменные. Эти переменные будут ассоциированы с бизнес-объектом на протяжении его существования. Доступ к ним может быть получен из любой процедуры, из которой доступен бизнес-объект.

С бизнес-объектом могут быть ассоциированы переменные простых и объектных типов. Следующие методы и свойства позволяют работать со связанными с бизнес-объектом переменными:

- **AddObjectItem**(Name: String) – добавляет в список новую объектную переменную с именем Name, инициализируя ее nil.
- **AddVariableItem**(Name: String) – добавляет в список новую переменную простого типа с именем Name, инициализируя ее пустым значением.

- **ObjectExists**(Name: String): Boolean – возвращает Истина, если в списке объектных переменных, связанных с текущим бизнес-объектом, присутствует переменная с именем Name, Ложь – в обратном случае.
- **Objects**(Name: String): IDispatch – список объектных переменных, связанных с текущим бизнес-объектом. Обращение к переменной идет по ее наименованию.
- **VariableExists**(Name: String): Boolean – возвращает Истина, если в списке переменных простых типов, связанных с текущим бизнес-объектом, присутствует переменная с именем Name, Ложь – в обратном случае.
- **Variables**(Name: String): Variant – список простых переменных, связанных с ткущим бизнес-объектом. Обращение к переменной идет по ее наименованию.

Рассмотрим пример использования связанных переменных. Довольно часто переменные простых типов используются как флаги, например в методе DoFieldChange.

```
'Если существует уже такая переменная, то выходим
if Self.VariableExists("BN_BillNDS_DoFieldChange") then
  exit sub
end if
'Иначе, добавляем переменную
call Self.AddVariableItem("BN_BillNDS_DoFieldChange")
'далее идет код процедуры
```

В рассмотренном выше примере связанная переменная используется для того, чтобы выполнить какую-то часть кода всего один раз. При входе в процедуру проверяется, существует ли такая переменная, если существует, значит процедура уже выполнялась. А т.к. повторного выполнения не требуется, то мы выходим из процедуры. Если связанной с бизнес-объектом (в данном случает это Self) переменной не существует, то мы добавляем эту переменную и выполняем процедуру.

В следующем примере мы создали какой-то внутренний объект и хотим его связать с нашим бизнес-объектом.

```
'Объявляем переменную
Dim NewObject
'Создаем объект с классом MyClass
set NewObject = new MyClass
'Создаем объектную переменную, связанную с бизнес-объектом
'gdcObject
call gdcObject.AddObjectItem("AssociateObject")
'Присваиваем нашей объектной переменной созданный объект
set gdcObject.Objects("AssociateObject") = NewObject
```

Внимание! Связанные с бизнес-объектом объектные переменные уничтожаются при уничтожении бизнес-объекта.

Создание своих бизнес-классов

Мы уже не раз сталкивались с понятиями стандартного и пользовательского бизнес-объектов. Уточним, что же они значат. Стандартный бизнес-объект – это бизнес-объект, который заложен в платформу Гедымин. Он будет присутствовать даже при ненастроенной (т.е. эталонной) базе. Пользовательский бизнес-объект – это бизнес-объект, созданный пользователем-настройщиком платформы.
Пользовательский бизнес-объект создается средствами Гедымина и представляет собой экземпляр одного из следующих классов (иерархию смотрите на Рисунок 0.9) :

- TgdcAttrUserDefined, TgdcAttrUserDefinedTree, TgdcAttrUserDefinedLBRBTree – бизнес-объекты, созданные на основании пользовательских таблиц. Подтип равен названию таблицы (первых 20 символов). Про создание таких бизнес-объектов читайте в главе 8.
- TgdcUserDocument, TgdcUserDocumentLine – бизнес-объекты, созданные на основании простых пользовательских документов. Подтип равен РУИД-у пользовательского документа. Про создание таких бизнес-объектов читайте в главе 11.
- TgdcInvDocument, TgdcInvDocumentLine – бизнес-объекты, созданные на основании складских пользовательских документов. Подтип равен РУИД-у документа. Про создание таких бизнес-объектов читайте в главе 11.

- TgdcInvPriceList, TgdcInvPriceListLine – бизнес-объекты, созданные на основании пользовательских прайс-листов. Подтип равен РУИД-у пользовательского прайс-листа. Про создание таких бизнес-объектов читайте в главе 11.

У пользовательского бизнес-объекта обязательно присутствует подтип. И напротив, изначально все стандартные бизнес-объекты идут без подтипа. Здесь есть одна оговорка: вы можете создать свою разновидность стандартного бизнес-объекта за счет создания для него некого уникального подтипа.

Любой созданный бизнес-объект обладает всеми возможностями базового бизнес-объекта. Кроме того, создание бизнес-объекта не потребует больших затрат по времени, т.к. платформа настраивает созданный объект в соответствии с принятой бизнес-логикой. Вы можете дополнить или изменить существующую функциональность за счет перекрытия методов или событий и настройки визуальных форм в соответствии с необходимыми требованиями.

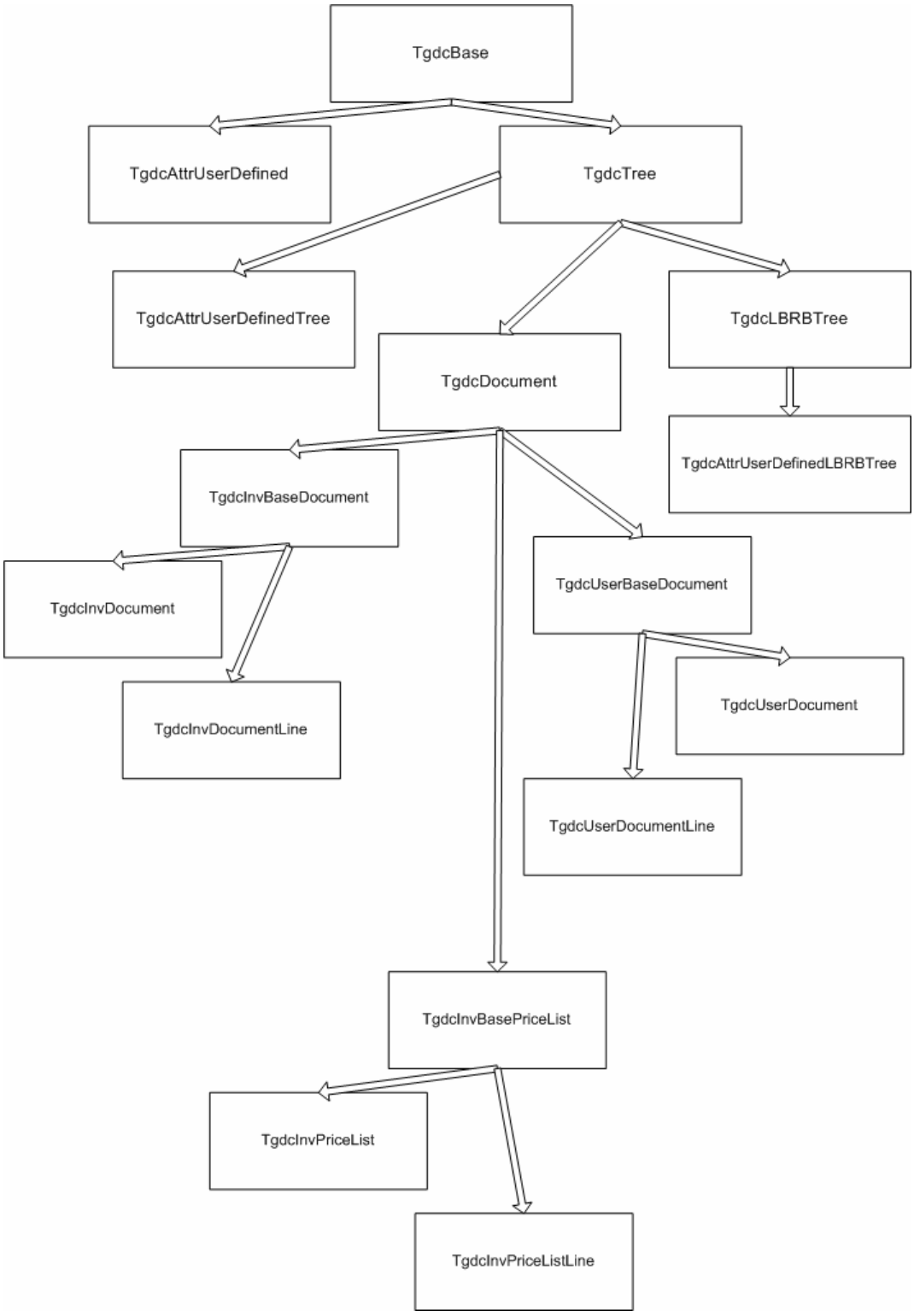


Рисунок 0.9

Модифицирование поведения бизнес-объектов. Перекрытие методов и событий

Итак, повторимся: в системе Гедымин используются понятия стандартного и пользовательского бизнес-объектов. Пользовательские бизнес-объекты создаются от определенных классов (смотрите предыдущую главу) и всегда имеют подтип. Попытка открыть пользовательский бизнес-объект с пустым подтипом приведет к генерации исключения. Стандартные бизнес-объекты изначально не имеют подтипов. Но, т.к.

платформа Гедымин не предоставляет настройщику стандартного механизма наследования, то за счет использования подтипов вы можете модифицировать существующие бизнес-классы.

Зарегистрировать подтип для стандартного бизнес-класса можно через исследователь. Для этого вам нужно создать новую ветку исследователя, выбрать переключатель «Бизнес-класс», в выпадающем списке указать стандартный бизнес-класс и ввести новый подтип. Создавая таким образом новое расширение стандартного класса, помните, что подтип должен быть уникален в пределах базы.

Если рассматривать, как это организуется на уровне работы с базой, то в таблицу GD_COMMAND, добавляется новая запись, которая и хранит в себе связку «стандартный бизнес-класс - подтип».

После того, как вы создали ветку в исследователе, в окне Редактор скрипт-объектов в дереве бизнес-классов появится бизнес-класс с новым подтипом. Вы можете перекрывать все доступные методы, чтобы определить его поведение. Кроме того, вам обязательно нужно перекрыть метод GetCurrRecordSubType в классе, который вы расширили. Например, вы создали новый тип Товара – Услугу, введя подтип SERVICE. Значит, в классе TgdcGood (без подтипа) вам необходимо перекрыть метод GetCurrRecordSubType, чтобы указать признак, по которым запись будет определяться как услуга. Данный метод используется для определения подтипа при создании / изменении / удалении записи. Если вы не перекроете GetCurrRecordSubType, то при работе с базовым классом Гедымин не сможет корректно определить тип изменяемой записи, а следовательно отработают не ваши перекрытые методы, а методы базового класса.

Ввод нового подтипа для стандартного класса всегда должен соответствовать существованию некого признака в базовой таблице. По данному признаку и будет определяться, что конкретная запись репрезентирует не просто стандартный объект, но объект с подтипом. В качестве признака вы можете использовать либо уже существующее поле, либо добавить новое поле. Данное поле должно заполняться значениями, которые будут соответствовать конкретному подтипу.

Например, в настройке Аренда расширен бизнес-класс Товар (TgdcGood, подтип пустой, базовая таблица GD_GOOD). В Аренде созданы бизнес-объекты Услуга (класс TgdcGood, подтип SERVICE, признак – поле USR$REN_ISSERVICE, для данного подтипа содержит 1) и Объект (класс TgdcGood, подтип OBJECT, признак - поле USR$REN_ISOBJECT, для данного подтипа содержит 1).

Теперь обратим внимание на перекрытие методов и событий. Итак, само понятие перекрытие метода означает его дополнение некой функциональностью. Чтобы перекрыть метод, вам необходимо в дереве бизнес-объектов найти требуемый класс и подтип (если он задан), выбрать метод. Двойной щелчок мыши по методу выводит окно редактирования метода.

Внимание: не желательно изменять название функции метода или подключать функцию другого метода! Все дело в том, что в названии функции метода зашито название класса, подтипа и метода, что облегчает чтение кода. Подключив же одну и ту же функцию к нескольким методам, вы можете элементарно забыть о том, что она используется в нескольких местах, и изменить ее в соответствии с требованиями конкретного метода.

При перекрытии метода или события тело скрипта изначально не пустое. В нем обязательно присутствует строка с вызовом Inherited. Вызов Inherited дает возможность отработать функциональности, заданной изначально в платформе Гедымин для данного класса. Обычно Inherited из скрипта не убирают, т.к. некорректное обращение с Inherited может привести к неправильной работе данного класса. Однако бывают случаи, когда вызов Inherited не нужен. Например, вы можете перекрыть метод бизнес-класса CreateDialogForm, который возвращает диалоговое окно бизнес-объекта, и сделать так чтобы он возвращал ваше диалоговое окно. В этом случае вызов Inherited не нужен. При перекрытии метода или события во входных параметрах обязательно присутствует объект, вызвавший метод или событие. Если вы перекрываете метод бизнес-объекта на его открытие, то во входных параметрах обязательно будет данный бизнес-объект, если вы перекрыли событие, вызванное нажатием кнопки, то во входных параметрах будет кнопка. Чаще всего входной параметр, представляющий объект, в методах называется Self, а в событиях Sender.

Чем отличается перекрытие методов от перекрытия событий?

Перекрывая метод, вы определяете поведение класса с заданным подтипом. Перекрывая событие, вы определяете поведение конкретного объекта. Например, на форме Адресная книга вы хотите кроме прочей информации отображать еще и главный счет для организаций. Для этого из запроса

```
SELECT z.*
FROM gd_contact z
```

вам нужно сделать запрос вида

```
SELECT z.*, ca.account, b.bankcode
FROM gd_contact z
  LEFT JOIN gd_company c ON z.id = c.contactkey
  LEFT JOIN gd_companyaccount ca ON ca.id = c.companyaccountkey
  LEFT JOIN gd_bank b ON b.bankkey = ca.bankkey
```

Согласитесь, что данный запрос является более громоздким, и его время выполнения несколько больше, чем время выполнения первоначального запроса. Расширить запрос можно двумя способами:

1. Перекрыть методы GetSelectClause и GetFromClause бизнес-класса TgdcBaseContact. В этом случае дополнительные таблицы будут добавляться в запрос для всех экземпляров, созданных от TgdcBaseContact или от классов, являющихся его наследниками.
2. Перекрыть события OnGetSelectClause и OnGetFromClause бизнес-объекта gdcContacts, непосредственно лежащего на форме Адресная книга. В этом случае дополнительные таблицы будут подключаться к запросу только в случае отображения записей на форме Адресная книга.

Теперь рассмотрим, как это выглядит на практике.

В первом случае мы перекрываем методы базового класса TgdcBaseContact для всех типов контактов:

```
option explicit
function TgdcBaseContactGetSelectClause(Self)
  TgdcBaseContactGetSelectClause = _
    Inherited(Self, "GetSelectClause", Array(Self))
  TgdcBaseContactGetSelectClause =
    TgdcBaseContactGetSelectClause + ", ca.account, b.bankcode "
end function

option explicit
function TgdcBaseContactGetFromClause(Self, ARefresh)
  TgdcBaseContactGetFromClause = _
    Inherited(Self, "GetFromClause", Array(Self, ARefresh))
  TgdcBaseContactGetFromClause =
    TgdcBaseContactGetFromClause + _
      " LEFT JOIN gd_company c ON z.id = c.contactkey " + _
      " LEFT JOIN gd_companyaccount ca " + _
      "   ON ca.id = c.companyaccountkey " + _
      " LEFT JOIN gd_bank b ON b.bankkey = ca.bankkey "
end function
Во втором случае перекрываются события:
option explicit
sub gdcContactsOnGetSelectClause(ByVal Sender, ByRef Clause)
  Dim ParamArr(1)
  Set   ParamArr(0) = Sender
  ParamArr(1) = Clause
  call   Inherited(Sender, "OnGetSelectClause", ParamArr)
  Clause.Value = ParamArr(1)

  Clause.Value = Clause.Value + " , ca.account, b.bankcode "
end sub

option explicit
sub gdcContactsOnGetFromClause(ByVal Sender, ByRef Clause)
  Dim ParamArr(1)
  Set   ParamArr(0) = Sender
  ParamArr(1) = Clause
  call   Inherited(Sender, "OnGetFromClause", ParamArr)
  Clause.Value = ParamArr(1)

  Clause.Value = Clause.Value + _
```

```
  " LEFT JOIN gd_company c ON z.id = c.contactkey " + _
  " LEFT JOIN gd_companyaccount ca " + _
  "   ON ca.id = c.companyaccountkey " + _
  " LEFT JOIN gd_bank b ON b.bankkey = ca.bankkey "
end sub
```

Если вы рассмотрели примеры внимательно, то заметили некоторые отличия в перекрытии методов и событий. В данном случае методы GetSelectClause и GetFromClause сами возвращают сформированную часть запроса в виде строки. События ничего не возвращают. Они срабатывают в определенное время и выполняют определенные действия. Если нам необходимо что-либо вернуть из события, мы должны передать для этого переменную в качестве входного параметра. Чтобы вернуть какое-либо значение именно платформе Гедымин, а не другому скрипту, вам необходимо использовать конструкцию <Переменная>.Value – т.е. своеобразное обращение к значению переменной.

А теперь обобщим различия между методами и событиями:

- метод перекрывается для класса и его наследников; обработчик события определяется только для конкретного объекта. Т.е. перекрытие методов несет более глобальный характер.
- в методе нельзя можно обращаться только к объекту, вызвавшему метод, или к глобальным объектам. Событие позволяет обращаться также к объектам, которые созданы в момент его вызова. Например, к форме, на которой лежит объект. Обращение к форме, на которой лежит объект из метода недопустимо. Метод срабатывает для всех объектов, созданных с определенным классом и подтипом. И эти объекты могут не лежать на форме.
- метод может непосредственно возвращать некоторое значение; событие ничего не возвращает. Вернуть что-либо из события можно через некоторый входной параметр. Работа со значением такого параметра происходит через .Value.

Пример перекрытия событий бизнес-объекта

Разработчики типового решения «Банк и Касса» не предусмотрели поле «Наименование банка получателя» в бизнес-объекте платежное поручение. Конечно, увидеть наименование банка можно, открыв платежное поручение в диалоговом окне на редактирование, однако намного удобнее было бы иметь соответствующую колонку в окне просмотра списка документов.

Попробуем решить данную проблему средствами Гедымина. Обратите внимание, что все, указанные ниже действия, необходимо выполнять под учетной записью Администратор.

Прежде всего, следует определиться, откуда можно получить необходимую информацию. Платежные поручения хранятся в таблице USR$BN_PAYMENT. Просмотреть структуру данной таблицы можно, если последовательно перейти в Исследователе системы в раздел «Сервис», «Атрибуты» и выбрать команду «Таблицы». Найдем в верхнем списке позицию, соответствующую USR$BN_PAYMENT и установим на нее курсор. В нижней части окна можно просмотреть список полей выбранной таблицы.

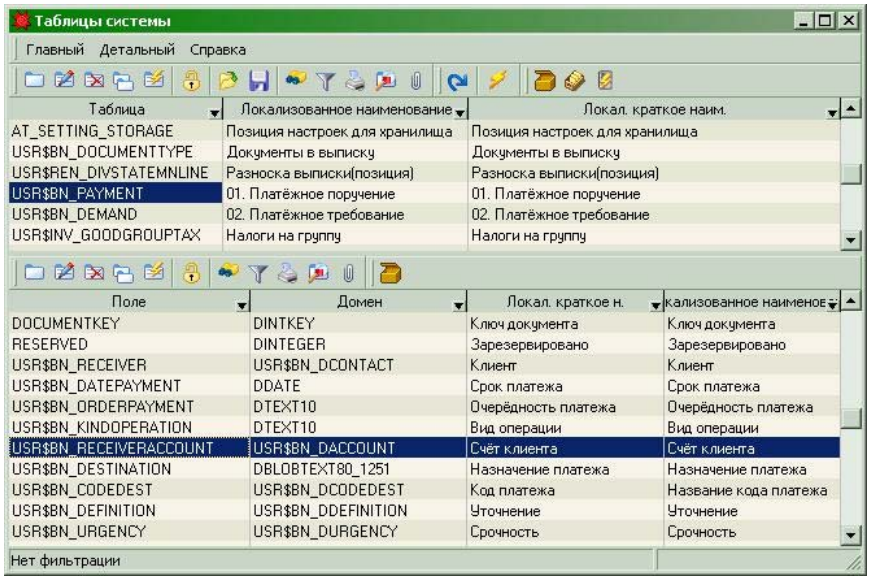


Рис. 40 Список таблиц системы.

Интересующее нас поле, ссылка на расчетный счет получателя денег, имеет наименование USR$BN_RECEIVERACCOUNT. Данное поле ссылается на таблицу GD_COMPANYACCOUNT. Последняя, в свою очередь, имеет поле BANKKEY — ссылку на таблицу GD_CONTACT, которая и содержит искомое наименование банка.

Итак, мы знаем, где находится нужная нам информация. Для ее извлечения необходимо изменить исходный запрос бизнес-объекта, подключив в секцию FROM две таблицы: GD_COMPANYACCOUNT и GD_CONTACT и добавив поле наименования банка в секцию SELECT.

Поскольку мы будем соединять таблицу GD_COMPANYACCOUNT с таблицей USR$BN_PAYMENT необходимо узнать, какой алиас имеет она в запросе. Для этого, находясь в списке платежных поручений, нажмем клавишу F10. На экране появится диалоговое окно «Мастер установок» таблицы. Нас интересует вкладка «Запрос».

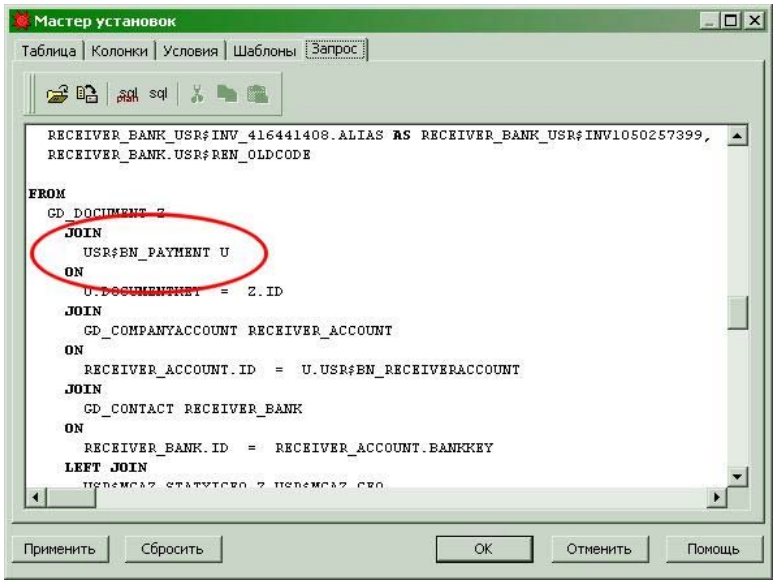


Рис. 41 Определение алиаса таблицы в запросе.

Как видно на рисунке, таблица USR$BN_PAYMENT имеет псевдоним U. Закроем окно «Мастера установок».

Теперь внесем изменения в запрос бизнес-объекта. Для этого активизируем форму просмотра списка платежных поручений и переведем ее в режим дизайнера путем нажатия комбинации клавиш Ctrl+Alt+E. Отыщем на ней и выделим компонент gdcUserDocument.

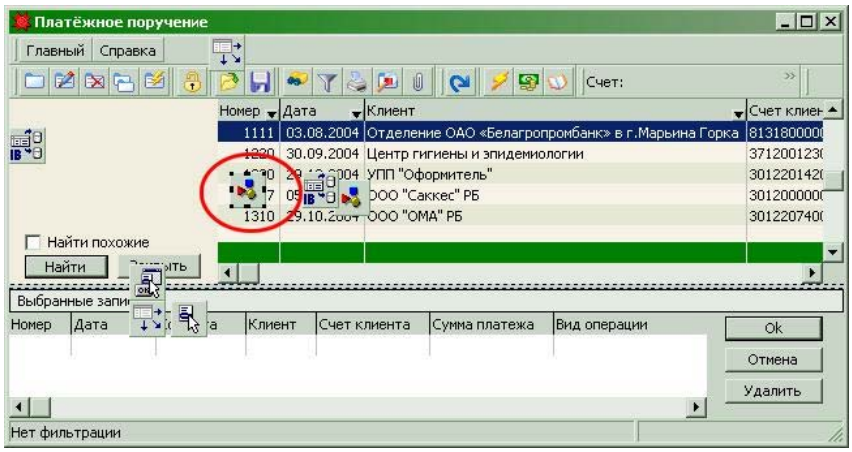


Рис. 42 Форма в режиме дизайнера.

Нажмем F11 для того, чтобы перейти в окно Инспектора объектов. Переключимся на вкладку События.

SQL запрос бизнес-объекта собирается из пяти частей: SELECT, FROM, WHERE, GROUP и ORDER. При формировании каждой из них вызывается соответствующее событие. Перекрыв событие можно изменить или даже целиком заменить текст соответствующей секции SQL запроса. В нашем случае мы будем

обрабатывать события OnGetSelectClause и OnGetFromClause. Для ввода программного кода найдем событие в списке и нажмем кнопку «…» справа от него, как показано на следующем рисунке.

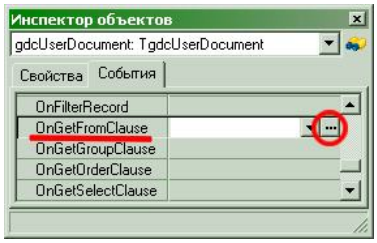


Рис. 43 Инспектор объектов.

На экране откроется редактор скрипт-объектов с текстом обработчика события по-умолчанию. Внесем в него свои коррективы и сохраним с помощью команды меню «Скрипт»—«Сохранить».

Ниже приводится исходный код перекрытых обработчиков. Текст, выделенный жирным начертанием, был добавлен нами.

```
option explicit
sub gdcUserDocumentOnGetFromClause(ByVal Sender, ByRef Clause)
'*** Данный код необходим для вызова встроенного обработчика ***
'*** В случае его удаления возможно нарушение работы системы ***
  Dim ParamArr(1)
  Set   ParamArr(0) = Sender
  ParamArr(1) = Clause
  call   Inherited(Sender, "OnGetFromClause", ParamArr)
  Clause.Value = ParamArr(1)
'*** конец кода поддержки встроенного обработчика            ***

  Clause.Value = Clause & _
    " JOIN gd_companyaccount receiver_account ON" & _
    " receiver_account.id = U.USR$BN_RECEIVERACCOUNT " & _
    " JOIN gd_contact receiver_bank ON" & _
    " receiver_bank.id = receiver_account.bankkey "

end sub


option explicit
sub gdcUserDocumentOnGetSelectClause(ByVal Sender, ByRef Clause)
'*** Данный код необходим для вызова встроенного обработчика ***
'*** В случае его удаления возможно нарушение работы системы ***
  Dim ParamArr(1)
  Set   ParamArr(0) = Sender
  ParamArr(1) = Clause
  call   Inherited(Sender, "OnGetSelectClause", ParamArr)
  Clause.Value = ParamArr(1)
'*** конец кода поддержки встроенного обработчика            ***

  Clause.Value = Clause & _
    ",receiver_bank.name AS receiver_bank_name "
end sub
```

Как видно из листингов все достаточно просто. Мы добавили две таблицы в секцию FROM, связав их с таблицей платежных поручений, и одно поле в секцию SELECT.

Испытаем наше творение в действии. Для этого необходимо пересоздать окно со списком платежных поручений. Именно пересоздать, а не закрыть-открыть. Для этого нажмем клавишу Shift и, удерживая ее, щелкнем мышью по крестику в правом верхнем углу окна. Оно исчезнет не только с экрана, но и удалится из оперативной памяти компьютера. Теперь заново откроем его используя Исследователь

системы[13]. Если на экран не выдалось никаких сообщений об ошибках, то все было сделано правильно. Осталось сделать нашу новую колонку видимой в таблице. Нажимаем F10, переходим на вкладку «Колонки», находим наше поле в списке, выделяем его и устанавливаем флаг «Колонка отображается».

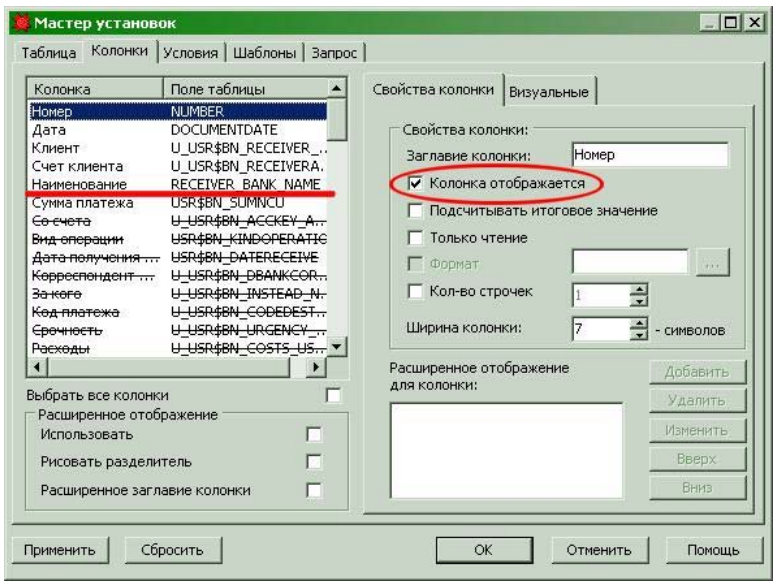


Рис. 44 Мастер установок.

Закроем «Мастер установок». Теперь в таблице, для каждого платежного поручения отображается наименование банка клиента.

Сохранение объектов в поток и загрузка их из потока

Итак, мы создали нечто на своей базе и теперь хотим это перенести на другую базу. Каким образом это сделать? Можно, конечно, воспользоваться механизмом репликации. Однако реплицировать базу из-за переноса двух документов или пары табличек нецелесообразно. В подобных случаях используется перенос данных при помощи файлов. Т.е. вы указываете, какие записи хотите перенести, сохраняете их в файл (файл хранит в себе упакованный поток). На второй базе просто указываете, из какого файла загрузить данные.

Для того чтобы сохранить данные в файл, вам необходимо открыть форму отображения бизнес-объекта, выделить записи, которые вы хотите сохранить и нажать на панели инструментов кнопку «Сохранить в объект файл». На второй базе на форме отображения нужно нажать кнопку «Загрузить объект из файла» и указать файл, который хранит данные (смотрите Рисунок 0.10).

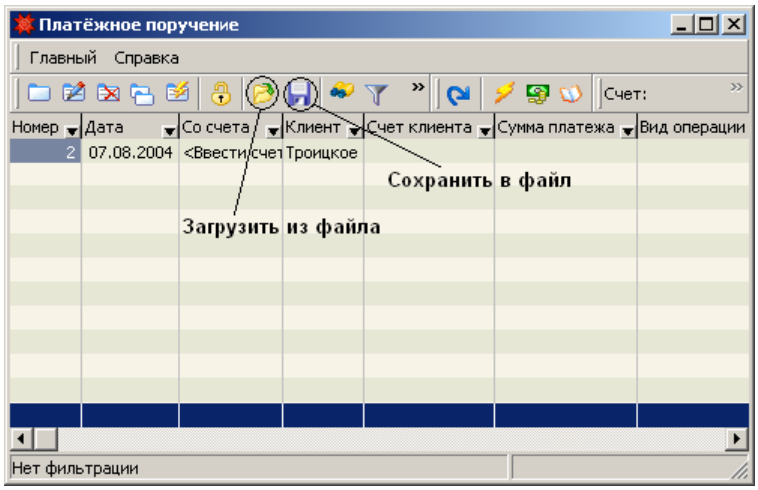


Рисунок 0.10

[13] Более долгий вариант — это выйти из Гедымина и запустить его снова.

К сожалению, из макросов вы не сможете работать с сохранением нескольких записей в один файл, используя механизм потоков. Однако это можно обойти, используя механизм настроек, который доступен и из макросов, и в виде мастеров.

Формы

В данной главе будут описаны основные виды форм, доступные при работе с платформой Гедымин, рассказано про особенности их поведения. Также будут указаны различия между просмотровыми и диалоговыми окнами, описано взаимодействие между формами и бизнес-объектами. Здесь мы познакомимся со способами создания новых форм, с возможностями изменения уже существующих, рассмотрим иерархию и наследование. Также эта глава содержит описание наиболее часто использующихся свойств и методов базовых форм.

Иерархия классов форм

В данном пункте мы рассмотрим иерархию классов форм, используемых при работе с платформой Гедымин.

Каждая форма – это по сути экземпляр определенного класса. Базовым классом для всех форм, используемых в Гедымине является класс TgdcCreateableForm, все остальные классы наследованы от него. На рисунке 7.1 вы можете увидеть иерархию классов форм.

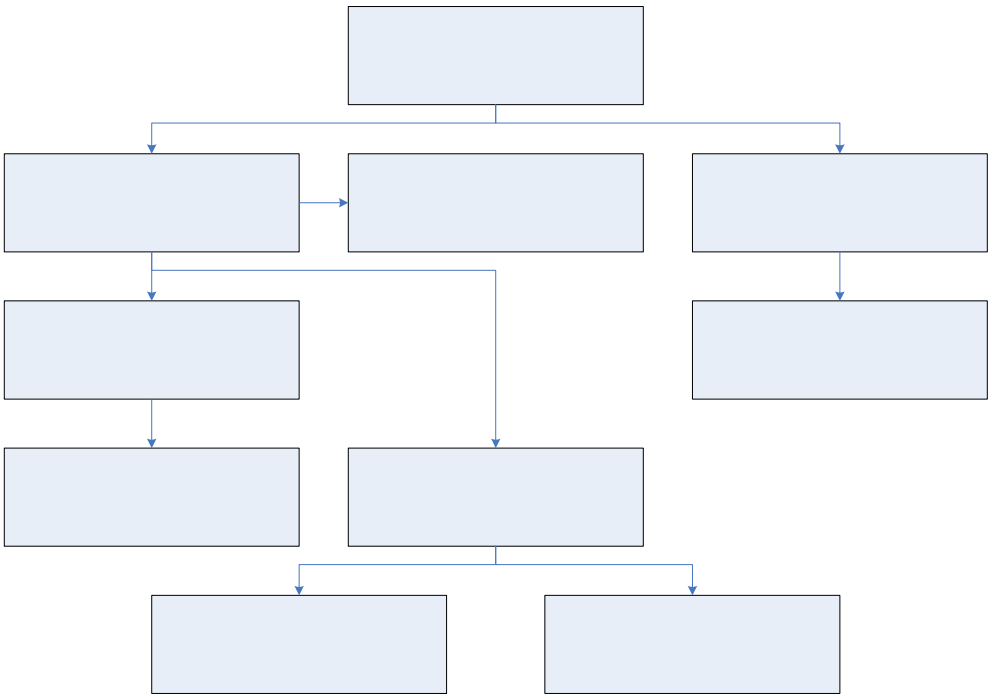


Рис. 45Иерархия классов форм просмотра

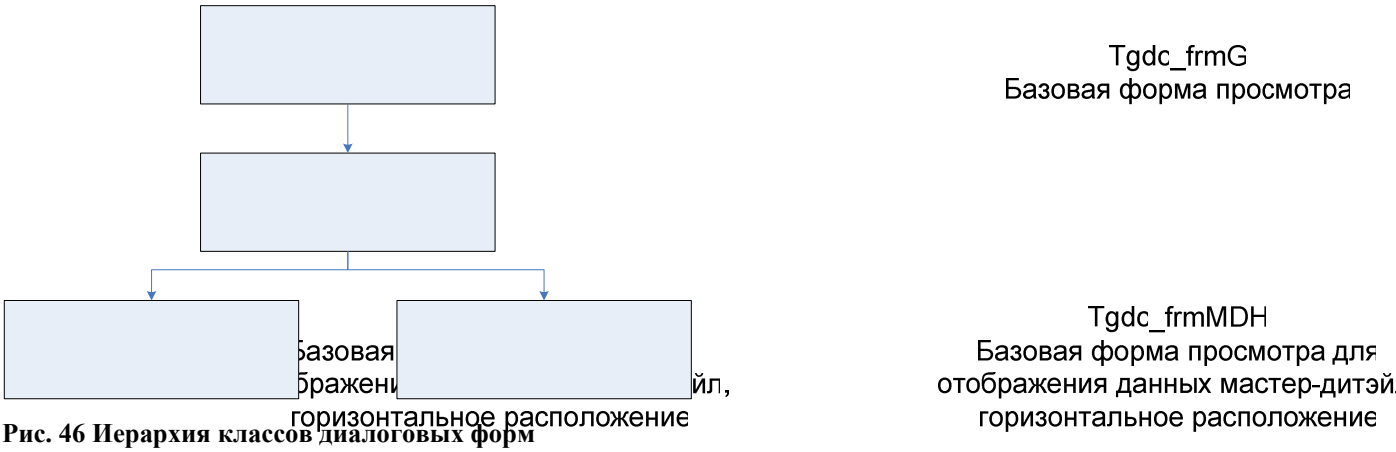


Рис. 46 Иерархия классов диалоговых форм

Как мы видим, от основного класса TgdcCreateableForm наследовано два класса Tgdc_dlgG (отвечает за работу с диалоговыми окнами) и Tgdc_frmG (отвечает за работу с формами просмотра). Более подробное описание классов можно увидеть в таблице 7.1.

Наименование класса	Описание класса
Tgdc_dlgG	Базовый класс для работы с диалоговыми окнами. Содержит основные методы для обработки изменения данных.
Tgdc_dlgTR	Наследован от Tgdc_dlgG. Содержит дополнительную транзакцию, которая может быть полезна при использовании выпадающих списков.
Tgdc_dlgHGR	Наследован от Tgdc_dlgTR. Используется для редактирования бизнес-объекта, у которого есть детальный объект. Содержит грид для отображения данных детального объекта и панель для работы с этими данными.
Tgdc_dlgTRPC	Наследован от Tgdc_dlgTR. Используется для редактирования бизнес-объекта, на базовую таблицу которого есть ссылка типа множество. Например, Товар имеет несколько связанных таблиц-множеств: драг. металлы (GD_GOODPRMETAL), налоги (GD_GOODTAX), единицы измерения (GD_GOODVALUE). Эти связки отображаются на отдельных закладках.
Tgdc_frmG	Наследован от TgdcCreateableForm. Является базовым классом для всех форм просмотра. Содержит основные методы и свойства, определяющие общее поведение всех форм просмотра.
Tgdc_frmMDH	Наследован от Tgdc_frmG. Является базовым классом для сложных форм просмотра (т.е. для master-detail форм).
Tgdc_frmMDHGR	Наследован от Tgdc_frmMDH. Отображает данные в двух гридах, расположенных на форме горизонтально.
Tgdc_frmMDHGRAccount	Наследован от Tgdc_frmMDHGR. Содержит дополнительную панель с выпадающим списком счетов текущей рабочей организации. Используется для отображения данных, которые имеют поле «счет текущей компании».
Tgdc_frmMDVGR	Наследован от Tgdc_frmMDH. Отображает данные в двух гридах, расположенных на форме вертикально.
Tgdc_frmMDV	Наследован от Tgdc_frmMDH. Предполагается вертикальное отображение master-detail данных.
Tgdc_frmMDVTree	Наследован от Tgdc_frmMDV. Содержит дерево (для master-объекта) и грид(для detail-объекта).
Tgdc_frmSGR	Наследован от Tgdc_frmG. Является базовым классом для простых форм просмотра. Содержит грид для отображения содержимого бизнес-объекта.
Tgdc_frmSGRAccount	Наследован от Tgdc_frmSGR. Содержит дополнительную панель с выпадающим списком счетов текущей рабочей организации. Используется для отображения данных, которые имеют поле «счет текущей компании».

Таблица 0.1

Используя данные классы, вы можете «с нуля» создать свою форму.

Существуют так называемые стандартные классы форм, которые «зашиты» в программу. Все они наследованы от вышеперечисленных классов. Например, форма для отображения справочника ТМЦ – экземпляр класса Tgdc_frmMainGood. Вы можете изменять поведение стандартных форм за счет перекрытия методов и событий. Создать форму, наследуясь от стандартного класса в платформе Гедымине, невозможно. Как мы уже знаем из главы 6 о бизнес-объектах, прямое наследование в Гедымине не реализовано. Но этот пробел можно обойти за счет введения подтипа. Понятие подтипа для формы напрямую связано с бизнес-объектом. Расширяя стандартный бизнес-объект пользовательским подтипом вы автоматически получаете все его формы (для просмотра и редактирования) с тем же подтипом, а следовательно можете вносить изменения, которые будут характерны только для конкретного подтипа.

База данных

В данной главе будет рассмотрена база данных, поставляемая с платформой Гедымин, ее объекты, даны их основные определения. Также вы сможете ознакомиться с механизмами изменения структуры базы данных, узнаете об ограничениях при работе с БД, заложенных в платформу Гедымин, и о том, как их обойти.

База данных состоит из различных объектов, таких как таблицы, представления (в некоторой литературе упоминаются как проекции), домены, хранимые процедуры, триггеры, индексы, ограничения (в данном случае имеются ввиду ограничения, накладываемые на данные в таблицах), генераторы, пользовательские функции (еще называют udf-функции, сокращение от английского User Defined Functions). Объекты базы данных содержат всю информацию о ее структуре и данных. Объекты базы данных также упоминаются как метаданные.

Для управления базой данных Interbase используется язык структурированных запросов SQL. SQL – это сокращение для Structured Query Language.

Если вы берете «голую» инсталляцию Гедымина (т.е. инсталляцию, которая включает в себя только платформу без настроек), то в ее пакете будет так называемая эталонная база данных. Эталонная база данных содержит только необходимые для начальной работы объекты, т.е. ее структура является общей частью для всех баз данных, предназначенных для работы с Гедымином. Настроенная база данных будет содержать кроме общих объектов еще и пользовательские. Пользовательские объекты БД имеют в своем наименовании префикс USR$. Такие объекты создаются посредством использования платформы Гедымин. Скажем также, что системные объекты БД Interbase имеют префикс RDB$. Объекты базы данных, не имеющие пользовательского и системного префиксов, будем называть стандартными.

Конечно, нарастить структуру БД можно и используя сторонние утилиты, но в этом случае вы рискуете столкнуться с некорректной работой платформы, особенно при переносе данных с одной базы на другую, при апгрейде (обновлении) базы данных и т.д. При использовании платформы Гедымин для изменения структуры БД помните, что вы можете изменять только пользовательские объекты БД. Изменение так называемых стандартных объектов может повлечь за собой некорректную работу, и поэтому непосредственно из-под платформы оно заблокировано.

Любое изменение метаданных возможно только под администратором сервера Interbase. Обычно это пользователь SYSDBA с паролем masterkey, если вы не изменили пароль при инсталляции. Администратор Гедымина (пользователь Administrator) по умолчанию является также администратором сервера. Если вы попытаетесь изменить метаданные под другим пользователем, то вам будет предложено ввести пароль администратора сервера Interbase.

Следующие главы будут посвящены пользовательским объектам базы данных, их видам, способам создания.

Версия структуры базы данных

История изменений структуры базы данных хранится в таблице FIN_VERSIONINFO. Запись с наибольшим ID определяет текущий номер версии структуры базы данных.

Информация из данной таблицы используется утилитой обновления структуры базы данных MODIFY.EXE.

Просмотреть информацию о текущей версии структуры базы данных можно так же на вкладке «База данных», диалогового окна «О системе». Вызов данного окна производится из меню «Справка» главного окна Гедымина.

FIN_VERSIONINFO

PK	FK	Поле Домен Тип данных	NN	Описание
1		ID DINTKEY INTEGER	☒	Уникальный идентификатор. Первая версия имеет ИД=1. С каждой последующей записью идентификатор увеличивается на 1. Обратите внимание, что при вставке новой записи значение идентификатора должно быть указано

				непосредственно, так как он не увеличивается автоматически.
		VERSIONSTRING DVERSIONSTRING VARCHAR(20)		Строка с номером версии. Четыре группы по четыре десятичные цифры, разделенные точками. Например, 0000.0001.0000.0055. Важно соблюдать увеличивающуюся последовательность номеров от более старшей версии структуры к более младшей.
		RELEASEDATE DDATE DATE	☒	Дата обновления структуры базы данных.
		COMMENT DTEXT254 VARCHAR(254)		Произвольный коментарий.

На момент написания данной книги последняя запись в таблице FIN_VERSIONINFO содержала следующие данные:

- ID — 64;
- VersionString — 0000.0001.0000.0092;
- ReleaseDate — 01.03.2005;
- Comment — Sync triggers procedures changed.

Типы данных. Домены

Типы данных – это базовые элементы любого языка программирования или сервера СУБД. Когда мы говорим о том, что в базе данных хранится какая-то информация, то осознаем, что эта информация хранится в упорядоченном виде. Данные обрабатываются, исходя из их типа. Тип данных является своего рода ограничителем; он указывает, что можно передавать объекту данного типа, а что нельзя.

Каждый тип данных имеет определенный набор операций, который можно выполнять над значениями данного типа. Interbase предоставляет следующий набор типов (более подробно о них можно узнать в документации Interbase):

- INTEGER – целое число, длиной 4 байта.
- SMALLINT – короткое целое число, длиной 2 байта.
- FLOAT – вещественный тип с плавающей точкой. Точность данного типа недостаточна для хранения больших дробных значений. Особенно данный тип не рекомендуется использовать для хранения денежных величин – в переменных типа FLOAT очень быстро нарастают ошибки за счет округления.
- DOUBLE PRECISION – вещественный тип с плавающей точкой двойной точности. Аналогичен FLOAT, за исключением того, что предоставляет наибольшую точность при округлении значения.
- NUMERIC – вещественный тип с фиксированной точкой. Имеет разрядность 18 знаков, точность – от нуля до разрядности. Разрядность – это общее количество цифр в числе, точность – количество знаков после запятой.
- DECIMAL – вещественный тип с фиксированной точкой. Аналогичен NUMERIC. Для хранения денежных величин рекомендуется использовать либо NUMERIC, либо DECIMAL. Дублирование практически одинаковых типов объясняется тем, что в более ранних версиях Interbase они имели некоторые отличия, и были сохранены для совместимости версий.
- DATE – хранит даты с точностью до дня. Диапазон возможных значений от 1 января 100 года н.э. до 29 февраля 32768 года.
- TIME – хранит данные о времени с точностью до десятитысячной доли секунды. Диапазон возможных значений – от 00:00 до 23:59.9999.
- TIMESTAMP – представляет собой комбинацию типов DATE и TIME, т.е. хранит и дату, и время.

- CHAR – предназначен для хранения текстовой информации. Максимальная длина – 32768 символов. Если длина заданного значения короче, чем длина типа, то текст дополняется в конце пробелами. Эти пробелы не играют роли при сравнении полей различных текстовых типов, но выводятся при отображении значения данного типа. Такое поведение было необходимо ранее, когда табличная информация выводилась в текстовом режиме. Дополнительные пробелы помогали выравнивать значения по колонкам. Теперь, когда информация выводится в графическом режиме использование данного типа не является целесообразным.
- VARCHAR – аналогичен типу CHAR, за исключением того, то значение не дополняется пробелами до заданной длины. И, следовательно, данный тип экономит трафик сети. Также называется текстовым типом переменной длины.
- BLOB – для хранения динамически расширяемых данных. Позволяет хранить такие данные, как картинки, музыку, видео, большие тексты переменной длины. Имеет подтипы: 0 – данные неопределенного типа, 1 – текстовые данные, 2 – данные в двоичном виде.

В символьных типах (CHAR, VARCHAR) следует обратить внимание на такие характеристики как:

- CHARACTER SET – набор символов. Набор символов определяется для всей базы данных и используется по умолчанию для всех текстовых полей, если не переопределяется явно при создании поля. База данных Гедымина имеет CHARACTER SET равный WIN1251.
- COLLATION ORDER – указание порядка сортировки в зависимости от национального языка. Для русского языка используется PXW_CYRL в паре с CHARACTER SET WIN1251.

Некоторые рассматривают домены как пользовательское расширение типов на основе уже имеющихся. Однако домен определяет не только тип данных, но и может дополнить его некоторыми ограничениями, весьма полезными для сохранения целостности.

В Гедымине при создании новых полей в таблице вы не сможете им указать явный тип: типизация полей происходит только через домен. Кроме того, в большинстве случаев, если домен не указан, но указан явный тип, Interbase создает сам новый домен с автоматическим (а следовательно не несущим смысловой нагрузки) названием.

Следующая таблица 8.1 содержит перечисление наиболее часто используемых стандартных для Гедымина доменов.

Домен Тип данных	Not Null	Для чего используется
DINTKEY INTEGER	x	Используется для определения ключевого целочисленного поля таблицы. Как мы помним, в основном таблицы в Гедымине имеют простой целочисленный ключ.
DFOREINGKEY INTEGER		Целочисленный тип, используется для указания того, что поле является ссылкой. Не рекомендуется использовать в пользовательских объектам.
DMASTERKEY INTEGER	x	Целочисленный тип, используется для указания ссылки на таблицу мастера. Непосредственно по данному домену определяются детальные объекты для главного объекта. Т.е., если мы ищем детальные объекты, то Гедымин сначала находит все объекты, которые имеют внешний ключ со ссылкой на мастера, затем офильтровывают только те ссылки, поля которых имеют домен DMASTERKEY. Не рекомендуется использовать самим. Данный домен подставляется сам для некоторых полей при создании сложных документов.
DINTEGER INTEGER		Целое число.
DINTEGER_NOTNULL INTEGER	x	Целое число, данные данного типа не могут содержать NULL.
DDOUBLE DOUBLE PRECISION		Число с плавающей точкой.

DNAME VARCHAR(60)	x	Текстовое значение переменной длины не более 60 символов, не может быть пустым. Используется обычно для обязательного наименования (например, в справочниках)
DTEXTx (x – число) VARCHAR(x)		Текстовое значение переменной длины не более x символов. Может быть NULL.
DPERCENT NUMERIC(7, 4)		Дробное число с разрядностью 7 и точностью 4. Используется для хранения процентов.
DPOSITIVE NUMERIC(15, 8)		Дробное число с разрядностью 15 и точностью 8. Используется для хранения положительных чисел или чисел равных нулю.
DCURRENCY NUMERIC(15, 4)		Дробное число с разрядностью 15и точностью 4. Используется для хранения денежных величин.
DQUANTITY		Дробное число с разрядностью 15и точностью 4. Используется для хранения количества.
DBOOLEAN		Логический тип. Т.к. Interbase напрямую не поддерживает использование логических типов, то он определен как целочисленный тип, который может принимать значения 0 (Ложь),1 (Истина) и NULL. Сам Interbase немного шире трактует понятие логических величин: для него ИСТИНА – это любое положительное значение, ЛОЖЬ – в обратном случае.
DBOOLEAN_NOTNULL	x	Аналогичен домену DBOOLEAN, за исключением того, что может принимать только значения 0 и 1.
DBMP		Представляет собой двоичный BLOB для хранения картинок.
DSECURITY INTEGER	x	Дескриптор безопасности. Используется для указания прав доступа к записям. По умолчанию принимает значение -1 (т.е. доступ для всех).
DDATE DATE		Домен для хранения даты.
DDATE_NOTNULL DATE	x	Домен для хранения даты. Не может быть пустым.
DTIME TIME		Домен для хранения времени.
DTIME_NOTNULL TIME	x	Домен для хранения времени. Не может быть пустым.
DTIMESTAMP TIMESTAMP		Домен для хранения даты и времени.
DTIMESTAMP_NOTNULL TIMESTAMP	x	Домен для хранения даты и времени. Не может быть пустым.
DBLOBTEXT		Представляет собой текстовый BLOB для хранения текстов переменной длины.
DDOCUMENTDATE	x	Дата документа. Обязательна для заполнения.
DDOCUMENTNUMBER	x	Номер документа. Обязателен для заполнения. Является текстовым типом переменной длины с кодировкой WIN1251 и сличением PXW_CYRL
DDESCRIPTION		Представляет собой текстовый BLOB для хранения текстов переменной длины. Используется для описания содержимого записи таблицы.

Таблица 0.1

В данной таблице перечислены далеко не все домены. Но даже среди перечисленных вы наверняка заметили дублирование типов, которые они описывают. По сути домены используются не только для

определения типа и накладываемых на него ограничений. Название домена играет информационную роль. Кроме того, при изменении некоторых визуальных характеристик домена можно менять эти характеристики и у полей, которым сопоставлен данный домен. Как это сделать мы узнаем ниже.

Создание домена в системе Гедымин

Бизнес-классом, отвечающим за работу с доменами, является класс TgdcField. Чтобы создать новый домен в системе Гедымин, необходимо выбрать в исследователе пункт Сервис \ Атрибуты \ Домены. Перед вами появится форма просмотра доменов. Т.к. создание домена (как впрочем и любого объекта БД) – это изменение структуры базы данных, то делать эту операцию дозволено только администраторам сервера Interbase.

Итак, при добавлении нового объекта на экране появится мастер для добавления доменов. На первой его закладке вам будет предложено ввести название на английском языке, локализованное название (которое видит пользователь) на русском, и описание (смотрите рисунок 8.1). При вводе названия домена на английском вам не обязательно самим указывать префикс USR$. Если вы не указали его, то Гедымин добавит префикс при сохранении объекта БД.

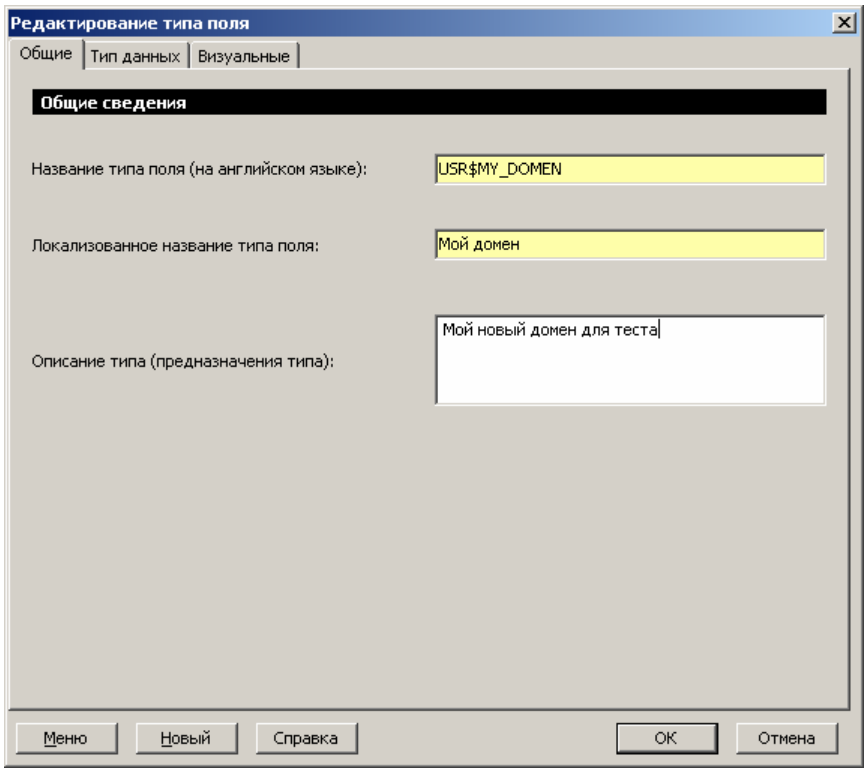


Рисунок 0.1

Закладка «Тип данных» предоставляет пользователю возможность выбрать тип, на основе которого будет создан домен, добавить ограничения и значение по умолчанию. Рассмотрим ее поподробнее. Итак, мы можем создать следующие типы данных:

- **Строковый** – переменной и постоянной длины. При этом вам необходимо указать максимальную длину. Если вы ходите хранить в поле информацию с использованием национальных символов, то желательно указать символьную кодировку и сличение.
- **Числовой** – здесь вам необходимо указать, какое число вы будете хранить: короткое целое, длинное целое, дробное с плавающей запятой, дробное с фиксированной запятой. Для дробного числа с фиксированной запятой нужно указать разрядность и количество цифр для дроби (точность).
- **Временной** – дата, время или дата и время.
- **Двоичный** – этот тип создается на базе BLOB-а. Вы можете создать строковое поле (т.е. текстовый BLOB), для него желательно указать символьную кодировку, или произвольный BLOB для хранения картинок и другой информации.

- **Ссылка** – один из наиболее сложных доменов. Создает целочисленное поле, которое является внешней ссылкой на другую таблицу. При этом вам необходимо указать таблицу, на первичный ключ которой будет ссылаться это поле, также поле для отображения из этой таблицы (используется в выпадающих списках, при формировании запроса парсером). Это поле необходимо для того чтобы пользователь мог видеть, что конкретно он выбирает для хранения в поле-ссылке, т.к. целочисленное значение ключа содержит весьма скромную информацию о записи. Также вы можете указать условие, по которому будут фильтроваться записи из таблицы, на которую создана ссылка. Это условие будет использоваться только в выпадающих списках. Не должно содержать алиасов. Кроме всего прочего, при создании поля с доменом-ссылкой для таблицы сразу создается внешний ключ. Таким образом, через домен, обходится одно из ограничений Гедымина – нельзя создавать внешние ключи напрямую.
- **Множество** – по набору параметров очень похож на тип Ссылка. Только, если для типа Ссылка создается целочисленное поле с внешним ключом на другую таблицу, то в случае множества создаваемое поле несет информационную нагрузку больше для внутреннего анализатора структуры БД. Создание поля с доменом Множество говорит Гедымину, что необходимо создать промежуточную таблицу-множество. Первичный ключ в данной таблице будет состоять из двух ключевых полей: поле-ссылка на таблицу, в которой создано поле-множество, и поле-ссылка на таблицу, которая указывается в параметрах домена-множества. Удаление поля-множества влечет за собой удаление таблицы-множества. Таблицы-множества имеют префикс USR$CROSS. Кроме уже известных нам параметров «Ссылка на таблицу», «Поле для отображения», «Условие выбора значений» домен типа Множество имеет такие параметры, как «Множество с текстовым полем» (указывает, что создаваемое поле-множество будет текстовым, если этот параметр не указывать, то создастся поле с типом короткое целое) и «Длина поля» (данный параметр работает в паре с предыдущим, содержит длину текстового поля). Также необходимо помнить, что при выборе текстового типа вы можете указать кодировку и сличение. Текстовый тип позволяет заполнять поле-множество указанными значениями поля для отображения через пробел. Заполнение этого поля происходит на триггере, который создается вместе с полем-множеством. Таким образом пользователь даже при помощи грида может увидеть, какие записи другой таблицы соответствуют текущей. При указании целочисленного типа просмотреть какие записи соответствуют текущей можно будет только через диалог редактирования записи.
- **Перечисление** – данный тип представляет собой перечисление возможных символов, которые в качестве значения может принимать поле. Каждому значению соответствует уникальное наименование, которое должно содержать краткую характеристику значения. При работе с типом Перечисление пользователь работает со списком наименований, а уже система сама подставляет в поле необходимое значение.

Также на закладке «Тип данных» есть раздел по контролю за содержимым поля. Этот раздел содержит три пункта:

- **Поле обязательно должно быть заполнено** – указывает, что поле не может содержать NULL-значения. О использовании NULL-значений читайте ниже.
- **Значение по умолчанию** – здесь указывается значение по умолчанию, которое будет принимать поле при добавлении новой записи (если при описании поля не будет указано иное значение по умолчанию).
- **Ограничение** – содержит правила по ограничению содержимого поля. Например, при создании логического поля на базе короткого целого типа используется ограничение, которое указывает, что из всего множества коротких целых чисел данный тип может использовать только 0 и 1.

Закладку «Тип данных» вы можете увидеть на рисунке 8.2.

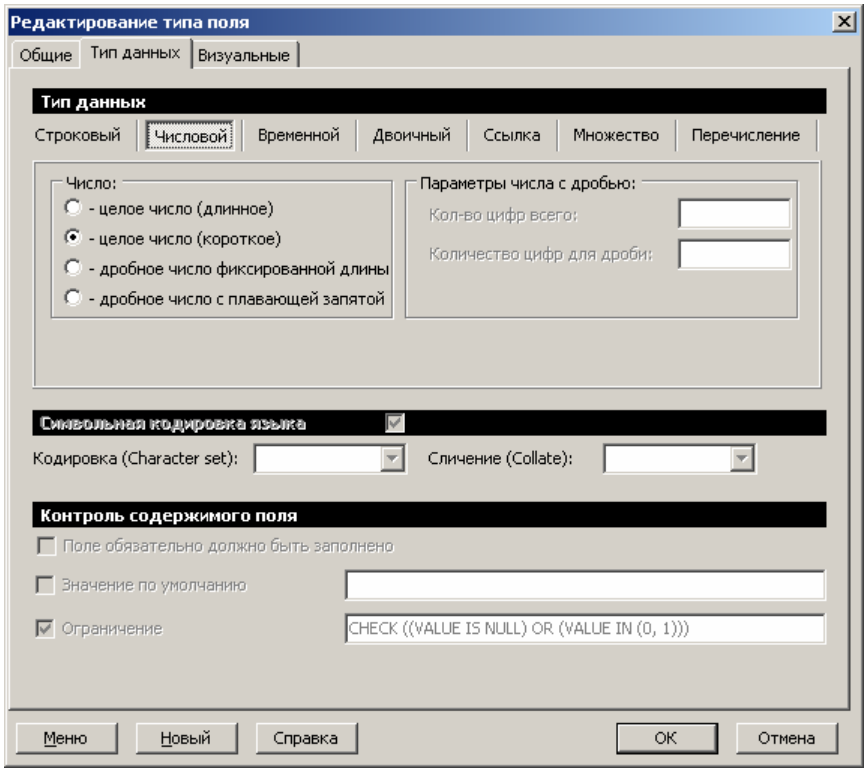


Рисунок 0.2

Нам остается рассмотреть последнюю закладку «Визуальные» (рисунок 8.3), которая содержит визуальные настройки для домена. Эти настройки по умолчанию устанавливаются для полей, создаваемых с этим доменом. Итак, на этой закладке мы имеем следующие пункты:

- **Ширина поля при отображении** – содержит **визуальную** ширину поля в пикселях, т.е., попросту говоря, ширину визуального элемента отображения, используемого для отображения / редактирования поля с данным доменом.

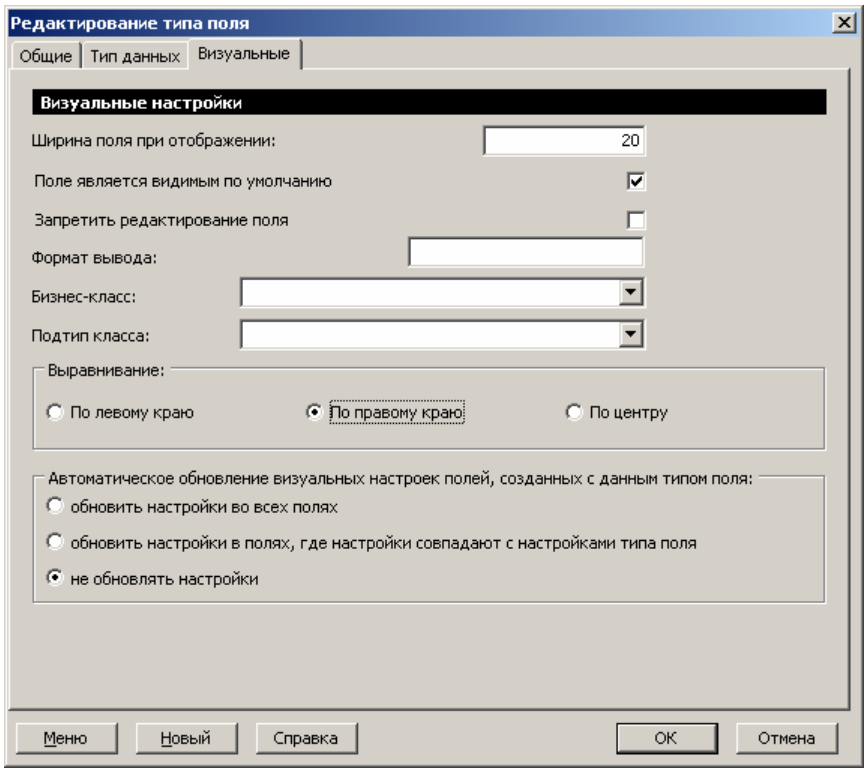


Рисунок 0.3

- **Поле является видимым по умолчанию** – указывает, что поле по умолчанию отображается в гриде. Наверняка вы согласитесь, что пользователю необязательно видеть ключевые поля, т.к. его скорее будет интересовать текстовое их представление. Для таких полей данный признак можно не устанавливать.
- **Запретить редактирование поля** – указывает, что поле не редактируется пользователем вручную. Такие поля заполняются системой на триггере или на каком-либо событии в скрипте. Используются по большей части для содержания признаков принадлежности записей к какой-нибудь конкретной группе.
- **Формат вывода** – указывает формат, в котором будут отображаться данные в колонке грида. Для указания формата используется маска, принятая в Делфи (смотрите Таблица 0.2).

Символ	Описание
!	Данный символ означает, что дополнительные символы представлены в тексте как ведущие пробелы, в обратном случае – как замыкающие пробелы.
>	Означает, что все символы, следующие за ним, будут в верхнем регистре, пока не закончится маска, или пока не встретится символ <.
<	Означает, что все символы, следующие за ним, будут в верхнем регистре, пока не закончится маска, или пока не встретится символ >.
<>	Если эти два символа встречаются в маске вместе, то регистр вводимых символов не меняется.
\	Означает, что следующий за ним символ – это буквенный символ. Используйте его для вывода символов, используемых в формате маски.
L	Означает, что только буква может быть на данной позиции. Для Соединенных Штатов это A-Z, a-z.
l	Разрешает ввести только букву на данной позиции, но не требует ее обязательного ввода.
A	Означает, что только буква или цифра может быть на данной позиции. Для Соединенных Штатов это A-Z, a-z, 0-9.
a	Разрешает ввести только букву или цифру на данной позиции, но не требует ее обязательного ввода.
C	Требует ввода любого символа на данной позиции.
c	Позволяет ввести любой символ на данной позиции, но не требует его ввода.
0	Требует ввода цифры на данной позиции.
9	Позволяет ввести цифру на данной позиции, но не требует ее ввода.
#	Позволяет ввести цифру или знак плюса или минуса на данной позиции, но не требует его ввода.
:	Используется как разделитель для часов, минут и секунд при вводе времени. При этом берется символ из региональных настроек компьютера.
/	Используется как разделитель для месяца, дня и года при вводе даты. При этом берется символ из региональных настроек компьютера.
;	Используется для разделения трех полей маски.
_	Данный символ автоматически вставляет пробел в текст. Когда пользователь водит символы в поле, курсор пропускает символ _.

Таблица 0.2

- **Бизнес-класс** – используется только для полей-ссылок. При автоматическом создании окна диалога бизнес-объекта для всех полей ссылок на форму кладутся выпадающие списки. Данный бизнес-класс будет подставлен в качестве параметра в выпадающий список, таким образом обеспечив работу со связанным бизнес-объектом. Внимание: данный бизнес-класс должен быть логически связан с таблицей, на которую идет ссылка. Еще лучше, если ссылку вы делаете на базовую таблицу указанного бизнес-класса. Если вы не указали никакого класса, но ваш домен –

ссылка, то внутренний настройщик форм сам подставит в выпадающий список наиболее подходящий бизнес-класс по указанной таблице (т.е. будет найден бизнес-класс, базовая таблица которого равна указанной, или указанная таблица ссылается на базовую).

- **Подтип класса** – используется в паре с предыдущим параметром. Содержит подтип класса, наиболее точно характеризующий связанный с нашим доменом-ссылкой объект.
- **Выравнивание** – указывает как будет выводится информация в гриде: выровнено по левому краю, по правому или по центру.
- **Автоматическое обновление визуальных настроек полей, созданных с данным типом поля** – данная опция говорит Гедымину, что делать при изменении характеристик домена. В отличие от других настроек она не хранится в базе, а работает как команда по окончанию редактирования домена. Может работать в трех режимах:
    - обновить настройки во всех полях – при подтверждении изменений для всех полей, созданных с данным доменом обновятся визуальные настройки в соответствии с указанными настройками домена. Например, вы забыли указать бизнес-класс при создании домена. Создали несколько полей с этим доменом и только тогда обнаружили оплошность. Достаточно отредактировать домен и установить данную опцию. **Важно**: при этом у вас на редактирование должен быть открыт только домен. Настройки у открытых на редактирование полей не изменятся!
    - обновить настройки в полях, где настройки совпадают с настройками домена – аналогичен предыдущему пункту за исключением того, что настройки будут обновляться только в тех полях, где они остались такими же как у домена, т.е. вы не изменяли их вручную.
    - не обновлять настройки – ничего не делает с настройками полей при редактировании домена. Установлено по умолчанию. Т.к. изменять визуальные настройки полей по домену нужно очень аккуратно.

Здесь еще необходимо добавить, что при изменении домена нужно быть очень внимательным: желательно не изменять домены, которые вы не создавали. Ваши изменения могут причинить ущерб уже существующей функциональности. Если вы не уверены, что ваши действия не причинят вреда, лучше создать свой аналогичный домен.

В уже созданном домене можно изменять все визуальные параметры, локализованные названия, а также (если это ссылка или множество) поле для отображения и условие.

Таблицы

Гедымин использует реляционную СУБД. Помимо всего прочего это означает, что все данные в ней хранятся в виде таблиц. И, как мы уже знаем, бизнес-объект – это набор данных, основанный на запросе. А в запросе участвуют таблицы.

Прежде дадим определение таблицы.

Таблица – это структура, состоящая из множества неупорядоченных горизонтальных строк (rows), каждая из которых содержит одинаковое количество вертикальных столбцов (colums). Пересечение отдельной строки и столбца называется полем (field), которое содержит специфическую информацию. Многие принципы работы реляционной базы данных взяты из определений отношений (relations) между таблицами.

Одно из требований реляционной базы данных – это то, что каждая таблица должна содержать первичный ключ. Конечно, можно создать таблицу и без первичного ключа, но это может привести к проблеме идентификации записи, а следовательно, вы не сможете корректно обработать запись. Итак, повторяя уже сказанное, первичный ключ – важнейший объект базы данных, используемый для однозначной идентификации записи. Бизнес-объект поддерживает только простые первичные ключи, за исключением режима множества, когда поддерживается составной ключ.

Виды пользовательских таблиц

Мы уже знаем, что пользовательские бизнес-объекты создаются на основе пользовательских таблиц. Гедымин предлагает несколько видов таблиц, предназначенных для решения различных задач.

- Простая таблица с идентификатором;

- Таблица с идентификатором;
- Таблица со ссылкой;
- Простое дерево;
- Интервальное дерево;

Каждый тип отличается от другого набором полей, триггерами, внешними ключами, хранимыми процедурами, создаваемых по умолчанию. И у всех этих типов есть одна общая черта: они все содержат ключевое поле ID с доменом DINTKEY (кроме таблицы-ссылки, которая создает для поля ID свой домен). Создать таблицу без полей нельзя.

Простая таблица с идентификатором

Данный тип создает самый простой вид таблиц. Таблица с таким типом по умолчанию содержит одно поле ID с доменом DINTKEY, которое и является ключевым. Также для этой таблицы создается первичный ключ, и триггер, который заполняет ключевое поле при вставке новой записи. Таблицы подобного типа обычно используются для небольших справочников, которые будут хранить редко редактируемые записи (например, справочник назначений платежа). Бизнес-объект, созданный на основе этой таблицы имеет класс TgdcAttrUserDefined и подтип, равный английскому названию таблицы.

Таблица с идентификатором

В отличие от предыдущего типа содержит также поля EDITORKEY (кто модифицировал), EDITIONDATE (когда модифицировал), DISABLED (отключено). На поле EDITORKEY создается внешний ключ на справочник контактов (в данном поле хранится ссылка не на пользователя, а на контактное лицо, который тот представляет). Кроме того, создаются два дополнительных триггера, которые перед вставкой записи и ее модификацией проверяют, указано ли контактное лицо, изменившее данные, и дата модификации. При отсутствии данной информации эти триггеры подставляют по умолчанию в качестве контакта администратора и в качестве даты модификации текущую дату. Данный тип таблиц используется уже для более сложных справочников, данные которых могут часто изменяться (для этого добавлено два поля для отслеживания изменений) и устаревать / становиться не актуальными (для этого используется поле DISABLED). Бизнес-объект, созданный на основе этой таблицы имеет класс TgdcAttrUserDefined и подтип, равный английскому названию таблицы.

Таблица со ссылкой

Данная таблица также по умолчанию содержит всего одно поле ID. Но в отличие от простой таблицы с идентификатором, ее первичный ключ является одновременно и внешним ключом. Здесь идет реализация связи один-к-одному. При создании такой таблицы указывается главная таблица, на которую будет ссылаться создаваемая. Создается пользовательский домен-ссылка со ссылкой на главную таблицу, который используется в качестве типа для поля ID. Такие таблицы используются для расширения справочников какими-либо редко используемыми признаками. Вынесение подобных признаков в отдельную таблицу помогает экономить физическое пространство, занимаемое базой. Однако увлекаться подобными таблицами не стоит, т.к. их приходиться отдельно обрабатывать. Бизнес-объект, созданный на основе этой таблицы имеет класс TgdcAttrUserDefined и подтип, равный английскому названию таблицы. Но более логично для подобной таблицы использовать не отдельный бизнес-объект, а подключать ее в запрос к основному бизнес-объекту по мере необходимости. Например, среди стандартных таблиц можно выделить таблицу gd_companycode, которая хранит некоторые признаки организаций. Данная таблица не имеет собственного бизнес-объекта, но подключается в запрос к бизнес-объекту Организации.

Простое дерево

Расширяет тип Таблица с идентификатором полем PARENT. Данное поле является ссылкой на собственное ключевое поле, следовательно, создается дополнительный внешний ключ. Таблицы подобного типа используются для организации структур небольшой вложенности. При очень большой вложенности вам будет тяжело извлечь все вложенные уровни по верхнему. Например, базовая таблица GD_COMMAND бизнес-объекта Исследователь является простым деревом. Бизнес-объект, созданный на основе этой таблицы имеет класс TgdcAttrUserDefinedTree и подтип, равный английскому названию таблицы.

Интервальное дерево

В дополнение к предыдущему типу содержит также такие поля, как LB (левая граница, сокращение от английского Left Bounday) и RB (правая граница, сокращение от английского Right Boundary). Самый сложный тип пользовательских таблиц. Левая и правая границы используются для определения вложенных уровней для текущей записи. Для их заполнения создается три хранимые процедуры (с названиями USR$_P_CHLDCT_<Название_таблицы>, USR$_P_EXLIM_<Название_таблицы>, USR$_P_RESTR_<Название_таблицы>), которые используются для пересчетов интервалов, специальные триггеры, которые вызывают пересчет интервалов через процедуры при изменении данных, и исключение USR$_E_TR_<Название_таблицы>, возникающее при попытке зациклить ветки дерева. На поля LB, RB создаются индексы. Создание подобной таблицы требует переподключения к базе. Таблицы данного типа используются для создания сложных вложенных структур с большим уровнем вложенности. Например, весь справочник клиентов реализован при помощи интервального дерева GD_CONTACT. Использование правых и левых границ помогает просматривать все контакты, которые входят не только в текущую папку, но и во вложенные папки. Бизнес-объект, созданный на основе этой таблицы имеет класс TgdcAttrUserDefinedLBRBTree и подтип, равный английскому наименованию таблицы.

Если первые четыре типа достаточно понятны, то интервальное дерево требует особого внимания. Рассмотрим его поподробнее.

Само название «Интервальное дерево» говорит нам о том, что работа с уровнями использует интервалы, при этом левая граница является началом интервала, а правая граница – окончанием. Самый верхний уровень дерева будет иметь наименьшую левую границу и наибольшую правую. Конечные узлы дерева, в которые никогда не входили никакие подуровни, будут иметь равные равные между собой правую и левую границы. Итак, что же означает интервал и его границы? Для простоты рассмотрим это на примере.

Предположим у нас есть папка «Организации», в которую вложены папки «ООО», «ОДО», «ИП». Все эти папки будут иметь родителем папку «Организации», т.е. в поле PARENT будет содержаться ключ папки «Организации». Пусть в каждой папке есть некий набор организаций. Мы имеем древовидную структуру, представленную на рисунке 8.4

Чтобы найти вложенные уровни по родителю, достаточно выполнить запрос вида:

```
SELECT *
FROM gd_contact z
WHERE
  z.parent = :parent
```

Однако если нам необходимо просмотреть все вложенные уровни, то такого запроса будет не достаточно. В нашем примере, нам придется выполнить подобный запрос четыре раза: первый раз для того, чтобы найти все вложенные папки в папку «Организация», и три раза, чтобы найти все вложенные организации в каждую из трех папок.

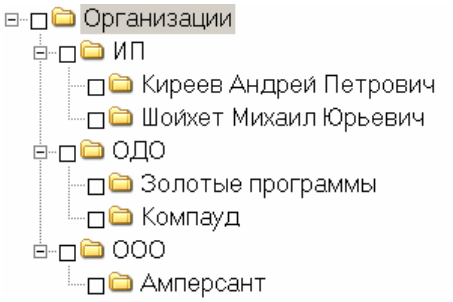


Рис. 47 Иллюстрация интервального дерева.

Использование правых и левых границ позволяет просмотреть все вложенные уровни, используя один запрос. Необходимо сопоставить каждому элементу выборки пару чисел так, чтобы пара чисел вложенного уровня входила в интервал, образованный парой чисел родителя. В отношении нашего примера можно представить следующую таблицу 8.2.

ID (Идентификатор)	PARENT (Идентификатор родителя)	LB (Левая граница)	RB (Правая граница)	NAME (Наименование)

147035451	**NULL**	**3000**	**6008**	**Организации**
147035452	147035451	3001	4000	ИП
147035459	147035452	3002	3002	Киреев Андрей Петрович
147035460	147035452	3003	3003	Шойхет Михаил Юрьевич
147035454	147035451	4102	5111	ОДО
147035455	147035454	4103	4103	Золотые программы
147035457	147035454	4104	4104	Компауд
147035453	147035451	4001	4101	ООО
147035458	147035453	4002	4002	Амперсант

Таблица 0.3

И этой таблицы видно, что самый верхний уровень имеет достаточно большой интервал от 3000 до 6008, вложенные уровни – интервалы поменьше, а листья дерева – интервал равный единице (т.е. правая граница совпадает с левой). Имея подобную информацию, можно вытянуть все вложенные уровни для указанного посредством одного запроса:

```
SELECT z.*
FROM gd_contact z
  JOIN gd_contact c ON c.lb <= z.lb AND c.rb >= z.rb
WHERE
  c.id = :id
```

Здесь в качестве параметра ID передается идентификатор узла, для которого мы будем выводить все вложенные уровни. Нестрогое сравнение правой и левой границ позволяет вывести и сам узел. Чтобы вывести только вложенные уровни необходимо в условии использовать строгое сравнение:

```
SELECT z.*
FROM gd_contact z
  JOIN gd_contact c ON c.lb < z.lb AND c.rb > z.rb
WHERE
  c.id = :id
```

Как же формируются интервалы, ведь нам известно, что пользователь при вводе новой записи их не задает? На триггерах перед вставкой и изменением записи осуществляется пересчет границ. При этом чем больше вложенности имеет узел, тем больше запаса в интервале ему предоставляется (сравните интервалы для самого высокого уровня и вложенных уровней). Это необходимо, чтобы пересчет и обновление интервалов не происходил при каждом изменении данных, что может существенно замедлить работу. Для пересчета границ используются следующие хранимые процедуры:

- USR$_P_CHLDCT_<Название_таблицы> - рекурсивно пробегается по всем вложенным уровням и проверяет их диапазоны. При необходимости присваивает новые значения границ. Используется в процедуре USR$_P_RESTR_<Название_таблицы>.
- USR$_P_EXLIM_<Название_таблицы> - проверяет диапазон для вставки новой записи и раздвигает его, если он мал. Используется в триггерах.
- USR$_P_RESTR_<Название_таблицы> - сжимает интервалы дерева.

Как мы видим в триггерах используется только одна процедура USR$_P_EXLIM_<Название_таблицы>. Другие две являются вспомогательными и могут использоваться для восстановления корректных интервалов после сбоев системы. При попытке зацикливания (например, при указании родителем собственного вложенного уровня) сработает внутренне исключение Interbase, и на экране появится сообщение:

```
Can not cycle tree branch!
```

Создание и редактирование таблиц средствами Гедымина

Бизнес-классом, отвечающим за работу с таблицами, является класс TgdcTable. При этом для каждого вида таблиц идет свой бизнес-класс, наследованный от TgdcTable:

- TgdcPrimeTable – отвечает за работу с простой таблицей с идентификатором;
- TgdcSimpleTable – для работы с таблицей с идентификатором;
- TgdcTableToTable – для работы с таблицей-ссылкой;
- TgdcTreeTable – для работы с простым деревом;
- TgdcLBRBTreeTable – для работы с интервальным деревом;
- TgdcUnknownTable – для работы с множествами. Также этот тип возвращается для всех остальных стандартных таблиц.

Здесь необходимо различать бизнес-объекты для работы с таблицами и бизнес-объекты, построенные на основе пользовательских таблиц. Бизнес-объекты для работы с таблицами отвечают за изменение структуры таблиц. Бизнес-объекты, построенные на основе таблиц, отвечают за изменение данных в этих таблицах.

Чтобы создать новую таблицу в системе Гедымин, необходимо выбрать в исследователе пункт Сервис \ Атрибуты \ Таблицы. Перед вами появится форма просмотра таблиц. Создание таблицы также дозволено только администраторам сервера Interbase.

При добавлении новой таблицы вам будет предложено выбрать тип таблицы. С возможными типами мы ознакомились ранее. В соответствии с выбранным типом идет настройка первой закладки мастера. Диалог создания / редактирования таблицы состоит из четырех закладок:

- Таблица
- Поля
- Триггеры
- Индексы

Закладка **Таблица** содержит следующие поля для заполнения:

- **Название таблицы на английском языке** – собственно название таблицы. Если вы не укажете префикс USR$, то система сама добавит его. Максимальная длина названия – 31 символ (как впрочем и любого объекта БД), однако рекомендуется использовать названия не длиннее 14 символов, т.к. они участвуют в формировании названий других объектов базы данных при обработке запросов бизнес-объекта парсером.
- **Локализованное название таблицы** – название таблицы на национальном языке, понятное простому пользователю. Это название будет использовано в качестве наименования для отображения (функция бизнес-объекта GetDisplayName) при использовании данной таблицы как базовой для бизнес-объекта. Должно быть уникально в пределах базы.
- **Краткое название таблицы** – сокращенный вариант локализованного названия таблицы.
- **Описание таблицы** – содержит описание данных, которые будет хранить таблица.
- **Ветка для команды вызова в Исследователе** – содержит ссылку на ветку исследователя, в которой будет содержаться ветка для вызова формы просмотра созданной таблицы. Т.е. мы указываем существующую ветку-родителя, а платформа сама создает в ней ветку для просмотра таблицы. Если этот параметр не указывать, то никакой ветки в исследователе создано не будет.
- **Ссылка на таблицу(связь один к одному)** – этот параметр виден только для таблиц-ссылок. Указывает главную таблицу, на которую будет ссылаться первичный ключ создаваемой таблицы. При этом у главной таблицы должен быть простой первичный ключ.
- **Поле для отображения(на английском)** – текстовый необязательный параметр. Если ничего не указывать, то будет использовано поле для отображения (свойство бизнес-объекта GetListField) по умолчанию. Т.е. система сначала поищет, есть ли в таблице поле USR$NAME, если нет, то возьмет в качестве поля для отображения первое строковое поле, если строковых полей в таблице нет, то возьмет первое поле любого типа. Т.е. если вы хотите, чтобы в качестве поля для отображения у вас использовалось весьма конкретное поле, то укажите его название в «поле для отображения (на английском)». При этом, если в названии будет ошибка, то система просто проигнорирует данный параметр.

- **Поля для расширенного отображения (на английском через запятую)** – текстовый необязательный параметр. Указывает какие поля для расширенного отображения вы хотите использовать для данной таблицы. Например, полем для отображения для таблицы GD_DOCUMENT является поле NUMBER. Однако один номер документа не несет достаточной информации. Желательно выводить также дату документа. Поэтому в данном параметре для таблицы GD_DOCUMENT можно указать DOCUMENTDATE. Теперь, если SQL-парсер найдет в запросе ссылку на таблицу GD_DOCUMENT, то он добавит в запрос не только поле для вывода номера документа, но и поле для вывода даты документа.
- **Соответствующий бизнес-класс** – параметр заполняется системой Гедымин. Является информационным. Указывает пользователю, какой бизнес-класс представляет таблица.
- **Соответствующий подтип** – работает в паре с предыдущим параметром. Указывает пользователю, какой подтип бизнес-класса представляет таблица. Для стандартных таблиц (без префикса USR$) будет всегда пустым.

Закладка **Поля** предлагает вам добавить / отредактировать / удалить поля в созданной таблице. При переходе на эту закладку в режиме добавления таблицы вы увидите, что поля, заданные по умолчанию, уже созданы. Эти поля вы не сможете удалить, т.к. их наличие определяет тип таблицы. Мастер для создания полей мы рассмотрим несколько позднее.

Закладка Триггеры доступна только при модификации уже существующей таблицы. Т.к. для создания триггера Гедымин уже должен знать, какие поля есть в таблице. Верхняя часть закладки представляет собой дерево, в котором перечислены триггеры. Отключенные триггеры выделяются серым цветом. Панель посредине предоставляет пользователю возможность добавить / отредактировать / удалить триггер. Все эти действия определены только над пользовательскими триггерами. Нижняя часть закладки отображает тело триггера, на котором стоит курсор (смотри рисунок 8.5).

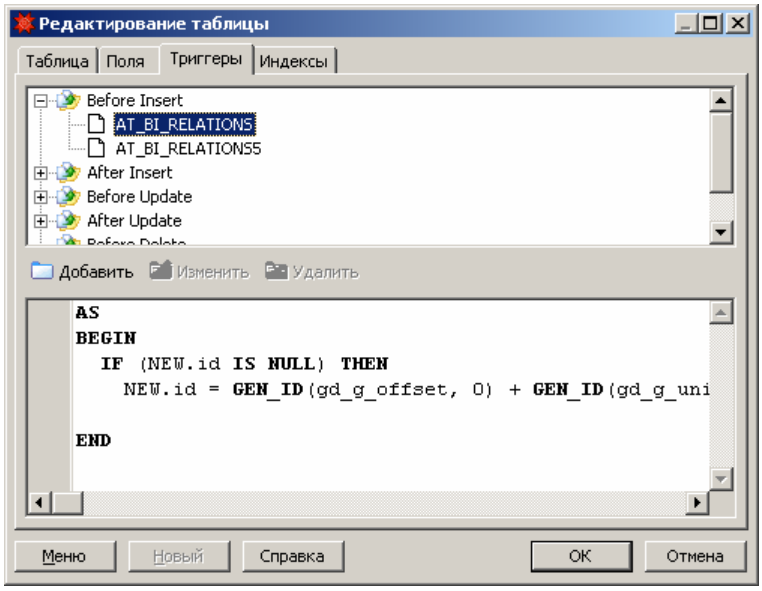


Рисунок 0.4

Закладка **Индексы** отображает созданные для текущей таблицы индексы, а также содержит панель с кнопками для создания / редактирования / удаления индекса. Диалог для модификации индекса будет рассмотрен несколько ниже.

В уже созданной таблице вы можете изменять параметры для локализации, указание полей для отображения, добавлять / изменять / удалять поля, триггеры, индексы.

Диалог для создания / изменения полей таблицы

Поле таблицы – это также объект базы данных. Но понятие таблицы без полей невозможно, поэтому мы рассмотрим диалог для создания / изменения полей в главе, посвященной таблице.

Бизнес-классом для работы с полями таблицы является класс TgdcTableField. Если рассматривать поле как бизнес-объект, то сразу можно оговориться, что его мастер-объектом будет являться таблица. Т.е.

базовая таблица AT_RELATION_FILEDS бизнес-объекта Поле имеет ссылку RELATIONKEY с доменом DMASTERKEY на базовую таблицу AT_RELATIONS бизнес-объекта Таблица.

Диалог бизнес-объекта Поле можно вызвать и с диалога редактирования таблицы, и с формы просмотра таблиц и полей. Различие в вызове имеет некоторые отличия при создании полей. Если мы создаем поля из диалога редактирования таблицы, то они будут созданы не при закрытии диалога редактирования поля, а по завершении работы с таблицей вообще. И их создание может идти не в том порядке, в котором мы их описывали (поэтому через диалог редактирования таблиц запрещено создавать вычисляемые поля – они могут содержать ссылки на другие поля таблицы, которые будут созданы позже). При создании полей через форму просмотра, поле создается каждый раз при закрытии диалога редактирования поля (или нажатии на нем кнопки Новый). Это позволяет более точно задать порядок в котором будут созданы поля, но несколько замедляет работу, если вам необходимо добавить много полей.

Мастер Гедымина для создания полей имеет три закладки:

- Общие
- Визуальные
- Объекты

Закладка Общие содержит следующие параметры:

- **Таблица** – нередактируемый параметр, указывает пользователю, для какой таблицы создается поле.
- **Название поля на английском языке** – собственно название поля, длиной не более 31 символа. Используется в запросах. Желательно не создавать поля с очень длинными названиями, т.к. их названия могут использоваться для формирования наименований других объектов БД.
- **Локализованное название** – название поля на национальном языке. Содержит краткую характеристику поля.
- **Краткое название** – укороченный вариант локализованного названия. Краткое название используется в качестве названия колонок грида на формах отображения.
- **Описание поля** – подробная характеристика поля.
- **Тип поля** – здесь вы можете указать домен или установить флаг вычисляемое поле. При выборе домена если у поля не указаны локализованные названия, то они подставляются из локализованных названий домена.
- **Not NULL** – используется только при указании домена поля. Говорит о том, что поле не может быть пустым. Первоначально устанавливается из установок домена. Если домен имеет флаг Not NULL, то снять этот флаг у поля нельзя. Обратное возможно.
- **Значение по умолчанию** – хранит значение подставляемое в поле при добавлении новой записи в таблицу. Первоначально устанавливается из установок домена. Недоступно, если поле вычисляемое.
- **Правило удаления** – видимо только для поля-ссылки. Может принимать одно из четырех значений:
    - NO ACTION - запрещать удаление при удалении главной записи (т.е. вы не сможете удалить главную запись, пока на нее существует ссылка);
    - CASCADE - удалять, если удаляется главная запись (этот принцип используется в сложных документах – нет смысла удалять сначала все позиции документа, а только затем шапку, гораздо логичнее, чтобы при удалении шапки позиции удалялись автоматически);
    - SET NULL - устанавливать значение NULL (главная запись удалиться, а ссылки на нее будут замещены NULL-значением);
    - SET DEFAULT - устанавливать значение по умолчанию (аналогично предыдущему, только вместо NULL на место ссылки будет установлено значение, которое задано для поля по умолчанию).
- **Выражение для вычисляемого поля на SQL** – хранит выражение, по которому будет вычисляться значение поля. Доступно только если поле является вычисляемым.

Вычисляемое поле – это поле, значение которого вычисляется динамически. В выражении для вычисления значения поля могут участвовать поля текущей таблицы, SQL-функции, также, используя подзапрос, можно обратиться к полям другой таблицы. Для вычисляемого поля всегда создается системный домен.

Закладка **Визуальные настройки** первоначально заполняется из установок домена. Содержит следующие параметры:

- Ширина поля при отображении
- Поле является видимым по умолчанию
- Запретить редактирование поля
- Формат вывода
- Выравнивание
- Бизнес-класс
- Подтип класса

Расшифровку этих параметров можно посмотреть в пункте 8.1.1 «Создание домена в системе Гедымин».

Закладка **Объекты** содержит дерево доступных бизнес-объектов. В этом дереве птичками обозначены объекты, для которых данное поле будет видимым. если не обозначен ни один объект, поле будет видимым для всех. Видимость распространяется не только на объекты отмеченных классов, но и на объекты дочерних классов. Т.е. если отметить TgdcBase (по умолчанию именно он и отмечен), то поле будет видимым для всех.

Осталось только пояснить, что мы понимаем под видимостью полей в бизнес-объектах. Нам известно, что бизнес-объект – это набор данных, выборка и изменение которых идет посредством выполнения SQL-запросов. В данном случае видимость поля означает попадание его в SQL-запрос. Если поле не попадет в SelectSQL, то оно, естественно, не будет отображаться. Кто-нибудь спросит: «Во-первых, зачем прятать поле, если мы его создали; во-вторых, какое отношение к данному полю имеют все бизнес-объекты, если известно, что каждой таблице соответствует один бизнес-объект?» Ответ на эти два вопроса будет один: данная закладка предназначена, чтобы прятать пользовательские поля в объектах, для которых текущая таблица не является базовой. Дело в том, что SQL-парсер при обработке SQL-запроса вытягивает все пользовательские поля для всех таблиц, входящих в запрос. Например, вам необходимо для каждой компании отображать ее главный счет, следовательно вам нужно в запрос бизнес-объекта Организации добавить таблицу GD_COMPANYACCOUNT и выводить только одно ее поле ACCOUNT. Однако, если в таблице GD_COMPANYACCOUNT есть пользовательские поля, то SQL-парсер добавит их автоматически в SELECT-часть запроса для объекта Организации. Чтобы этого не произошло, для всех пользовательских полей таблицы GD_COMPANYACCOUNT необходимо указать, что они должны отображаться только для бизнес-объекта Расчетный счет (класс TgdcAccount). Чтобы пользовательские поля не попадали в запрос, можно также использовать свойство бизнес-объекта SQLSetup.

Генераторы

Генератор – это специфический объект базы данных. Употребляя слово «специфический» мы хотим сказать, что не все базы данных предоставляют механизм генераторов. В Interbase генераторы используются достаточно широко.

Мы уже говорили о первичных ключах таблиц, и о том, что чаще всего это простые целочисленные ключи. Как же обеспечить их уникальность? Можно каждый раз выполнять запрос к таблице и искать максимальное значение ее ключа и затем найденное значение увеличивать на единицу. Однако это неудобно, может быть ощутимо по времени, а также может вызвать конфликт, если другой пользователь работает с этой же таблицей при открытой транзакции на изменение. В данном случае генератор является достаточно удобным и быстрым средством.

Что же такое генератор? Генератор – это, попросту говоря, именованный счетчик. При этом приращение генератора работает на собственной внутренней транзакции. Т.е. вне зависимости от того, добавите вы запись в таблицу или откатите ее добавление, если вы уже успели затребовать новое значение из генератора, то он будет хранить это новое значение.

В основном генераторы используются в триггерах до вставки записи для присвоения нового значения ключа. Кто-то использует для каждой таблицы свой генератор, платформа Гедымин использует для всех таблиц один генератор GD_G_UNIQUE, что обеспечивает уникальность ключей в пределах базы.

Чтобы получить новое значение генератора через запрос, необходимо выполнить следующую инструкцию:

```
SELECT GEN_ID(gd_g_unique, 1)
FROM rdb$database
```

Функция GET_ID указывает SQL, что будет работа с генератором, первый параметр содержит название генератора, второй – шаг приращения. Если вторым параметром передать ноль, то запрос вернет текущее значение генератора.

В триггере же достаточно указать, что вы новому значению ключевого поля присваиваете результат от выполнения функции GEN_ID.

```
NEW.ID = GEN_ID(gd_g_unique, 1)
```

В существующих триггерах вы можете увидеть, что при присвоении нового ключа используется еще один триггер GD_G_OFFSET. Этот генератор устарел, и уже нигде не используется. Вы можете смело его опускать.

Создать новый генератор средствами Гедымина нельзя, и бизнес-класса для него не существует.

Стандартные генераторы

Ниже приводится список генераторов, поставляемых с эталонной базой данных.

Генератор	Комментарий
gd_g_attr_version	Увеличивается каждый раз, когда происходит изменение структуры базы данных или изменяется информация в at_ таблицах. Вместо того, чтобы каждый раз при подключении к базе данных считывать ее структуру, Гедымин кэширует необходимую информацию, сохраняя ее на локальном жестком диске. Вместе с кэшированными данными сохраняется значение генератора gd_g_attr_version. При очередном подключении проверяется значение генератора из кэша и полученное из базы данных, если они совпадают, значит структура не изменялась и можно воспользоваться кэшем, если нет, то необходимо перечитать структуру базы данных.
gd_g_block	Используется для блокировки периода. Если равен 0, то блокировка выключена. В случае, если блокировка включена, генератор хранит приведенную к целому числу дату блокировки.
gd_g_block_group	Данный генератор хранит битовую маску групп пользователей на которых не распространяется блокировка периода. Используется только, если блокировка периода включена.
gd_g_dbid	Уникальный идентификатор базы данных. Если равен 0, то при очередном подключении Гедымин сгенерирует новый идентификатор и запишет его в этот генератор. Значение генератора используется при формировании РУИДа записи. Текущее значение идентификатора базы данных можно получить через свойство DBID глобального объекта IBLogin.
gd_g_functionch	Аналогично генератору gd_g_attr_version, данный генератор используется для контроля актуальности кэша скрипт-функций. Увеличивается каждый раз, когда изменяются данные в таблице gd_function.
gd_g_offset	На данный момент не используется. Значение 0.
gd_g_session_id	Порядковый номер сессии. Увеличивается при каждом подключении к базе данных. Текущий номер сессии можно получить через поле SessionID глобального объекта IBLogin.
gd_g_triggercross	При добавлении в пользовательскую таблицу поля типа множество в базе данных создается т.н. кросс таблица, которая содержит две ссылки: на таблицу, в которой добавлено поле-множество и на справочник, содержащий элементы множества. Генератор gd_g_triggercross используется для обеспечения уникальности имени

	кросс таблицы, которое формируется как USR$CROSS<NNN>, где <NNN> — значение генератора. Увеличивается, каждый раз при создании очередной кросс таблицы.
gd_g_unique	Используется для генерации уникального идентификатора записи. Начальное значение 147000000.
inv_g_balancenum	Используется для визуальной организации выбора складских остатков.

Индексы

Индексы – это механизм для улучшения быстродействия поиска данных. Индекс определяет столбцы которые могут быть использованы для эффективного поиска и сортировки в таблице. InterBase автоматически определят индексы для первичных и внешних ключей таблицы.

Существует достаточно распространенное мнение, что по индексированному полю поиск идет быстрее, чем по неиндексированному. Однако это не всегда так. Для небольших таблиц нет необходимости создавать индекс, т.к. запрос отработает быстрее, если переберет все записи, чем будет отбирать их по индексу. В случае маленькой таблицы при использовании индекса будет больше обращений к страницам базы данных, нежели при простом переборе.

Структура индексов - наиболее важный момент при проектировании структуры базы данных. Эффективность оптимизации запросов в InterBase в огромной степени зависит от правильной структуры индексов. В большинстве случаев используется примерно следующий алгоритм создания индексов - сначала определяются первичный ключ и уникальные индексы по тем полям, где необходимо исключить дублирование значений, потом создаются неуникальные индексы для внешних ссылок (foreign key) и для полей, по которым наиболее часто происходит поиск/сортировка. В большинстве случаев это работает. Однако надо иметь ввиду, что эффективность неуникальных индексов резко падает при большом количестве одинаковых величин в столбце и в результате наличие индекса в некоторых ситуациях может тормозить, а не ускорять выполнение запроса.

Не следует перегружать таблицу индексами, т.к. каждый индекс требует дополнительной памяти. Interbase автоматически создает индексы для первичного ключа и внешних ключей (в отличие от некоторых других баз данных). При создании дополнительных индексов необходимо сначала проанализировать, по каким параметрам будет чаще идти поиск (возможно поиск записей будет идти по нескольким параметрам, тогда лучше создать один сложный индекс, чем несколько простых индексов). Кроме того, если у вас на таблице есть несколько индексов, которые затрагивают одно и то же поле, то лучше проанализируйте вашу структуру еще раз. Возможно некоторые из этих индексов лишние.

Существует еще такое понятие, как статистика индексов. Статистика, показывает насколько у вас разные значения в поле таблицы, на основании которого у вас построен индекс. Считается она следующим образом:

Величина статистики индекса = 1 / Количество различных значений, содержащихся в данном поле таблицы.

Т.е., например, в таблице GD_CONTACT поле CONTACTTYPE может принимать значения от 0 до 5. Если мы создадим индекс на это поле и внесем хотя бы по одной записи с каждым типом, то величина статистики будет 1 / 6 = 0,1666(6), что является не очень хорошим результатом. Здесь, однако, нужно учитывать, что статистика только в некоторых случаях (как в примере с GD_CONTACT и CONTACTTYPE) не будет сильно изменяться при добавлении данных. Чаще всего статистика индексов достаточно сильно зависит от создания новых записей.

Например, статистика индекса, созданного на первичный ключ, может быть равна 1, если у вас всего одна запись. Как вы понимаете, в данном случае использование индекса будет нецелесообразно. Однако, если мы добавим в таблицу 300 записей, то статистика изменится и станет равной 1 / 300 = 0,00333(3), и использование индекса в данном случае будет оправдано.

Теперь поговорим о том, какие механизмы для работы с индексами предоставляет Гедымин. Вы можете создавать / изменять / удалять пользовательские индексы, и в данном случае не иметь практически никаких ограничений. Но есть еще и так называемые стандартные индексы (здесь мы не затрагиваем индексов, создаваемых Interbase автоматически при добавлении первичных и внешних ключей). Стандартные индексы можно сделать неактивными и оптимизатор запроса не будет их учитывать.

Операции над индексами можно производить из диалога редактирования таблицы. Существует также и форма просмотра индексов, представляющая собой форму с master-detail связью, где мастер-объект – это таблица, а детальный объект – это индекс. Изначально это форма не доступна из исследователя. Однако вы сами можете добавить эту ветку при желании. Для этого необходимо добавить новый пункт исследователя, выбрать переключатель «Бизнес-объект», в выпадающем списке выбрать класс TgdcIndex (бизнес-класс для работы с индексами).

При открытии формы просмотра индексов вам будет задан вопрос, синхронизировать ли индексы с базой данных. Как мы помним для ускорения часть данных кэшируется на локальном компьютере. Синхронизация таких метаданных, как таблицы, домены, поля происходит достаточно быстро, а потому не требует подтверждения пользователя. Синхронизация же индексов иногда требует ощутимых временных затрат.

Итак, что представляет собой диалог редактирования индекса (смотрите Рис. 48 Диалоговое окно создания или изменения индекса):

- **Название индекса** – собственно название индекса на английском языке. Не более 31 символа. Желательно давать более менее осмысленные названия, которые бы говорили о том, какие поля входят в индекс и для какой таблицы. Название индекса, в отличие от названия полей является уникальным в пределах базы.
- **Активный** – флаг активности индекса. При создании нового индекса всегда включен. При отключении данного флага оптимизатор запросов перестает учитывать индекс.
- **Уникальный** – говорит о том, что поле (или совокупность полей) может иметь только уникальные значения в пределах таблицы.
- **Порядок** – порядок построения индекса. Может быть по возрастанию и по убыванию.

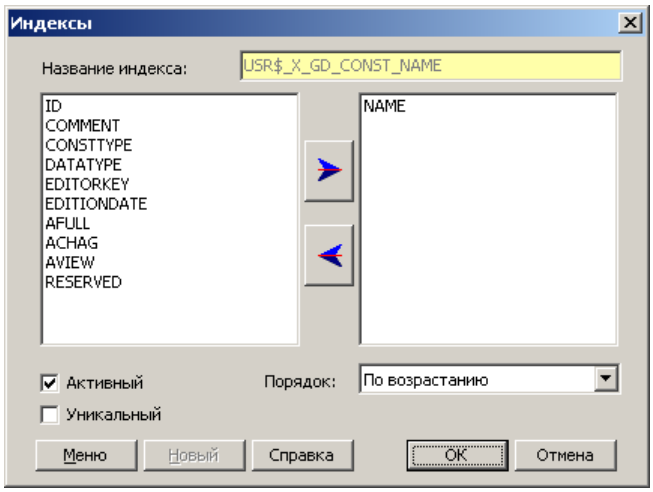


Рис. 48 Диалоговое окно создания или изменения индекса

Также мы видим два списка и кнопки навигации между ними. Левый список предоставляет все поля таблицы, за исключением тех, что пользователь уже выбрал в индекс, правый – выбранные для индекса поля. Порядок полей (если их выбрано больше одного) в индексе имеет значение. Сначала поиск будет идти по первому полю в индексе, затем по второму и т.д.

Хранимые процедуры

Хранимая процедура – это объект базы данных, представляющий собой откомпилированную во внутреннее представление InterBase подпрограмму, написанную на специальном языке. Компилятор данного языка встроен в ядро сервера InterBase Хранимую процедуру можно вызывать из клиентских приложений, из триггеров и других хранимых процедур, использовать в запросах. Хранимая процедура выполняется внутри серверного процесса и может манипулировать данными в базе данных, а также возвращать результаты своего выполнения.

В хранимых процедурах можно использовать как обычные SQL-запросы (на выборку и изменение данных), так и средства организации условных ветвлений и циклов (IF, WHILE), а также генерировать исключения. Язык хранимых процедур позволяет реализовать сложные алгоритмы работы с данными, а, благодаря ориентированности на работу с реляционными данными, хранимые процедуры получаются

значительно компактнее аналогичных процедур на традиционных языках. Кроме того в некоторых случаях хранимая процедура будет работать быстрее, чем, например, макрос.

Бизнес-классом, отвечающим за работу с хранимыми процедурами, является класс TgdcStoredProc. Чтобы вызвать форму просмотра вам необходимо выбрать в исследователе пункт Сервис / Атрибуты / Процедуры. Вы можете добавлять / изменять / удалять любые пользовательские процедуры.

Диалог редактирования процедуры содержит две закладки:

- Наименование
- Текст процедуры

Закладка Наименование содержит два поля для ввода:

- **Наименование процедуры** – наименование процедуры на английском языке. Уникально в пределах базы. Не более 31 символа.
- **Описание процедуры** – содержит краткую характеристику процедуры, что она делает, какие данные необходимо подать на вход, что она вернет. Необязательно для заполнения.

Закладка **Текст процедуры** содержит поле для ввода текста процедуры.

Триггеры

Триггер в InterBase - это особый вид хранимой процедуры, которая выполняется автоматически при вставке, удалении или модификации записи таблицы или представления (view). Триггеры могут "срабатывать" непосредственно до или сразу же после указанного события. Т.е. вы можете создать шесть видов триггеров:

- До вставки (Before Insert)
- После вставки (After Insert)
- До изменения (Before Update)
- После изменения (After Update)
- До удаления (Before Delete)
- После удаления (After Delete)

Триггеры «до события» обычно используются для проверки введенных значений, указания значений по умолчанию и других операций, которые нужно выполнить пока данные в базе еще не успели измениться. Триггеры «после события» используются для синхронизации данных. Например, после добавления записи в одну таблицу, вам необходимо добавлять запись в другую таблицу. Все действия, производимые в триггерах, выполняются на одной транзакции с самой операцией изменения данных. Т.е., если у вас в триггере возникла ошибка, то все действия будут откачены.

В отличие от обычных хранимых процедур триггеры не могут иметь входных и выходных параметров, использоваться в запросах или непосредственно вызываться по имени.

Операции над триггерами можно производить из диалога редактирования таблицы. Существует также и форма просмотра триггеров, представляющая собой форму с master-detail связью, где мастер-объект – это таблица, а детальный объект – это триггер. Изначально это форма не доступна из исследователя. Однако вы сами можете добавить эту ветку при желании. Для этого необходимо добавить новый пункт исследователя, выбрать переключатель «Бизнес-объект», в выпадающем списке выбрать класс TgdcTrigger (бизнес-класс для работы с триггерами).

При открытии формы просмотра триггеров вам будет задан вопрос, синхронизировать ли триггеры с базой данных. Все дело в том, что в отличие от остальных объектов базы данных, отследить изменение триггеров практически невозможно. Сам по себе процесс синхронизации триггеров не очень быстрый, поэтому производить его каждый раз при загрузке системы нецелесообразно.

Диалог редактирования триггера (смотрите рисунок 8.7) содержит следующие поля для заполнения:

- **Наименование** – наименование триггера на английском языке. Уникально в пределах базы. Хотя, это и не обязательно, рекомендуется чтобы наименования триггеров соответствовали следующему шаблону: PREFIX_TYPE_TABLE_SUFFIX. Где,

- **PREFIX — префикс подсистемы таблицы, для которой определен настоящий триггер;**
- **TYPE — тип триггера. Двухбуквенная аббревиатура.**
  - **BI — Before Insert;**
  - **AI — After Insert;**
  - **BU — Before Update;**
  - **AU — After Update;**
  - **BD — Before Delete;**
  - **AD — After Delete;**
- **TABLE — имя таблицы (без префикса) для которой определен триггер;**
- **SUFFIX** — суфикс, делающий имя тригера уникальным. Суфикс необходим, если для одной и той же таблицы на одну и ту же операцию создано более одного триггера. В качестве суффикса может использоваться номер позиции триггера.

- **Позиция** – позиция триггера. Вы можете создать несколько триггеров на одно событие. Позиция указывает порядок их выполнения: первым выполнится триггер с нулевой позицией, вторым – с первой позицией и т.д. При совпадении позиций Interbase сам будет решать какой триггер выполнить первым.
- **Тип** – тип триггера. Указывает для какого события он вызывается.
- **Активный** – говорит о том, что триггер включен. Если этот флаг снять, триггер не будет вызываться при возникновении события.
- **Поле для ввода тела триггера** – содержит тело триггера без шапки.

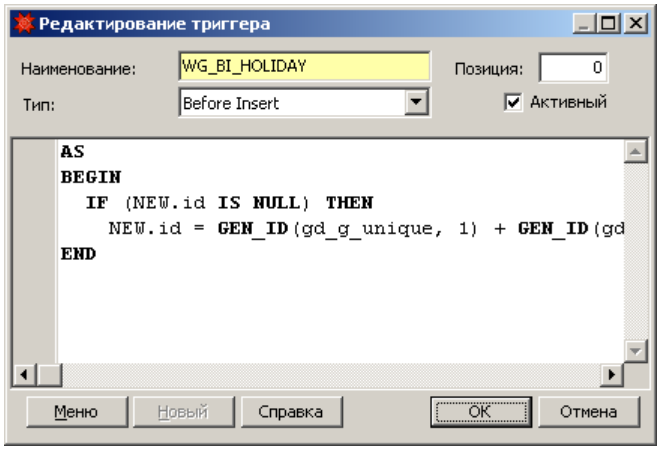


Рис. 49 Диалоговое окно просмотра и изменения триггера

Представления (проекции)

Представление (VIEW) - это виртуальная таблица, созданная на основе запроса к обычным таблицам. Представление реализовано как запрос, хранящийся на сервере и выполняющийся всякий раз, когда происходит обращение к представлению. В некоторой вы также можете встретить также термин «Проекция».

Для чего нужны представления? Например, один и тот же запрос вы используете в различных местах. Этот запрос может быть достаточно массивным, содержать много связанных таблиц. Представление предлагает «зашить» (спрятать) ваш запрос. Теперь, если вам нужно обратиться к данным, то достаточно выполнить коротенький запрос. Например, представление GD_V_COMPANY возвращает всю необходимую информацию для работы с организациями. При этом «лишние» (служебные) поля не отображаются. Сам по себе запрос выглядит следующим образом:

```
CREATE VIEW GD_V_COMPANY
 (ID,COMPNAME, COMPFULLNAME, COMPANYTYPE, COMPLB, COMPRB, AFULL,
```

```
   ACHAG, AVIEW, ADDRESS, CITY, COUNTRY, PHONE, FAX, ACCOUNT,
   BANKCODE, BANKMFO, BANKNAME, BANKADDRESS, BANKCITY,
   BANKCOUNTRY, TAXID, OKPO, LICENCE, OKNH, SOATO, SOOU)
AS
SELECT
  C.ID, C.NAME AS COMPNAME, COMP.FULLNAME AS COMPFULLNAME,
  COMP.COMPANYTYPE, C.LB AS COMPLB, C.RB AS COMPRB,
  C.AFULL, C.ACHAG, C.AVIEW,
  C.ADDRESS, C.CITY, C.COUNTRY, C.PHONE, C.FAX,
  AC.ACCOUNT, BANK.BANKCODE, BANK.BANKMFO,
  BANKC.NAME AS BANKNAME, BANKC.ADDRESS AS BANKADDRESS,
  BANKC.CITY AS BANKCITY, BANKC.COUNTRY AS BANKCOUNTRY,
  CC.TAXID, CC.OKPO, CC.LICENCE, CC.OKNH, CC.SOATO, CC.SOOU

FROM
  GD_CONTACT C
  JOIN GD_COMPANY COMP ON COMP.CONTACTKEY = C.ID
    LEFT JOIN GD_COMPANYACCOUNT AC
      ON COMP.COMPANYACCOUNTKEY = AC.ID
    LEFT JOIN GD_BANK BANK
      ON AC.BANKKEY = BANK.BANKKEY
    LEFT JOIN GD_COMPANYCODE CC
      ON COMP.CONTACTKEY = CC.COMPANYKEY
    LEFT JOIN GD_CONTACT BANKC
      ON BANK.BANKKEY = BANKC.ID
```

Согласитесь, что писать подобный запрос каждый раз, когда вам понадобятся данные по организациям в высшей степени утомительно. Однако, создав представление на основе этого запроса, вы сможете ограничиться следующим:

```
SELECT *
FROM gd_v_company
```

Конечно, имеются и ограничения. В системе Гедымин вы не можете изменять данные непосредственно через представление.

Бизнес-классом для работы с представлениям является класс TgdcView, а для работы с полями представления TgdcViewField. Как вы понимаете, добавить поле в представление (или удалить его), можно только изменив запрос, на основании которого создано представление. Чтобы вызвать форму просмотра вам необходимо выбрать в исследователе пункт Сервис / Атрибуты / Представления. Вы можете добавлять / изменять / удалять любые пользовательские представления.

Диалог редактирования представления содержит четыре закладки:

- Представление
- Текст представления
- Поля
- Триггеры

Закладка **Представление** содержит следующие поля для заполнения:

- **Название представления(на английском языке)** – собственно название представления. До 31 символа. Внимание: название представлений и название таблиц должны быть уникальны в пределах базы, т.е. они не могут пересекаться.
- **Локализованное название представления** – название на национальном языке.
- **Краткое название представления** – аналогично локализованному, только в краткой форме.
- **Описание представления** – характеристика представления, какие данные оно отображает.
- **Ветка для команды вызова в Исследователе** – содержит ссылку на ветку исследователя, в которой будет содержаться ветка для вызова формы просмотра созданной таблицы. Т.е. мы указываем существующую ветку-родителя, а платформа сама создает в ней ветку для просмотра таблицы. Если этот параметр не указывать, то никакой ветки в исследователе создано не будет.

- **Поле для отображения(на английском)** – аналогично параметру на диалоге редактирования таблиц.
- **Поля для расширенного отображения (на английском через запятую)** – аналогично параметру на диалоге редактирования таблиц.
- **Соответствующий бизнес-класс** – параметр заполняется системой Гедымин. Является информационным. Указывает пользователю, какой бизнес-класс представляет проекция.
- **Соответствующий подтип** – работает в паре с предыдущим параметром. Указывает пользователю, какой подтип бизнес-класса представляет таблица. Для стандартных таблиц (без префикса USR$) будет всегда пустым.

Закладка **Текст представления** содержит поле для ввода запроса, на основании которого будет создано представление, и кнопку «Создать», по нажатию которой оно будет создано (или пересоздано – в режиме редактирования).

Закладка **Поля** появляется после создания представления. На ней вы можете просмотреть поля представления, а также отредактировать их визуальные характеристики.

Закладка **Триггеры** позволяет добавить триггеры, появляется после создания представления. Однако механизм изменения данных через представление в первой версии Гедымина еще не доработан. Поэтому особой пользы от этой закладки пока нет.

При создании представления через диалог редактирования будьте внимательны: для нормальной работы все as-наименования полей необходимо задать непосредственно в самом запросе. Вы можете даже не писать шапку с перечислением полей, платформа создаст ее сама. Т.е., возвращаясь к примеру GD_V_COMPANY. Для создания подобного представления вам было бы достаточно в поле для ввода запроса написать:

```
SELECT
  C.ID, C.NAME AS COMPNAME, COMP.FULLNAME AS COMPFULLNAME,
  COMP.COMPANYTYPE, C.LB AS COMPLB, C.RB AS COMPRB,
  C.AFULL, C.ACHAG, C.AVIEW,
  C.ADDRESS, C.CITY, C.COUNTRY, C.PHONE, C.FAX,
  AC.ACCOUNT, BANK.BANKCODE, BANK.BANKMFO,
  BANKC.NAME AS BANKNAME, BANKC.ADDRESS AS BANKADDRESS,
  BANKC.CITY AS BANKCITY, BANKC.COUNTRY AS BANKCOUNTRY,
  CC.TAXID, CC.OKPO, CC.LICENCE, CC.OKNH, CC.SOATO, CC.SOOU

FROM
  GD_CONTACT C
  JOIN GD_COMPANY COMP ON COMP.CONTACTKEY = C.ID
    LEFT JOIN GD_COMPANYACCOUNT AC
      ON COMP.COMPANYACCOUNTKEY = AC.ID
    LEFT JOIN GD_BANK BANK
      ON AC.BANKKEY = BANK.BANKKEY
    LEFT JOIN GD_COMPANYCODE CC
      ON COMP.CONTACTKEY = CC.COMPANYKEY
    LEFT JOIN GD_CONTACT BANKC
      ON BANK.BANKKEY = BANKC.ID
```

Если же вы зададите в шапке представления другие наименования для полей, то при переносе представления на другую базу вы получите неприятности.

Создание представления требует переподключения в базе данных. О переподключении к БД читайте далее.

Исключения

Одной из рассматриваемых особенностей языка хранимых процедур и триггеров InterBase является возможность использовать «исключения». Исключения InterBase во многом похожи на исключения других языков высокого уровня, однако имеют свои особенности. Фактически исключение InterBase – это сообщение об ошибке, которое имеет собственное, задаваемое программистом имя и текст сообщения об ошибке на английском языке. Гедымин предоставляет пользователю возможность локализовать сообщение исключения.

Бизнес-классом, отвечающим за работу с исключениями интербейза, является TgdcException. Открыть форму просмотра исключений можно из исследователя Сервис / Атрибуты / Исключения. Диалог редактирования вызывается стандартно и выглядит как на рисунке 8.8. Диалог содержит следующие поля для заполнения:

- **Наименование** – наименование исключения на английском языке, не более 31 символа. Уникально в пределах базы.
- **Сообщение на английском** – сообщение исключения на английском языке.
- **Локализованное сообщение** – перевод сообщения с английского на национальный язык.

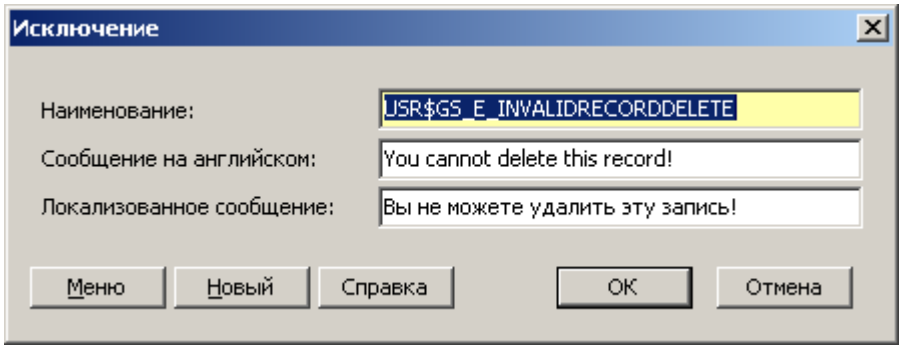


Рисунок 0.5

Вы можете создавать / изменять / удалять пользовательские исключения и модифицировать локализованное сообщение стандартных исключений.

Чтобы вызвать исключение из тела процедуры или триггера, необходимо написать следующую команду:

```
EXCEPTION <имя исключения>
```

Возникновение исключения откатывает целиком всю операцию.

Переподключение к базе данных. Окно состояния SQL

Некоторые операции по изменению структуры базы данных достаточно трудоемки и не могут выполняться на одной транзакции. Более того, часть подобных операций требует переподключения к базе данных. Например, создание внешних ключей требует монопольного подключения к базе данных. Когда вы создаете или изменяете объекты БД, используя средства Гедымина, то мастера сами выполняют пачку запросов. Те запросы, которые требуют переподключения, записываются в специальную таблицу AT_TRANSACTION (структура таблицы будет рассмотрена ниже). При следующем подключении к базе пользователю-администратору будет предложено в монопольном режиме (т.е. отключив всех остальных пользователей) выполнить данные запросы. В подобном режиме платформа сама определяет необходимость переподключения к базе для успешного выполнения всех операций. Те операции, которые не могут быть выполнены, удаляются из списка.

Кроме того, вы наверное обратили внимание, что при любом изменении структуры БД, на экране появляется окно Выполнение SQL команд (смотрите рисунок 8.9). Данное окно отображает последовательность выполняемых команд, состояние системы (имеется ввиду требуется ли переподключение к БД), возникшие ошибки. При этом служебная информация в обычном режиме выводится синим цветом, в режиме переподключения –зеленым; выполняемые команды – черным цветом, ошибки – красным цветом. Кроме того имеется еще строка состояния внизу окна, в которой отображается возникли ли ошибки в процессе выполнения и требуется ли переподключение к базе. Максимально окно может хранить до 64 Кб текста. При достижении предела окно очищается. Однако, если вас интересует полностью весь лог выполнения сложной операции, вы можете нажать кнопку Сохранить вверху окна и сохранить его в текстовый файл.

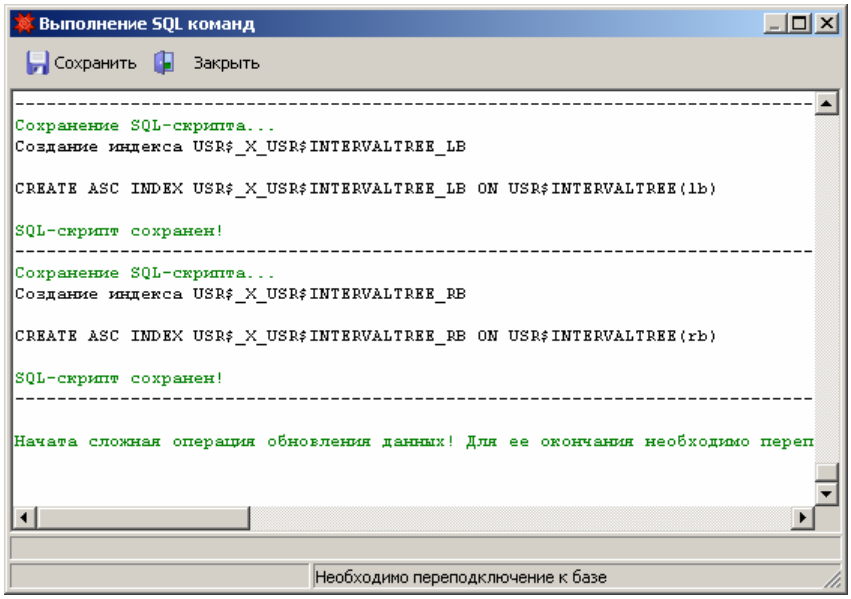


Рисунок 0.6

Если вас не интересует статистика выполнения команд, вы можете отключить ее, сняв флаг в Сервис / Опции / Выводить лог при загрузке / сохранении в поток.

Теперь рассмотрим подробнее структуру таблицы AT_TRANSACTION, в которую помещаются отложенные команды.

Поле	Домен	Первичный ключ	Описание
TRKEY	DINTKEY	x	Ключ транзакции. Данная таблица может хранить несколько операций, производимых на разных транзакциях.
NUMORDER	DSMALLINT	x	Порядковый номер команды внутри одной транзакции.
SCRIPT	DSCRIPT		Текст SQL-команды.
SUCCESSFULL	DBOOLEAN		Должна быть успешно завершена. Если этот флаг снят, то при возникновении ошибки команда просто пропускается. В обратном случае, пропускается не только текущая команда но и все последующие за ней на одной транзакции. Кроме того пользователю выдается запрос на удаление невыполненных команд.

Таблица 0.4

В заключение к выше сказанному можем повториться: все операции переподключения требуют монопольного доступа к БД. Если во время выполнения подобной операции кто-либо подключится к базе, то администратору будет выдано окно со списком подключившихся пользователей. Продолжить выполнение отложенных команд можно будет только после отключения лишних пользователей от базы.

Хитрый NULL

Данная глава относится не столько к базе данных, сколько к SQL вообще. Однако с NULL значениями связано столько ошибок, что мы считаем своим долгом заострить на них внимание.

Итак, что такое NULL? NULL – это ПУСТО. В принципе, это означает, что поле не содержит никакого значения. NULL может присутствовать как в текстовых, так и в числовых полях, или, проще говоря, поле любого типа может содержать NULL, если это не оговорено специально. Под фразой «оговорено специально» мы понимаем ограничения, налагаемые на содержимое полей.

Чем же так коварен NULL? Поля, содержащие NULL-значения не могут напрямую участвовать в операциях сравнения. Операции сравнения (>, <, = и т.д.), в которых хотя бы с одной стороны участвует NULL значение всегда вернут ЛОЖЬ (FALSE). Например, нет никакого смысла в следующем сравнении:

```
z.field = NULL
```

Даже если z.field содержит NULL подобное сравнение вернет FALSE. Если вам необходимо отфильтровать записи по NULL-содержимому, то следует использовать следующую конструкцию:

```
z.field IS NULL
```

Оператор IS выполняет те же функции для NULL значений, что и оператор = для остальных значений. Если же вам необходимо сравнить содержимое двух полей, и при этом вы считаете, что совпадение должно учитывать NULL-значения, то ваше сравнение будет выглядеть следующим образом:

```
(z.field1 = z.field2) OR (z.field1 IS NULL AND z.field2 IS NULL)
```

Теперь рассмотрим обработку NULL-значений полей из макросов. Обращение к содержимому поля через AsVariant или Value вернет вам значение поля изначального типа. Т.е., если поле хранило NULL, то именно NULL вы и получите.

```
MyValue = gdcObject.FieldByName("field").AsVariant
```

Однако обращение через AsInteger, AsFloat, AsCurrency – преобразет NULL значение в 0. Обращение через AsString вернет вместо NULL пустую строку «».

NULL-значения не могут содержаться в ключевых полях. Также не все версии Interbase и его клонов поддерживают NULL-значения в уникальных полях. Самая большая опасность таится при добавлении не NULL-полей (т.е. полей, которые не могут содержать NULL-значения) в уже используемые таблицы. Если при добавлении такого поля, вы не заполните его каким-то значением, то в вашей таблице могут остаться записи, в которых это поле будет NULL (те записи, которые были созданы до добавления поля). База данных с такой таблицей просто не восстановится из архива. Поэтому при архивировании БД средствами Гедымина можно установить флаг «Проверять целостность данных». Этот флаг проверяет, нет ли у вас в не NULL-полях NULL значений.

Глобальный объект atDatabase. Таблицы для хранения метаданных

Итак, практически любой объект базы данных может быть представлен в Гедымине как бизнес-объект. Это становится возможным за счет использования таблиц для хранения метаданных. Любая база данных Interbase имеет собственные системные объекты. Их название начинается с префикса RDB$. Однако содержимое данных системных таблиц не полностью отвечает требованиям бизнес-объекта, например, в них нет целочисленного первичного ключа. Поэтому в базе данных Гедымина есть дополнительные системные таблицы (их название начинается с AT_), которые хранят дополнительные данные об объектах базы данных (например, локализацию, визуальные настройки). Таблица 8.4 содержит перечисление служебных таблиц Гедымина.

Название таблицы	Описание
AT_EXCEPTIONS	Информация о исключениях БД.
AT_FIELDS	Информация о доменах БД.
AT_INDICES	Информация об индексах.
AT_PROCEDURES	Информация о хранимых процедурах.
AT_RELATIONS	Информация о таблицах и представлениях.
AT_RELATION_FIELDS	Информация о полях таблиц и представлений.
AT_SETTING	Информация о заголовках настроек
AT_SETTINGPOS	Информация о позициях настроек для хранения данных о бизнес-объектах.
AT_SETTING_STORAGE	Информация о позициях настроек для хранения веток хранилища Гедымина.
AT_TRANSACTION	Хранит отложенные операции (читайте о переподключении к БД)
AT_TRIGGERS	Информация о триггерах

Таблица 0.5

Данные служебных таблиц Гедымина синхронизируются с системными таблицами Interbase по мере необходимости. Единственно, что не хранят AT-таблицы, так это информацию о системных таблицах Interbase.

Используя данные AT-таблиц в совокупностями с системными таблицами Interbase можно узнать все связи между объектами базы данных. Иногда подобная информация может быть весьма полезна. Однако, чтобы ее получить, необходимо выполнить запрос к соответствующим служебным таблицам. Если вы в текущем сеансе работы не изменяли структуру базы данных, то для получения информации об объектах БД можно обойтись без лишних запросов и воспользоваться глобальным объектом atDatabase.

Глобальный объект **atDatabase** кэширует информацию о метаданных на локальном компьютере. Использование atDatabase, а не прямых запросов, имеет ряд преимуществ (впрочем как и любой кэш):

- некоторая информация может быть получена быстрее;
- не перегружается сеть;
- не используется лишний раз сервер базы данных.

Хотя, повторимся, использование кэша имеет и минусы. Информация в кэше обновляется в определенные промежутки времени (если взять atDatabase, то информация с БД в нем синхронизируется только при подключении к базе), а потому может устареть. Требование о переподключении к базе данных при определенных операциях выставляется также и для синхронизации atDatabase с измененной структурой. Хотя некоторые операции обновляют локальный кэш на том компьютере, где они были произведены, сами по себе.

Далее следует перечень свойств и объектов, к которым вы можете получить доступ через глобальный объект atDatabase. Объект atDatabase представляет собой экземпляр класса TatDatabase.

Свойства и методы TatDatabase

- FindRelationField(ARelationName: string; ARelationFieldName: string): TatRelationField – по названию таблицы (ARelationName) и названию поля (ARelationFieldName) находит объект Поле и возвращает на него ссылку. Если поле не найдено, то возвращает nil.
- Fields: TatFields – возвращает список всех доменов, созданных на текущей базе, включая системные домены. Это необходимо, т.к. системные домены создаются на пользовательские вычисляемые поля и поля в представлениях.
- ForeignKeys: TatForeignKeys – возвращает список всех внешних ключей.
- InMultiConnection: Boolean – возвращает Истина, если база данных требует переподключения, Ложь – в обратном случае.
- PrimaryKeys: TatPrimaryKeys – возвращает список всех первичных ключей.
- Relations: TatRelations – возвращает список всех таблиц, исключая системные.

Класс TatFields представляет собой список для хранения всех доступных доменов в системе Гедымин.
Свойства и методы TatFields

- ByFieldName(AFieldName: string): TatField – возвращает ссылку на домен по его английскому названию.
- ByID(ID: Integer): TatField – возвращает ссылку на домен по его идентификатору. В данном случае подразумевается идентификатор из таблицы AT_FIELDS.
- Count: Integer – возвращает количество доменов.
- Items(Index: Integer): TatField – т.к. данный класс представляет собой список, то мы можем обратиться к каждому элементу списка по индексу. Данное свойство возвращает ссылку на домен по его индексу в списке.

Класс TatField предназначен для описания каждого домена по отдельности.

Свойства и методы TatField

- Alignment: Integer (перечисление) – указывает тип выравнивания данных. Примечание, что данных тип является перечислением означает, что он может принимать одну из ряда констант,

преобразованную в целое число. В данном случае имеет место следующий ряд и сопоставленные ему целые значения: faLeft = 0 (влево), faRight = 1 (вправо), faCenter = 2 (по центру).

- ColWidth: Integer – ширина колонки при отображении поля с данным доменом.
- Description: String – описание домена.
- Disabled: Boolean – указывает, что домен отключен.
- FieldLength: Integer – размер домена в байтах.
- FieldName: String – название домена на английском языке.
- FieldType: Integer (перечисление) – тип домена на Делфи(ftUnknown, ftString, ftSmallint, ftInteger, ftWord, ftBoolean, ftFloat, ftCurrency, ftBCD, ftDate, ftTime, ftDateTime, ftBytes, ftVarBytes, ftAutoInc, ftBlob, ftMemo, ftGraphic, ftFmtMemo, ftParadoxOle, ftDBaseOle, ftTypedBinary, ftCursor, ftFixedChar, ftWideString, ftLargeint, ftADT, ftArray, ftReference, ftDataSet, ftOraBlob, ftOraClob, ftVariant, ftInterface, ftIDispatch, ftGuid). Во всех перечисляемых типах нумерация идет с нуля.
- FormatString: String – строка форматирования вывода поля. Задает маску формата с использованием правил, установленных в Делфи.
- gdClassName: String – название класса бизнес-объекта. Используется только для доменов-ссылок. Для остальных возвращает пустую строку.
- gdSubType: String – название подтипа бизнес-объекта. Используется в паре с gdClassName.
- ID: Integer – идентификатор домена (из таблицы AT_FIELDS).
- IsNullable: Boolean – если Истина, значит домен поддерживает Null-значения. В обратном случае – Ложь.
- IsSystem: Boolean – указывает, что домен является системным, т.е. начинается с префикса RDB$.
- IsUserDefined: Boolean – указывает, что домен является пользовательским, т.е. начинается с префикса USR$.
- LName: String – локальное наименование домена на национальном языке.
- ReadOnly: Boolean – если Истина, то поле с данным доменом предназначено только для чтения. Т.е. данные в подобное поле не могут быть занесены пользователем.
- RefCondition: String – условие, накладываемое на выборку данных из таблицы, на которую ссылается домен. Только для доменов-ссылок.
- RefListField: TatRelationField – поле для отображения в таблице, на которую ссылается домен. Только для доменов-ссылок.
- RefListFieldName: String – название поля для отображения в таблице, на которую ссылается домен. Только для доменов-ссылок.
- RefTable: TatRelation – таблица, на которую ссылается домен. Только для доменов-ссылок.
- RefTableName: String – название таблицы, на которую ссылается домен. Только для доменов-ссылок.
- SetCondition: String - условие, накладываемое на выборку данных из таблицы, на основании которой будет создано множество. Только для доменов-множеств.
- SetListField: TatRelationField – поле для отображения в таблице, на основании которой будет создано множество. Только для доменов-множеств.
- SetListFieldName: String – название поля для отображения в таблице, на основании которой будет создано множество. Только для доменов-множеств.
- SetTable: TatRelation – таблица, на основании которой будет создано множество. Только для доменов-множеств.
- SetTableName: String – название таблицы, на основании которой будет создано множество. Только для доменов-множеств.
- Visible: Boolean – флаг видимости.

Класс TatRelations представляет собой список для хранения таблиц и представлений.

Свойства и методы TatRelations

- ByID(ID: Integer): TatRelation – возвращает ссылку на таблицу по ее идентификатору. Идентификатор берется из таблицы AT_RELATIONS. Если таблица не найдена, возвращает nil.
- ByRelationName(RelationName: String): TatRelation – возвращает ссылку на таблицу по ее английскому наименованию. Если таблица не найдена, возвращает nil.
- Count: Integer – возвращает количество таблиц, созданных в текущей базе, исключая системные таблицы.
- Items(Index: Integer): TatRelation – т.к. данный класс представляет собой список, то мы можем обратиться к каждому элементу списка по индексу. Данное свойство возвращает ссылку на таблицу по ее индексу в списке.

Класс TatRelation предназначен для описания каждой таблицы (представления) по отдельности.

Свойства и методы TatRelation

- Description: String – описание таблицы.
- HasSecurityDescriptors: Boolean – указывает, содержит ли таблица поля-дескрипторы безопасности (т.е. поля с названиями AVIEW, ACHAG, AFULL).
- ID: Integer – идентификатор таблицы.
- IsLBRBTreeRelation: Boolean – указывает, является ли таблица интервальным деревом.
- IsStandartTreeRelation: Boolean – указывает, является ли таблица простым деревом.
- IsSystem: Boolean – указывает, является ли таблица системной (т.е название таблицы содержит префикс RDB$). В данном случае свойство не используется, т.к. список таблиц не содержит системных.
- IsUserDefined: Boolean – указывает, является ли таблица пользовательской (т.е. название таблицы содержит префикс USR$).
- ListField: TatRelationField – возвращает ссылку на поле отображения данной таблицы. Если поле для отображения не задано в Гедымине изначально (для всех стандартных таблиц, имеющих бизнес-объекты, оно зашито в программе, кроме того, вы можете указать поле для отображения для каждой таблицы, используя диалог редактирования таблицы), то оно ищется по следующему алгоритму: если в таблице есть поле NAME, то оно используется для отображения; если нет, то ищется поле USR$NAME. Если таких полей нет, и при этом задан список полей для расширенного отображения, то берется первое существующее поле по списку. Если список не задан, то берется первое текстовое поле. Если текстовых полей нет, то берется вообще первое в таблице поле.
- LName: String – возвращает локализованное название таблицы.
- LShortName: String – возвращает локализованное короткое название таблицы.
- PrimaryKey: TatPrimaryKey – возвращает ссылку на первичный ключ таблицы, если он существует. Если у таблицы нет первичного ключа, возвращает nil.
- RelationFields: TatRelationFields – возвращает ссылку на список полей таблицы. Поля в списке отсортированы по английскому названию.
- RelationID: Integer – идентификатор таблицы внутри базы Interbase (берется из поля RDB$RELATION_ID таблицы RDB$RELATIONS). Не мог быть использован в качестве обычного идентификатора, т.к. может принимать значение 0, что не допустимо для целочисленных идентификаторов в системе Гедымин.
- RelationName: String – наименование таблицы на английском языке.
- RelationType: Integer (перечисление) – тип таблицы. Здесь и выше при описании свойств TatRelation мы использовали понятие «таблица». Однако класс TatRelation может хранить не только описания таблиц, но и описания представлений. Для различимости введен тип, который может принимать следующие значения 0 = таблица, 1 = представление.

Класс TatRelationFields предназначен для хранения списка полей. Список создается для каждой таблицы (представления).

Свойства и методы TatRelationFields

- ByFieldName(AName: String): TatRelationField – возвращает ссылку на поле по его наименованию. Если поле с таким именем не найдено, возвращает nil.
- ByID(ID: Integer): TatRelationField – возвращает ссылку на поле по его идентификатору. Значение идентификатора берется из таблицы AT_RELATION_FIELDS. Если поле с таким идентификатором не найдено, возвращает nil.
- Count: Integer – возвращает количество полей в списке.
- Items(Index: Integer): TatRelationField – возвращает ссылку на поле из списка по его индексу (порядковому номеру в списке).
- Relation: TatRelation – возвращает ссылку на таблицу, которой принадлежит данный список полей.

Класс TatRelationField предназначен для описания каждого поля таблицы (представления) по отдельности.

Свойства и методы TatRelationField

- Alignment: Integer (перечисление) – указывает тип выравнивания данных: faLeft = 0 (влево), faRight = 1 (вправо), faCenter = 2 (по центру).
- ColWidth: Integer – ширина колонки при отображении поля.
- CrossRelation: TatRelation – если поле, является полем-множеством, то ему сопоставляется таблица-множество. Данное свойство возвращает ссфлку на таблицу-множество.
- CrossRelationField: TatRelationField – поле для отображения таблицы-множества. Используется только для полей-множеств.
- CrossRelationFieldName: String – название поля для отображения таблицы-множества. Испольуется только для полей множеств.
- CrossRelationName: String – название таблицы-множества. Используется только для полей-множеств.
- Description: String – описание поля.
- Field: TatField – возвращает ссылку на домен.
- FieldName: String – название поля на английском языке.
- FieldPosition: Integer – позиция поля в таблице.
- ForeignKey: TatForeignKey – если поле является полем-ссылкой, то возвращает ссылку на внешний ключ. Иначе возвращает nil.
- FormatString: String – строка форматирования вывода поля. Задает маску формата с использованием правил, установленных в Делфи.
- gdClassName: String – название класса бизнес-объекта. Используется только для полей-ссылок. Для остальных возвращает пустую строку.
- gdSubType: String – название подтипа бизнес-объекта. Используется в паре с gdClassName.
- ID: Integer – идентификатор поля из таблицы AT_RELATION_FIELDS.
- IsComputed: Boolean – указывает, является ли поле вычисляемым.
- IsSecurityDescriptor: Boolean – указывает, является ли поле дескриптором безопасности (т.е. поле имеет название AVIEW или ACHAG, или AFULL).
- IsUserDefined: Boolean – указывает, является ли поле пользовательским, т.е. его английское название начинается с префикса USR$.
- LName: String – локализованное название поля.

- LShortName: String – краткое локализованное название поля.
- ObjectsList: TStringList – список бизнес-классов, для которых поле является видимым. Если список пустой, то поле видимо для всех классов.
- ReadOnly: Boolean – если Истина, то поле с предназначено только для чтения. Т.е. данные в подобное поле не могут быть занесены пользователем.
- ReferenceListField: TatRelationField – поле для отображения в таблице ссылке. Используется только для полей-ссылок.
- References: TatRelation – таблица-ссылка. Используется только для полей-ссылок.
- ReferencesField: TatRelationField – ключевое поле таблицы-ссылки. Используется только для полей-ссылок. (!!!! проверить)
- Relation: TatRelation – таблица, которой принадлежит данное поле.
- Visible: Boolean – флаг видимости поля.

Класс TatPrimaryKeys предназначен для хранения списка первичных ключей.

Свойства и методы TatPrimaryKeys

- ByConstraintName(AConstraintName: String): TatPrimaryKey – возвращает ссылку на первичный ключ по его названию. В данном случае имеется ввиду название первичного ключа, а не название индекса, создаваемого на каждый первичный ключ. Если объект с таким названием не найден, возвращает nil.
- Count: Integer – количество первичных ключей в списке.
- Items(Index: Integer): TatPrimaryKey – возвращает ссылку на первичный ключ по его индексу в списке.

Класс TatPrimaryKey предназначен для описания каждого первичного ключа по отдельности.

Свойства и методы TatPrimaryKey

- ConstraintFields: TatRelationFields – возвращает ссылку на список полей, входящих в первичный ключ. Поля в списке отсортированы по порядку следования их в первичном ключе.
- ConstraintName: String – название первичного ключа.
- IndexName: String – название индекса, созданного для данного первичного ключа.
- Relation: TatRelation – ссылка на таблицу, для которой создан первичный ключ.

Класс TatForeignKeys предназначен для хранения списка внешних ключей.

Свойства и методы TatForeignKeys

- ByConstraintName(AConstraintName: String): TatForeignKey – возвращает ссылку на внешний ключ по его наименованию. В данном случае имеется ввиду наименование именно внешнего ключа, а не название создаваемого для каждого внешнего ключа индекса.
- ByRelationAndReferencedRelation (ARelationName: String; AReferencedRelationName: String): TatForeignKey – вернет первый найденный внешний ключ для таблицы с названием ARelationName со ссылкой на таблицу AReferencedrelationName.
- ConstraintsByReferencedRelation(RelationName: String; List: TObjectList; ClearList: Boolean) – возвращает список внешних ключей (List), ссылающихся на таблицу с названием RelationName. Параметр ClearList указывает, очищать ли список перед добавлением в него найденных внешних ключей.
- ConstraintsByRelation(RelationName: String; List: TObjectList) – возвращает список внешних ключей (List) для таблицы с названием RelationName. Перед заполнением список всегда очищается.
- Count: Integer – количество внешних ключей в списке.
- Items(Index: Integer): TatForeignKey – возвращает ссылку на внешний ключ по его индексу в списке.

Класс TatForeignKey предназначен для описания каждого внешнего ключа по отдельности.

Свойства и методы TatForeignKey

- ConstraintField: TatRelationField – для простых внешних ключей возвращает ссылку на поле, для которого создан внешний ключ (на поле-ссылку). Для составных внешних ключей вернет nil.
- ConstraintFields: TatRelationFields – возвращает список полей, для которых создан внешний ключ. Поля в списке отсортированы по порядку следования во внешнем ключе.
- ConstraintName: String – название внешнего ключа.
- IndexName: String – название индекса, созданного для данного внешнего ключа.
- IsSimpleKey: Boolean – указывает, является ли данный внешний ключ простым или составным.
- ReferencesField: TatRelationField – для простых внешних ключей возвращает поле, на которое ссылается внешний ключ. Для составных внешних ключей вернет nil.
- ReferencesFields: TatRelationFields – возвращает список полей, на которые ссылается внешний ключ. Поля в в списке отсортированы по порядку следования во внешнем ключе.
- ReferencesRelation: TatRelation – таблица, на которую ссылается внешний ключ.
- Relation: TatRelation – ссылка на таблицу, в которой создан внешний ключ.

Фильтрация данных

В данной главе мы познакомимся с компонентом фильтрации, узнаем об автоматическом применении фильтров для бизнес-объектов, об изменении запросов при помощи фильтров и нюансах их использования.

Использование фильтров

Каждый из нас наверняка сталкивался с необходимостью оперативного изменения выборки данных. Конечно, можно изменить выборку, за счет добавления условий в запрос, используя либо подмножества бизнес-объекта (свойство SubSet), либо дополнительные условия (свойство ExtraConditions). Однако для этого нам нужно будет написать макрос. Это не всегда удобно. Компонент фильтрации позволяет без создания новых скриптов добавлять дополнительные условия в запрос бизнес-объекта. Кроме того пользоваться фильтрами, их создавать и изменять сможет и просто опытный пользователь.

Для каждого бизнес-объекта создается свой компонент фильтрации (свойство QueryFilter). Он применяется автоматически (т.е. при открытии бизнес-объекта ему применяется последний использовавшийся фильтр), если свойство бизнес-объекта QueryFiltered установлено в Истину. Данное свойство устанавливается в Истину, если объект лежит на форме. Для объектов, создающихся в макросах, свойство QueryFiltered по умолчанию равно Ложь. Ссылка на последний примененный фильтр сохраняется при закрытии окна.

Как работает компонент фильтрации? Он изменяет запрос в соответствии с указанными условиями и переоткрывает бизнес-объект (сравните с фильтрацией на форме отображения бизнес-объекта, глава 7).

Внимание: компонент фильтрации привязан к классу и подтипу бизнес-объекта! Это значит, что он будет применятся ко всем объектам, имеющим такой же класс и подтип. Например, мы применили фильтр к мастер-объекту Папка, лежащему на форме Адресная книга. Закрыли форму. Открываем окно Организации, где мастер-объектом также является бизнес-объект Папка. К мастер-объекту автоматически применяется фильтр, использовавшийся на форме Адресная книга. Если вы не хотите, чтобы у бизнес-объекта работал фильтр, то установите свойство QueryFiltered в Ложь в событии OnCreate формы, перед вызовом метода Inherited. Например:

```
option explicit
sub gdc_frmCompanyOnCreate(ByVal Sender)
  Sender.GetComponent("gdcFolder").QueryFiltered = False
  call Inherited(Sender, "OnCreate", Array(Sender))
end sub
```

Заметьте, в данном примере мы не можем обратиться к мастер-объекту через свойство формы gdcObject, т.к. это свойство устанавливается именно при вызове Inherited. Там же и происходит применение фильтра, если это необходимо. Поэтому, мы указываем именно до вызова Inherited, что для компонента gdcFolder, лежащего на форме, применение фильтра не нужно.

Фильтры могут использоваться не только для ограничения выборки записей, но и для сортировки выборки по одному или нескольким полям.

Меню компонента фильтрации

Каждый компонент фильтрации имеет меню. Чтобы вывести это меню, бизнес-объект использует метод PopupFilterMenu. Меню фильтра (смотрите рисунок 9.1) условно делится на три области:

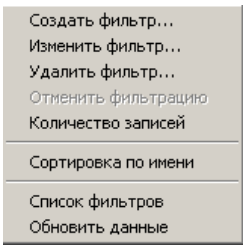


Рисунок 0.1

- Команды работы с фильтрами:
  - Создать фильтр – выводит окно для создания нового фильтра. Помните, наименование фильтра уникально в пределах бизнес-объекта под определенным пользователем. Если фильтр используется для все пользователей, то его наименование должно быть уникально для бизнес-объекта в пределах базы.
  - Изменить фильтр – выводит окно для изменения текущего фильтра.
  - Удалить фильтр – удаляет текущий фильтр.
  - Отменить фильтрацию – отменяет фильтрацию, если она применена.
  - Количество записей – выводит реальное количество записей в выборке.
- Последние 10 использовавшихся фильтров – перечисление имен десяти наиболее часто использовавшихся фильтров. Выбор фильтра означает его применение.
- Команды обновления фильтров:
  - Список фильтров – выводит окно со списком всех фильтров (смотрите рисунок 9.2). Во второй части меню выводится 10 наиболее часто использовавшихся фильтров. Это сделано для того, чтобы не перегружать меню лишней информацией. Данное окно позволяет вывести список всех фильтров, заданных для бизнес-объекта. Центральную часть окна занимает список фильтров в алфавитном порядке. При этом для каждого фильтра выводится последнее время его выполнения и количество обращений к данному фильтру. Кнопки внизу окна позволяю изменить указанный фильтр, удалить его или применить (кнопка «Выбрать»). Кнопка «Закрыть» закрывает окно.

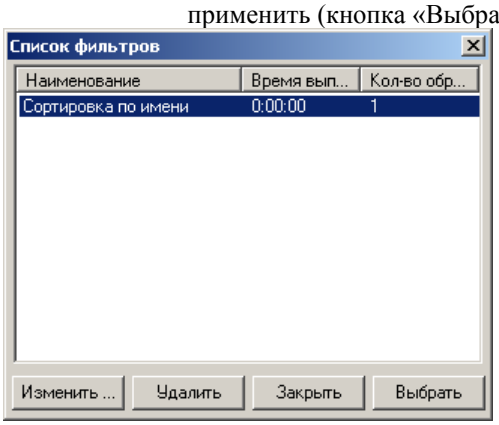


Рисунок 0.2

  - Обновить данные – перечитывает данные в соответствии с примененным фильтром. Если применение фильтра требует ввода некоторых параметров, то пользователю предлагается ввести новые параметры или оставить текущие (например, вы создали фильтр для вывода информации за определенный период. Для указания периода на экране появляется окно с вводом параметров).

Окно изменения фильтра

При создании нового фильтра или изменении существующего на экран выводится окно изменения фильтра. Данное окно имеет четыре закладки:

- Наименование
- Фильтр
- Сортировка
- SQL

Закладка **Наименование** содержит поля для ввода наименования фильтра, комментарий, режим отображения полей, по которым формируется фильтр (закладка Фильтр), признак «Только для меня», кнопку с вызовом формы установления прав доступа к фильтру, время последнего его выполнения (смотрите рисунок 9.3). Если признак «Только для меня» установлен, то фильтр виден и используется только для текущего пользователя. В этом случае кнопка Права недоступна, т.к. пользователь-создатель фильтра имеет на него все права. Если данный признак не установлен, то фильтр может использоваться другими пользователями. Уровень использования фильтра можно установить по кнопке Права. По нажатию кнопки «Права» на экран выводится диалог установки прав на использование фильтров для групп пользователей (смотрите рисунок 9.4). Диалог установки прав имеет три закладки: Полный

(установка всех прав: на просмотр, изменение, удаление), Изменение (пользователи групп имеют права только на просмотр и изменение), Просмотр (пользователи могут только применять фильтры).

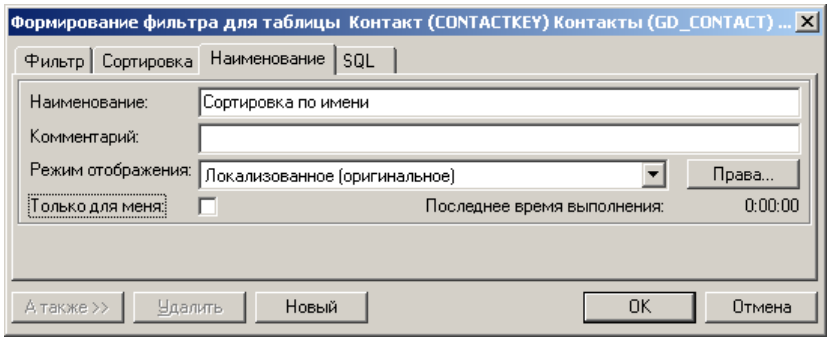


Рисунок 0.3

Закладка **Фильтр** (смотрите рисунок 9.5) содержит следующие поля для заполнения: наименование поля, условие фильтра, переключатели «Должны быть выполнены все условия» (т.е. условия добавляются с оператором AND), «Должно быть выполнено хотя бы одно условие» (т.е. условия добавляются с оператором OR), признак «Только уникальные записи» - позволяет избавится от дублирования записей, которое становится возможным при использовании некоторых условий. Переключали «Должно …» используются при указании нескольких условий. Чтобы добавить несколько условий, нужно после ввода первого условия нажать на кнопку «А также >>» на нижней панели формы. Чтобы удалить лишнее условие, нужно нажать на кнопку «Удалить» на нижней панели формы или на крестик в верхнем правом углу условия.

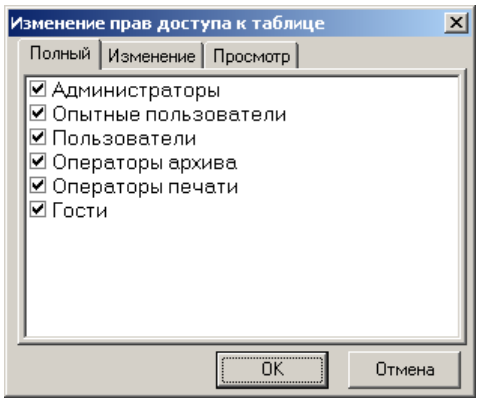


Рисунок 0.4

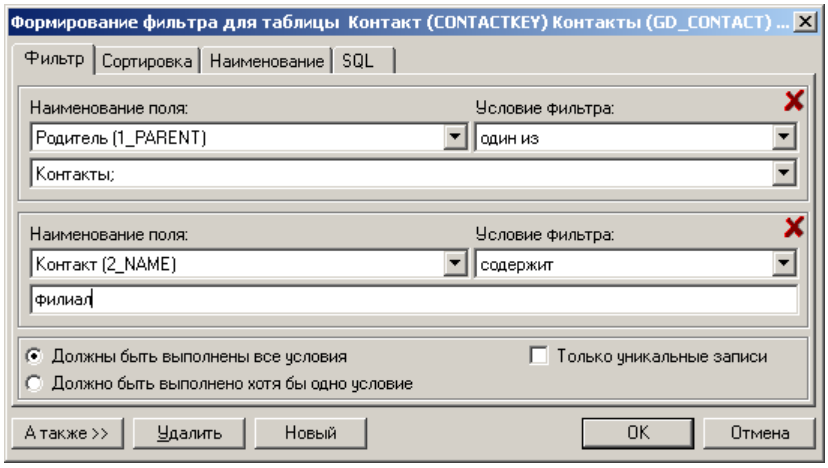


Рисунок 0.5

Теперь рассмотрим подробнее содержимое области «Наименование поля». Данное поле может принимать следующие значения:

- Формула – предлагает настройщику добавить условия в запрос вручную. При этом поле «Условия фильтра» предлагает выбрать одну из двух операций:
  - ввести условия – настройщику выводится окно с областью для добавления условий в существующий запрос
  - создать запрос – настройщику выводится окно с областью для изменения существующего запроса. Изменяя запрос бизнес-объекта, помните: не следует изменять набор полей бизнес-объекта!
- Выбранное поле – т.е. данная область содержит список таблиц, использующихся в базовом запросе бизнес-объекта, для каждой таблицы список полей и список ссылок на эту таблицу. В зависимости от типа выбранного поля область «Условия фильтра» заполняется определенными операциями.

Выбранное поле может быть одним из следующих типов:

- Ссылка на другую таблицу  - поле текущей таблицы списка, являющееся ссылкой на другую таблицу.
- Поле данных  - поле текущей таблицы списка, хранящее числовые, текстовые, временные (дата, время) или BLOB данные.
- Множество  - поле внешней таблицы-множества, которое ссылается на ключевое поле текущей таблицы. При этом в расшифровке указывается название таблицы-множества и название поля, являющегося внешней ссылкой на текущую таблицу.
- Внешняя ссылка  - поле внешней таблицы, которое ссылается на ключевое поле текущей таблицы. При этом в расшифровке указывается название таблицы и название поля, являющегося внешней ссылкой на текущую таблицу.

Область «Условия фильтра» может принимать следующие значения:

- **больше** – используется для полей числовых и временных типов. Требует указания фиксированного порога. Если значение указанного поля будет больше данного порога, то условие верно.
- **больше или равно** – используется для полей числовых и временных типов. Требует указания фиксированного порога. Если значение указанного поля будет больше или равно данного порога, то условие верно.
- **включает** – используется для полей множеств или внешних ссылок. Требует указания фиксированного набора значений. Если данные значения содержатся в таблице-множестве или внешней таблице, то они будут отображены.
- **вне вкл. границы** – используется для полей числовых и временных типов. Требует указания фиксированных границ. Если значение указанного поля вне или равно данным границам, то условие верно.
- **вне искл. границы** – используется для полей числовых и временных типов. Требует указания фиксированных границ. Если значение указанного поля вне данных границ, то условие верно.
- **доп. фильтрация** – используется только для полей-множеств. Позволяет открыть диалог для фильтрации по полям таблицы-множества.
- **за выбранный день** – используется для полей временных типов. Требует указания фиксированной даты. Если значение указанного поля будет равно этой дате, то условие верно.
- **за сегодня** – используется для полей временных типов. Если значение указанного поля будет равно текущей дате, то условие верно.
- **заканчивается на** – используется для текстовых полей. Требует указания фиксированного текста. Если значение указанного поля будет заканчиваться на данный текст, то условие верно. Поиск текста не зависит от регистра.
- **запрос параметра** – используется для всех типов полей, за исключением полей типа BLOB. Выводит окно с запросом параметра, соответствующего типу поля. При применении фильтра пользователь должен указать значение параметра. Последний указанный параметр сохраняется в настройках пользователя.
- **между вкл. границы** – используется для полей числовых и временных типов. Требует указания фиксированных границ. Если значение указанного поля внутри или равно данным границам, то условие верно.

- **между искл. границы** – используется для полей числовых и временных типов. Требует указания фиксированных границ. Если значение указанного поля внутри данных границ, то условие верно.
- **меньше** – используется для полей числовых и временных типов. Требует указания фиксированного порога. Если значение указанного поля будет меньше данного порога, то условие верно.
- **меньше или равно** – используется для полей числовых и временных типов. Требует указания фиксированного порога. Если значение указанного поля будет меньше или равно данного порога, то условие верно.
- **начинается с** – используется для текстовых полей. Требует указания фиксированного текста. Если значение указанного поля начинается на данный текст, то условие верно. Поиск текста не зависит от регистра.
- **не включает** – используется для полей-ссылок, множеств или внешних ссылок. Требует указания фиксированного набора значений. Для полей ссылок используется условие «не в списке». Для полей множеств или внешних ссылок данные выводятся в соответствии с существованием на них ссылок из внешних таблиц или таблиц-множеств, исключая указанный набор значений.
- **не включает ветвь** – используется для полей-ссылок или множеств, если текущая таблица является интервальным деревом. Требует указания фиксированного набора значений. Исключает из выборки не просто значения из набора, но и целиком всю ветку, которая включена в значение.
- **не равно** – используется для полей текстовых, числовых и временных типов. Требует указания фиксированного значения. Если текущее поле не равно указанному значению, то условие верно. Для текстовых полей сравнение идет без учета регистра. (!!!! проверить про регистр)
- **не содержит** – используется для текстовых полей. Требует указания фиксированного текста. Если значение указанного поля не содержит данный текст, то условие верно. Поиск текста не зависит от регистра.
- **не существует** – используется для полей всех типов. Проверяет поле на NULL. Если поле содержит NULL, то условие верно.
- **один из** – используется для полей-ссылок, множеств, внешних ссылок. Требует указания фиксированного набора значений. Если указанное поле входит в данных набор, то условие верно. При работе с полями-множествами и внешними ссылками данные значения должны также присутствовать в этих таблицах.
- **одна ветвь из** – используется для полей-ссылок и множеств, если текущая таблица – интервальное дерево. Требует указания фиксированного набора значений. Если указанное поле входит в данных набор или является вложенным значением для значения из набора, то условие верно. При работе с полями-множествами данные значения должны также присутствовать в таблице-множестве.
- **последние __ дней** – используется только для временных типов. Требует указания фиксированного значения количества дней. Записи будут отфильтрованы по указанному полю, содержащему значения от текущей даты минус фиксированное количество дней.
- **равно** – используется для полей текстовых, числовых и временных типов. Требует указания фиксированного значения. Если текущее поле равно указанному значению, то условие верно. Для текстовых полей сравнение идет без учета регистра. (!!!! проверить про регистр)
- **скрипт** – используется для всех типов полей. Позволяет указать условие для выбранного поля при помощи скрипта.
- **содержит** – используется для текстовых полей. Требует указания фиксированного текста. Если значение указанного поля содержит данный текст, то условие верно. Поиск текста не зависит от регистра.
- **существует** – используется для полей всех типов. Проверяет поле на NULL. Если поле содержит какое-либо значение, то условие верно.
- **фильтрация** – используется для полей-ссылок, множеств и внешних ссылок. Позволяет открыть диалог для фильтрации по полям таблиц, на которые идет ссылка.

Закладка **Сортировка** предоставляет возможность добавить в запрос сортировку по одному или нескольким полям (сравните с сортировкой данным в гриде). Чтобы добавить следующее условие сортировки, необходимо нажать на кнопку «А также >>» на нижней панели формы. Чтобы удалить лишнее условие, нужно нажать на кнопку «Удалить» на нижней панели формы или на крестик в верхнем правом углу условия. Внизу закладки находится признак «Разрешать сортировку только по полям, для

которых создан индекс в базе». При установке данного признака, в списке полей будут отображаться только индексированные поля. Вы можете упорядочить записи по возрастанию или по убыванию. При указании признака только индексированных полей порядок сортировки берется из индекса.

Закладка **SQL** содержит две кнопки: текст исходного SQL-запроса (т.е. запроса бизнес-объекта без применения фильтрации), текст SQL-запроса с условиями фильтрации (т.е. запроса с применением текущих условий). По нажатию одной из этих кнопок перед вами появляется SQL-редактор, в котором вы можете отладить запрос. **Внимание!** Изменения в SQL-редакторе не сохраняются. Он предназначен только для тестирования запроса.

Хранение фильтров в базе

В данной подглаве рассматривается структура таблиц, предназначенных для хранения фильтров. Все таблицы для работы с фильтрами начинаются с префикса FLT_ (смотрите таблицу 9.1).

Название таблицы	Описание
FLT_COMPONENTFILTER	Каждый фильтр как компонент имеет уникальное название, состоящее из flt_ + РУИД бизнес-объекта + название бизнес-объекта. Если фильтр привязан не к бизнес-объекту, а просто к IB-датасету, то РУИД, естетственное опускается. Данная таблица хранит связку между названием фильтра и объектом, к которому он привязан
FLT_LASTFILTER	Хранит ссылку на последний использовавшийся фильтр для указанного компонента и пользователя системы.
FLT_PROCEDUREFILTER	Не используется.
FLT_SAVEDFILTER	Хранит отдельные фильтры, созданные пользователем, их описание, права доступа.

Таблица 0.1

Теперь рассмотрим структуру каждой таблицы по отдельности.

Таблица **FLT_COMPONENTFILTER**.

Первичный ключ	Название поля	Не NULL	Описание
x	ID	x	Идентификатор записи
	FILTERNAME		Название компонента фильтра
	FORMNAME		Название набора данных, к которому привязан компонент фильтрации. Название FORMNAME осталось с тех времен, когда фильтр был привязан к форме.
	APPLICATIONNAME		Название приложения. По умолчанию GEDEMIN.
	CRC		Код контроля данных. Введен для ускорения поиска.
	FULLNAME		Полное наименование компонента фильтрации. За счет длины РУИД-а поле FILTERNAME не может хранить слишком длинное наименование.

Таблица 0.2

Таблица **FLT_LASTFILTER**.

Первичный ключ	Название поля	Не NULL	Описание
x	COMPONENTKEY	x	Ссылка на компонент фильтрации (таблица FLT_COMPONENTFILTER).
x	USERKEY	x	Ссылка на пользователя, который использовал фильтр (таблица GD_USER).
	LASTFILTER	x	Ссылка непосредственно на последний

		использовавшийся фильтр (таблица FLT_SAVEDFILTER).
	CRC32	Код контроля данных.
	DBVERSION	Версия БД, с которой был создан фильтр.

Таблица 0.3

Таблица **FLT_SAVEDFILTER**.

Первичный ключ	Название поля	Не NULL	Описание
x	ID	x	Идентификатор фильтра.
	NAME	x	Название фильтра.
	USERKEY		Ссылка на пользователя, для которого создан фильтр (таблица GD_USER). Если фильтр создан для всех, поле пустое.
	COMPONENTKEY	x	Ссылка на компонент фильтрации (таблица FLT_COMPONENTFILTER).
	DESCRIPTION		Описание фильтра
	LASTEXTIME		Последнее время выполнения фильтра.
	READCOUNT		Количество обращений к фильтру.
	DATA		Тело фильтра.
	AVIEW, ACHAG, AFULL		Дескрипторы безопасности.
	DISABLED		Не используется.
	EDITIONDATE		Дата последней модификации
	EDITORKEY		Ссылка на контакт (таблица GD_CONTACT) пользователя, последнего изменявшего фильтр.
	RESERVED		Не используется

Таблица 0.4

Использование фильтрации для TIBDataset и TIBQuery

Итак, для бизнес-объекта компонент фильтрации создается автоматически. Что же делать, если вы используете обычный TIBDataSet или TIBQuery, и при этом хотите использовать фильтрацию? Для таких случаев создан компонент TQueryFilterGDC. Его достаточно положить на форму и в свойстве IBDataSet указать, какой набор данных вы желаете фильтровать. Затем вам нужно создать кнопку, которой будет вызываться меню фильтра и на OnClick прописать вызов.

```
option explicit
sub usrg_ButtonFilterOnClick(ByVal Sender)
  Dim FilterComponent
  'находим компонент фильтрации на форме
  set FilterComponent = _
    Sender.OwnerForm.GetComponent("usrg_QueryFilterGDC")
  'вызываем меню филтрации
  call FilterComponent.PopupMenu(-1, -1)
end sub
```

Пример использования фильтрации для TIBDataSet можно посмотреть в Разноске по счетам (настройка «Бухгалтерия»).

Отчеты

Схематически, компоненты процесса построения отчета в системе Гедымин можно изобразить следующим образом:

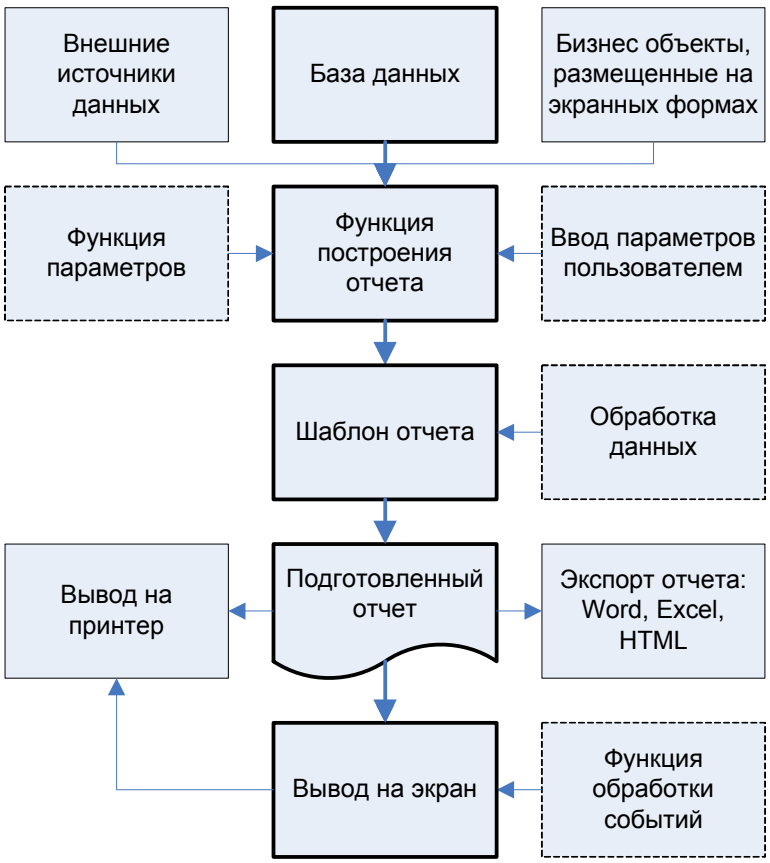


Рис. 50 Компоненты процесса создания отчета.

Вкратце, схема создания отчета такова: функция построения отчета извлекает данные из базы данных или внешних источников и помещает их в специальную внутреннюю структуру. На вход функции могут быть переданы параметры, которые могут быть запрошены у пользователя или сформированы автоматически функцией параметров. Далее, Гедымин подставляет данные из внутренней структуры в шаблон, в результате чего получается готовый, сформированные отчет. Этот отчет может быть отображен на экране, отпечатан на принтере, сохранен на диске или экспортирован в другие программы (например, в Microsoft Excel). Отчет, отображаемый на экране, можно сделать интерактивным. Для этого создается функция обработки событий, которая вызывается в ответ на щелчок мышью по той или иной части отчета.

Рассмотрим каждый компонент, участвующий в описанном процессе, более подробно:

Функция построения отчета

Функция построения отчета, она же основная функция отчета, возвращает список наборов данных (dataset), которые будут доступны в шаблоне отчета. Работа со списком происходит через глобальный объект BaseQueryList. Как правило, в первых строках функции объект BaseQueryList очищается вызовом метода Clear. Далее, с помощью метода BaseQueryList.Add создаются и добавляются в список новые наборы данных, при этом задается имя набора и его тип. Поддерживаются два типа: набор данных, являющийся результатом выполнения SQL запроса к базе данных и таблица в оперативной памяти компьютера. После создания набора данных первого типа, необходимо задать текст запроса, присвоить параметры (если они есть) и выполнить запрос. Во втором случае, необходимо создать поля в таблице и заполнить ее вручную, добавляя записи по-одной. Ниже приводится пример основной функции, в котором создаются по одному набору данных каждого типа:

```
function rp_MyReport

  ' объект BaseQueryList глобальный, поэтому
  ' его необходимо очистить перед использованием
  BaseQueryList.Clear

  ' создадим набор данных на основании SQL запроса
  ' который извлечет из базы данных список пользователей системы
  ' назовем его SQLQuery (под этим именем набор данных
  ' будет виден в списке наборов в дизайнере шаблона отчета
  dim Q
  set Q = BaseQueryList.Query(BaseQueryList.Add("SQLQuery", 0))
  Q.SQL = "SELECT * FROM gd_user"
  ' укажем, что набор следует показывать с списке
  ' наборов данных в дизайнере
  Q.IsResult = 1
  ' выполняем запрос
  Q.Open

  ' создадим набор данных типа таблица с двумя полями:
  ' целочисленным полем "ID" и строковым полем "Name"
  ' оба поля обязательны для заполнения
  ' назовем набор данных "MemTable"
  dim MT
  set MT = BaseQueryList.Query(BaseQueryList.Add("MemTable", 1))
  MT.AddField "ID", "ftInteger", 0, True
  MT.AddField "Name", "ftString", 60, True
  MT.Open
  ' добавим одну запись
  MT.Append
  MT.FieldByName("ID").AsInteger = 1
  MT.FieldByName("Name").AsString = "Name"
  MT.Post
  MT.IsResult = 1

  ' перед выходом из функции не забудем вернуть
  ' сформированный список наборов данных
  set rp_MyReport = BaseQueryList

end function
```

Данные

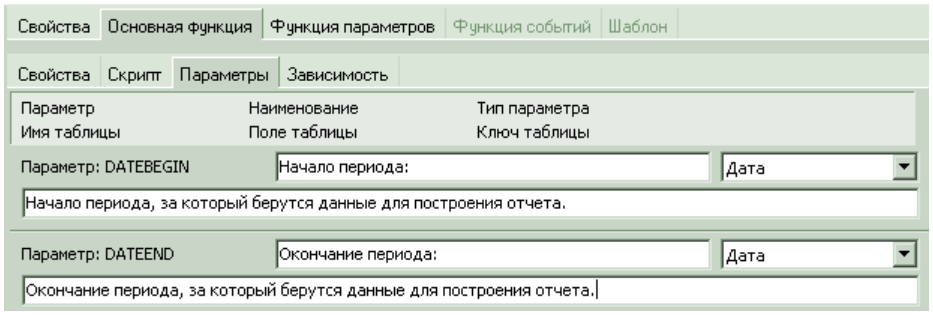
Большинство отчетов строится на основании информации из базы данных. Для ее извлечения применяются SQL запросы. Кроме этого, возможно использование любых других источников данных: массивов, StringGrid-ов, данных из файлов и т.п. Подключение к внешним источникам данных может осуществляться через объекты ADO, при наличии соответствующих драйверов доступа ODBC или провайдеров OLE DB.

Параметры

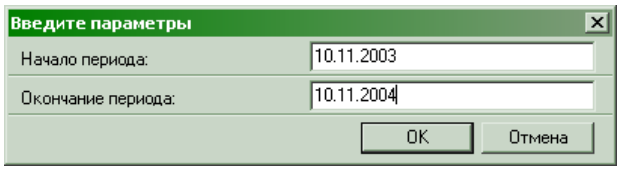
Функция построения отчета может иметь входные параметры. Например, параметрами могут быть начальная и конечная даты периода, за который берутся данные для построения отчета и т.п. Если просто задать параметры в заголовке функции, Гедымин автоматически будет запрашивать их у пользователя при выполнении данного отчета. Каждый параметр можно настроить: указать его тип, снабдить, понятным пользователю названием. Рассмотрим следующий пример: пусть функция построения отчета определена как:

```
function rp_CustomerList(DateBegin, DateEnd)
```

Выполнена настройка параметров:



При запуске отчета на выполнение на экран будет выведено окно запроса параметров:



Шаблон

Пример простого отчета

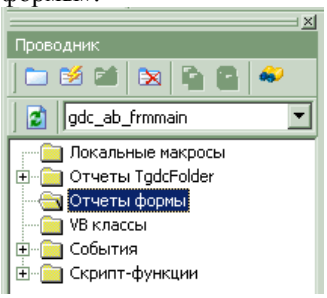
Рассмотрим пример создания простого отчета — списка организаций клиентов, содержащихся в базе данных, с выводом наименования организации, ее адреса и контактного телефона.

В Гедымине существует три типа отчетов: отчеты, привязанные к конкретной форме (будут отображаться в меню отчетов этой формы), отчеты, привязанные к определенному типу бизнес объекта (будут выводиться в меню отчетов любой формы, на которой лежит объект такого типа) и, наконец, отчеты ни к чему не привязанные. Такие отчеты доступны через форму Отчеты, вызываемую из Исследователя. Стоит заметить, что для любого отчета можно просто добавить команду вызова в Исследователь. В этом случае доступ к отчету можно будет получить непосредственно, без открытия промежуточных форм.

В нашем случае мы создадим отчет, привязанный к форме справочника клиентов. Последовательность действий следующая:

1. Откроем справочник клиентов. Для этого зайдем в Исследователь, найдем в нем раздел Справочники и дважды щелкнем по ветке Клиенты в этом разделе;
2. На панели инструментов отыщем изображение принтера (команда «Печать») и щелкнем по нему;
3. На экране откроется всплывающее меню. Выберем в нем команду «Редактор скрипт-объектов»;
4. На экране откроется окно редактора скрипт-объектов;

5. В Проводнике редактора скрипт-объектов необходимо установить курсор на папку «Отчеты формы»:



Если окно проводника закрыто, его необходимо открыть, вызвав соответствующую команду в меню «Окна» редактора скрипт-объектов.

6. По правой кнопке мыши вызовем контекстно-зависимое меню и выберем команду «Добавить отчет»;

7. На рабочей области редактора скрипт-объектов откроется набор закладок, предназначенных для работы с основными компонентами отчета: функцией построения (закладка «Основная функция»), функцией параметров, функцией событий и шаблоном отчета, а также для изменения свойств отчета (закладка «Свойства»).

Свойства отчета

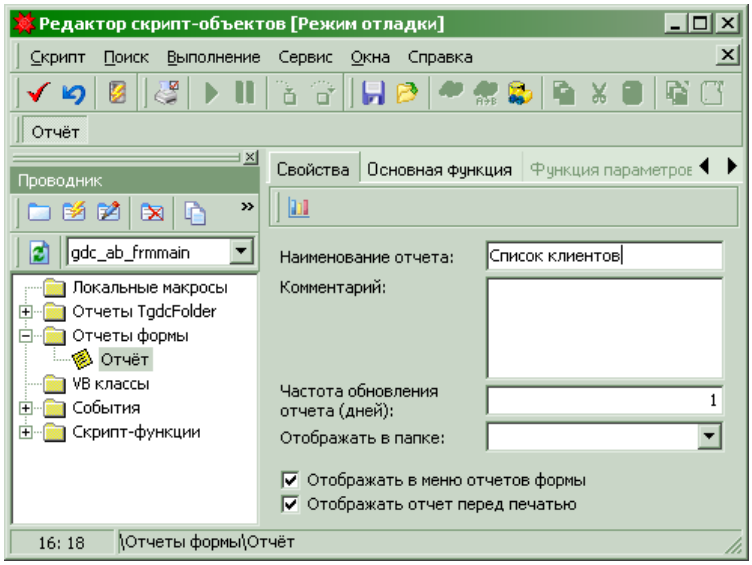


Рис. 51 Свойства отчета.

На закладке «Свойства» располагаются следующие поля:

- Наименование отчета. Наименование отчета так, как оно будет отображаться в меню отчетов формы;
- Комментарий. Произвольный текстовый комментарий;
- Частота обновления отчета. В настоящий момент данное поле не используется. Может быть заполнено произвольным образом;
- Отображать в папке. Из выпадающего списка можно выбрать папку Исследователя, куда будет помещен отчет. Если поле заполнить, то отчет можно будет вызвать как из меню «Печать» на форме, так и путем вызова соответствующей команды в Исследователе системы. Команда будет иметь такое же наименование, как и наименование отчета и располагаться в заданной папке. Если в последствии потребуется удалить команду из Исследователя, то для этого достаточно очистить поле и пересохранить отчет;
- Отображать в меню отчетов формы. Если галочка снята, то наименование отчета не будет отображаться в меню отчетов формы;

- Отображать отчет перед печатью. Если галочка снята, то при выполнении отчета он будет сразу же выводиться на печать без предварительного просмотра на экране.

Кроме этого, на вкладке свойства можно настроить права доступа к данному отчету. Соответствующая кнопка называется «Дополнительные свойства» и располагается на панели инструментов, непосредственно над полем «Наименование отчета».

Основная функция

На вкладке «Основная функция» осуществляется редактирование функции построения отчета.

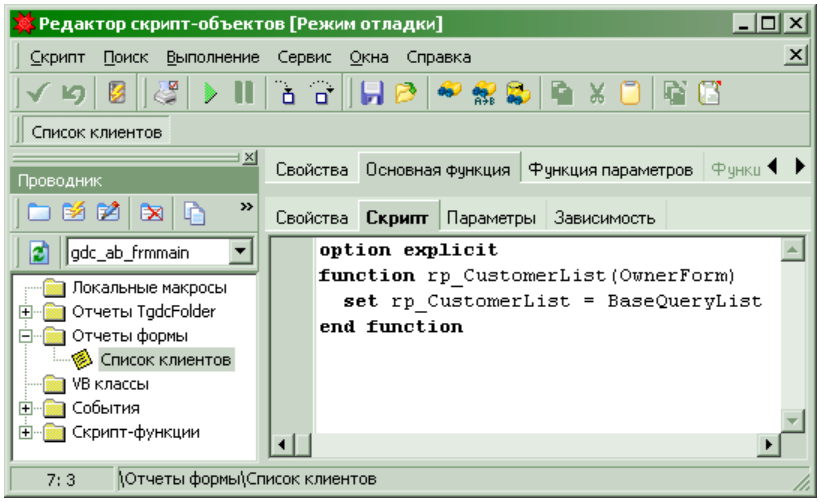


Рис. 52 Вкладка "Основная функция"

Разграничение прав доступа

Только пользователя, являющиеся членами групп «Администраторы», «Опытные пользователи» или «Операторы печати» имеют право редактировать построенный отчет в режиме просмотра, перед его отправкой на принтер.

FastReport

Функции

- SumStr(Number: Double; Precision: Integer)
  Возвращает число Number прописью, количество знаков после запятой ограничивается параметром Precision (не более 4).
  Пример: SumStr(0.005, 3) — “Ноль целых пять тысячных”.

xFastReport

Описание формата файла-формы

Файл-форма представляет собой набор блоков. Каждый блок содержит формат вывода той или иной части отчета.

Расположение блоков

Для обозначения начала блока используется строка, состоящая из символа % (в первой позиции) и следующего за ним названия блока (не отделенного пробелами). Все содержимое файла до начала первого блока игнорируется. Весь текст, следующей за именем блока (в той же строке) тоже игнорируется.

Файл - форма должен содержать следующие блоки (строго в указанном порядке):

⇒ **%Report** - текст, печатаемый в начале отчета

⇒ **%Page** - текст, печатаемый в начале каждой страницы

⇒ **%TableHeader** - заголовок таблицы (печатается на каждой странице)

⇒ **%TableRecord** - формат вывода одной записи (каждая запись будет напечатана на одной странице)

⇒ **%TableSeparator** - текст, отделяющий одну запись от другой

⇒ **%TableFooter** - текст, печатаемый внизу таблицы на каждой странице (кроме последней)

⇒ **%TableEnd** - текст, печатаемый в конце таблицы (на последней странице)

⇒ **%Page** - текст, печатаемый в конце каждой страницы

⇒ **%Report** - текст, печатаемый в конце отчета

Кроме указанных блоков файл-форма может содержать следующие блоки (в произвольном месте файла):

⇒ **%Group** - текст, отделяющий блоки записей. После объявления группы в той же строке в квадратных скобках нужно указать переменную, по которой формируются группы

Содержимое блока

Каждый блок содержит набор произвольного текста, который будет вставлен в файл отчета в соответствующих местах и из полей, значения которых вычисляются в ходе создания отчета. Для смены шрифта используются последовательности символов @x:

@b - режим выделенного шрифта

@i - режим курсива

@u - режим подчеркнутого шрифта

@n - режим нормального шрифта

@@ - данная последовательность трактуется как один символ @

Если установлен новый шрифт, то он действует до тех пор, пока не будет установлен другой шрифт (независимо от смены блока).

Для обозначения поля используются символы [ ] . Так, последовательность символов [FriendName] будет считаться полем. Текст внутри символов [ ] обозначает название поля и, возможно, опции. Если опции есть, то они должны быть отделены от имени поля символом /. После вычисления значения поля, оно будет напечатано вместо названия поля (т.е. вместо блока [xxxxxxxxxxx] за исключением случая, когда указана опция /a – в этом случае поле будет расширено/сжато до реальных размеров подставляемого значения).

Поддерживаются поля четырех типов:

a) ссылки на встроенные переменные
b) ссылки на пользовательские переменные
c) ссылки на поля, которые должны быть вычислены пользователем
d) ссылки на поля базы дынных (названия этих полей должны начинаться с символа #).

Встроенные переменные

В данной версии поддерживаются следующие встроенные переменные:

Page - номер страницы

Date - текущая дата

Time - текущее время

PageBreak - символ конца страницы

- номер текущей записи. Данное поле может быть использовано и без квадратных скобок (в этом случае печатаются только две последние цифры номера записи: например, вместо 126 будет напечатано 26).

Пользовательские переменные

Произвольные переменные, значения которых определяются в свойстве Vars. Формат строки в свойстве Vars:

<имя-переменной> = “ присваемое-значение ”

Все пробелы, кроме указанных в кавычках игнорируются. Если присваиваемое значение указано без кавычек, то все пробелы перед этим значением и после него будут утеряны (проигнорированы).

Вычисляемые переменные

Все переменные, которые не являются встроенными, и не указаны в переменной Vars, должны быть вычислены программно в процедуре, на которую ссылается переменная OnEvalField. Если переменная не будет вычислена и там, то в качестве значения переменной берется имя этой переменной с добавленным символом ‘?’ в начале и в конце.

Поля базы данных

В качестве значения данного поля берется значение соответствующей колонки в текущей записи в базе данных на которую ссылается DataSource.

Опции

Опции используются для указания выравнивания значения поля (если его длина меньше выделенного под поле места), для управления переносом слов в сложных системах полей, для суммирования полей и для перевода полей в текстовое (‘словесное’) представление.

Поддерживаются следующие опции:

Выравнивание по левому краю (по умолчанию)

/R Выравнивание по правому краю

/C Центрирование

/V, /W Указывают режим переноса слов. Если в текущем блоке есть ещё поля с одной из этих опций и таким же именем, то значение будет распределено между этими полями. При использовании опции /V значение поля обрезается точно по ширине поля. При использовании же опции /W производится попытка переноса слова целиком (word wrap). Если значение последнего поля превышает отведенное ему место, то это значение обрезается, и в конце добавляется строка ‘…’.

Замечание: для различных полей с одним именем разрешается смешивать опции /L, /R, /C, /W /V.

/A Опция, отменяющая обрезание поля. Если указана эта опция и длина поля превосходит отведенную под него длину, то при печати поле будет расширено до необходимого размера.

Замечание: опция /A отменяет действие опций /L, /R, /C, /V, /W

/S Эта опция имеет смысл только в разделах TableEnd, GroupFooter, ReportEnd. В качестве значения поля к которому применена данная опция, будет использоваться сумма значений этого поля в соответствующем разделе (таблице, группе, отчете). Использование этой опции в любом другом разделе приведет к тому, что данное поле будет печататься пустым. Эта опция может комбинироваться с ранее описанными опциями. Для определения своего способа суммирования значений полей используйте OnSummarize event.

/T Эта опция должна использоваться только с числовыми полями. Она приводит к переводу числового поля в текстовое представление (для подробной информации смотри описание компоненты TNumberConvert).

```
1234567890 234567890 234567890 234567890 234567890 234567890 234567890 234567890
%variable
[quantity]=Q
[quantity]*([1]-[orderprint])=Q1
[Weight]*[orderprint]=W
CostNCU=F2
Round([USR$AmountVAT])=N
USR$BeforUse=USE
Round([AmountNCU]+[USR$AmountVAT])=F5
Round([AllPaymentNCU])=F0
Round([AmountNCU])=F6
PackQuantity=F7
PackInQuant=F8
VALUENAME=V
USR$AMOUNTVATPERC=P
PackName=F10
[Weight]*([1]-[orderprint])=F9
[Weight]*[orderprint]=W1
[AMT$USR$AMOUNTVATNCU0]-[TaraNCU100]=AMTT
[AMT$USR$AMOUNTVATNCU10]+[AMT$USR$AMOUNTVATNCU18]+[AMT$USR$AMOUNTVATNCU0]-[TaraNCU100]=SumGood
Round([AmountWithOutTaraNCU])=F20
[AmountTaraNCU]=F21
Round([AllPaymentVATNCU])=AllVAT
[BRUTTO]=BR
%endvariable
%report
[#compressedon]                          Оператор:     [#OperatorName/a]Дата и время: [#PrintDate/a]

                                  [#SenderUNN/c  ]     [#CustomerTAXID     ] [#CustomerTAXID ]

                                  [#SenderOKPO/c ]     [#CustomerOKPO      ] [#CustomerOKPO ]

                             [#SenderLicence/r      ]            [#CustomerLICENCE  ]  [#CustomerLICENCE ]

[#Number/c          ]
[#DocumentDate/c    ]                Автомобиль          [#Car/a]  К путевому листу [#WaySheetNumber/a]
Владелец транспорта [#CarOwner/c                    ]   Водитель  [#Driver/c           ] Вид перевозки:[#TypeTransName/A] авто
Заказчик(плательщ) [#PayerName/a] [#PayerCity/a] [#PayerAddress/a] p\c[#PayerAccount/a]
[#PayerBank/a],[#PayerBankCity/a],[#PayerBankAddress/a] код:[#PayerBankCode/a]
Грузоотправитель  [#Sender/a] [#SenderCity/a] [#SenderAddress/a] P/C [#SenderAccount/a]  [#SenderBank/a],[#BankCity
/a],[#BankAddress /a] код:[#SenderCode/a]
Грузополучатель  [#CustomerName/a] [#CustomerCity/a] [#CustomerAddress/a] p\c[#CustomerAccount/a]
[#CustomerBank/a],[#CustomerBankCity /a],[#CustomerBankAddress /a] код:[#CustomerBankCode/a]
Пункт погрузки    [#LoadingPoint/a] Пункт разгрузки [#UnLoadingPoint/a]      Маршрут №
Договор/оплата    [#contractnum/a ] от [#date/l     ] Форма оплаты: [#paycond/a]     Цель приобретения: [#purpose/a]
Переадресовка     [#readdressing                                                    ] Прицеп [#hooked/a] Гар.№
Отпуск разрешил РУКОВОДИТЕЛЬ_____ ГЛ.БУХГАЛТЕР_____
Прейскурант цен [#Pricename                          ], Цена: [#LName                        ]

                                          СВЕДЕНИЯ О ГРУЗЕ
%page
%tableHeader
-------------------------------------------------------------------------------------------------------------------------
|     Наименование товара          | Ед. |         |  Масса  |          | Цена за  |   Ст-сть  |  %  | Сумма   |   Всего
|                                  | изм.| кол-во  |  тара   |  нетто   | единицу  |           | НДС |  НДС    |   с НДС
|                                  |     |         |         |          |[#PRICE/c]|           |     |         |
-------------------------------------------------------------------------------------------------------------------------
%tableRecord
[#ShortName                       ][#V/R][#Q/Z/Q/R][#W/Z/Q/R][#F9/Z/R/Q][#F2/R/M  ][#F6/R/M   ][#P/C][#N/R/M  ][#F5/R/M    ]
%tableSeparator
%tableFooter
%tableEnd
          Вес упаковки [#USR$PACKWEIGHT/a] кг.
          Сумма НДС (руб) [#N/S/A] ([#N/S/T/A])
          По ставке 18% - НДС: [#USR$AMOUNTVATNCU18] Товар: [#AMT$USR$AMOUNTVATNCU18]  Продукция производства
                    10% - НДС: [#USR$AMOUNTVATNCU10] Товар: [#AMT$USR$AMOUNTVATNCU10]  Слуцкого мясокомбината РБ
                     0% - НДС: [#USR$AMOUNTVATNCU0 ] Товар: [#AMT$USR$AMOUNTVATNCU0 ]
          Сумма транспортных с НДС [#USR$TRANSSUMNCU/M/a] руб. в том числе 18% НДС [#USR$TRANSVATNCU/M/a] руб.
-------------------------------------------------------------------------------------------------------------------------
 Итого :                              [#Q1/M/S/R][#W/M/S/R][#F9/M/S/R]          [#F6/R/S/M ]     [#N/R/S/M ][#F20/R/M   ]
                                                                                        в том числе тара: [#F21/R/Z/M ]

Удостоверение о качестве № [#USR$CERTIFICATENUMBER/a] от [#DOCUMENTDATE/a] Дата и час выработки продукции [#DOCUMENTDATE/a]
Дата конечного срока реализации_____
Всего отпущено на сумму: [#F5/S/M/A]([#F5/S/T/A])руб.
По доверенности № [#WARRANTNUMBER/A] от [#WARRANTDATE/A]
Указанный груз за исправной пломбой                |Указанный груз за исправной пломбой,              [#F6/S/R/M    ]Товар
 тарой и упаковкой_____                            | тарой и упаковкой_____                          |[#F6/S/R/M    ]Товар
без НДС
Кол-во мест [#F7/S/T/c/Z                          ]|Кол-во мест [#F7/S/T/c/z                        ]|[#F5/S/R/M    ]Товар
с НДС
Масса нетто [#F9/S/a/M/Z] кг. Масса брутто [#BR/a/M/Z] кг.
к перевозке сдал [#SURRENDER                       ]|водитель-экспедитор  [#FORWARDERKEY_NAME        ]|[#F0/R/M      ]К
оплате
принял водитель-экспедитор [#RECEPTION             ]|принял _____                                   |[#ALLVAT/R/M  ]НДС
Получил _____
%page_
%report
```

Документы

Документом в терминах системы Гедымин называется особый вид бизнес-объекта, предназначенный для отражения событий, произошедших в процессе хозяйственной жизнедеятельности предприятия. Одно из основных отличий документов от прочих бизнес-объектов заключается в том, что при записи документа в базу данных могут быть автоматически сформированы связанные с этим документом хозяйственные операции и, соответственно, проводки по счетам бухгалтерского или управленческого учета. Кроме этого, в базовый класс документа встроена функция генерации и присваивания следующего порядкового номера, который может быть как простым целым числом, так и более сложной последовательностью символов, составленной на основании заданного шаблона.

Ниже приведены несколько примеров документов, часто встречающихся в повседневной практике предприятия:

- Товарная накладная — фиксирует факт передачи товарно-материальных ценностей;
- Платежное поручение — описывает распоряжение банку на перевод денежных средств с расчетного счета предприятия;
- Договор на оказание услуг — определяет сроки и условия взаимоотношений между организацией и ее клиентом;

Виды документов

С точки зрения Гедымина существуют четыре вида документов:

1. Встроенные документы («зашиты» непосредственно в платформу. Код встроенного бизнес объекта написан в Делфи, откомпилирован и находится непосредственно в файле gedemin.exe. Необходимые для хранения данных встроенного объекта таблицы находятся в эталоне базы данных и, как правило, не имеют префикса USR$. Пользователь, настройщик системы имеет возможность изменять поведение встроенного документа путем перекрытия методов и добавления своих полей в таблицы бизнес объекта);
2. Документы пользователя (поставляются в составе настройки или создаются и конфигурируются пользователем системы, настройщиком);
3. Складские документы (поставляются в составе настройки или создаются и конфигурируются пользователем системы, настройщиком. От документов пользователя складские документы отличаются тем, что при их создании, изменении или удалении происходит изменение информации в специальных таблицах, которые отвечают за учет движения ТМЦ, контроль остатков и т.п.);
4. Документы типа «Прайс-лист» (по сути, это вовсе и не документы, а справочники, но так исторически сложилось, что прайс-лист был отнесен к категории документов. Для хранения информации прайс-листов используются таблицы INV_PRICE, INV_PRICELINE в связке с таблицей GD_DOCUMENT.)

Ниже приведена иерархия бизнес-классов документов:

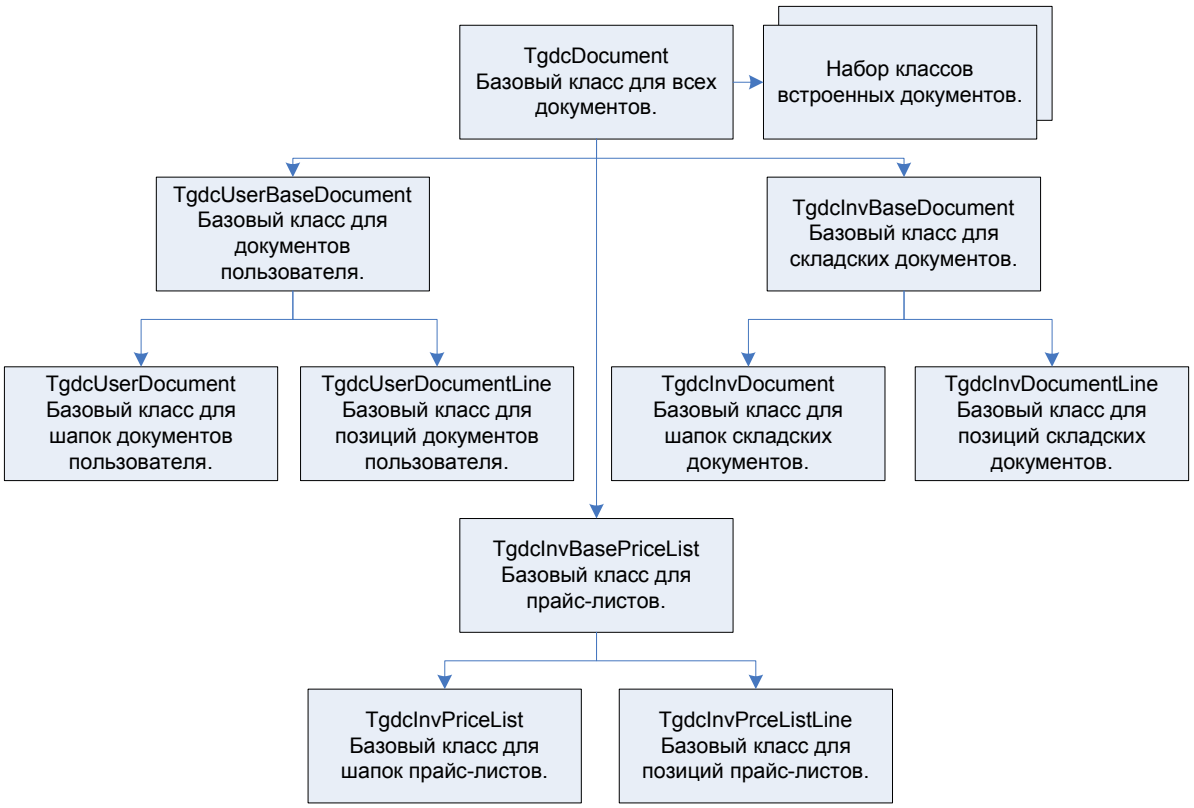


Рис. 53 Иерархия базовых классов документов.

Структура метаданных

Документы могут быть однопозиционными и многопозиционными. В первом случае существует одна таблица, содержащая специфические данные документа. Каждая запись из этой таблицы связана один-к-одному с записью в таблице GD_DOCUMENT, которая содержит общие данные документа, такие как: идентификатор, номер, дата, дескрипторы безопасности и т.п. В случае многопозиционного документа существуют две таблицы: одна содержит шапки документов, а вторая — позиции. Каждая запись как из таблицы шапок, так и из таблицы позиций связана один-к-одному с записью в таблице GD_DOCUMENT. Связь мастер-дитэйл между шапкой и ее позициями организуется внутри таблицы GD_DOCUMENT через поле PARENT, которое для записей-позиций документа содержит ссылку на запись его шапки.

Схематически, организацию данных документов в базе данных можно представить следующим образом:

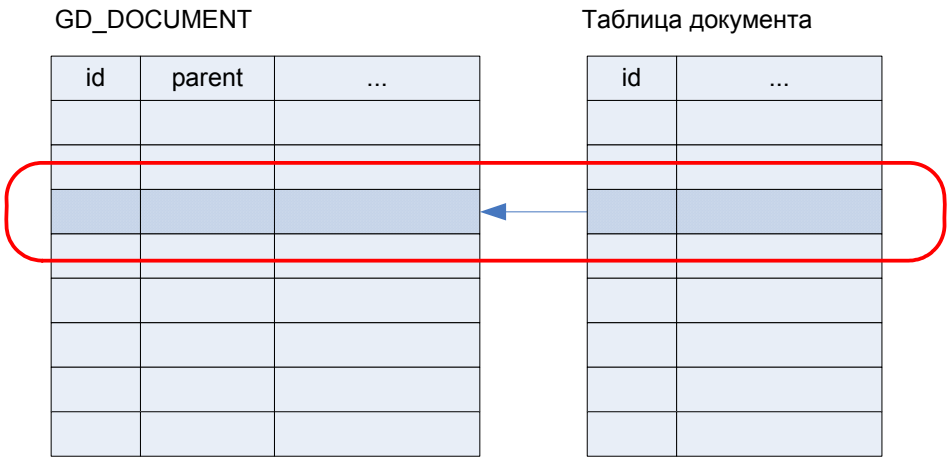


Рис. 54 Хранение данных документов в базе.

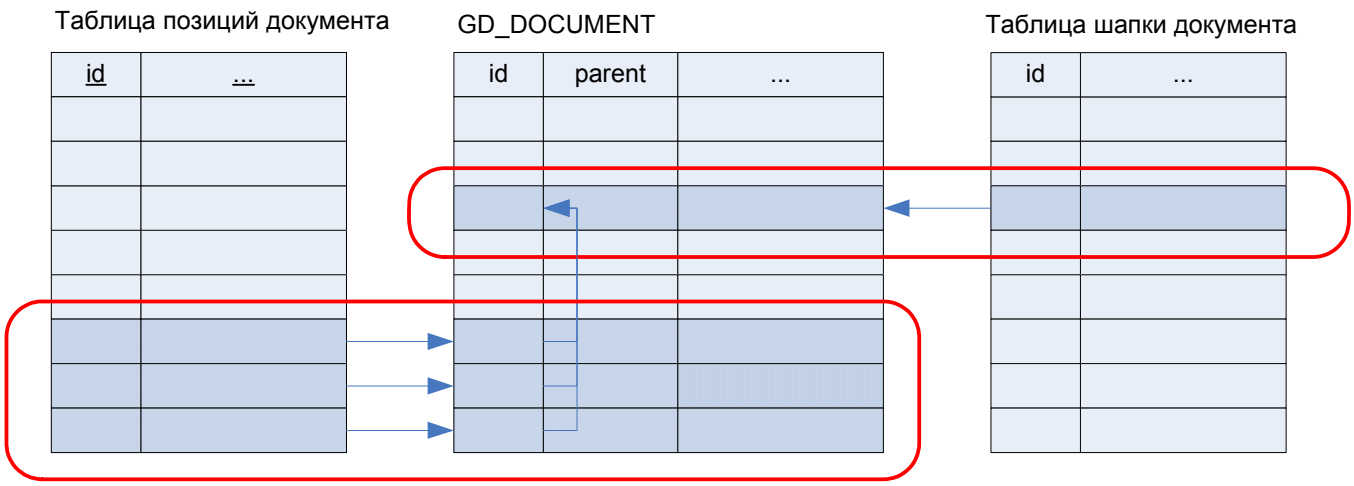


Рис. 55 Организация хранения данных документов в БД.

GD_DOCUMENT

PK	FK	Поле Домен Тип данных	NN	Описание
1		ID DINTKEY INTEGER	☒	Уникальный идентификатор.
	F	PARENT DFOREIGNKEY INTEGER		Для позиции документа — ссылка на шапку. Для шапки поле содержит NULL. Т.е. таблица документов может иметь максимум два уровня вложенности.
	F	DOCUMENTTYPEKEY DINTKEY INTEGER	☒	Ссылка на таблицу GD_DOCUMENTTYPE — тип документа.
	F	TRTYPEKEY		Поле сейчас не используется.
	F	TRANSACTIONKEY		Ссылка на хозяйственную операцию, связанную с данной записью. Ссылка на таблицу AC_TRANSACTION.
		NUMBER	☒	Номер документа.
		DOCUMENTDATE	☒	Дата документа.
		DESCRIPTION		Произвольное описание документа, коментарий.
		SUMNCU		Сумма в НДЕ.
		SUMCURR		Сумма в валюте.
		SUMEQ		Сумма в эквиваленте.
		DELAYED		Флаг отложенного документа. Если установлен, то по такому документу (позиции документа) не будет сформирована хозяйственная операция и, соответственно, проводки. Если это складской документ, то не будет сформировано движение ТМЦ.
		AFULL		Дескриптор безопасности. Полный доступ.
		ACHAG		Дескриптор безопасности. Просмотр и изменение.
		AVIEW		Дескриптор безопасности. Только просмотр.

		CURRKEY		Ссылка на валюту, в которой выражена сумма, находящаяся в поле SUMCURR.
		COMPANYKEY		Ссылка на рабочую организацию, к которой относится данный документ.
		CREATORKEY		Создатель документа. Ссылка на таблицу GD_CONTACT.
		CREATIONDATE		Дата создания.
		EDITORKEY		Редактор документа. Ссылка на таблицу GD_CONTACT.
		EDITIONDATE		Дата последнего изменения.
		PRINTDATE		Дата печати документа. Фактически, это дата, когда документ был открыт для просмотра перед печатью. Был ли послан документ на принтер и, если да, то напечатал ли он его не фиксируется.
		DISABLED		Документ отключен.
		RESERVED		Зарезервировано.

Тип документа

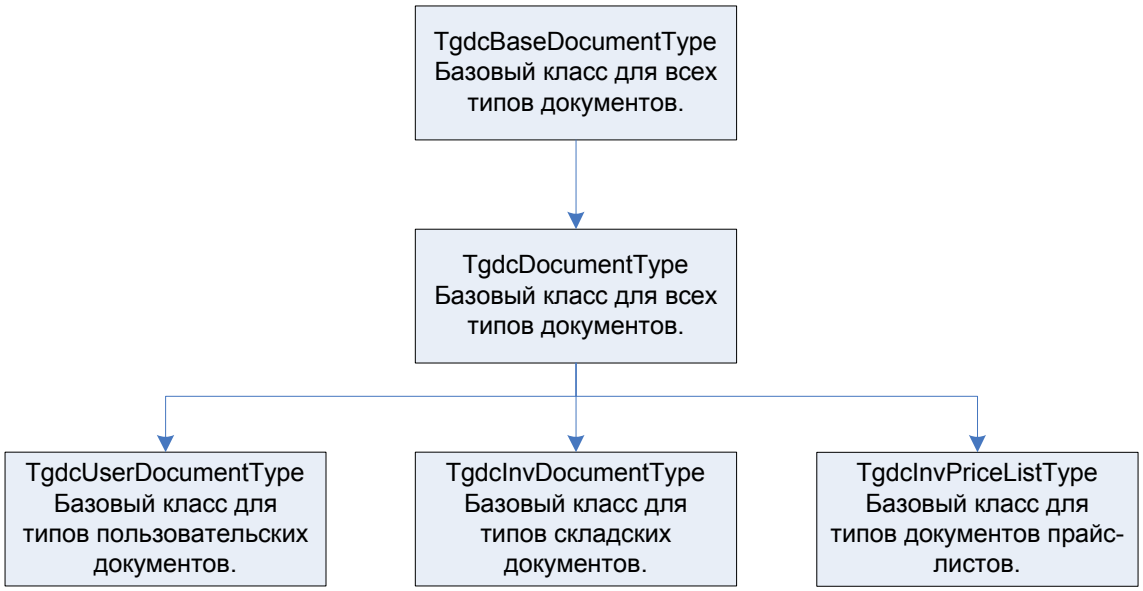


Рис. 56 Иерархия классов типов документов.

GD_DOCUMENTTYPE

Для упорядочения визуального отображения типы документов организованы в иерархическую древовидную структуру. И ветви дерева и, непосредственно, типы документов хранятся в одной таблице GD_DOCUMENTTYPE. Как это принято в Гедымине, связь между дочерними и родительским объектом организована с помощью поля-ссылки PARENT. С помощью поля DOCUMENTTYPE мы различаем, что сохранено в конкретной записи: ветка или тип документа.

PK	FK	Поле Домен Тип данных	NN	Описание
1		ID DINTKEY INTEGER	☒	Уникальный идентификатор.

		PARENT DFOREIGNKEY INTEGER		Ссылка на родительский объект.
		LB DLB INTEGER		Левая граница интервала данной записи.
		RB DRB INTEGER		Правая граница интервала данной записи.
		NAME DNAME VARCHAR(60)		Наименование ветки или типа документа.
		DESCRIPTION DTEXT180 VARCHAR(180)		Произвольное текстовое описание.
		CLASSNAME DCLASSNAME VARCHAR(40)		Имя Делфи класса типа документа. Одно из трех возможных значений: TgdcUserDocumentType, TgdcInvDocumentType и TgdcInvPriceListType.
		DOCUMENTTYPE DDOCUMENTTYPE VARCHAR(1)		Определяет тип информации, которая хранится в данной записи: D — тип документа, B — ветка в дереве типов документов.
		OPTIONS DBLOB1024 BLOB		Двоичные данные. Параметры типа документа.
		HEADERRELKEY DFOREIGNKEY INTEGER		Ссылка на таблицу at_relations. Таблица шапки документа.
		LINERELKEY DFOREIGNKEY INTEGER		Ссылка на таблицу at_relations. Таблица позиций документа.
		AFULL DSECURITY INTEGER		Дескриптор безопасности полного доступа.
		ACHAG DSECURITY INTEGER		Дескриптор безопасности доступ на изменение.
		AVIEW DSECURITY INTEGER		Дескриптор безопасности доступ на просмотр.
		DISABLED DDISABLED INTEGER		
		RUID DRUID VARCHAR(21)		Идентификатор типа документа. В качестве такового используется сформированная строка РУИД.
		BRANCHKEY DFOREIGNKEY INTEGER		
		REPORTGROUPKEY DFOREIGNKEY INTEGER		

		RESERVER DRESERVED INTEGER		Зарезервировано для будущих версий или для использования настройщиком системы по своему усмотрению.
		ISCOMMON DBOOLEAN SMALLINT		Если флаг установлен, то документы такого типа являются Общими. Т.е. при просмотре списка будут выводиться все документы, не зависимо от того, к какой из рабочих организаций они относятся. Если флаг не установлен, то в списке документов будут отображаться только те, которые относятся к текущей рабочей организации. Пример общего документа: прайс лист. Все прайс листы будут доступны при просмотре под любой из рабочих организаций. Пример не общего документа: платежное поручение. При работе под конкрентой рабочей организацией будут отображаться платежные поручения только этой организации.
		ISCHECKNUMBER DBOOLEAN SMALLINT		Данное поле определяет: осуществлять проверку на уникальность номера документа или нет.
		HEADERFUNCTIONKEY DFOREIGNKEY INTEGER		Ссылка на скрипт-функцию, формирующую хозяйственные операции по шапке документа. Ссылка на таблицу GD_FUNCTION.
		HEADERFUNCTIONTEMPLATE DBLOB80 BLOB		Шаблон скрипт-функции, формирующей хозяйственные операции по шапке документа.
		LINEFUNCTIONKEY DFOREIGNKEY INTEGER		Ссылка на скрипт-функцию, формирующую хозяйственные операции по позиции документа.
		LINEFUNCTIONTEMPLATE DBLOB80 BLOB		Шаблон скрипт-функции, формирующей хозяйственные операции по позиции документа. Ссылка на таблицу GD_FUNCTION.

Пример создания типа документа

Представим такую ситуацию: наша организация оказывает некоторые услуги клиентам. С каждым клиентом мы заключаем договор. Договор имеет следующие реквизиты:

- Номер, дату заключения;
- Реквизиты контрагента;
- Дату, до которой действует договор;
- Услуга, которая оказывается по настоящему договору;
- Стоимость работ.

При заключении договора у клиента возникает задолженность перед нами на сумму договора. При поступлении денег на наш расчетный счет эта задолженность погашается.

Необходимо организовать ввод и учет договоров в базе данных.

Для решения поставленной задачи необходимо выполнить следующие шаги:

1. Создать тип документа Договор;
2. Настроить форму просмотра договоров;
3. Настроить диалоговое окно для ввода или изменения Договора;
4. Создать выходную форму для печати одного договора;
5. Создать выходную форму для печати реестра договоров за указанный период;

Создание типа документа

Для создания типа документа необходимо открыть окно «Типовые документы». Соответствующая команда находится в Исследователе, в разделе Сервис. Далее, необходимо выполнить следующие действия:

1. В дереве, в левой части окна, установить курсор на раздел «Документ пользователя»;
2. В правой части окна открыть на панели инструментов меню создания нового типа и выбрать команду «Добавить документ пользователя» (см. рисунок внизу).

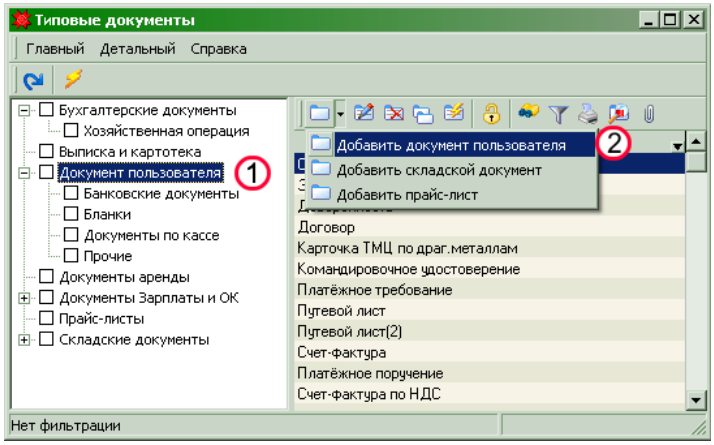


Рис. 57 Вызов команды создания пользовательского типа документа.

На экране откроется диалоговое окно создания нового типа документа. В этом окне выполним следующие действия:

1. Установим курсор в поле «Наименование документа» и введем наименование типа документа «Договор на оказание услуг»;
2. Установим курсор в поле «Наименование на английском» и введем «TST_CONTRACT»;
3. Установим курсор в поле «Ветка для команды вызова» и выберем из выпадающего списка раздел «Бухгалтерия»;
4. Установим курсор в поле «Таблица шапки документа» и нажмем на клавиатуре клавишу F2 для того, чтобы создать новую таблицу в базе данных.

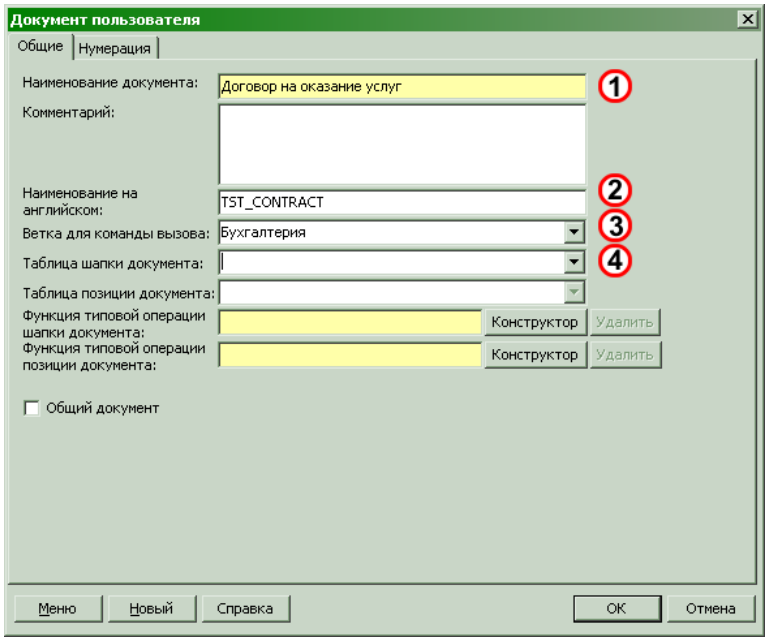


Рис. 58 Создание документа пользователя.

На экране откроется диалоговое окно создания новой таблицы:

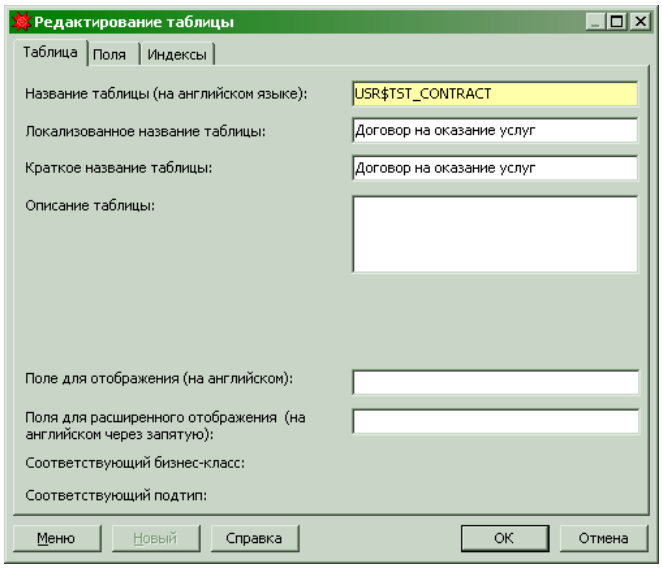


Рис. 59 Создание новой таблицы.

Обратите внимание, что система автоматически сгенерировала наименование таблицы, прибавив префикс USR$ к английскому наименованию типа документа. Так же автоматически были заполнены поля «Локализованное название таблицы» и «Краткое название таблицы».

Перейдем на закладку поля:

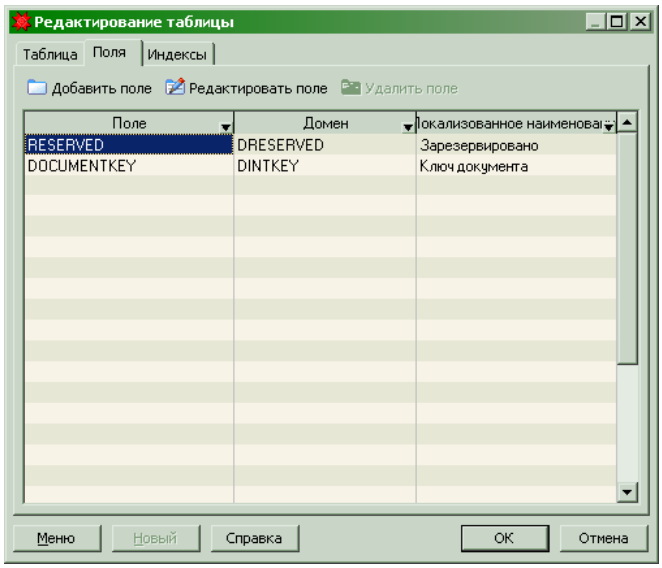


Рис. 60 Добавление полей документа.

Обратите внимание, что два поля уже добавлены системой автоматически. Это поле RESERVED и поле DOCUMENTKEY — ссылка на таблицу GD_DOCUMENT.

Над списком полей расположена кнопка «Добавить поле», которой мы и воспользуемся для того, чтобы добавить следующие поля:

- Ссылку на организацию котрагента;
- Дату истечения срока действия договора;
- Ссылку на услугу, оказываемую по договору;
- Стоимость работ по договору.

Поля «Номер документа» и «Дата документа» добавлять нет необходимости, поскольку они уже присутствуют в таблице GD_DOCUMENT. Возникает вопрос: зачем добавлять поле «Стоимость работ»,

если оно так же присутствует в таблице GD_DOCUMENT[14]? Ответ очень простой: добавлять поля в таблицу конкретного документа или нет зависит от разработчика. Мы придерживаемся такого правила, что все стоимостные (ценовые) поля должны быть вынесены в таблицы конкретного документа. Почему? Потому что, стоимостей (цен) может быть больше, чем ценовых полей в таблице GD_DOCUMENT: учетная цена, цена с НДС, цена с налогом с продаж (НП), цена первого поставщика и т.п, а держать часть цен в одной таблице, а часть в другой не очень хорошая практика.

Добавление ссылки на контрагента

Нажмем кнопку «Добавить поле» и заполним поля «Название поля на английском языке», «Локализованное название» и «Краткое название» значениями, как показано на рисунке.

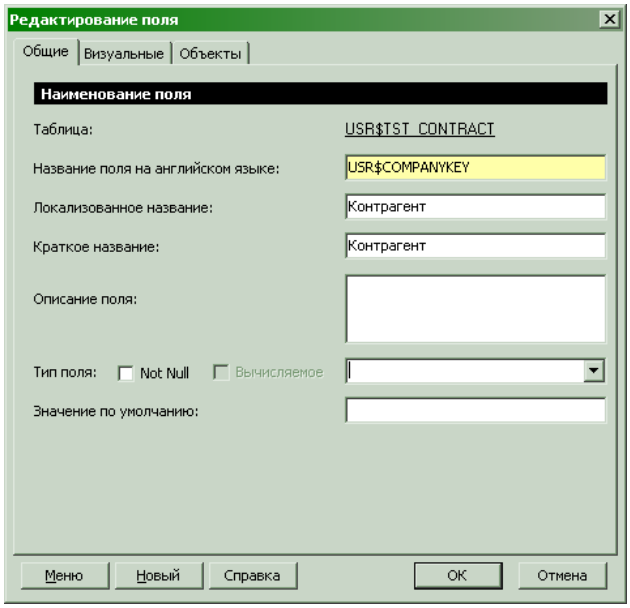


Рис. 61 Добавление ссылки на контрагента.

Теперь необходимо создать тип поля (домен). Установим курсор на выпадающий список «Тип поля» и нажмем клавишу F2 на клавиатуре. Заполним поля на первой вкладке диалогового окна значениями, как показано на рисунке ниже. Обратите внимание на то, что наименование типа поля — USR$TST_DCOMPANYKEY — имеет сразу три префикса и один суффикс. Префикс USR$ показывает, что это пользовательский тип поля, т.е. тип, созданный настройщиком (пользователем системы), а не входящий в эталон базы данных. Префикс TST_ мы избрали для нашего примера и, наконец, префикс D показывает, что с данное имя принадлежит объекту типа домен. Единственный суффикс KEY указывает на то, что данный тип — это ссылка на запись в другой таблице (FOREIGN KEY).

[14] Имеется ввиду поле SUMNCU.

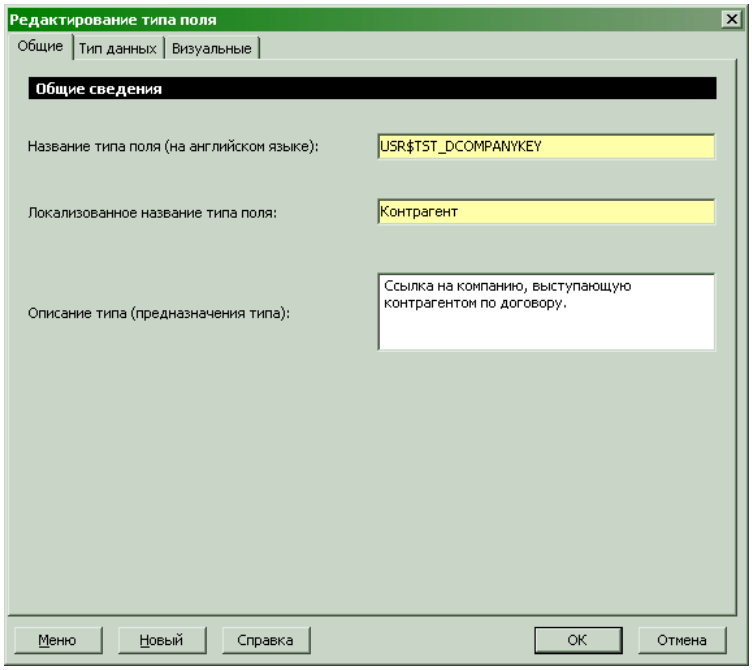


Рис. 62 Создание типа поля.

Перейдем на вторую вкладку — «Тип данных». Из выпадающего списка «Ссылка на таблицу» выберем таблицу «Компания» (GD_COMPANY). После этого перейдем в следующее поле и выберем поле для отображения — «Полное наименование». Пометим флажок «Поле обязательно должно быть заполнено» в нижней части вкладки.

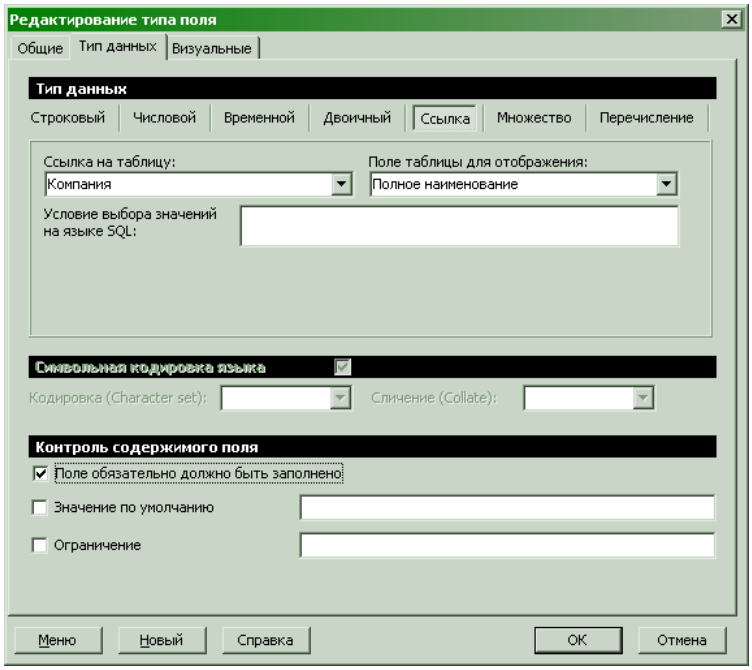


Рис. 63 Создание типа поля.

На последней, третьей по счету вкладке выберем из выпадающего списка «Бизнес-класс» строчку «Организация(TgdcCompany)».

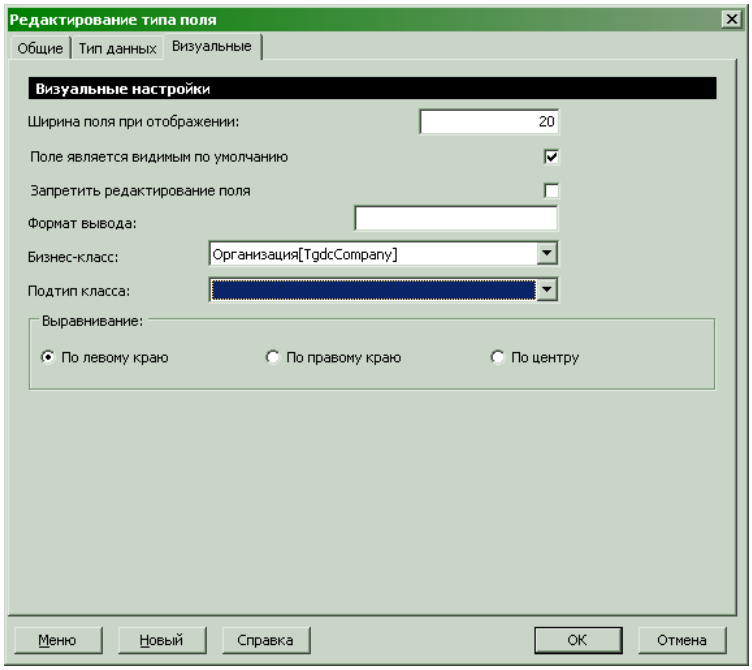


Рис. 64 Создание типа поля.

На этом создание типа поля можно считать завершенным. Нажимаем кнопку «Ок» для того, чтобы записать изменения в базу данных.

На экране откроется окно, в котором будет показана команда создания нового домена в базе данных.

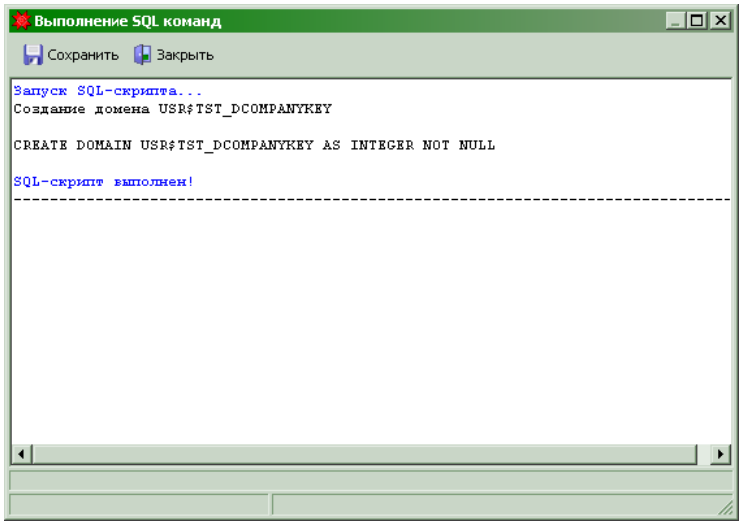


Рис. 65 Создание типа поля.

Если все прошло успешно, закрываем это окно с помощью кнопки «Закрыть» на панели инструментов и возвращаемся в диалоговое окно создания нового поля. Выпадающий список «Тип поле» теперь содержит значение «Контрагент». Поскольку, больше никаких параметров указывать не надо, нажимаем кнопку «Ок» для того, чтобы создать поле.

Добавление срока действия договора

Снова воспользуемся кнопкой «Добавить поле». На это раз последовательность действий будет гораздо проще. Нам не придется создавать свой тип поля, мы воспользуемся стандартным типом для хранения даты (ddate), выбрав его из выпадающего списка.

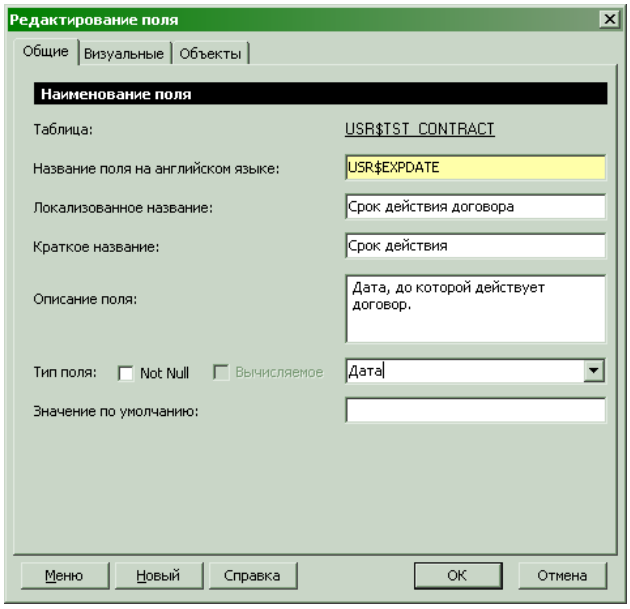


Рис. 66 Добавление срока действия договора.

Срок окончания действия договора может быть и не указан, поэтому мы не делаем это поле обязательным для заполнения.

После ввода всех параметров нажимаем кнопку «Ок».

Добавление ссылки на услугу

Добавление поля ссылки на услугу, оказываемую по договору, во многом похоже на проделанную нами ранее операцию по добавлению реквизита «Контрагент». Выберем команду «Добавить поле» на панели инструментом и заполним поля диалогового окна значениями, как показано на рисунке:

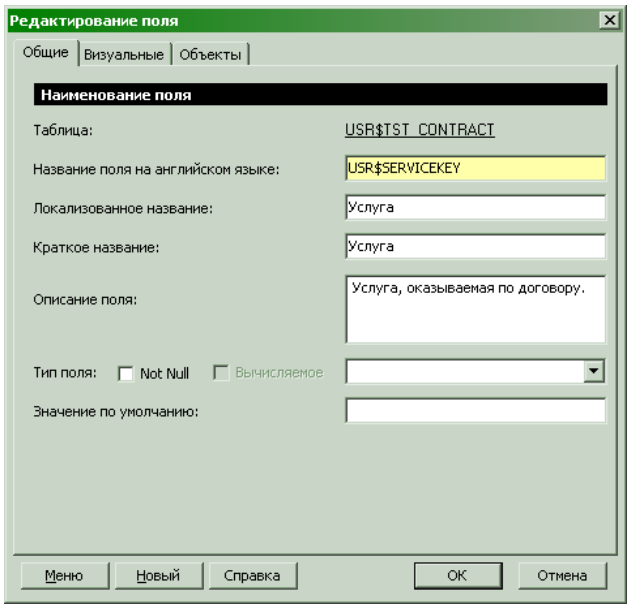


Рис. 67 Добавление ссылки на услугу.

Нам снова предстоит создать тип поля. Поэтому установим курсор в выпадающий список «Тип поля» и нажмем F2. На экране появится диалоговое окно создания типа поля (домена).

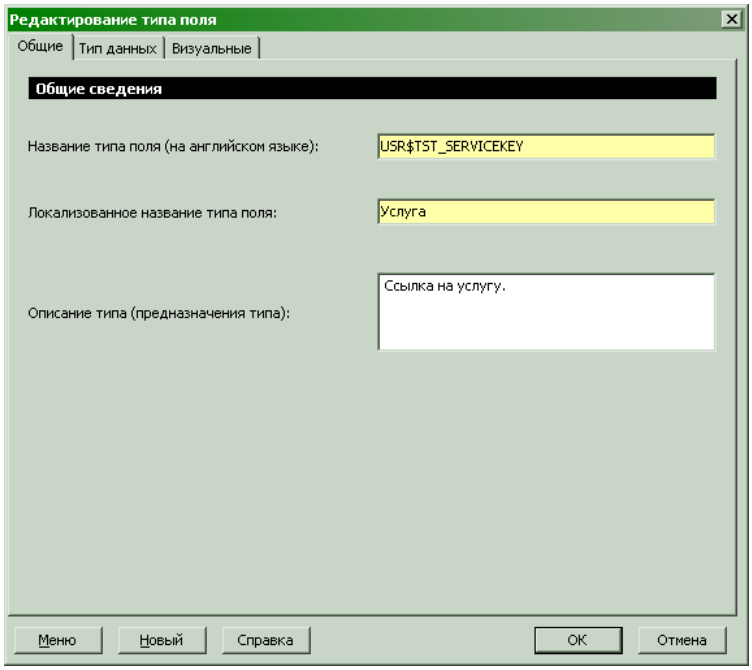


Рис. 68 Добавление типа поля.

После заполнения полей на первой вкладке окна, как показано на предыдущем рисунке перейдем на вторую вкладку:

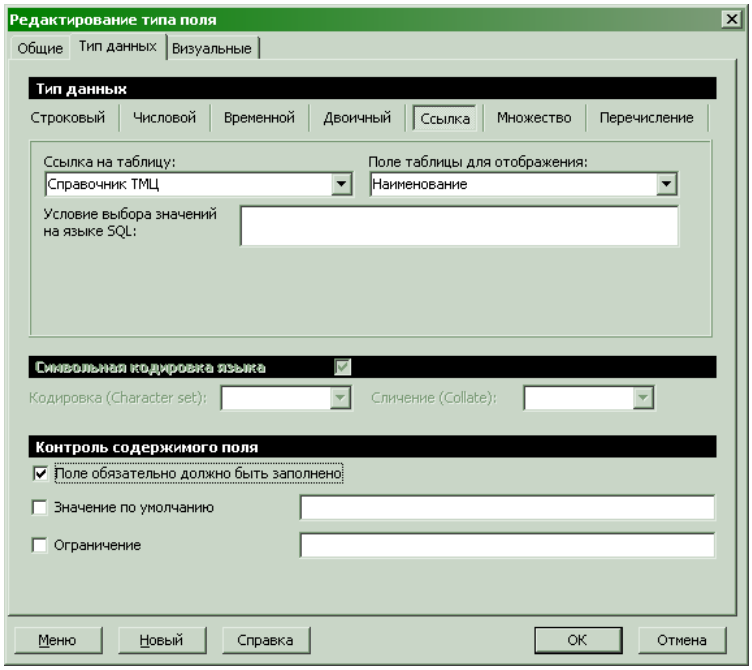


Рис. 69 Добавления типа поля.

Выберем тип поля «Ссылка». Из выпадающего списка «Ссылка на таблицу» выберем запись «Справочник ТМЦ» (можно искать в списке как указанное наименование, так и имя таблицы в базе данных — gd_good). В выпадающем списке «Поле таблицы для отображения» выберем поле «Наименование» (Name). В разделе «Контроль содержимого поля» внизу окна пометим флаг «Поле обязательно должно быть заполнено» и нажмем кнопку «Ок». На экране должно появится окно со списком выполненных SQL команд для создания домена в базе данных. Если все прошло успешно, закроем это окно, воспользовавшись кнопкой «Закрыть» на панели инструментов. Теперь, осталось только нажать кнопку «Ок» в диалоговом окне создания новго поля.

Добавление поля стоимости работ

Последнее поле, которое нам предстоит добавить, предназначено для хранения стоимости работ по договору. Снова воспользуемся командой «Добавить поле». Заполним поля диалогового окна значениями, как показано на рисунке ниже. Из выпадающего списка «Тип поля» выберем тип «Сумма» (домен DCURRENCY). Установим флаг NOT NULL, т.е. поля является обязательным для заполнения.

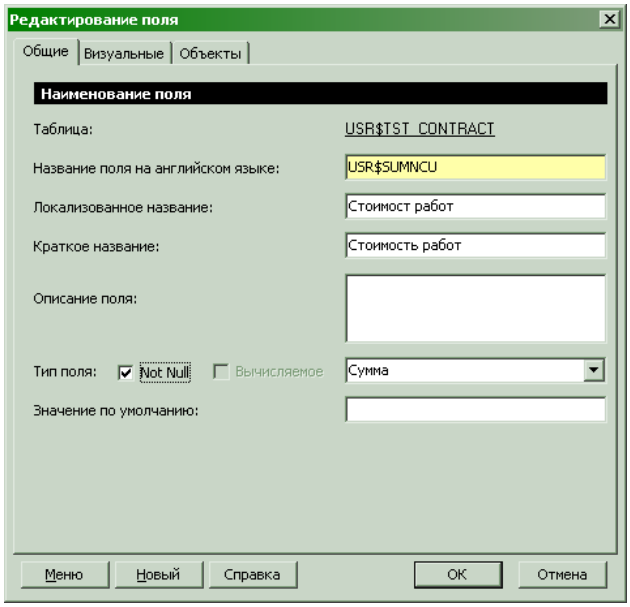


Рис. 70 Добавление поля Стоимост работ.

По окончании ввода нажимаем «Ок». Теперь, окно со списком полей создаваемого документа, должно выглядеть так, как показано на следующем рисунке.

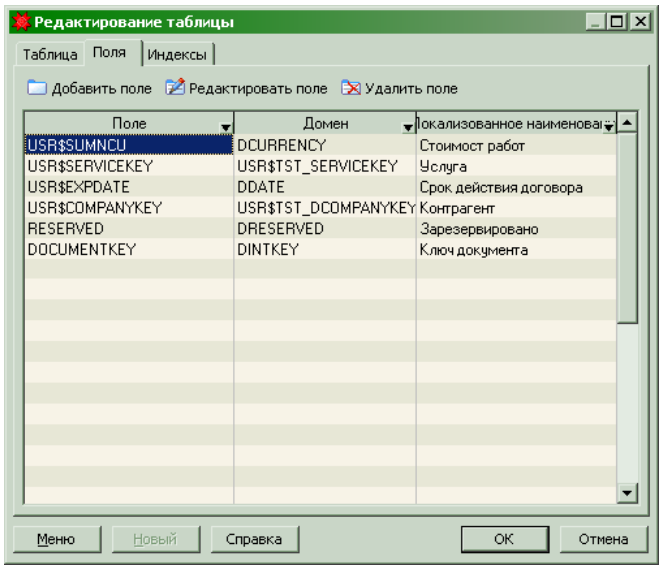


Рис. 71 Список полей документа.

Для создания таблицы документа в базе данных нажимаем «Ок». На экране появится уже знакомое нам окно, в котором будут показаны выполняемые SQL команды.

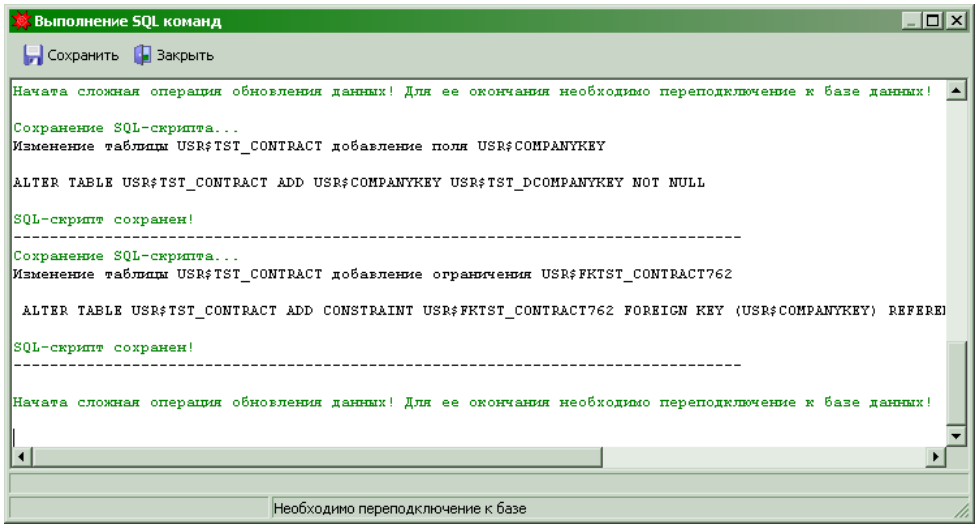


Рис. 72 Ход создания метаданных.

Обратите внимание, что часть сообщений выводится зеленым цветом, при этом в строке состояния окна появляется надпись «Необходимо переподключение к базе». Это означает, что некоторые команды изменения структуры базы данных (как правило, это команды создания внешних ссылок) не могут быть выполнены в рамках текущего сеанса. Такие команды записываются в отдельную таблицу (AT_TRANSACTION) и будут выполнены при следующем подключении к базе данных. Таким образом, для создания типа документа, по завершении выполнения всех SQL команд, необходимо отключиться от базы данных используя соответвующую команду в меню «База данных» главного окна и подключиться к ней вновь, используя команду «Подключиться к базе данных» там же[15]. Если при подключении к базе данных в таблице AT_TRANSACTION присутствуют отложенные операции, то на экран будет выдан вопрос о их выполнении:

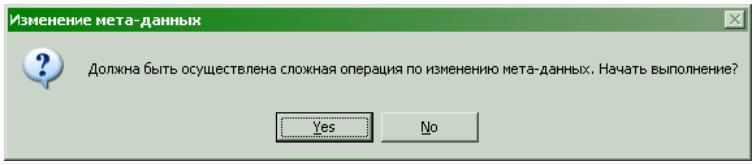


Рис. 73 Вопрос о выполнении операции по изменению мета-данных.

Следует ответить на него утвердительно.

Еще раз напомним, что создание типа документа и последующее переподключение к базе данных следует производить только под учетной записью Администратор.

Поскольку мы добавили несколько обязательных для заполнения полей, в процессе создания соответствующих структур в базе данных будет предложено установить для них значения.

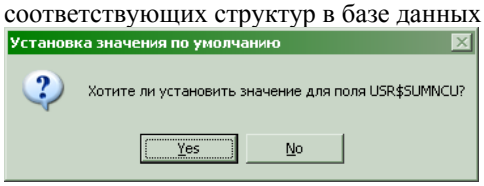


Рис. 74 Вопрос об установке значения обязательного поля.

Очевидно, что в нашем случае устанавливать значения бесполезно, так как создается новая таблица, в которой нет ни одной записи.

Настройка экранных форм

Итак, мы создали новый тип документа. Попробуем обратиться к нему. При создании, мы указали, что команду вызова формы просмотра данного документа следует разместить в ветке «Бухгалтерия» Исследователя системы, поэтому откроем Исследователь и развернем указанную ветвь.

---

[15] Альтернативный способ переподключения — это выйти из Гедымина и запусть его вновь.

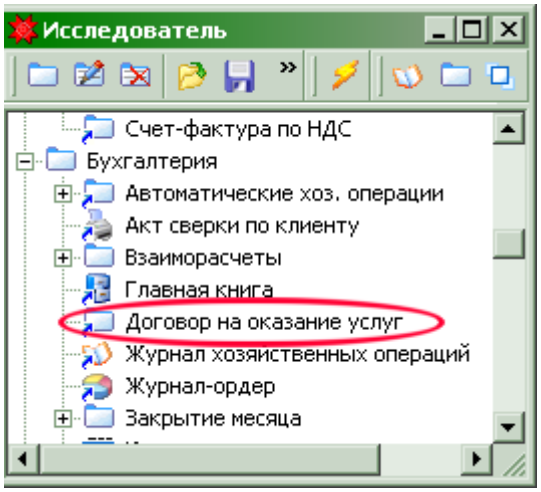
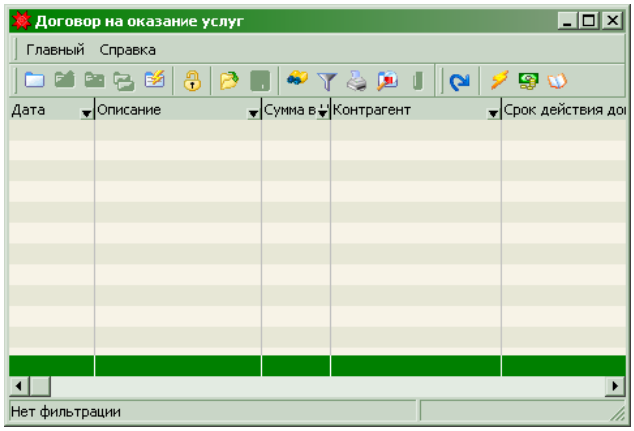


Рис. 75 Команда вызова документа в Исследователе.

Как видно на представленном рисунке, в разделе Бухгалтерия действительно появилась команда «Договор на оказание услуг». Если мы дважды щелкнем по ней, то на экране откроется окно с пустым списком наших документов:



Первым делом следует настроить внешний вид таблицы. Ведь система, ничего не знает о том, какие поля являются значимыми для нас, а какие нет. Нажмем F10 на клавиатуре и выполним действия по настройке. Мы оставим следующие колонки видимыми (в скобках приведены имена полей из запроса), скрыв все остальные:

- Номер договора (NUMBER);
- Дата договора (DOCUMENTDATE);
- Срок действия договора (USR$EXPDATE);
- Контрагент (U_USR$COMPANYKEY_FULLNAME);
- Услуга (U_USR$SERVICEKEY_NAME);
- Стоимость (USR$SUMNCU);

Нам также придется переименовать поля Контрагент (называлось «Полное наименование), Услуга (называлось «Наименование») , Стоимость (называлось «Стоимость работ») и Срок действия (называлось «Срок действия договора»).

Обратите внимание на имена полей в запросе. С полями NUMBER и DOCUMENTDATE все достаточно просто: отсутствие префикса USR$ указывает на то, что это стандартные поля, присутствующие в эталонной структуре базы данных. Эти поля находятся в таблице GD_DOCUMENT. Не намного сложнее с полями USR$EXPDATE и USR$SUMNCU. Это поля из созданной нами таблицы USR$TST_CONTRACT. Но, что за странные имена у полей Контрагент и Услуга? Откуда они появились?

Для того, чтобы дать ответ на поставленный вопрос, следует вспомнить, что мы добавили в таблицу USR$TST_CONTRACT две ссылки, на компанию и на услугу. Но ссылка, это всего лишь целочисленное поле хранящее идентификатор (значение первичного ключа) записи. Вряд ли, пользователю будет какой-то толк от информации, что контрагентом по договору выступает организация 153699432, а оказанная услуга — 245910718. Для того, чтобы упростить нам жизнь, Гедымин автоматически добавил в запрос две таблицы GD_COMPANY и GD_GOOD и связал их с таблицей USR$TST_CONTRACT. Соответственно, под алиасом U_USR$COMPANYKEY_FULLNAME[16] скрывается поле FULLNAME из таблицы GD_COMPANY, т.е. полное наименование компании контрагента, а под алиасом U_USR$SERVICEKEY_NAME — поле NAME из таблицы GD_GOOD — наименование услуги.

Для наиболее дотошных читателей приведем весь результирующий SQL запрос:

```
/* Текст запроса созданный компонентом фильтрации */
SELECT
  Z.ID,
  Z.PARENT,
  Z.DOCUMENTTYPEKEY,
  Z.TRTYPEKEY,
  Z.TRANSACTIONKEY,
  Z.NUMBER,
  Z.DOCUMENTDATE,
  Z.DESCRIPTION,
  Z.SUMNCU,
  Z.SUMCURR,
  Z.DELAYED,
  Z.AFULL,
  Z.ACHAG,
  Z.AVIEW,
  Z.CURRKEY,
  Z.COMPANYKEY,
  Z.CREATORKEY,
  Z.CREATIONDATE,
  Z.EDITORKEY,
  Z.EDITIONDATE,
  Z.PRINTDATE,
  Z.DISABLED,
  Z.RESERVED,
  U.DOCUMENTKEY,
  U.RESERVED,
  U.USR$COMPANYKEY,
  U.USR$EXPDATE,
  U.USR$SERVICEKEY,
  U.USR$SUMNCU,
  Z.USR$SORTNUMBER,
  Z.USR$EQRATE,
  U_USR$SERVICEKEY.NAME AS U_USR$SERVICEKEY_NAME,
  U_USR$COMPANYKEY.FULLNAME AS U_USR$COMPANYKEY_FULLNAME

FROM
  GD_DOCUMENT Z
    JOIN
      USR$TST_CONTRACT U
    ON
```

[16] Алиас U_USR$COMPANYKEY_FULLNAME можно расшифровать следующим образом:

- U — это префикс таблицы (в нашем случае USR$TST_CONTRACT);
- USR$COMPANYKEY — имя поля ссылки в таблице U;
- FULLNAME — имя поля в таблице, на которую ведет ссылка USR$COMPANYKEY, используемое для отображения записи (наименование записи).

```
      U.DOCUMENTKEY  =  Z.ID
    LEFT JOIN
      GD_GOOD U_USR$SERVICEKEY
    ON
      U_USR$SERVICEKEY.ID  =  U.USR$SERVICEKEY
    LEFT JOIN
      GD_COMPANY U_USR$COMPANYKEY
    ON
      U_USR$COMPANYKEY.CONTACTKEY  =  U.USR$COMPANYKEY

WHERE
  G_SEC_TEST ( Z.AVIEW,  - 1 )  <>  0
     AND
  Z.DOCUMENTTYPEKEY  =  147049339
     AND
  Z.PARENT IS NULL
     AND
  Z.COMPANYKEY IN ( 147049280 )
```

В итоге, после всех проделаных манипуляций, мы должны получить таблицу, вроде той, которая отображена на следующем рисунке:

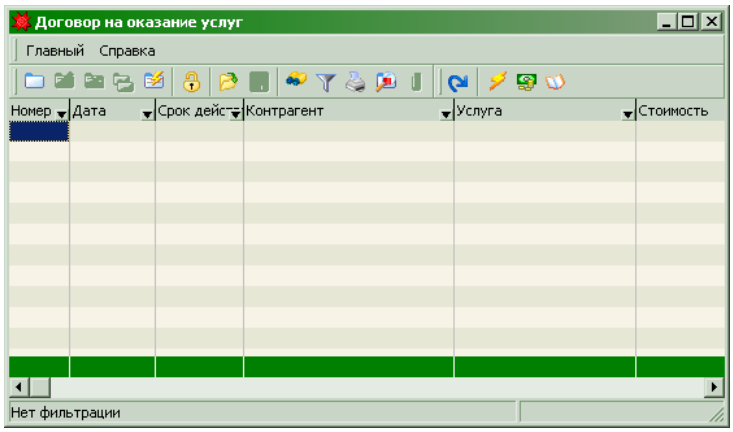
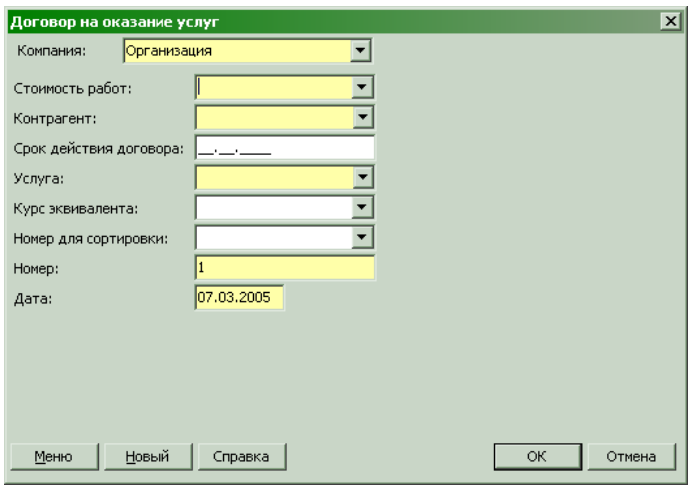


Рис. 76 Настроенная таблица документа.

Теперь создадим экземпляр документа — добавим первый договор на оказание услуг. Для этого выберем соответствующую кнопку на панели инструментов или, находясь в таблице нажмем Insert на клавиатуре.



Как видно на скриншоте, Гедымин обладает лишь самыми начальными представлениями в области дизайна. Не мудрствуя лукаво, он просто расположил поля нашего документа сверху вниз на диалоговом окне. Придется нам вмешаться. Воспользуемся коомбинацией клавишь Ctrl-Alt-E для того, чтобы перевести окно в режим дизайнера. Скроем ненужные поля, установив свойство Visible=False и

расположим видимые поля, так чтобы работа с ними была удобна пользователю. Поскольку понятие удобства достаточно субъективно, у вас может получится свой вариант оформления, отличный от того, который вышел у автора настоящей документации.

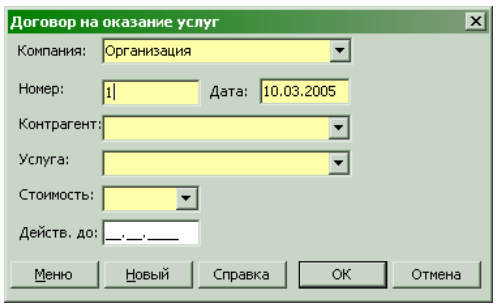


Рис. 77 Диалоговое окно "Договор на оказание услуг"

Настройка хозяйственных операций

Итак, мы создали новый тип документа и настроили его форму просмотра и диалоговое окно. Для того, чтобы сделать наш документ полноценным участником документооборота предприятия необходимо подключить к нему типовую хозяйственную операцию и настроить типовые проводки по этой хозяйственной операции.

При регистрации договора в базе данных, на сумму этого договора, должна формироваться бухгалтерская проводка дебет счета 62 «Расчеты с покупателями и заказчиками», кредит 98 «Доходы будущих периодов»[17]. Проводка должна регистрироваться на дату договора. Аналитические признаки проводки должны заполняться значениями «Клиент» и «Услуга» из договора.

Добавление новой хозяйственной операции

Откроем Исследователь системы, перейдем в раздел «Бухгалтерия» и вызовем команду «Типовые хоз. операции». Окно типовых хозяйственных операций разделено на две части по вертикали. Слева находится дерево типовых операций, а справа отображается список типовых проводок для выделенной в данный момент типовой хозяйственной операции.

Перейдем в левую часть окна и из контекстного меню вызовем команду «Добавить» (можно также воспользоваться клавишей Insert на клавиатуре). Заполним поля, как показано на рисунке ниже и нажмем клавишу «Ок».

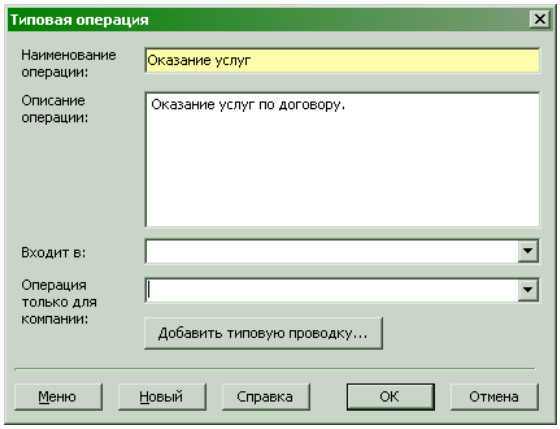


Рис. 78 Создание новой типовой операции.

Добавление типовой проводки

Теперь создадим типовую проводку. Оставим курсор в дереве на позиции «Оказание услуг» и перейдем в правую часть окна. Добавим новую запись и заполним поля, как показано на рисунке:

---

[17] Корреспонденция счетов приводится в соответствии с планом счетов бухгалтерского учета, введенным в Республике Беларусь с 1 января 2004г.

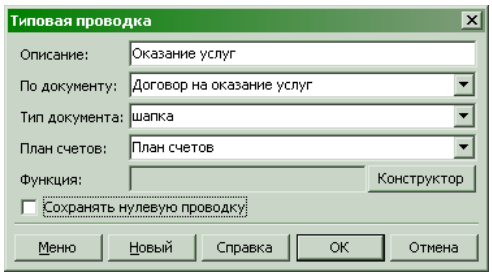


Рис. 79 Добавление новой типовой проводки.

Теперь мы должны задать алгоритм по которому будет генерироваться проводка для документа. Для того, чтобы настройщику системы не писать код вручную в Гедымине предусмотрен «Конструктор функций», с помощью которого можно составлять функцию из готовых блоков. Откроем «Конструктор» с помощью одноименной кнопки в диалогово окне «Типовая проводка».

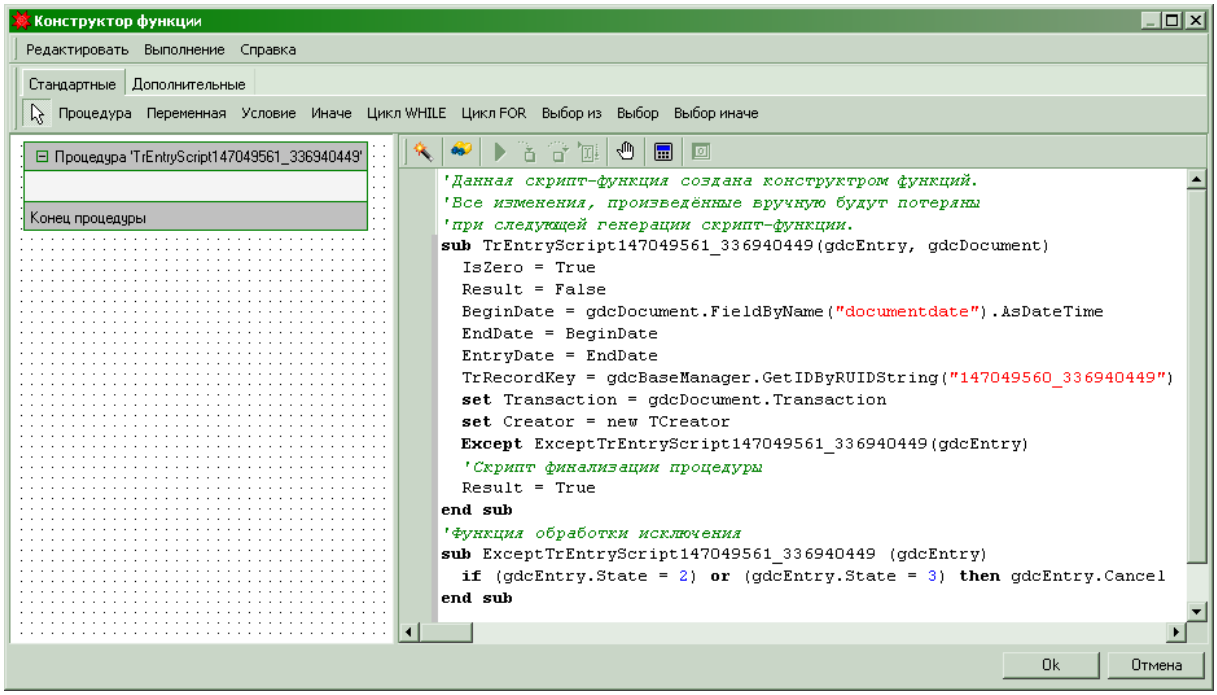


Рис. 80 Конструктор функции.

Основное рабочее пространство конструктора функции разделено по вертикали на две части. В левой части отображаются блоки, из которых состоит функция, в правой — выводится ее исходный код. Над рабочей областью располагается палитра структурных блоков (по аналогии с палитрой компонентов в Делфи). Палитра имеет две вкладки: Стандартные и Дополнительные. Для того, чтобы добавить структурный блок в тело функции необходимо выделить его на палитре и затем щелкнуть мышью внутри того блока, в правой части окна, куда необходимо добавить новый блок. Данный процесс проиллюстрирован на рисунке ниже:

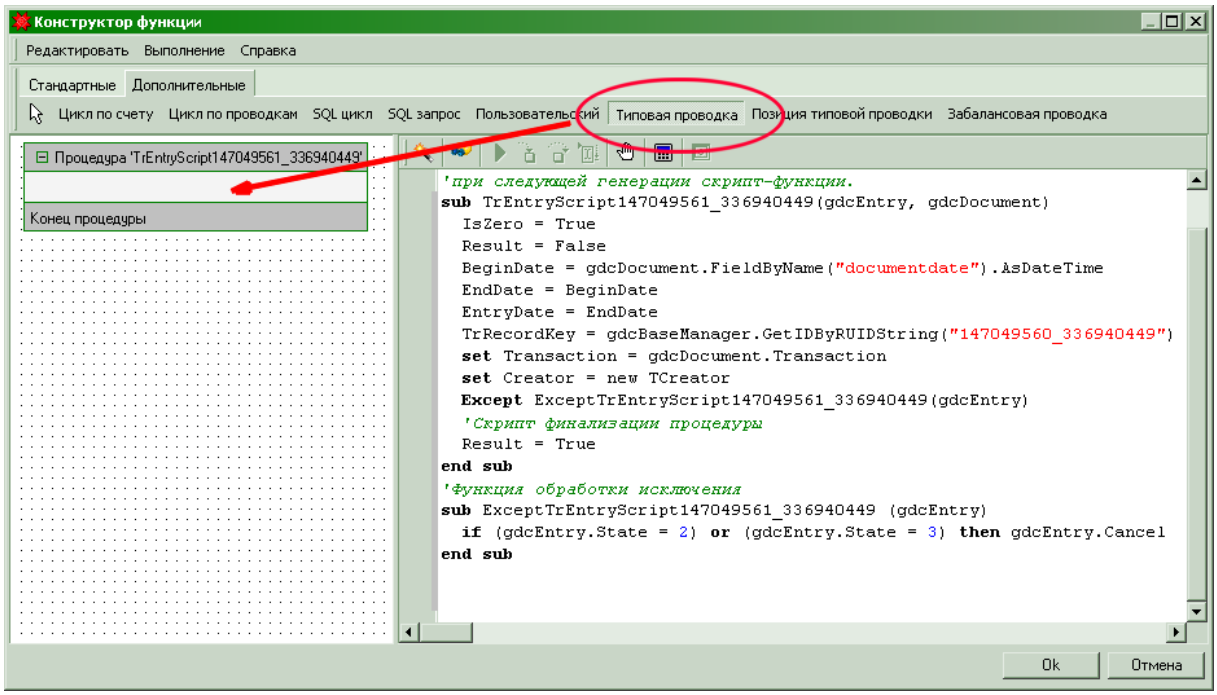


Рис. 81Добавление нового структурного блока.

В нашем примере добавим новый блок «Типовая проводка», который находится на вкладке «Дополнительные». При вставке новго блока на экране возникнет диалоговое окно для настройки параметров этого блока. Вид дилогового окна зависит от типа структурного блока. В случае типовой проводки диалоговое окно имеет три вкладки:

- Общие;
- Кредит;
- Дебет.

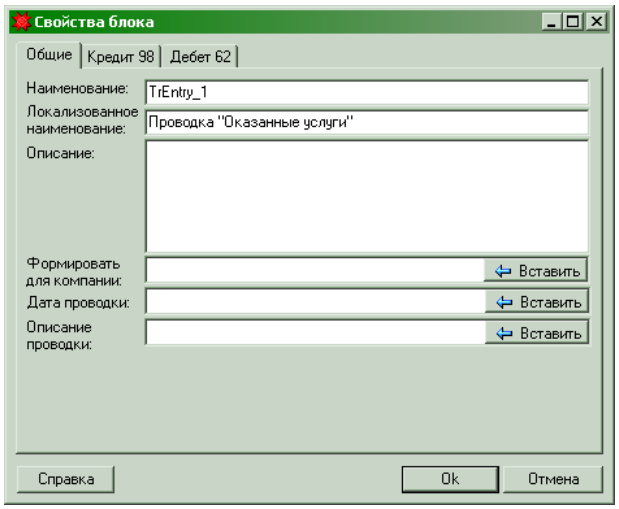


Рис. 82 Вкладка "Общие" диалогового окна типовой проводки.

Как показано на предыдущем рисунке, на вкладке «Общие» указываются такие параметры, как Наименование блока, локализованное наименование, описание и т.п. Заполним поле «Локализованное наименование», как показано на рисунке и перейдем на следующую вкладку.

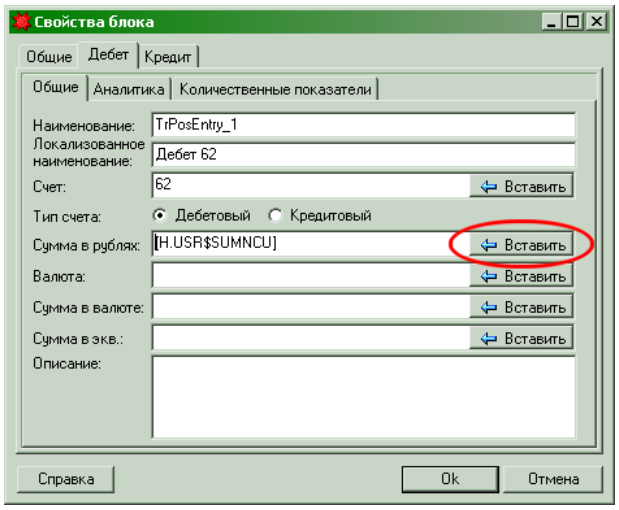


Рис. 83 Вкладка "Дебет" диалогового окна типовой проводки.

На вкладке «Дебет», как это не трудно предположить, происходит настройка дебетовой части проводки. Здесь выбирается счет из плана счетов и указывается формула для расчета суммы. Формулу можно вписать непосредственно в поле ввода, либо воспользоваться редактором выражений. Редактор вызывается с помощью кнопки «Вставить» справа от поля ввода. Подробно, работа с редактором выражений будет описана в последующих главах. Для нашего примера будет достаточно ввести в поле «Сумма в рублях» выражение [H.USR$SUMNCU], как показано на рисунке. Данное выражение означает, что сумма по дебету счета 62 будет взята из поля USR$SUMNCU шапки документа.

Обратите внимание, что на вкладке «Дебет», в свою очередь, присутствуют три вкладки:

- Общие;
- Аналитика;
- Количественные показатели.

Выше мы определили, что вкладка «Общие» предназначена для ввода таких параметров, как номер счета, сумма, сумма в валюте и т.п. Вкладка «Аналитика» предназначена для ввода выражений, на основании которых будут устанавливаться аналитические признаки для проводки. Конкретное наполнение вкладки зависит от настройки аналитического учета по выбранному счету. В нашем примере по 62-му счету ведется учет в разрезе клиентов. Соответственно, на вкладке «Аналитика» будет одно поле:

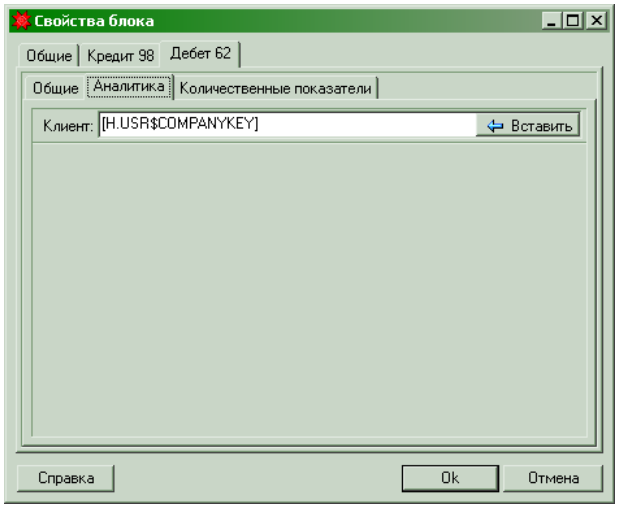


Рис. 84 Формирование аналитических признаков по дебетовому счету.

Заполним поле Клиент выражением [H.USR$COMPANYKEY]. Как вы уже наверное догадались, данное выражение означает, что при формировании проводки поле аналитического признака будет заполнено идентификатором клиента из договора.

Наш пример не подразумевает использование количественных показателей, поэтому мы не будем рассматривать здесь использование одноименной вкладки, а отложим его до следующих глав настоящего руководства.

Аналогично дебету, заполним поля на вкладке «Кредит».

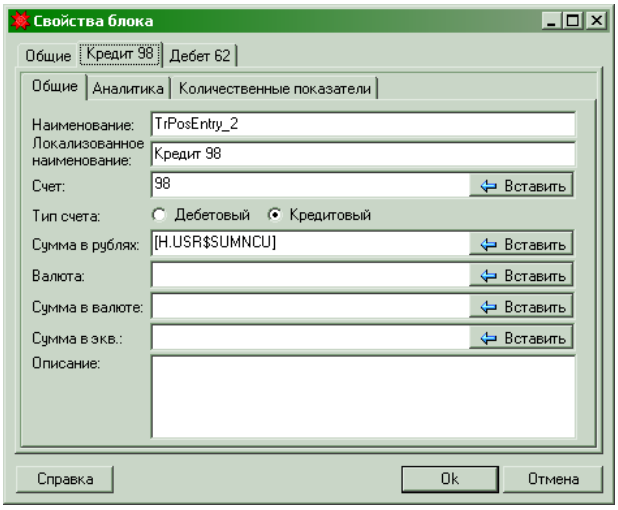


Рис. 85 Ввод кредитовой части типовой проводки.

Как видно на предыдущем рисунке, мы используем 98-й счет. Формула для вычисления суммы проводки совпадает с формулой, которую мы указали в дебетовой части проводки.

Может возникнуть естественный вопрос: какой смысл в том, чтобы указывать идентичную формулу и по дебету и по кредиту проводки? Дело в том, что рассматриваемый нами пример, достаточно прост и в нем мы используем т.н. простую проводку в которой один счет по кредиту корреспондирует с одним счетом по дебету. Гедымин позволяет так же формировать сложные проводки, когда несколько счетов по дебету корреспондируют с одним счетом по кредиту и наоборот. Очевидно, что в этом случае формулы для вычисления сумм будут различаться.

Но, вернемся к составлению типовой проводки. Нам осталось указать как будут заполняться аналитические признаки по 98 счету: Клиент по договору и, собственно, сам Договор.

Заполним поля на вкладке «Аналитика», как показано на рисунке ниже:

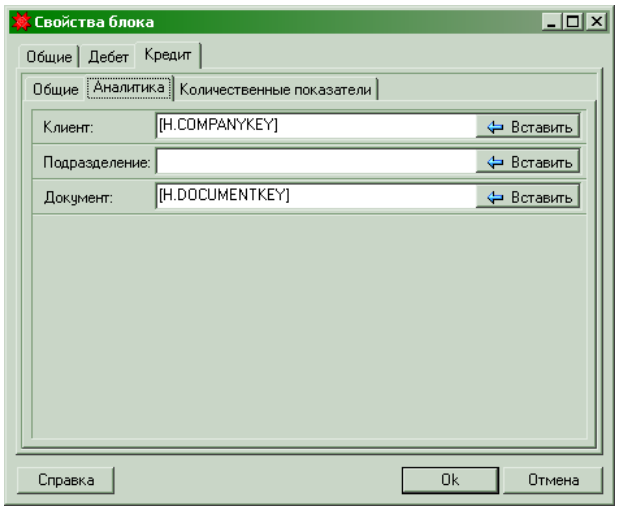


Рис. 86 Формирование аналитических признаков по кредиту.

После заполнения полей с параметрами кредитовой части проводки нажмем «Ок» для завершения работы с блоком. Теперь в конструкторе функции, внутри блока «Процедура» появился добавленный нами блок «Проводка».

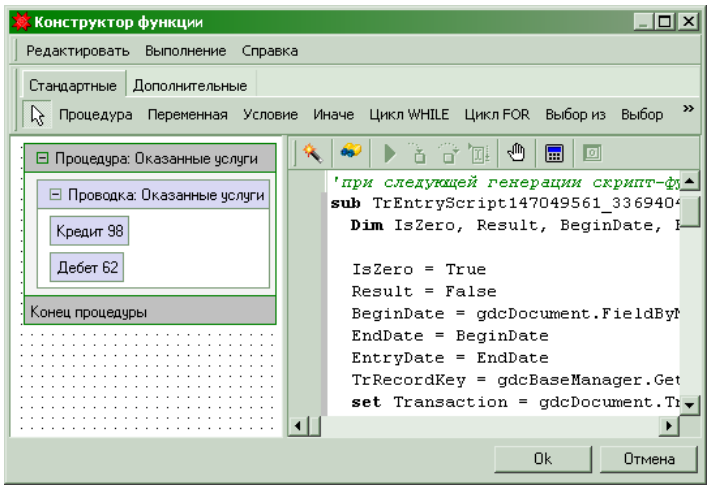


Рис. 87 Конструктор функции.

С помощью мыши мы можем переключаться с одного блока на другой. При этом, в правой части окна будет подсвечиваться код функции, относящийся к выбранному блоку. С помощью контекстно-зависимого меню над выделенным блоком можно выполнить следующие операции:

- Открыть диалоговое окно свойств блока;
- Удалить блок;
- Скопировать блок в буфер обмена;
- Вырезать блок и поместить его в буфер обмена;
- Вставить блок из буфера обмена;

Завершим формирование типовой проводки. Сначала выберем кнопку «Ок» внизу окна конструктора функции, а затем, также по «Ок», закроем диалоговое окно ввода типовой проводки.

Итак, мы создали новую типовую операцию и новую типовую проводку для этой операции. Результат наших действий можно наблюдать в окне «Типовые операции».

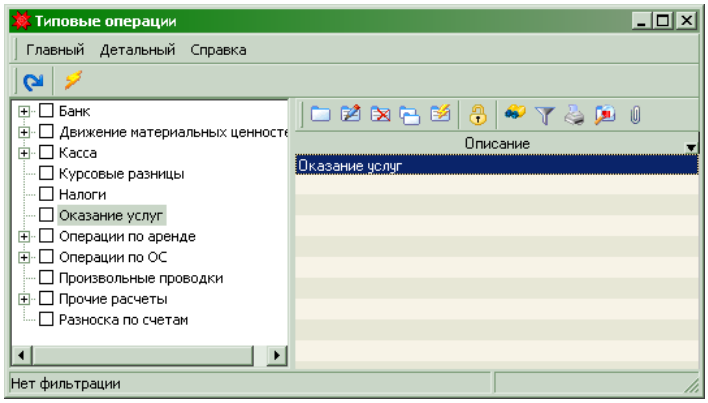


Рис. 88 Окно "Типовые операции"

Дело осталось за малым: указать Гедымину, что при создании нового экземпляра типа «Договор на оказание услуг» следует формировать хозяйственную операцию «Оказание услуг» и заданные для нее проводки. Для связи документа с хозяйственной операцией снова вернемся в раздел «Типовые документы» и откроем на редактирование, ранее созданный нами тип документа «Договор на оказание услуг».

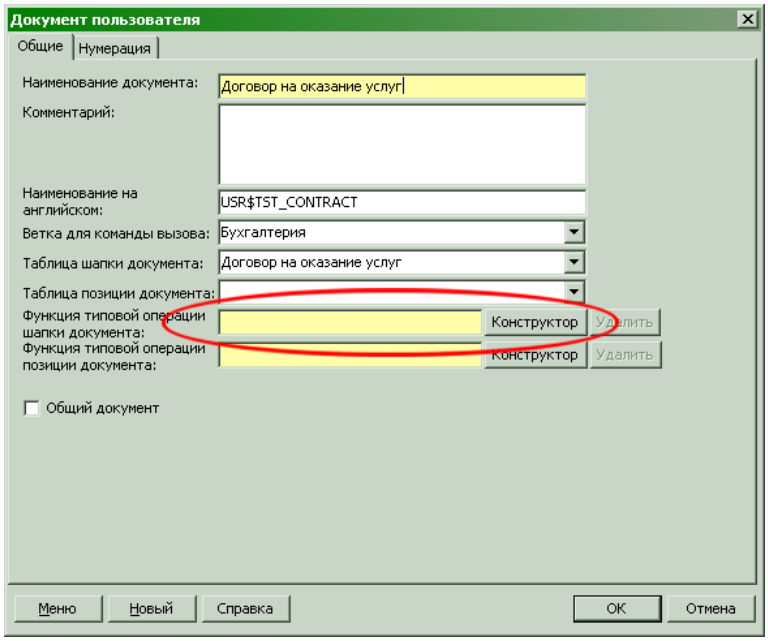


Рис. 89 Редактирование типа документа.

Как видно на представленном выше скриншоте, в диалоговом окне редактирования типа документа так же присутствуют кнопки вызова конструктора функции. И, если в типовой хозяйственной операции, с помощью конструктора можно было задать алгоритм, по которому для данной операции формировались бухгалтерские проводки, то для типа документа, как вы уже наверняка догадались, с помощью конструктора можно задать алгоритм, по которому для документов такого типа будут формироваться хозяйственные операции.

Откроем конструктор для функции формирования типовой операции по шапке документа. На палитре структурных блоков, на вкладке «Дополнительные» найдем блок «Типовая операция и добавим» его.

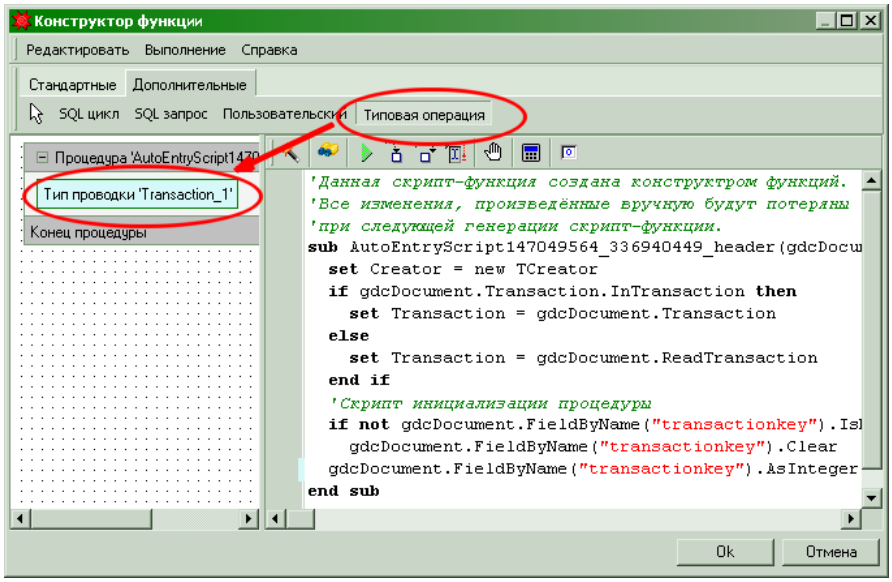


Рис. 90 Конструктор функции

В появившемся на экране окне «Свойства блока» переместим курсор в поле «Типовая операция» и выберем из выпадающего древовидного списка, созданную нами ранее операцию «Оказание услуг».

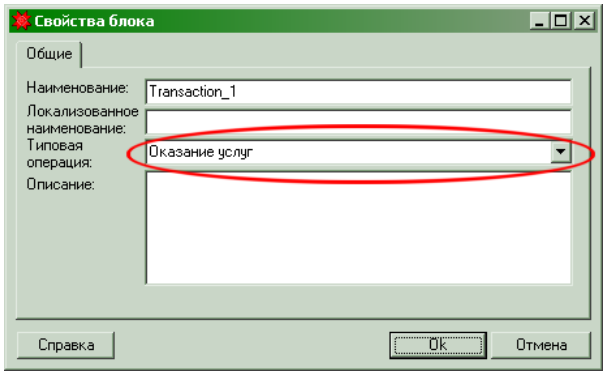


Рис. 91 Блок "Типовая операция"

Трижды воспользуемся кнопкой «Ок» для того, чтобы закрыть окно «Свойства блока», «Конструктор функции» и «Документ пользователя». На этом создание нового типа документа и его настройку можно считать оконченным.

Работа с созданным типом документа

Проверим, как работает созданный нами документ на небольшом примере. Откроем форму просмотра списка наших документов: «Исследователь»—«Бухгалтерия»—«Договор на оказание услуг». Введем пять тестовых записей, как показано на следующем рисунке.

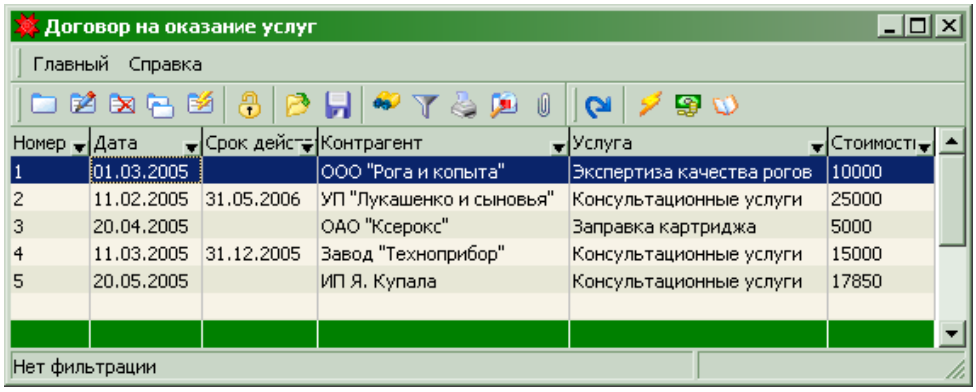


Рис. 92 Набор тестовых данных.

При вводе данных в диалоговом окне договора на указание услуг вы навряд ли отыщете в списке таких контрагентов или услуги, как показано на рисунке. Вам придется добавить новые записи в соответствующие справочники. Напомним, что создать новую запись (новую компанию, новую услугу) вы можете находясь непосредственно в поле ввода с выпадающим списком. Для этого достаточно нажать F2 и заполнить необходимые поля в появившемся диалоговом окне.

После того, как тестовые данные введены, можно просмотреть список сформированных бухгалтерских проводок. Откроем журнал хозяйственных операций: «Исследователь»—«Бухгалтерия»—«Журнал хозяйственных операций».

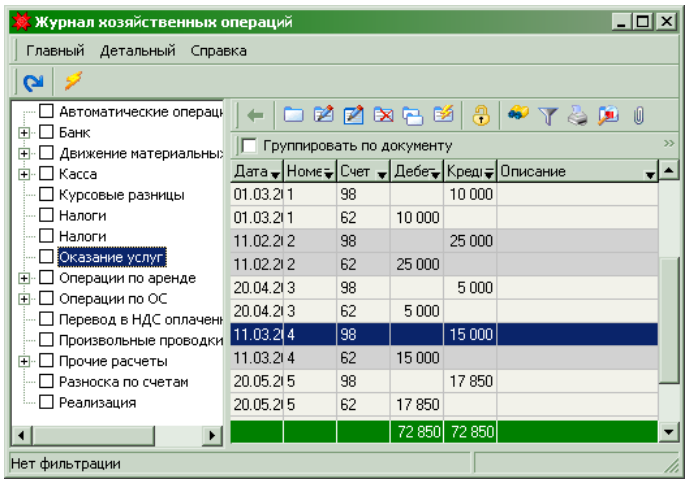


Рис. 93 Журнал хозяйственных операций.

В левой части окна располождагается древовидный список типовых хозяйственных операций, а в правой — список проводок для выбранного типа хозяйственных операций.

В нашем примере мы использовали операцию «Оказание услуг». Отыщем ее в списке и установим курсор. Справа откроется список, содержащих пять проводок Дт 62—Кт 98. Каждая проводка имеет дату и сумму документа, для которого она была сформирована.

Окна журнала хозяйственных операций и просмотровая форма документа связаны между собой. Установив курсор на документ, можно перейти в журнал хозяйственных операций на проводки, сформированные по этому документу. И наоборот, находясь в журнале хозяйственных операций на конкретной проводке, можно перейти на документ.

Алгоритм действий при создании типа документа

1. Перейти в раздел «Типовые документы»;
2. Открыть диалоговое окно создания нового типа документа;
3. Заполнить поля
    a. Наименование
    b. Наименование на английском языке;
4. Открыть диалоговое окно создания таблицы шапки документа;
5. Заполнить поля;
6. Открыть диалоговое окно создания нового поля;
7. Открыть диалоговое окно создания нового типа данных (домена);
8. Заполнить поля;
9. Сохранить созданный тип данных (домен);
10. Сохранить созданное поле таблицы;
11. Повторить шаги 5-9 для всех полей шапки документа;
12. Сохранить созданную таблицу;
13. Сохранить созданный тип документа;
14. Перейти в раздел «Типовые операции»;
15. Создать новую типовую операцию;
16. Создать новую типовую проводку;
17. Вернуться в раздел «Типовые документы»;
18. Открыть тип документа на редактирование;

19. Используя конструктор функции связать тип документа с типовой хозяйственной операцией;
20. Сохранить тип документа;

Встроенные документы

Перенос документов между базами

Нумерация

Бизнес-объект Документ имеет функцию автоматической генерации порядкового номера. Номер генерируется при вставке записи на основании заданной маски и инкремента.

Нумерация ведется отдельно для каждого типа документа и для каждой из рабочих организаций. Настройка маски осуществляется в окне редактирования типового документа на вкладке Нумерация. Окно редактирования типового документа вызывается из окна со списком типовых документов, которое открывается из раздела Сервис Исследователя системы.

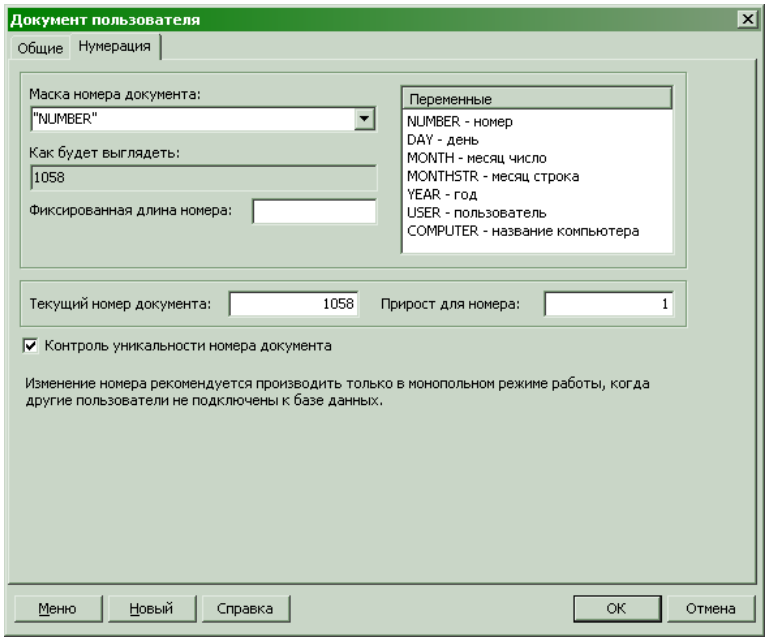


Рис. 94 Настройка нумерации документа.

Совет

Из диалогового окна редактирования документа можно всенгда открыть окно для его типа. Для этого необходимо выбрать кнопку «Меню», которая обычно располагается в нижней части окна, и вызвать команду «Тип документа».

При задании маски можно использовать переменные. Список переменных расположен справа от поля ввода маски. Для того, чтобы добавить переменную достаточно дважды щелкнуть по ее наименованию.

Доступны следующие переменные:

- NUMBER — порядковый номер документа;
- DAY — день из текущей даты;
- MONTH — номер месяца из текущей даты;
- MONTHSTR — наименование месяца из текущей даты;
- YEAR — год из текущей даты (четыре цифры);
- USER — наименование учетной записи пользователя;
- COMPUTER — наименование компьютера, на котором добавляется документ.

Ниже приводятся примеры задания маски номера:

Маска	Получающийся номер
“NUMBER”	1956
“NUMBER”/”YEAR””MONTH””DAY”	1956/20050212
“NUMBER”-“USER”	1956-Janak

Кроме маски номера, в диалоговом окне можно задать следующие параметры:

- Фиксированная длина номера. Если строка номера документа начинается с цифры и ее длина меньше указанной фиксированной длины номера, то строка номера будет дополнена сначала нулями до указанной фиксированной длины. Например, если указана фиксированная длина 8 символов и маска номера «NUMBER», то будут получаться следующие значения: 00002048, 00002049 и т.д.
- Текущий номер документа. Определяе текущий номер. При генерации номера, если в маске используется переменная «NUMBER», вместо нее будет подставлено значение «Текущий номер документа» + «Прирост для номера».
- Прирост для номера. Определяет на сколько будет увеличиваться номер документа. По умолчанию установлено значение 1.
- Контроль уникальности номера. Если флаг установлен, то перед сохранением документа будет осуществляться проверка: нет ли в базе данных для текущей рабочей организации документа с таким же номером.

Как работает код присвоения номера

Сброс нумерации

Присвоение номера из кода программы

«Пропадающие» номера

При одновременной работе нескольких пользователей с документами одно типа (например, счетами-фактурами) возможна ситуация, когда нумерация документов будет не последовательной. Например, при вводе 10-ти документов, они могут получить следующие номера: 1, 2, 3, 6, 7, 9, 14, 15, 16, 17. Происходит это в том случае, если один или несколько пользователей сначала создадут документ, откроют диалоговое окно для заполнения его полей, а затем откажутся от сохранения документа в базе данных — выйдут из окна по кнопке Отмена, в то время как другие пользователи будут продолжать вводить документы.

Метаданные

Таблица GD_LASTNUMBER.

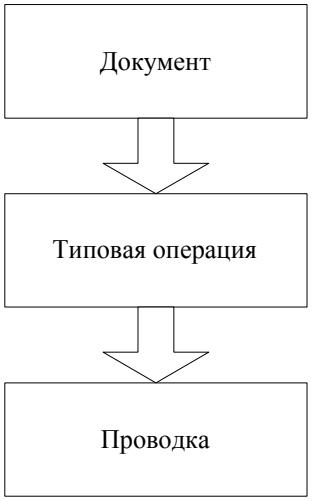
PK	FK	Поле Домен Тип данных	NN	Описание
1	F	DOCUMENTTYPEKEY DINTKEY INTEGER	☒	Ссылка на тип документа.
2	F	OURCOMPANYKEY DINTKEY INTEGER	☒	Ссылка на рабочую организацию.
		LASTNUMBER DINTEGER		Последний номер.

		INTEGER		
		ADDNUMBER DINTEGER INTEGER		Прирост номера.
		MASK DTEXT80 VARCHAR(80)		Маска для генерации строки номера.
		DISABLED DBOOLEAN SMALLINT		
		FIXLENGTH DFIXLENGTH INTEGER		Фиксированная длина строки номера.

Организация и настройка бухгалтерского учета на Гедымине

Общие положения

Принцип построения



Бухгалтерский учет в Гедымине строится от документа, любая хозяйственная операция может быть проведена только при наличии документа.

Проводка по документу проводится автоматически при сохранении документа или позиции документа, если в нем задана типовая операция.

Процесс настройки бухгалтерского учета соответственно сводится к следующим операциям:

1. Настройка плана счетов, видов аналитики и количественных показателей – раздел «План счетов»
2. Создать дерево типовых операций – раздел «Типовые хозяйственные операции»
3. По каждой операции определить какие проводки и по каким документам будут формироваться.
4. Для каждого типа документа определить какие операции при каких условиях будут формироваться – раздел «Типовые документы», редактирование типового документа.
5. Настройка автоматических операций
6. Настройка журналов-ордеров, карты счета, оборотной ведомости, главной книги
7. Настройка бухгалтерских отчетов.

План счетов, аналитика, количественные показатели

Настройка бухгалтерского учета всегда начинается с определения необходимых регистров учета, в нашем случае это определение необходимых счетов, субсчетов (n-го порядка), определение необходимых уровней аналитики и количественных показателей.

Все эти операции производятся в разделе «План счетов», где происходит добавление счетов, видов аналитик и определение необходимых количественных показателей.

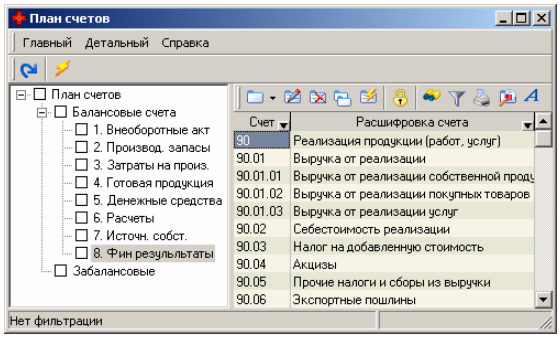


Рис.1 План счетов

Система поддерживает возможность ведения нескольких планов счетов для одной организации, один из них активный (базовый), все остальные вспомогательные. Реальным применением нескольких планов счетов может стать параллельное ведение учета для национального законодательства и европейского, ведение налогового учета и т.д. Для этого заводится второй план счетов и в разрезе данного плана определяются дополнительные проводки по тем же операциям, что и основные, соответственно параллельно будут формироваться проводки в двух системах, затем останется настроить отчеты, уже используя счета из дополнительного плана счетов.

План счетов делится на разделы, количество разделов и подразделов не ограничено, не ограничена и их вложенность, далее в разделы входят счета и счетов могут быть субсчета (у субсчета может быть субсчет второго уровня и т.д., количество вложенных субсчетов также не ограничено). При добавлении новой записи необходимо выбрать, что будет добавляться План счетов, Раздел, Счет или Субсчет. В зависимости от выбора на экран будет выведено соответствующее окно редактирования:

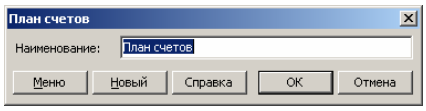


Рис.1а Диалог «План счетов»

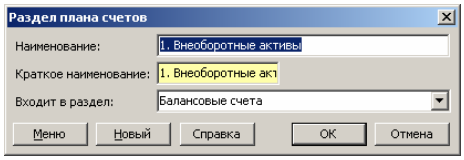


Рис. 1б Диалог «Раздел плана счетов»

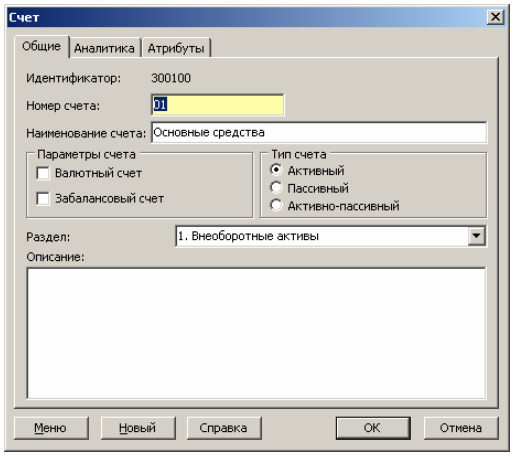


Рис. 1в Диалог «Счет»

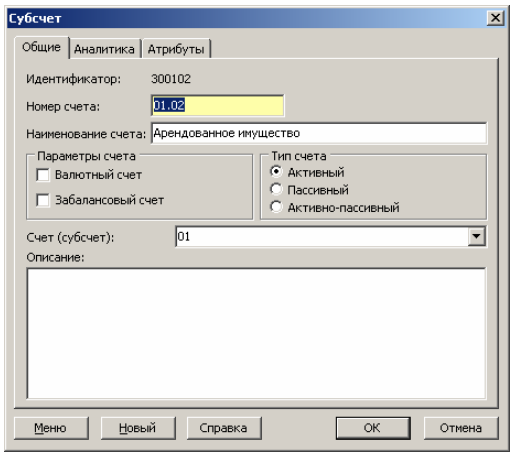


Рис. 1г Диалог «Субсчет»

Кроме синтетического учета большой интерес представляет аналитический учет. Система поддерживает достаточно большое (но ограниченное) количество видов аналитики, ограничение связано с тем, что каждый вид аналитики – это поле в таблице проводок (ориентировочное количество 128 независимых видов аналитик). Добавление нового вида аналитики, представляет собой добавление поля в таблицу с проводками и в таблицу с планом счетов. В таблицу с проводками поля добавляется того типа, какой необходим для учета (ссылка на справочник, ссылка на документ, строка, число, логическое, дата и т.д.), а в справочник плана счетов добавляется поле с тем же наименованием, но логического типа (dboolean).

Для автоматического добавления полей в две таблице в разделе план счетов есть кнопка добавить аналитику. По нажатию данного ней поля создаются сразу в двух таблицах. Соответственно каждый вид аналитики ведется параллельно со всеми остальными, что позволяет формировать достаточно гибкие отчеты. Отдельный вид аналитики может быть ссылкой на справочник древовидной структуры, что позволяет вложенные аналитики организовывать через один вид аналитики – ссылка на древовидный справочник.

Для каждого счета (субсчета) в диалоге редактирования, на закладке «Аналитика», отмечаются те виды аналитик, по которым будет вестись учет. Эта информация затем используется при настройке типовых проводок и при их ручном вводе проводок.

Кроме суммового учета, система предполагает возможность ведения количественного учета, в разрезе неограниченного числа количественных показателей. В качестве справочника количественных показателей используется справочник единиц измерения, по каждому счету можно указать в разрезе каких единиц измерения идет учет. В отличии от складского учета, здесь не рекомендуется дробить учет на такие единицы измерения как шт, кг, тонны и т.д. имеет смысл вводить укрупненную единицу измерения, скажем, количество, человеко-день, человеко-час и т.д. Использование количественного учета рекомендуется в тех случаях когда необходимо получить, какую-то обобщенную суммарно-количественную информацию по движению в разрезе счетов и объектов аналитик по этим счетам. Примером может служить расчет фактической себестоимости готовой продукции, для производства или же необходимость получения суммарных временных затрат человеко-дни и т.д. Все стандартные отчеты позволяют получить информацию по количественному движению, а также количественные показатели могут быть задействованы при формировании автоматических операций.

Типовые операции

Весь учет описывается деревом типовых операций, каждая из которых может содержать неограниченное количество типовых проводок, каждая из которых предназначена для конкретного документа (несколько разнотипных проводок может формироваться по одной операции одному документу), каждая типовая проводка привязана к конкретному плану счетов. Список типовых операций вызывается «Исследователь/Бухгалтерия/Типовые операции»

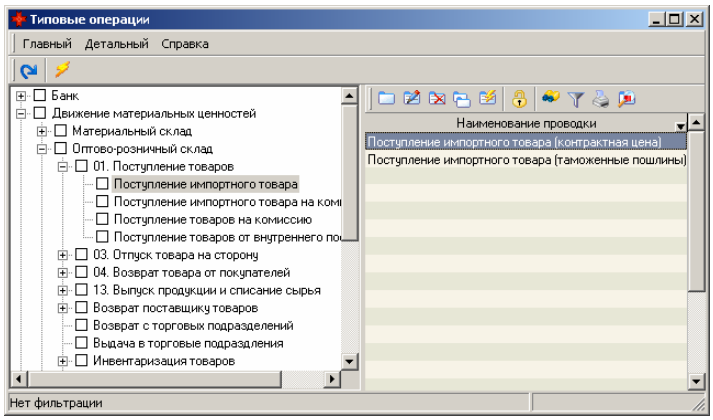


Рис. 2 «Дерево типовых операций»

Правилом хорошего тона является – для документов с позициями формировать проводки по каждой позиции, в этом случае система сама при каждом изменении значений в позиции или шапки документа, будет перестраивать проводки. Если проводки формируются по шапке документа, но ее сумма или другие параметры зависят от значений в позиции, то настройщику надо самому побеспокоиться о формировании проводок при изменении позиции документа.

Рассмотрим дальше, как задается типовая проводка. Задание типовой проводки состоит в написании функции, которая должна добавлять проводку в журнал хозяйственных операций. Данная функция вызывается системой в момент сохранения документа (позиции документа), если по нему указана типовая операция. Функция имеет два входящих параметра – объект проводки и объект документ, по которому формируется проводка. Для автоматизации процесса написания такой функции служит «Конструктор функции».

«Конструктор функции»

Конструктор содержит блоки, отвечающие за различные конструкции программирования из которых мы и создаем свою функцию. Все блоки размещены на 2-х закладках: 1 – это постоянные и 2 – переменные. На 1-й закладке находятся блоки, которые присутствуют в конструкторе всегда, для какой бы функции он не вызывался, (конструктор используется при формировании типовой проводки, автоматической операции и формирование отчетов бухгалтерии), на 2-й закладке располагаются блоки, которые зависят от того, для создания какой функции вызывается конструктор. Каждый блок конструктора настраивается через свое окно «Свойство блока».

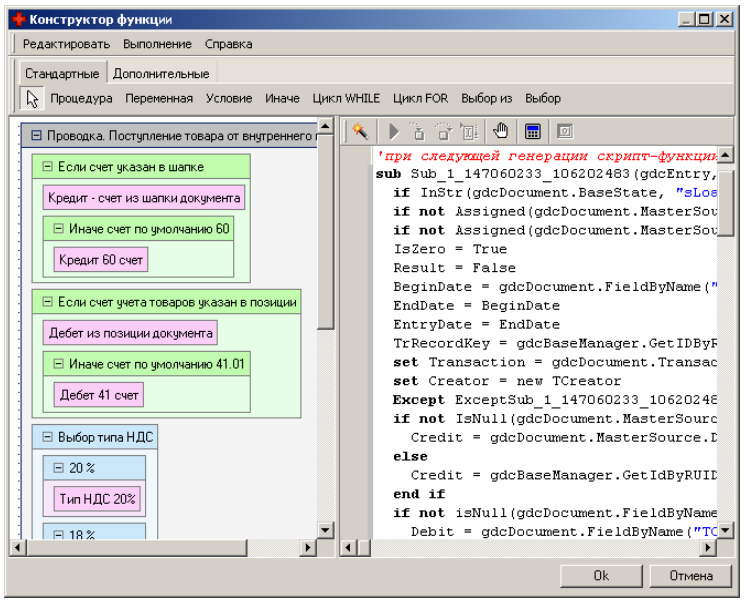


Рис 3. Конструктор функций.

Блоки конструктора функций:

1. Процедура. Конструкция sub …end sub. Function … end function

Данный блок предназначен для формирования процедуры или функции. В конструкторе может быть одна основная и сколько угодно вспомогательных процедур. Все остальные блоки вставляются внутрь блока процедуры.

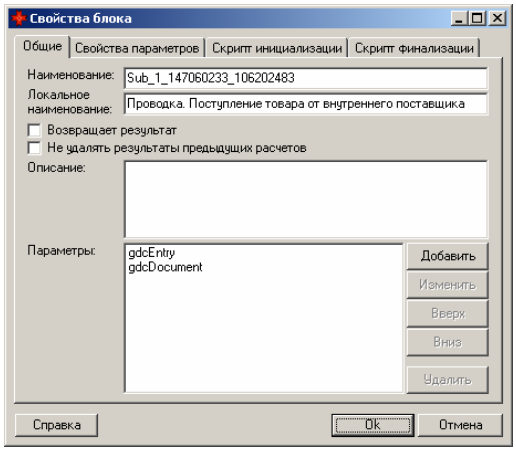


Рис. 4 Окно «Свойства блока». Процедура.

Настройка блока:

Наименование	Описание
Основные	
Наименование	Наименование процедуры (функции). Присваивается автоматически, можно изменить, можно оставить. Наименование может содержать только латинские буквы и цифры. Начинаться может только с буквы.
Локальное наименование	Наименование процедуры (функции)
Возвращает результат	Будет ли возвращать результат наша процедура или нет.
Описание	Описание блока
Параметры	Список параметров процедуры. Для добавления параметра используется окно «Свойство параметра», в котором мы указываем наименование параметра и способ передачи (ByVal, ByRef) ByVal – процедура создает копию переданного параметра, что сохраняет оригинальное значение этого параметра. ByRef – в процедуру передается адрес переменной, что позволяет изменять значение данной переменной в теле процедуры.
Свойства параметров. На этой закладке для каждого указанного параметра, указывается его тип. Система поддерживает следующие типы	
Число целое	
Число дробное	
Дата	
Дата и время	
Время	
Строка	
Логический	True, False
Следующие типы используются в тех случаях, когда необходимо обеспечить ввод данных параметров конечным пользователем, чаще всего они задаются для основной процедуры.	

Ссылка на элемент	Будет передан значение поля в какой-то таблице. В процедуру будет передан массив из одного элемента. Для данного типа необходимо указать на какую таблицу идет ссылка, какое поля в этой таблице использовать для отображения информации и значение, какого поля будет передано в процедуру.
Ссылка на множество	Будет передан набор значений. В процедуру передается массив, с размерностью = количеству выбранных записей. Настраивается аналогично предыдущему.
Не запрашивается	В окне для ввода параметров, данный параметр будет отсутствовать.
Скрипт инициализации. На этой закладке, можно написать код программы, который необходимо выполнить в самом начале процедуры. В основном используется в главной процедуре, которая обычно имеет, какой-то код по умолчанию и там его можно изменить	
Скрипт финализации. На этой закладке, можно написать код программы, который необходимо выполнить в самом конце процедуры. В основном используется в главной процедуре, которая обычно имеет, какой-то код по умолчанию и там его можно изменить	

2. Переменная. Конструкция variable = value, set variable = object.

Блок предназначен для задания переменной. Для переменных ведется список, затем для их использования достаточно выбрать нужную переменную из списка.

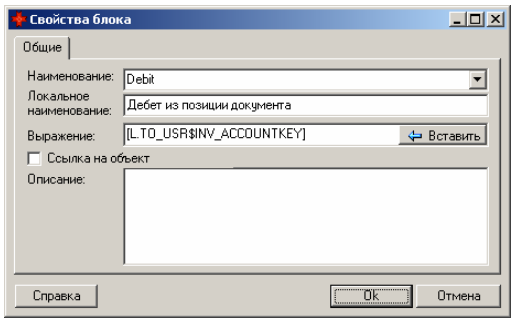


Рис. 5 Свойства блока. Переменная.

Настройка блока:

Наименование	Описание
Наименование	Наименование переменной (может быть выбрано из списка) в этом случае блок изменяет значение ранее созданной переменной или же указать новое имя, в этом случае будет создана новая переменная, которой будет присвоено значение. Наименование согласно правил составления наименований переменных, процедур и функций.
Локализованное наименование	Понятное для других наименование переменой (может совпадать с наименованием).
Выражение	Выражение, которое будет присвоено данной переменой. Для формирования переменной можно воспользоваться отдельным окном « Редактор выражений», которое вызывается при нажатии на кнопку вставить. Описание данного окна будет приведено ниже.
Ссылка на объект	Данная переменная является ссылкой на объект.
Описание	Описание использования данной переменной

Окно «Редактор выражений».

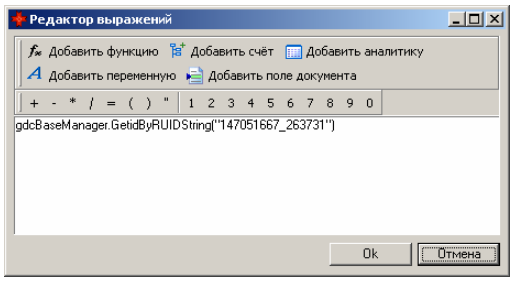


Рис. 6 Редактор выражений.

Данное окно служит для упрощения написания различного рода формул и выражений. Оно содержит несколько кнопок предназначенных для выбора различного рода конструкций. Часть кнопок этого окна постоянные – «Добавить функцию», «Добавить счет», «Добавить аналитику», «Добавить переменную», часть зависит от того для формирования, какой функции был задействован конструктор, для формирования проводок это кнопка «Добавить поле документа». Рассмотрим подробно каждую кнопку:

-«Добавить функцию». По данной кнопке на экране появится окно «Доступные функции», в котором выводится весь список функций, которые можно вставить в выражение.

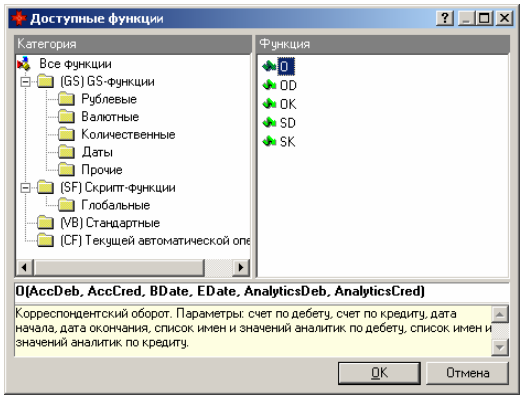


Рис. 7 Доступные функции.

Все функции разделены на четыре больших раздела:

GS-функции– это функции, которые описаны, в самой платформе и предназначены для бухгалтерских расчетов и вспомогательные функции, они делятся на рублевые, валютные, количественные, даты и прочее. Рублевые – функции для расчета оборотов, сальдо по счетам в национальной валюте. Валютные – функции для расчета оборотов, сальдо по счетам в указанной валюте. Количественные – функции для расчета оборотов, сальдо по счетам в разрезе количественного показателя. Даты – функции для работы с датами. Прочие – вспомогательные функции, используемые чаще всего самим конструктором.

Наименование функции	Входящие параметры	Описание
Рублевые		
O	Идентификатор счета по дебету, Идентификатор счета по кредиту, Дата начала, Дата окончания, Список аналитик по дебету, Список аналитик по кредиту	Функция расчет суммы оборота с дебета одного счета в кредит другого счета за указанный период, в разрезе необходимых аналитик. Список аналитик передается в следующем виде: Наименование поле = Значение; и т.д. Если список пустой, расчет идет без учета аналитики.
OD, OK	Идентификатор счета, Дата начала, Дата окончания, Список аналитик	Функция возвращает дебетовый (кредитовый) оборот по указанному

		счету, за указанный период, по выбранной аналитике
SD, SK	Идентификатор счета, Дата окончания, Список аналитик	Функция возвращает дебетовое (кредитовое) сальдо по указанному счету, на указанную дату (дата включается в расчет), по выбранной аналитике.
Валютные		
V_O	Идентификатор счета по дебету, Идентификатор счета по кредиту, Дата начала, Дата окончания, Список аналитик по дебету, Список аналитик по кредиту, Идентификатор валюты	Функция расчет суммы оборота с дебета одного счета в кредит другого счета за указанный период, в разрезе необходимой аналитик, в указанной валюте.
V_OD, V_OK	Идентификатор счета, Дата начала, Дата окончания, Список аналитик, Идентификатор валюты	Функция возвращает дебетовый (кредитовый) оборот по указанному счета, за указанный период, по выбранной аналитике, в указанной валюте
V_SD, V_SK	Идентификатор счета, Дата окончания, Список аналитик, Идентификатор валюты	Функция возвращает дебетовое (кредитовое) сальдо по указанному счету, на указанную дату (дата включается в расчет), по выбранной аналитике, в указанной валюте.
Количественные		
K_O	Идентификатор количественного показателя (единицы измерения) Идентификатор счета по дебету, Идентификатор счета по кредиту, Дата начала, Дата окончания, Список аналитик по дебету, Список аналитик по кредиту	Функция расчет суммы оборота с дебета одного счета в кредит другого счета за указанный период, в разрезе необходимой аналитик, в количественном выражении.
K_OD, K_OK	Идентификатор количественного показателя (единицы измерения) Идентификатор счета, Дата начала, Дата окончания, Список аналитик	Функция возвращает дебетовый (кредитовый) оборот по указанному счета, за указанный период, по выбранной аналитике в количественном выражении
K_SD, K_SK	Идентификатор количественного показателя (единицы измерения) Идентификатор счета, Дата окончания, Список аналитик	Функция возвращает дебетовое (кредитовое) сальдо по указанному счету, на указанную дату (дата включается в

		расчет), по выбранной аналитике, в количественном выражении.
Даты		
IncMonth	Дата, число месяцев	Изменяет переданную дату на указанное количество месяцев.
KM	Дата	Возвращает дату конца месяца, соответствующую переданной дате
NG	Дата	Возвращает дату начала года, соответствующую переданной дате
NK	Дата	Возвращает дату начала квартала, соответствующую переданной дате
NM	Дата	Возвращает дату начала месяца, соответствующую переданной дате
NP	-	Возвращает дату начала периода
OP	-	Возвращает дату окончания периода
Прочие		
CreditValue	Сумма	Если сумма < 0, то будет возвращено абсолютное значение суммы, если нет то 0
DebitValue	Сумма	Если сумма > 0, то будет возвращена сама сумма, если нет то 0
GetAnalytics	Analytics - строка, содержащая имена и значения аналитик; Names - список имен требуемых аналитик, разделенных точкой с запятой.	Возвращает строку требуемых аналитик
GetAnaliticStr	SQL-запроса	Формирует строку аналитики из имен и значений полей, переданного SQL-запроса
GetAnaliticValue	Analytics - строка содержащая список аналитик и их значений, разделенных точкой с запятой; AnalyticName - имя аналитика, значение которой нужно получить	Возвращает значение аналитики
GetFullRubSumStr	Сумма	Возвращает сумму прописью с рублями и

		копейками
GetRubSumStr	Сумма	Возвращает сумму прописью без копеек
GetSumCurr	Сумма	Возвращает сумму прописью, в указанной валюте
GetSumStr	Число, количество знаков после запятой	Возвращает число прописью
GetValue	Неиспользуется	

- (SF) Скрипт функции. Это функции, которые определены в той настройке, которую Вы используете.

- (VB) Стандартные. Это функции доступные в VB script.

- (CF) Текущей автоматической операции. Это функции, которые добавлены в конструкторе, в текущей типовой операции.

При выборе нужной функции, далее на экране появится окно для заполнения параметров данной функции, после чего она будет вставлена в текст выражения.

Кнопка «Добавить счет». По данной кнопке в выражение будет вставлен РУИД счета или же функция, которая получает идентификатор счета по его РУИД. При выборе счета, если необходимо получить его идентификатор необходимо установить признак Получить ИД.

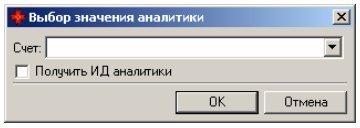


Рис. 8 Добавить счет.

Кнопка «Добавить аналитику». Данная кнопка предоставляет возможность работы как с фиксированной аналитикой (аналитика, которая определена для ведения бухгалтерского учета), так и работу с произвольной таблицей. При выборе «Аналитика», на экран выводится окно, в котором выводится список видов аналитики, для каждой из которых можно указать значение 3-мя способами: а) выбрать поле документа из появившегося списка, б) установить конкретное выражение через «Редактор выражений», в) установить конкретное значение аналитики. Результатом будет выбранное значение (выражение) или же если установлен флаг «С именами аналитики», то будет формироваться строка типа «имя поля аналитики =» + Значение аналитики.

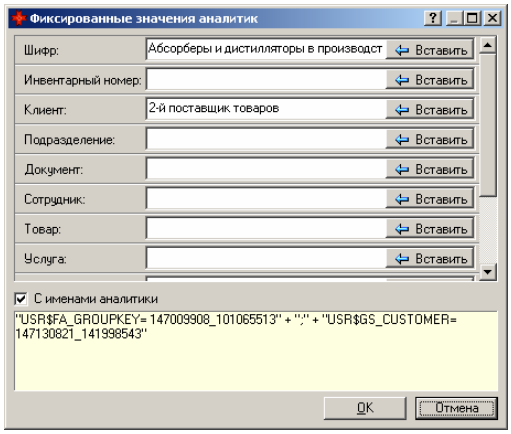


Рис.9 Аналитика.

При выборе прочие справочники, в появившемся окне выбирается необходимый справочник и из него выбирается интересующее нас значение, при этом получается или РУИД этой записи или функция которая вернет ИД этой записи по ее РУИД.

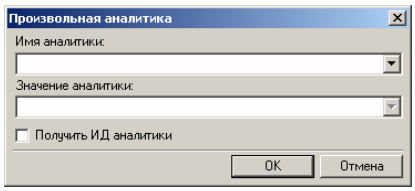
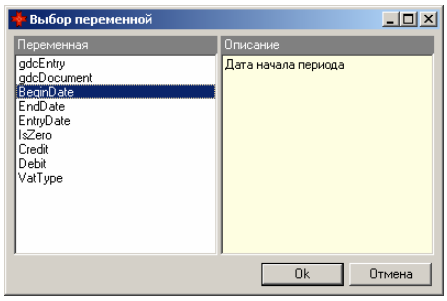


Рис.10. Прочие справочники.

Кнопка «Добавить переменную». По данной кнопке в выражение можно вставить одну из ранее объявленных переменных.



При формировании типовой проводки в редакторе выражений будет доступна кнопка «Добавить поле документа». По этой кнопке мы можем выбрать значение нужного нам поля из документа, по которому формируется проводка. Все поля разделены на 4 группы – Ссылки и ключи, Числовые, Даты, прочее. Если проводка формируется по позиции документа, то будут доступны поля и шапки, и позиции.

На этом описание «Редактора выражений» закончено. Заметим, что выражение это одна строка, поэтому если в редакторе Вы осуществляете переход на следующую строку, не забывайте ставить знак «_».

Продолжим описание блоков конструктора.

3. Условие. Конструкция if then end if.

Данный блок предназначен для формирования условий. В окне свойства блока, через редактор выражений указывается условие, при выполнении которого будет выполняться то, что находится внутри блока.

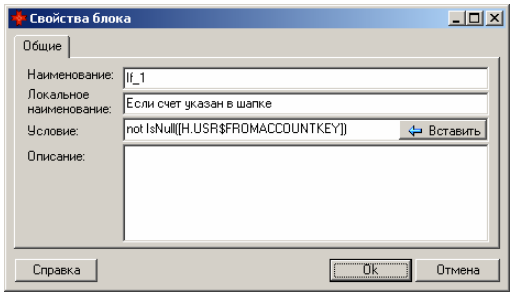


Рис. 12 Свойство блока. Условие.

4. Иначе. Конструкция else.

Обязательно находится внутри блока условие. Внутри данного блока располагаются инструкции, которые будут исполняться при нарушении условия указанного в блоке условие.

5. Цикл WHILE. Конструкция do while loop.

Данный блок предназначен для формирования циклов с произвольным условием. В окне свойства блока, через редактор выражений указывается условие, пока выполняется которое, будут выполняться блоки находящиеся внутри данного блока.

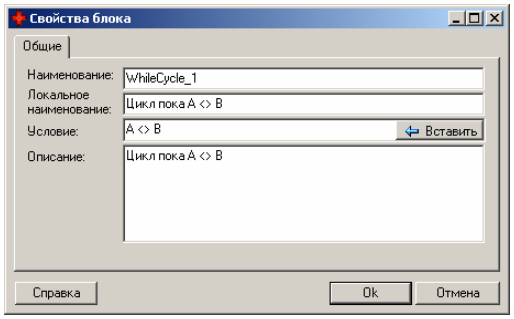


Рис. 13 Свойство блока Цикл WHILE

6. Цикл FOR. Конструкция for to step next.

Данный блок предназначен для организации инкрементного цикла. В окне свойства блока, указывается от какого значения до какого, будет выполняться цикл с каким шагом. Наименование блока в этом случае будет наименованием переменной счетчика.

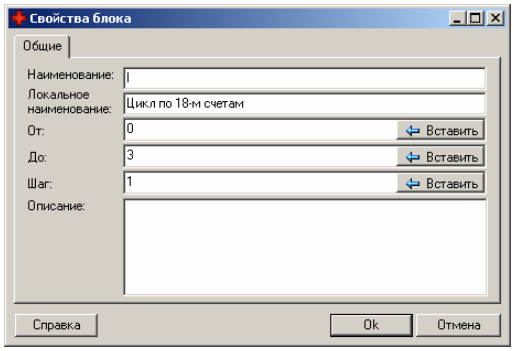


Рис. 14 Свойство блока Цикл FOR

7. Выбор из. Конструкция select case end select

Данный блок предназначен для организации выбора из множества значений. В свойствах блока указывается выражение, по различным значениям которого будет выполняться свой блок.

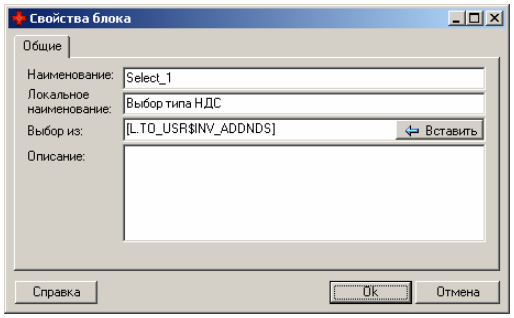


Рис. 15 Свойство блока. Выбор из.

8. Выбор. Конструкция case. Используется внутри блока «Выбор из».

Данный блок предназначен для размещения внутри него блоков, которые будут выполняться при конкретном значении выражения из блока «Выбор из».

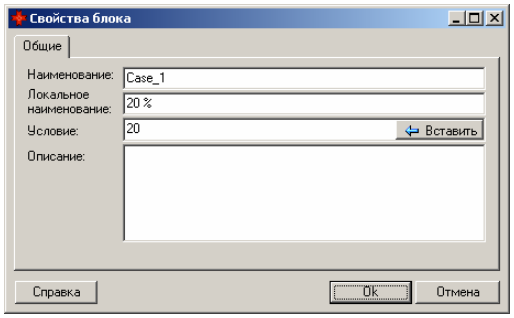


Рис. 16 Свойство блока. Выбор.

9. Цикл по счету. Конструкция организует цикл по SQL запросу, построенному по проводкам, которые ограничены набором бухгалтерских счетов и при необходимости набором аналитик, и периодом действия. Запрос выбирает значения счетов и аналитик из проводок, полученных с данными ограничениями, эти значения доступны через соответствующие переменные, имена которых сформированы, как наименование SQL цикла + наименование поля.

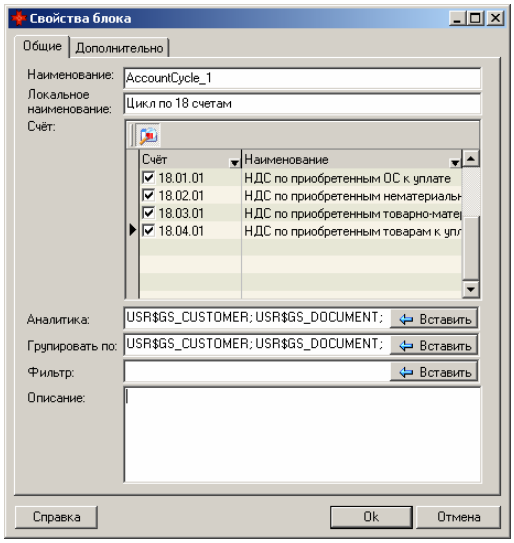
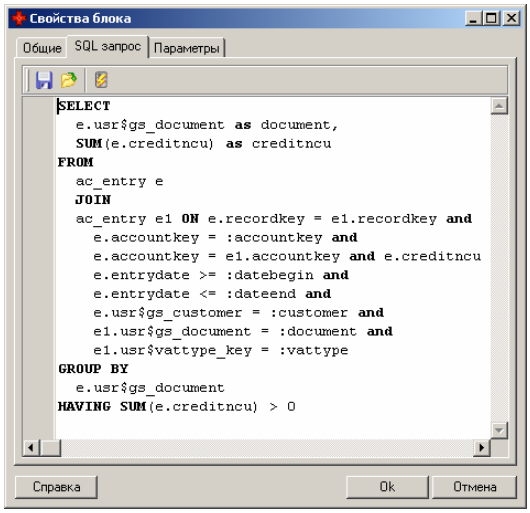


Рис. 17 Свойство блока. Цикл по счету.

В окне свойства блока, отмечаются счета, по которым будет организован цикл, аналитики, значения которых будут доступны внутри цикла, указывается в разрезе каких аналитик необходимо группировать данные, при необходимости на конкретные виды аналитик могут быть установлены ограничения, как конкретные значения или выражение, составленное через «Редактор выражений». На закладке «Дополнительно» указывается значение (выражение) даты начала периода, даты окончания периода. Здесь же можно скорректировать сам запрос – добавить поля в SELECT часть, таблицы – во FROM часть, ограничение – в WHERE часть, дополнительные поля в группировку – в GROUP BY часть.

10. Цикл по проводкам. Конструкция организует цикл по SQL запросу, построенному по проводкам, которые ограничены набором бухгалтерских счетов и при необходимости набором аналитик, и периодом действия. Запрос выбирает конкретные проводки, полученные с указанными ограничениями. Настройки блока полностью совпадают с предыдущим блоком.

11. SQL цикл. Конструкция организует цикл по произвольному SQL запросу.



В настройке блока указывается SQL запрос, который может содержать произвольный запрос с параметрами. Для каждого параметра на закладке «Параметры» указывается значение (выражение), которое будет присвоено данному параметру.

12. SQL запрос. Конструкция позволяет сформировать SQL запрос.

Блок аналогичен предыдущему, с тем ограничением, что доступна только одна запись.

13. Пользовательский блок. Позволяет написать произвольный скрипт.

В свойствах данного блока при написании скрипта, доступны кнопки аналогично окну «Редактор выражений».

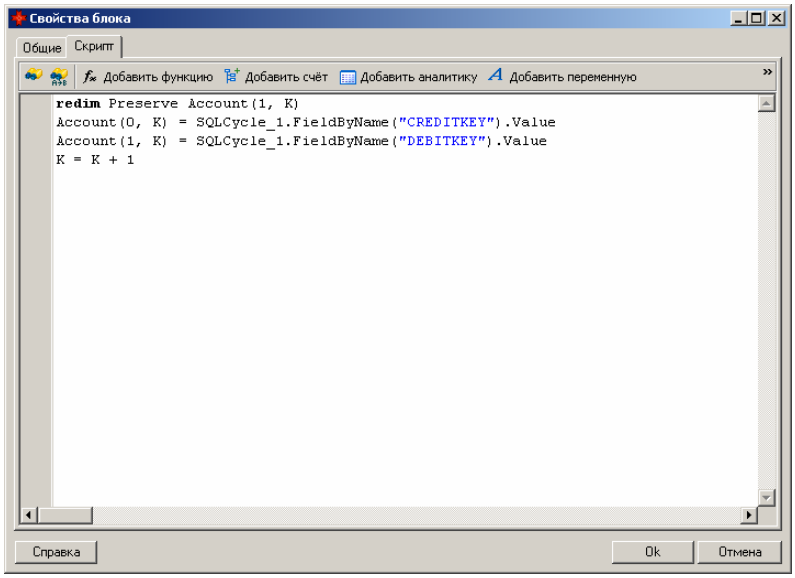


Рис.19 Свойства блока. Пользовательский.

14. Типовая проводка. Конструкция предназначена для определения создаваемой проводки. Блок доступен только при создании функции типовой проводки.

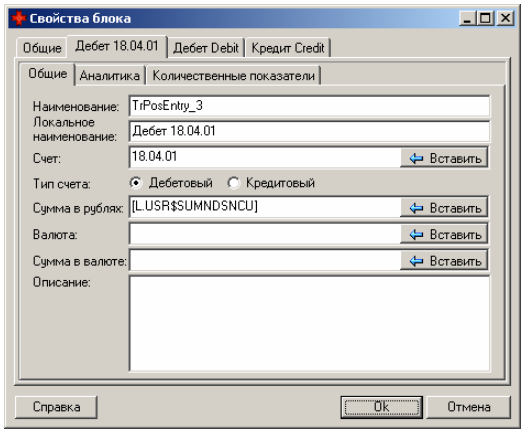


Рис.20 Свойства блока. Типовая проводка.

При настройке проводки на закладке «Общие» - указываем наименование блока, локальное наименование, описание блока. Для проводки можно указать, что она формируется только для конкретной рабочей компании, указать дату проводки (если она отличается от даты документа, который формирует проводку) – произвольное выражение, которое возвращает дату, описание проводки – произвольное выражение, возвращающее строку (например: для проводок по банковской выписке это может быть комментарий, введенный в позиции выписки). Закладка «Дебет» – настраивается дебетовая часть проводки. Наименование блока, локальное наименование (формируется автоматически Дебет + счет), Счет – номер конкретного счета или выражение, которое формирует этот счет, Сумма в рублях – выражение которое рассчитывает сумму дебетовой части проводки, Валюта – указывается валюта для мультивалютного счета (выражение или конкретная валюта), Сумма в валюте – для мультивалютного счета, указывается выражение для расчета валютной суммы. На закладке «Аналитика», указываются значения аналитик, если в поле счет был указан конкретный счет, то на данной закладке будут доступны только те виды аналитик, которые установлены для данного счета, если для счета указано выражение, то на закладке будут доступны все виды аналитик. На закладке «Количественные показатели» добавляются

записи для каждого количественного показателя, который учитывается по счету. Закладка «Кредит» - настраивается кредитовая часть проводки, полностью аналогично дебетовой части проводки. Блок типовая проводка формирует простую проводку, для превращения ее в сложную проводку, внутрь этого блока добавляется блок «Позиция типовой проводки».

15. Позиция типовой проводки. Конструкция предназначена для добавления новой позиции в типовую проводку. Настройка блока аналогична настройке закладке «Дебет» или «Кредит» блока типовой проводки.

16. Забалансовая проводка. Конструкция предназначена для формирования проводки состоящей только из кредитовой или дебетовой части.

Пример создания типовой проводки

Для примера создадим типовую проводку по документу «Поступление товара от поставщика», входящую в стандартную настройку оптово-розничного склада. Нам необходимо, что бы по данному документу формировалась следующая проводка

Дебет	Кредит	Сумма	Аналитка
41.01 Товары на складе		Сумма покупки без НДС	Подразделение
18.04.01 НДС полученный		Сумма НДС	Клиент Документ Тип НДС
	60 Расчеты с поставщиками	Сумма с НДС	Клиент Документ Тип НДС

Документ «Поступление товара от поставщика» состоит из шапки и позции, Шапка содержит номер, дату, поставщика и подразделение, балансовый счет расчета с поставщиками. Позиция содержит товар, количество, сумму покупки без НДС, ставку НДС, сумму НДС и сумму с НДС, балансовый счет учета товарно-материальных ценностей (ТМЦ).

При формировании проводки мы хотим, что бы по умолчанию формировалась, указанная выше проводка, но пользователь при желании мог установить свой счет расчета с поставщиками и счет учета ТМЦ.

Проанализируем аналитику, которая используется в проводке. Клиент – соответствует полю «поставщик» из шапки документа. Документ – это идентификатор шапки документа. Подразделение – полю подразделение из шапки документа. Тип НДС ссылка на запись из справочника типов НДС, который содержит следующие записи НДС 20%, НДС 18%. НДС 10%, НДС РФ, НДС 0%, Без НДС – значение типа НДС зависит от ставки НДС и страны поставки в позиции документа.

Для настройки данной проводки нам необходимо сделать следующее:

- Создаем типовую операцию «Поступление товара от поставщика»

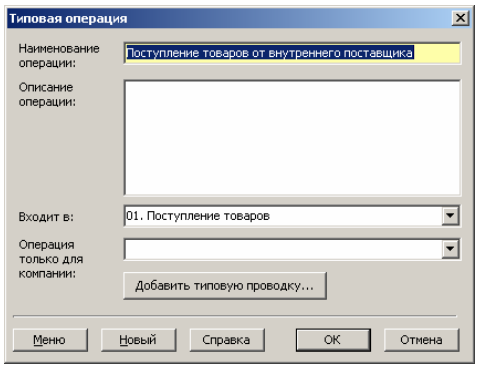


Рис. 21. Диалог «Типовая операция»

- Добавляем для этой операции типовую проводку, в которой указываем ее наименование, документ «Поступление товара от поставщика», указываем, что проводка формируется по позиции документа, и нажимаем на кнопку «Конструктор».

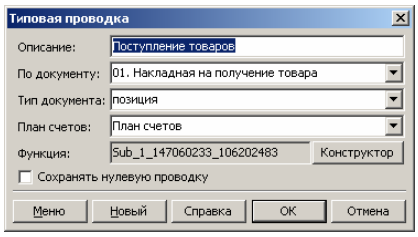


Рис. 22. Диалог «Типовая проводка»

Начнем формирование нашей функции с определением счетов по дебету и кредиту. По условию задачи нам необходимо, что бы в проводке в качестве счета учета расчетов с поставщиками использовался либо счет, который указал пользователь в шапке документа, либо если он этого не сделал 60 счет. Для реализации этого мы будем использовать переменную Credit (Кредит), которую будем определять внутри блока «Условие», «Иначе»

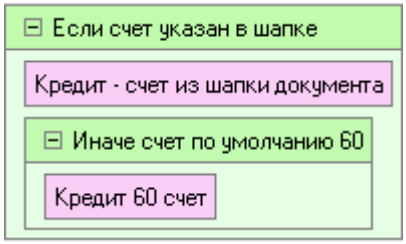


Рис. 23. Блок установки счета по кредиту.

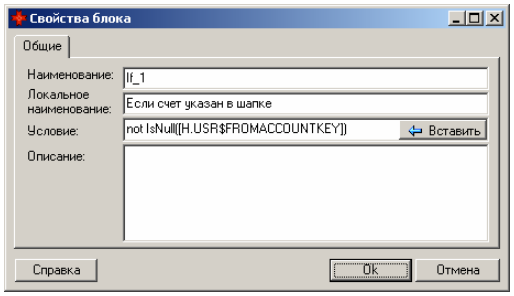


Рис. 24 Настройка блока условие, для выбора счета по кредиту.

В условии данного блока записано, что если поле счет в шапке документа (USR$FROMACCOUNTKEY – название поля «счет» в шапке документа) не пусто, то далее в следующем блоке присваиваем переменной Credit, значение этого счета:

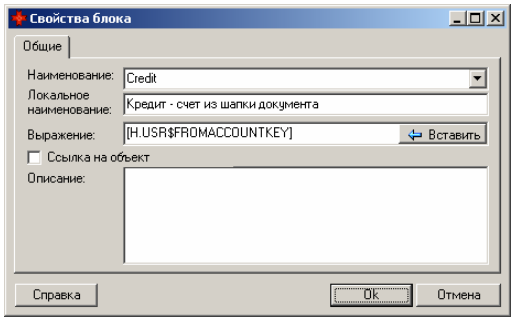


Рис. 25. Настройка блока переменная, счет Кредит.

Если условие не выполнится (счет в шапке документа не задан), то будет выполнен блок «Иначе», в котором переменной Credit будет присвоено значение идентификатора записи 60 счета.

Аналогичная конструкция используется для определения счета учета ТМЦ. Здесь у нас появится переменная Debit, которая будет равна значению поля «Счет» из позиции документа, если он задан или же счету 41.01 если не задан.

Следующим шагом иедт определения аналитики тип НДС. Для этого воспользуемся блоком «выбор из» и блоком «выбор». В блоке «выбор из», в строке условие мы укажем поле позиции документа Ставка НДС прихода (TO_USR$INV_ADDNDS). Для каждого возможного значения этого поля будут вставлены блоки «выбор», с соответствующими значениями

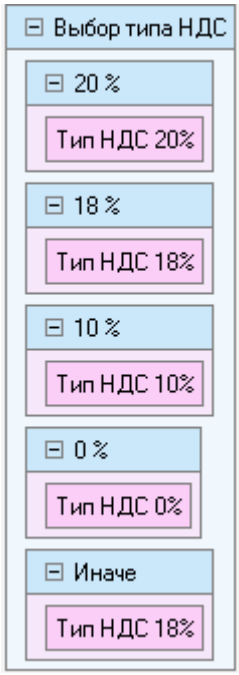
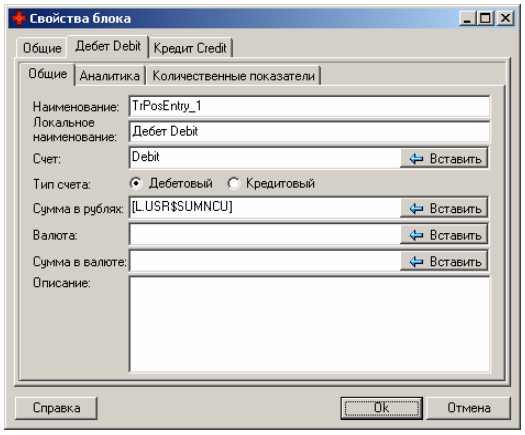


Рис. 26 Выбор типа НДС.

В каждом блоке «Выбор», присваивается переменная VatType значению соответствующей аналитике из справочника Типов НДС.

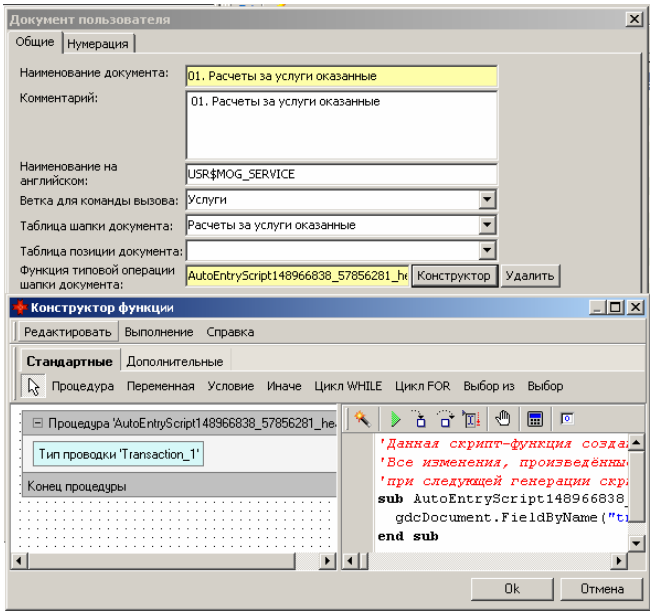
Теперь у нас все готово для формирования типовой проводки. Следующий блок «Типовая проводка». В этом блоке на закладке Дебет/Общие указываем Счет – переменная Debit, Сумма в рублях – значение поля сумма без НДС из позиции документа (L.USR$SUMNCU – L – позиция документа, USR$SUMNCU – поле сумма без НДС). На закладке Аналитика – в строке подразделение – значение поля подразделение из шапки документа (H.USR$DEPTKEY). На закладке Кредит/Общие указываем в строке Счет – переменную Credit, в строке Сумма в рублях – зарезервированную переменную DebitNCU. Переменная DebitNCU (и соответствующая ей CreditNCU) содержит общую сумму дебетовой (кредитовой) части проводки, рекомендуется использовать данную переменную в простой части проводки, в блоке типовой проводки, простая часть должна идти последней, что бы можно было корректно использовать эту переменную. На закладке «Кредит/Аналитика» устанавливаем значения для следующих видов аналитик: клиент – [H.USR$CONTACTKEY], документ – [H.ID], тип НДС – переменная VatType. Выбор полей и переменных осуществляется через команду Вставить/Выражение.



В нашу проводку осталось добавить еще одну строку по дебету, используем блок «позиция типовой проводки», в котором указываем счет 18.04.01 (счет учета полученного НДС по товарам), сумму

[L.USR$SUMNDSNCU], тип счета – Дебет, аналитику устанавливаем аналогично кредитовой части проводки. Все на этом создание типовой проводки завершено.

Для того, что бы проводка по документу сформировалась необходимо в позиции или шапке документа присвоить типовую операцию. Обычно это осуществляется двумя путями или вручную пользователь сам при формировании документа указывает операцию (так происходит при вводе и обработке выписки), или настройщик пишет функцию, которая присваивает операцию в документе автоматически. Данная функция задается в режиме типовых документов «Исследователь/Сервис/Типовые документы».



В диалоге редактирования типового документа по шапке или позиции, используя, конструктор можно создать функцию присвоения типовой операции (функция типовой операции по шапке документа, функция типовой операции по позиции документа). Все блоки конструктора, совпадают с ранее описанными блоками, исключение составляет новый блок «Типовая операция», который как раз и отвечает за присвоение типовой операции.

Автоматические операции (АО)

АО предназначены для формирования проводок по заданному алгоритму. Используются для настройки автоматического закрытия сальдо по счета, в разрезе аналитик, расчета реализации (оплаченной и отгруженной продукции, тмц), перевода НДС в оплаченный, расчета начисленного НДС по 60-ти дням и т.д.

Что собой представляет АО?

АО – является обычной типовой операцией, проводки по которой не привязаны к конкретному документу. Проводки по АО создаются аналогично типовым проводкам с помощью такого же конструктора. Конструктор АО отличается от типовой проводки следующим:

Отсутствуют следующие блоки:

- Типовая проводка
- Позиция типовой проводки
- Забалансовая проводка

Присутствует новый блок

- Проводка, предназначен для формирования простой проводки

В редакторе выражений отсутствует кнопка «Добавить поле документа». Рассмотрим новый блок «Проводка».

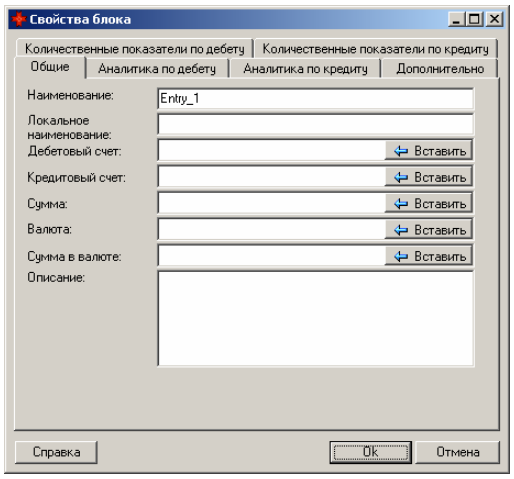


Рис. 28 Свойства блока «Проводка».

Данный блок предназначен для формирования простой проводки (один дебет и один кредит)

При настройке блока, необходимо указать значение (выражение) для дебетового и кредитового счета, указать выражение для суммы, валюты и суммы в валюте (если проводка валютная). На закладках «Аналитика по дебету», «Аналитика по кредиту» указывается значение (выражение) для интересующей аналитики. Аналитики на соответствующих закладках отображаются либо конкретные (соответствующие указанному счету, если счет указан точно) или все которые есть (если счет указан, как выражение). На закладках «Количественные показатели по дебету», «Количественные показатели по кредиту», указываются значения (выражения) для необходимых количественных показателей.

Пример автоматической операции

Ниже на рисунке изображена типовая операция, формирования зачета НДС. Суть задачи необходимо с 18-х счетов перенести сальдо в дебет 68 счета, причем сальдо на 18 счете хранится в разрезе 3-х видов аналитики, во вторых сальдо по 68 счету не должно стать в результате этой операции дебетовым. Рассмотрим, как формировалась данная АО. Сначала в переменную заносим кредитовый оборот по 68 счету, используя встроенную функцию ОК. Затем в пользовательском блоке определяем список 18 счетов, которые подлежат закрытию. Далее по эти счетам организуем цикл (блок «Цикл FOR»), далее организуем, цикл по запросу, которые высчитывает сальдо по конкретному 18 счету в разрезе 3-х видов аналитик (usr$gs_customer – клиент, usr$gs_document - документ, usr$vattype_key – тип НДС). В цикле формируется проводка Дебет 68 счета – Кредит 18 счета, причем перед тем как ее сформировать проверяется сумма, что бы не выйти за пределы кредитового оборота, после каждой проводки, уменьшается переменная, которая хранит кредитовый оборот.

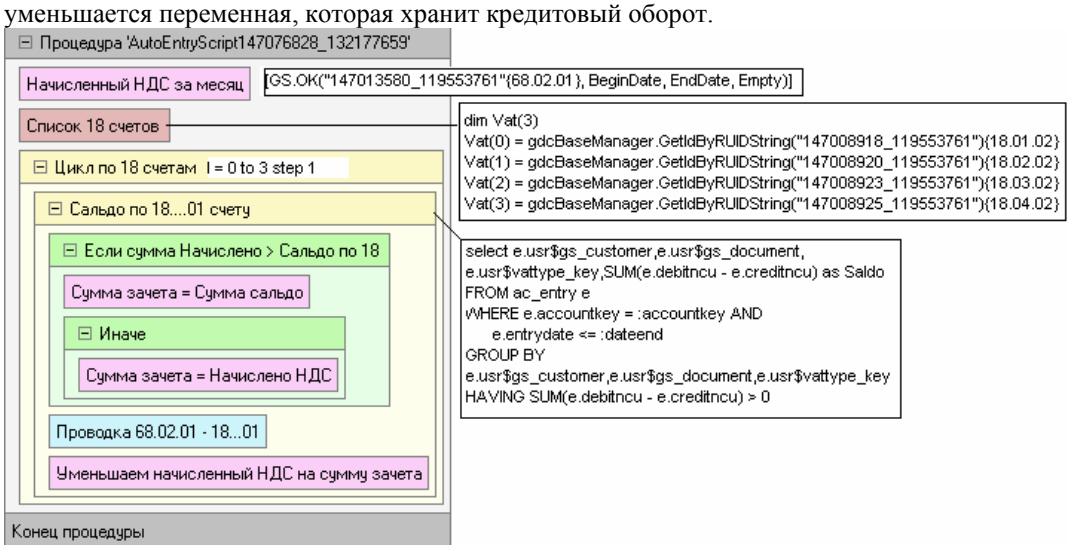
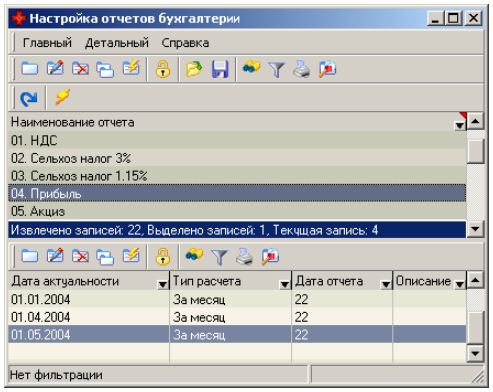


Рис. 29 Пример АО.

Расчет налогов и формирование проводок

Расчет налогов производится в разделе «бухгалтерские отчеты» БО. БО представляет собой совокупность функции расчета и печатной формы к нему. Данный тип отчетов предназначен для автоматизации расчета налогов, баланса, приложений и других аналогичных расчетов. Бухгалтерский отчет состоит из заголовка и расчета, который привязан к конкретной дате, это позволяет хранить в базе историю изменения данных расчетов, и проводить эти расчеты за любой расчетный период.

Настройка БО производится в разделе «Настройка функций»



Для каждого расчета, указывается дата актуальности данного расчета (дата с которой он будет применяться), тип расчета (ежемесячный, ежеквартальный, за произвольный период), отчетное число данного отчета (число до которого отчет должен быть сдан) и функция расчета.

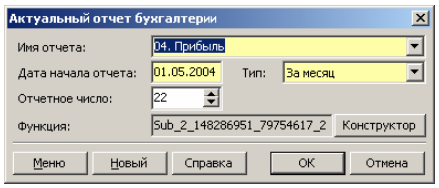


Рис. 31 Актуальный отчет бухгалтерии.

Функция расчета формируется с помощью все того же конструктора. В конструкторе присутствуют все блоки, как и в АО, плюс добавляется новый блок «Позиция отчета». Данный блок полностью аналогичен блоку «Переменная» и отличается от него только дальнейшей обработкой, все переменные объявленные в блоке «Позиция отчета», затем доступны в привязанной к данному расчету печатной форме. Для формирования проводок по отчету, служит так же, как и в АО блок «Проводка».

Расчет и печать отчета производится в разделе «Расчет». Расчет отчета производится по кнопке  «Рассчитать за период», по этой кнопке появится окно с выбором отчетов:

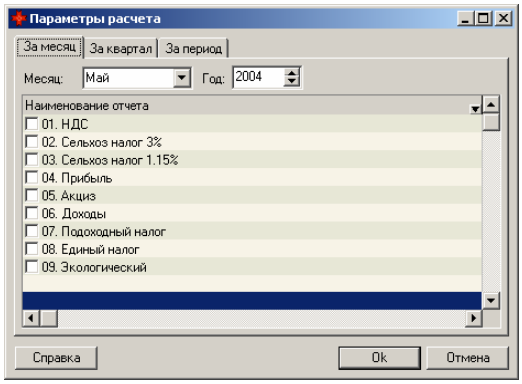


Рис. 32 Параметры расчета.

В этом окне отмечаем нужные отчеты, указываем за какой месяц (квартал, период), они будут рассчитаны и нажимаем OK – отчеты рассчитаны, программа спросит, сохранить результаты или нет, после можно печатать нужный отчет. Таким образом, пользователь рассчитывает налоги, баланс, приложения и т.д.

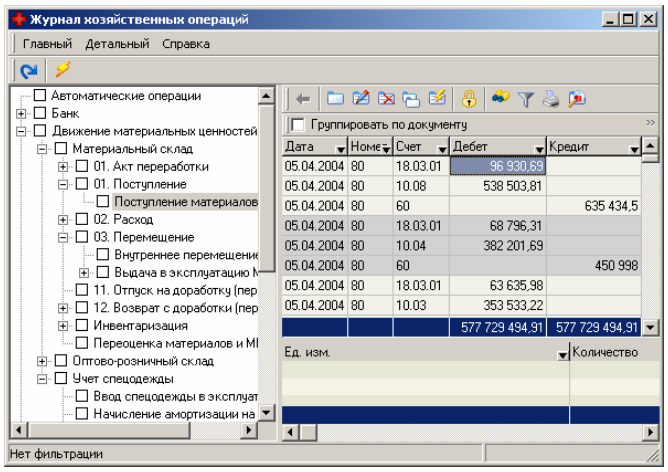
Рассмотрим, как формируется новый отчет. Сначала добавляем его в список отчетов (верхний список в окне «Настройка отчетов бухгалтерии»). Затем по данному отчету в нижнем списке добавляем строку актуальный отчет бухгалтерии, где указываем дату начала, периодичность, отчетное число. С помощью конструктора формируем функцию расчета. Для всех показателей, которые должны быть в печатной форме отчета используем блок «Позиция отчета». После этого рассчитываем сформированный отчет. Затем возвращаемся в «Настройку функций», где формируем печатную форму нашего отчета. Печатная форма формируется в «Редакторе скрипт-объектов» в группе отчетов, которая называется так же как наш отчет и у которой в скобках прописана соответствующая нашему расчету дата актуальности. Формирование отчета, полностью аналогично формирование обычных отчетов в «Редакторе скрипт-объектов» о которых Вы читали ранее, в стандартных настройках бухгалтерских отчетов идет стандартная функция, которая позволяет формировать печатную форму для любого отчета бухгалтерии, воспользовавшись этой функцией тело которой приводится ниже, Вам остается только нарисовать нужную форму.

Функция подготовки данных для печатной формы БО.

```
  BaseQueryList.Clear
  set gdcTaxResult = OwnerForm.FindComponent("gdcTaxResult")
  B = gdcTaxResult.GetBookMark
  gdcTaxResult.DisableControls
  gdcTaxResult.Close
  gdcTaxResult.Open
  gdcTaxResult.First
  call BaseQueryList.Add("TaxResult", 1)
  set tr = BaseQueryList.Query(0)
  CountRecort = 0
  While not gdcTaxResult.Eof
    FunctionName = gdcTaxResult.FieldByName("name").AsString + "_R"
    call TR.AddField(FunctionName, "ftString", 60, 0)
    CountRecord = CountRecord + 1
    gdcTaxResult.Next
  Wend
  gdcTaxResult.First
  TR.Open
  TR.Append
  for I = 0 to CountRecord - 1
    TR.Fields(I).AsString =gdcTaxResult.FieldByName("result").AsString
    gdcTaxResult.Next
  next
  TR.Post
  gdcTaxResult.EnableControls
  gdcTaxResult.GotoBookMark(B)
```

Внесение вручную

Не всегда есть возможность полностью описать все действия организации соответствующим документооборотом, а учет вести необходимо, отсюда вытекает необходимость ручного внесения проводок. Все проводки, которые были созданы по документам, автоматически или же вручную содержатся в «Журнале хозяйственных операций». Все проводки в журнале отображаются в разрезе типовых операций, сюда же попадают и автоматические операции и операции связанные с расчетом бухгалтерских отчетов.



В журнале хозяйственных операций пользователь может вносить произвольные проводки вручную, причем он может вносить как простые , так и сложные проводки.

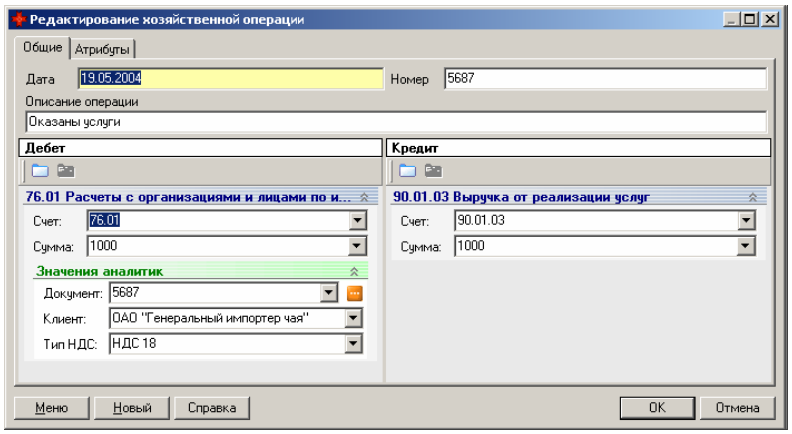


Рис. 34 Добавление хозяйственной операции.

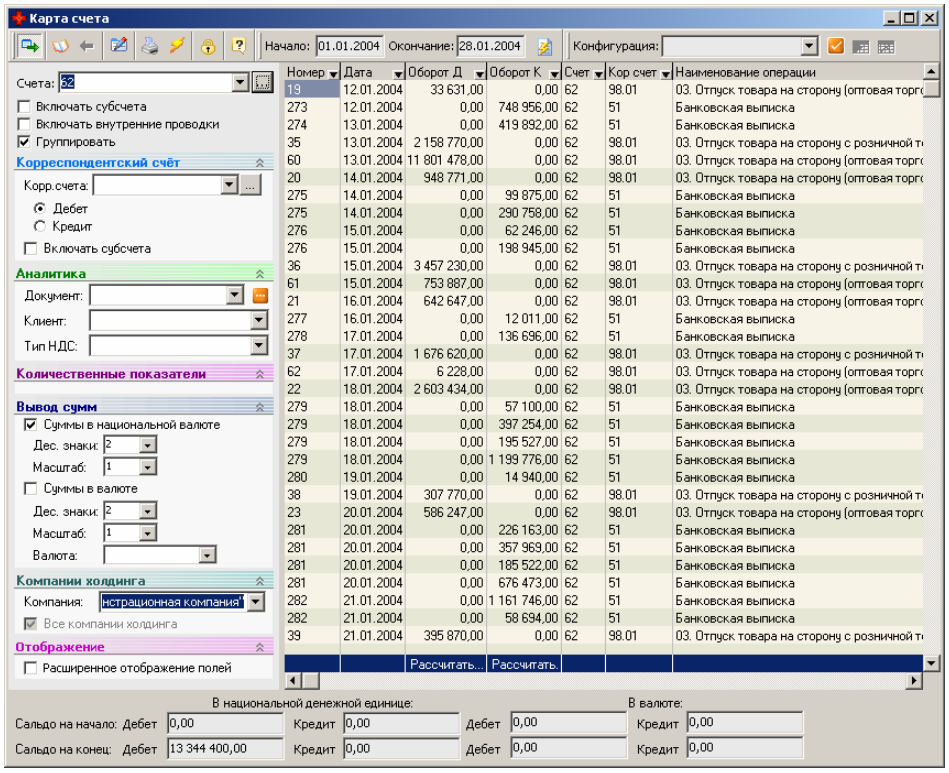
Внешне это нарушает декларируемую выше формулу, на самом деле это не совсем так. Ручная проводка на самом деле тоже документ, который просто скрыт от пользователя. Это означает, что каждая ручная проводка это отдельный документ, к которому эта проводка относится. В системе сразу заложено, что если по счету в ручной проводке ведется аналитика по документу и в проводке эта аналитика не указана, она будет заполнена автоматически – ссылкой на документ этой проводки. Ниже мы рассмотрим, как можно уже на уровне данных отличить проводку созданную вручную от проводки, созданной по реальному документу. Еще заметим, что каждую проводку можно изменить, только через тот документ, который ее создал. Это означает, что при редактировании произвольной проводки в журнале хозяйственных операций, система откроет окно с редактированием того документа, который создал эту проводку.

Стандартные отчеты их использование и настройка.

К стандартным отчетам, относятся следующие отчеты: Главная книга, Журнал-ордер, Карта счета, Оборотная ведомость. Данные отчеты заложены в самой платформе и не зависят от загруженных настроек. Все отчеты имеют одинаковый механизм конфигурирования и сохранения конфигураций. Рассмотрим все отчеты по порядку:

Карта счета

Данный отчет отображает набор проводок за определенный промежуток времени, в разрезе выбранной конфигурации:



В окне «Карта счета» левая панель предназначена для конфигурирования данного отчета.

Счета – номера счетов через запятую.

Включать счета – признак будут ли включаться по выбранным счетам, еще и все их субсчета (включены будут все вложенные субсчета всех уровней)

Включать внутренние проводки – признак будут ли отображаться в отчете внутренние проводки. Под внутренними проводками понимается проводка, у которой дебет = кредиту.

Группировать – признак группировки проводок. Проводки будут сгруппированы по аналитике и идентификатору шапки документа, по которому были созданы проводки.

Корреспондентский счет – указывается счет, в корреспонденции с которым необходимо выбрать проводки.

Дебет/Кредит – Относится к корреспондентскому счету, в какой части проводки он должен находиться (дебетовой или кредитовой).

Включать субсчета – признак необходимо ли включать субсчета корреспондентского счета.

Аналитика – в зависимости от выбранных счетов в данном блоке будут доступны те виды аналитик, которые определены по данным счетам. По каждой аналитике можно указать конкретное значение в разрезе, которого будут ограничен набор проводок.

Количественные показатели – в зависимости от выбранных счетов в данном блоке доступны те показатели, которые определены по данным счетам. В случае выбора данных показателей в отчете, кроме суммовых показателей будут еще и количественные.

Вывод сумм – в данном блоке указываются, какие суммы (рублевые, валютные) будут отображаться в отчете, сколько десятичных знаков отображается, масштаб в котором будут выводиться эти суммы.

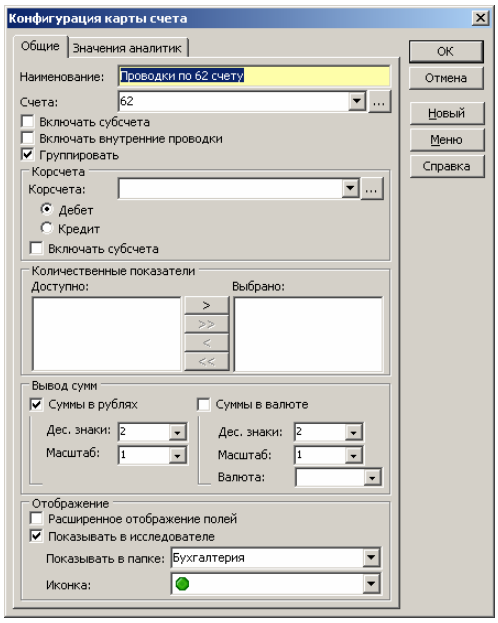
Компания холдинга – из списка можно выбрать рабочую организацию, по которой будет строиться отчет, по умолчанию там стоит текущая организация.

Все компании холдинга – для компании, являющейся холдингом установка данного переключателя позволяет построить отчет по всем дочерним компаниям холдинга, если переключатель отключен, то отчет строится только по текущей компании.

Расширенное отображение полей – признак, выводить ли дополнительные поля, для аналитик, которые ссылаются на другие справочники или документы (например: для документа можно сделать, чтобы кроме номера выводилась еще и дата документа).

Заполнение данной конфигурации позволяет создать отчет, который будет отображать хозяйственные операции в указанном разрезе по указанным ограничениям.

Если какая-то конфигурация используется достаточно часто ее можно сохранить, с возможностью создания отдельной команды в «Исследователе», которая будет строить уже сконфигурированный отчет. Для этого необходимо: если конфигурация уже настроена в текущем окне, то нажать на кнопку рядом с полем выбора конфигурации, в появившемся окне ввести имя и конфигурация будет сохранена. Для того, что бы добавить эту конфигурацию как самостоятельную команду в «Исследователь» необходимо зайти в редактирование конфигурации. В окне «Конфигурация карты счета» необходимо установить флаг «Отображать в исследователе» и указать, в какой ветке данный отчет будет вызываться. В данном окне также можно полностью изменить конфигурацию данного отчета. На закладке «Значения аналитик» можно установить конкретные значения аналитик или установить переключатель запрашивать значение, тогда при вызове отчета из «Исследователя» у пользователя будут запрошены соответствующие значения аналитик. Для уже существующей конфигурации можно сохранить и настройки таблицы, в которой выводится отчет, для этого необходимо выбрать нужную конфигурацию, по этой конфигурации построить отчет, настроить таблицу и нажать кнопку , текущая настройка таблицы будет сохранена в выбранной конфигурации. Заметим, что если в результате изменения конфигурации или других изменений в отчете появятся новые поля, то если была сохранена настройка таблицы отчета, эти поля не отобразятся – они будут скрыты, их необходимо открыть (если они нужны) и сохранить настройку таблицы еще раз.



Из любой строки отчета можно попасть соответствующую ей проводку в журнале хозяйственных операций, по кнопке или сразу в редактирование документа, который создал данную проводку, по кнопке .

Журнал ордер

Данный отчет предназначен для отображения обобщенной информации по счетам в различном разрезе с разной группировкой, с необходимыми ограничениями, как частный случай, данный отчет позволяет настроить стандартные формы бухгалтерских журналов ордеров. Отчет может быть сконфигурирован, конкретная конфигурация может быть сохранена и выведена отдельной командой в «Исследователь». Используя данный механизм, можно настроить такие отчеты, как отчеты по задолженностям поставщиков и покупателей в разрезе наименований, документов и т.д., реестры поступления денежных средств и т.д.

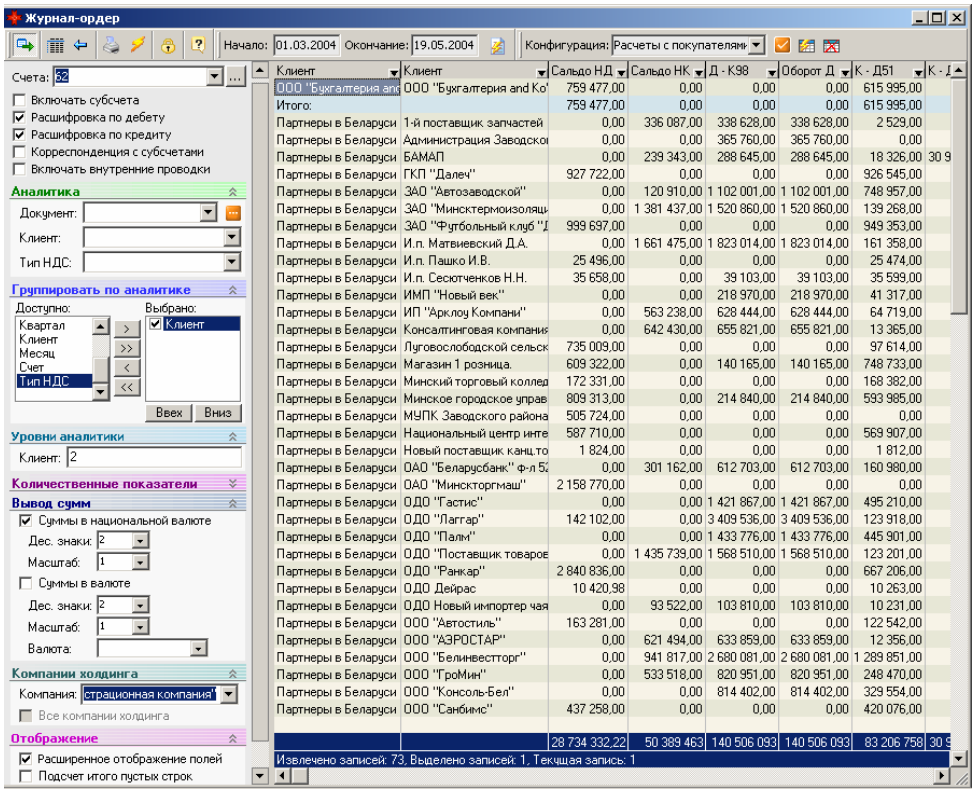


Рис. 38 Журнал-ордер.

Настройка полностью аналогична карте счета за исключением добавленных новых пунктов:

Группировать по аналитике – Основной пункт настройки журнала ордера. Здесь указывается, в каком разрезе будет строиться отчет. Для группировки всегда доступны следующие показатели –

Валюта – Отчет будет сгруппирован по видам валют, которые ведутся на выбранных счетах.

Год, Квартал, Месяц, Дата – Отчет будет сгруппирован соответственно по годам, кварталам, месяцам и числам

Счет – Отчет будет сгруппирован по счетам (субсчетам), на основании, указанных счетов, по которым строится отчет

Кроме этого в зависимости от выбранных счетов, здесь будут доступны разделы аналитики, которые ведутся по указанным счетам.

Вид группировки определят расчет сальдовых показателей по строкам отчета. Если в качестве первых уровней, указаны виды аналитики, валюта или счета, то сальдо будет считаться отдельно по каждой строке (например: группировка по клиентам, т.е. каждая строка – это клиент, а сальдо в строке – это сальдо по этому клиенту). Если первым уровнем аналитики будут идти временные параметры (год, месяц, квартал, дата), то сальдо будет считаться с нарастающим итогом, т.е. сальдо на конец предыдущей строки будет переходить в сальдо на начало следующей строки.

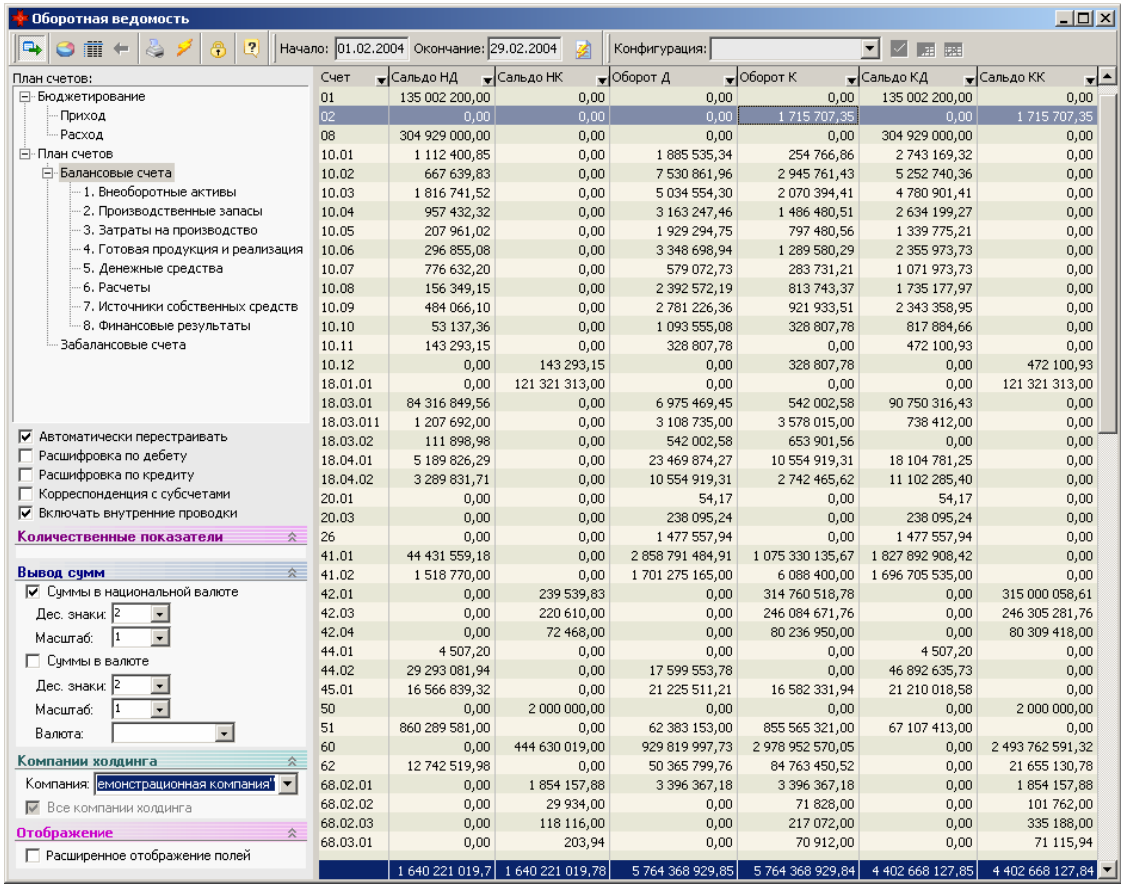
Уровни аналитики – в данный блок попадают те доступные виды аналитик, которые имеют древовидную структуру. Для этих видов аналитик, можно указать номер уровня, по которому будет осуществлена дополнительная группировка данных.

Находясь в журнале по любой цифре можно получить расшифровку, откуда она получилась. Для этого есть возможность быстрого построения «Карты счета» по текущим параметрам исходя из строки и столбца выбранной цифры. Карта счета строится по нажатию на кнопку  или по двойному щелчку мыши на этой цифре.

Сохранение конфигурации и настроек таблицы аналогично предыдущему отчету.

Оборотная ведомость

Данный отчет отображает оборотно-сальдовую ведомость по балансовым счетам, активного плана счетов. Отчет может быть сконфигурирован, аналогично предыдущему, конфигурацию можно сохранить и вызывать отдельной командой «Исследователя».



Счет	Сальдо НД	Сальдо НК	Оборот Д	Оборот К	Сальдо КД	Сальдо КК
01	135 002 200,00	0,00	0,00	0,00	135 002 200,00	0,00
02	0,00	0,00	0,00	1 715 707,35	0,00	1 715 707,35
08	304 929 000,00	0,00	0,00	0,00	304 929 000,00	0,00
10.01	1 112 400,85	0,00	1 885 535,34	254 766,86	2 743 169,32	0,00
10.02	667 639,83	0,00	7 530 861,96	2 945 761,43	5 252 740,36	0,00
10.03	1 816 741,52	0,00	5 034 554,30	2 070 394,41	4 780 901,41	0,00
10.04	957 432,32	0,00	3 163 247,46	1 486 480,51	2 634 199,27	0,00
10.05	207 961,02	0,00	1 929 294,75	797 480,56	1 339 775,21	0,00
10.06	296 855,08	0,00	3 348 698,94	1 289 580,29	2 355 973,73	0,00
10.07	776 632,20	0,00	579 072,73	283 731,21	1 071 973,73	0,00
10.08	156 349,15	0,00	2 392 572,19	813 743,37	1 735 177,97	0,00
10.09	484 066,10	0,00	2 781 226,36	921 933,51	2 343 358,95	0,00
10.10	53 137,36	0,00	1 093 555,08	328 807,78	817 884,66	0,00
10.11	143 293,15	0,00	328 807,78	0,00	472 100,93	0,00
10.12	0,00	143 293,15	0,00	328 807,78	0,00	472 100,93
18.01.01	0,00	121 321 313,00	0,00	0,00	0,00	121 321 313,00
18.03.01	84 316 849,56	0,00	6 975 469,45	542 002,58	90 750 316,43	0,00
18.03.011	1 207 692,00	0,00	3 108 735,00	3 578 015,00	738 412,00	0,00
18.03.02	111 898,98	0,00	542 002,58	653 901,56	0,00	0,00
18.04.01	5 189 826,29	0,00	23 469 874,27	10 554 919,31	18 104 781,25	0,00
18.04.02	3 289 831,71	0,00	10 554 919,31	2 742 465,62	11 102 285,40	0,00
20.01	0,00	0,00	54,17	0,00	54,17	0,00
20.03	0,00	0,00	238 095,24	0,00	238 095,24	0,00
26	0,00	0,00	1 477 557,94	0,00	1 477 557,94	0,00
41.01	44 431 559,18	0,00	2 858 791 484,91	1 075 330 135,67	1 827 892 908,42	0,00
41.02	1 518 770,00	0,00	1 701 275 165,00	6 088 400,00	1 696 705 535,00	0,00
42.01	0,00	239 539,83	0,00	314 760 518,78	0,00	315 000 058,61
42.03	0,00	220 610,00	0,00	246 084 671,76	0,00	246 305 281,76
42.04	0,00	72 468,00	0,00	80 236 950,00	0,00	80 309 418,00
44.01	4 507,20	0,00	0,00	0,00	4 507,20	0,00
44.02	29 293 081,94	0,00	17 599 553,78	0,00	46 892 635,73	0,00
45.01	16 566 839,32	0,00	21 225 511,21	16 582 331,94	21 210 018,58	0,00
50	0,00	2 000 000,00	0,00	0,00	0,00	2 000 000,00
51	860 289 581,00	0,00	62 383 153,00	855 565 321,00	67 107 413,00	0,00
60	0,00	444 630 019,00	929 819 997,73	2 978 952 570,05	0,00	2 493 762 591,32
62	12 742 519,98	0,00	50 365 799,76	84 763 450,52	0,00	21 655 130,78
68.02.01	0,00	1 854 157,88	3 396 367,18	3 396 367,18	0,00	1 854 157,88
68.02.02	0,00	29 934,00	0,00	71 828,00	0,00	101 762,00
68.02.03	0,00	118 116,00	0,00	217 072,00	0,00	335 188,00
68.03.01	0,00	203,94	0,00	70 912,00	0,00	71 115,94
	1 640 221 019,7	1 640 221 019,78	5 764 368 929,85	5 764 368 929,84	4 402 668 127,85	4 402 668 127,84

Рис.37 Оборотная ведомость.

План счетов – Позволяет выбрать нужный план счетов или раздел, по счетам которого будет построен отчет.

Автоматически перестраивать – При установленном переключателе отчет будет перестраиваться автоматически при выборе плана счетов (раздела планов счетов).

Расшифровка по дебету (кредиту) – При установленных переключателях дебетовая (кредитовая) часть отчета будет расшифрована в разрезе корреспондирующих счетов.

Включать внутренние проводки – Отображать в обороте, проводки у которых дебет совпадает с кредитом.

Количественные показатели – Выбранные количественные показатели добавятся в отчет по тем счетам, по которым они установлены

Вывод сумм – в данном блоке указываются, какие суммы (рублевые, валютные) будут отображаться в отчете, сколько десятичных знаков отображается, масштаб в котором будут выводиться эти суммы.

Компания холдинга – из списка можно выбрать рабочую организацию, по которой будет строиться отчет, по умолчанию там стоит текущая организация.

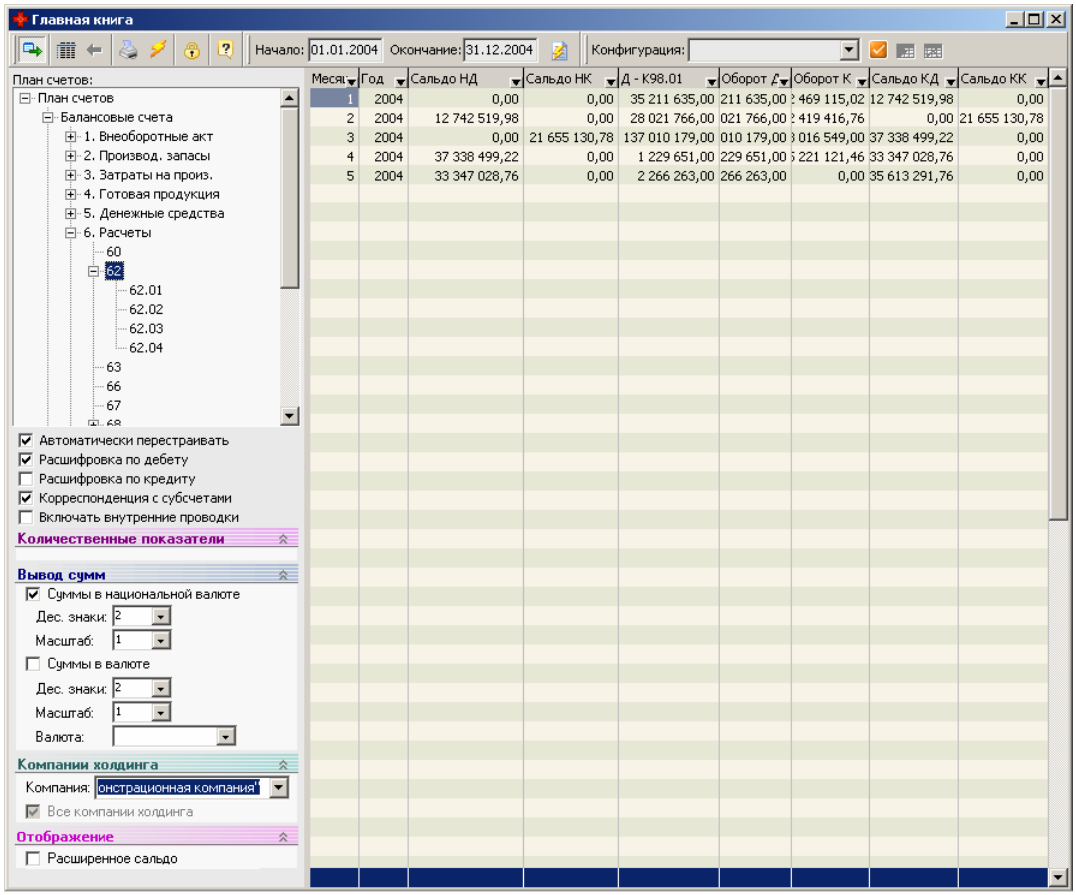
Все компании холдинга – для компании, являющейся холдингом установка данного переключателя позволяет построить отчет по всем дочерним компаниям холдинга, если переключатель отключен, то отчет строится только по текущей компании.

Расширенное отображение полей – признак, выводить ли дополнительные поля по плану счетов или нет (можно настроить, что бы кроме счета выводилось еще и его наименование).

При работе с отчетом каждую цифру можно расшифровать. По конкретному счету, можно построить или журнал ордер (CTRL+L) или карту счета (CTRL+G) или двойной щелочек мыши по необходимой сумме. Отчеты будут построены за выбранный для «Оборотной ведомости» период и по текущему счету. Журнал ордер будет построен, только по тем счетам, по которым ведется аналитический учета, по аналитике, указанной как главная аналитика счета, или идет первой в списке.

Главная книга

Отчет предназначен для построения главной книги предприятия.



Настройка данного отчета полностью совпадает с настройкой оборотной ведомости. Отличие только в переключателе «Расширенное сальдо», который есть в разделе «Отображение» в «Главной книге», он отвечает за то, будет ли по активно-пассивным счетам, выводится расширенное сальдо или нет.

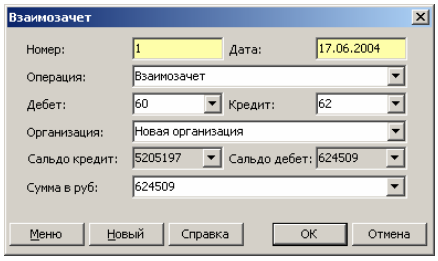
Дополнительные документы, отчеты и автоматические операции, реализуемые для бухгалтерского учета в стандартных настройках

Взаиморасчеты

Данный раздел содержит два документа, которые позволяют проводить взаиморасчеты по организациям и между организациями.

Взаимозачет

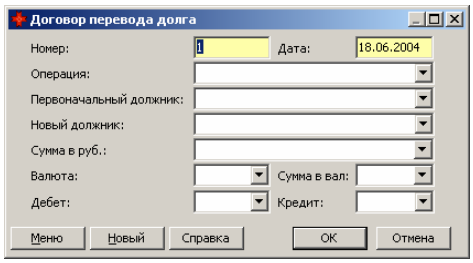
Данный документ предназначен для проведения взаимозачета по конкретной организации, сальдо по которой имеется по разным счетам. Например: организация является и поставщиком какой-либо продукции, материальных ценностей и соответственно имеется сальдо по 60 счету, и эта же организация является потребителем товаров и имеет сальдо по 62 счету. Для закрытия сальдо по данной организации используется документ «Взаимозачет».



При вводе документа, указывается операция, по которой будет формироваться проводка, по умолчанию есть общая операция «Взаимозачет», которая использует счета и организацию из документа. Счет по дебету, счет по кредиту, организация, по которой проводится взаимозачет. После выбора организации документ автоматически показывает сальдо по выбранным счетам в разрезе организации, после чего указывается сумма взаимозачета – обычно меньшее из указанных сумм сальдо по счетам.

Договор перевода долга

Данный документ переводит сальдо с одной организации на другую организацию по выбранным счетам. Например: организация получила продукцию, другая организация оплатила в счет долга первой или одна организация получила продукцию, другая организация отгрузила продукцию и между ними производится зачет согласно их желанию. Все такие операции проводятся через документ «Договор перевода долга».



Разноска по счетам

Данный режим предназначен для автоматизации процесса формирования проводок по закрытию сальдовых сумм по счетам в разрезе аналитик, на какие-то другие сальдовые суммы. Чаще всего данный режим используется для формирования проводок связки документов оплаты и получения (отгрузки). Настройка данного режима позволяет создать отдельную конфигурацию и использовать ее как самостоятельную команду «Исследователя». Рассмотрим, как настраивается конкретная разноска. Настройка производится в разделе «Исследователь/Разноска по счетам/Профили разноски».

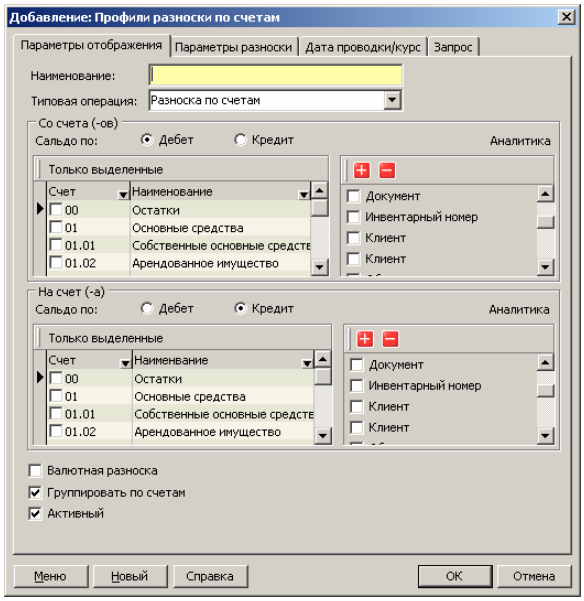


Рис. 40 Профиль разноски по счетам.

Указываем наименование профиля, под этим именем профиль будет отображаться в исследователе. Для каждой разноски можно создать свою типовую операцию. Затем необходимо указать какое (дебетовое или кредитовое) сальдо, и по каким счетам будет выводиться в верхней части разноски, и какое и по каким счетам будет выводиться в нижней части разноски. Для верхней и нижней части разноски указываются виды аналитик, которые должны выводиться в этих частях. Сразу отметим, что при формировании разноски, она раскрывает сальдо по всем уровням используемой аналитики и формирует проводки по всем уровням аналитики, здесь же, указываются именно отображаемые уровни аналитик. Для валютной разноски устанавливается соответствующий флаг. Переключатель группировать по счетам означает, что если в разноске участвует более одного счета, то закрытие все равно должно идти счет в счет (или же игнорируя это в противном случае).

На закладке «Параметры разноски», указывается, по каким признакам должна идти автоматическая разноска:

- По совпадению валют (для валютной разноски)
- По совпадению аналитик. В этом случае необходимо указать совпадение каких видов аналитик необходимо.

На закладке «Дата проводки/Курс», указывается на какую дату будут сформированы проводки:

- Дебетовая дата. Дата из дебетовой части разноски.
- Кредитовая дата. Дата из кредитовой части разноски.
- Минимальная дата. Минимальная дата из дебетовой и кредитовой дат.
- Дата, на которую строится сальдо. Дата, которая будет указана при построении разноски.
- Максимальная дата. Максимальная дата из дебетовой и кредитовой дат.

Для определения, что понимать под дебетовой и кредитовой датами ниже это необходимо указать. Для дебетовой и кредитовой даты можно выбрать следующие варианты:

- Дата построения сальдо
- Дата из документа (если существует аналитика документ)
- Прочие даты, которые могут быть видами аналитик.

Для валютной разноски, в случае, когда могут закрываться разные валюты друг на друга необходимо указать, как они будут пересчитываться. Есть два варианта:

- Кросс-курс. Для этого в справочнике валют необходимо вести курсы этих валют по отношению друг к другу.
- Через НДС. В этом случае пересчет будет идти через курсы данных валют к национальной денежной единице.

На закладке запрос, можно добавить в получаемый запрос, какие-то свои поля, подключить дополнительные таблицы, добавить группировку и сортировку, там же можно просмотреть полученный запрос.

Сформированные разноски попадут в ветку «Исследователь/Разноска по счетам».

Книга покупок

Книга покупок – отчет, который отражает суммы НДС, которые можно принять к зачету в текущем месяце. Отчет строится в разрезе документов поступления ТМЦ, оказания услуг и соответствующих документов оплат.

Стандартный пакет настроек «Бухгалтерия» включает в себя полную систему действий для получения, данного отчета.

Регистрация операций по полученному НДС. Варианты регистраций могут быть различные от ручного ввода проводок в журнал хозяйственных операций, до использования соответствующих блоков

(Материальный учет, Торговля, Учет основных средств, Учет услуг входящих и т.д.), главное чтобы данные попадающие в журнал хозяйственных операций отвечали следующим требованиям:

1. По счетам учета расчетов с поставщиками и по счетам учета входящего НДС, учет по документам поступления должен вестись по следующим объектам аналитики

- Клиент

- Документ поступления

- Тип НДС

2. По счетам расчетов с поставщиками по документам оплаты

- Клиент

- Документ оплаты

- Тип НДС (не обязательно)

Следующим шагом необходимо отразить в журнале хозяйственных операций, какой документ оплаты закрывает какой документ отгрузки. Система требует наличие проводок, в которых по дебету и по кредиту идут счета расчета с поставщиками. По дебету в аналитике указан Клиент (поставщик товаров или услуг), Документ – получения (товаров или услуг), Тип НДС, а по кредиту в аналитике указан Клиент (поставщик товаров или услуг), Документ – оплаты.

Следующий шаг, основан на предыдущем, по закрытым документам поступления (т.е. по тем документам, по которым есть оплаты), производится перевод НДС из полученного в оплаченный. В системе это должно быть отражено проводкой дебет счет учета НДС оплаченный, кредит счет учета НДС полученный, причем по дебету должна присутствовать следующая аналитика: Клиент, Документ оплаты, Тип НДС, а по кредиту – Клиент, Документ поступления, Тип НДС.

Пройдя данные шаги, мы почти готовы к построению книги покупок. Теперь нам осталось сформировать нужные нам конфигурации книги (система предполагает возможность ведения нескольких книг покупок) и книга покупок готова. Рассмотрим как происходит конфигурирование книги покупок. Для этого воспользуемся командой «Исследователь/Бухгалтерия/Книга покупок/Конфигурация книги покупок».

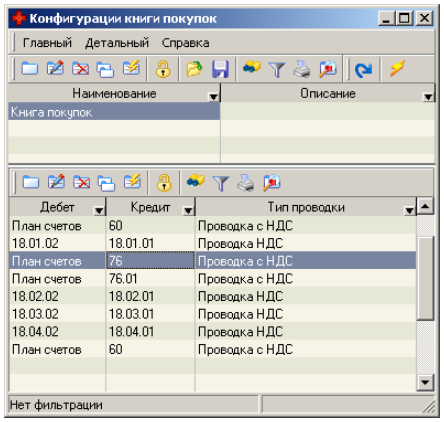
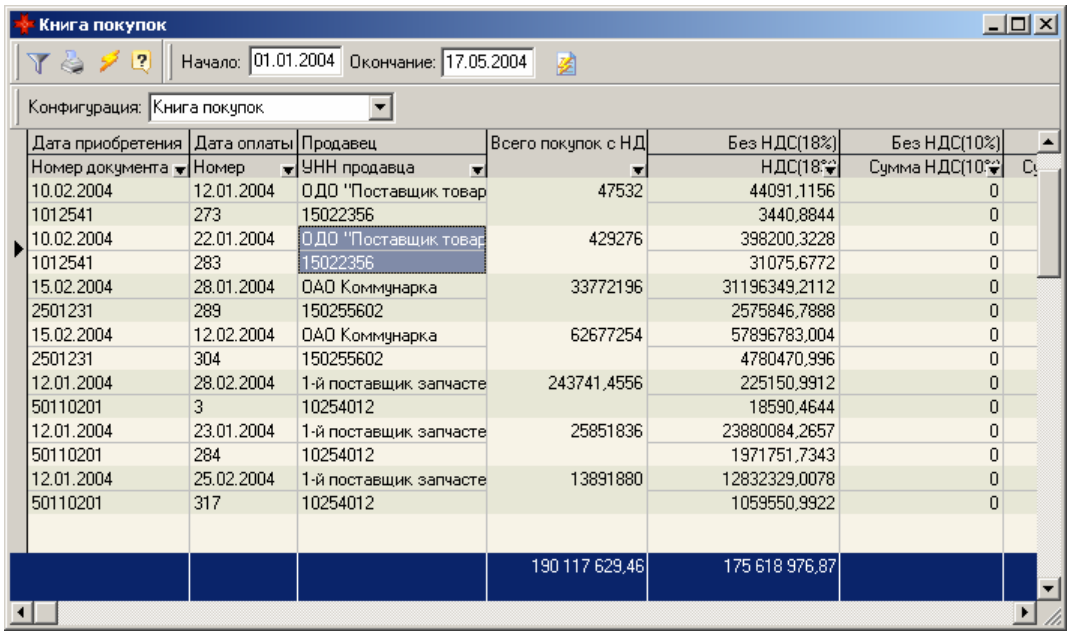


Рис. 40 Конфигурация книги покупок.

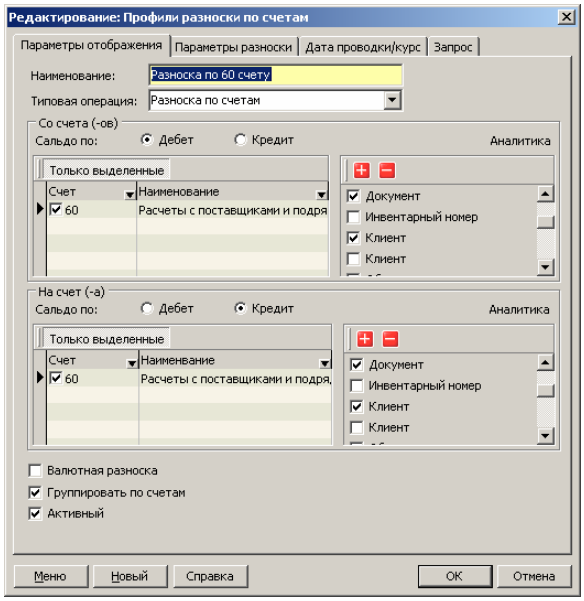
По данной команде открывается окно, которое содержит список конфигураций, и по каждой конфигурации указывается, из каких проводок она состоит. Необходимо указать какие счета, формируют суммы по документам поступления и оплаты, какие проводки учитывают перевод НДС полученный – оплаченный. Для этого при добавлении проводки для 1-го типа проводок необходимо указать тип поводки – Проводка с НДС и счет по кредиту (счет расчета с поставщиками), по дебету указывается произвольный счет. Заметим, что если для счета расчета с поставщиками суммы идут без НДС, то тип проводки устанавливается Проводка без НДС. Для 2-го типа проводок необходимо указать тип проводки – Проводка НДС, счет по дебету счет учета оплаченного НДС, счет по кредиту счет учета полученного НДС.

Теперь все готово для получения книги покупок.



Дата приобретения	Дата оплаты	Продавец	Всего покупок с НД	Без НДС(18%)	Без НДС(10%)
Номер документа	Номер	УНН продавца		НДС(18	Сумма НДС(10
10.02.2004	12.01.2004	ОДО "Поставщик товар	47532	44091,1156	0
1012541	273	15022356		3440,8844	0
10.02.2004	22.01.2004	ОДО "Поставщик товар	429276	398200,3228	0
1012541	283	15022356		31075,6772	0
15.02.2004	28.01.2004	ОАО Коммунарка	33772196	31196349,2112	0
2501231	289	150255602		2575846,7888	0
15.02.2004	12.02.2004	ОАО Коммунарка	62677254	57896783,004	0
2501231	304	150255602		4780470,996	0
12.01.2004	28.02.2004	1-й поставщик запчасте	243741,4556	225150,9912	0
50110201	3	10254012		18590,4644	0
12.01.2004	23.01.2004	1-й поставщик запчасте	25851836	23880084,2657	0
50110201	284	10254012		1971751,7343	0
12.01.2004	25.02.2004	1-й поставщик запчасте	13891880	12832329,0078	0
50110201	317	10254012		1059550,9922	0
			190 117 629,46	175 618 976,87	

Сейчас подробнее остановимся на автоматическом варианте формирования подготовительных шагов для книги покупок. Сначала нам необходимо сформировать проводки на привязку документов оплаты и поступления. Для этого служит стандартный механизм разноски по счетам, описанный выше. Используя этот механизм, создана конфигурация, позволяющая закрыть оплаты по 60 и 76 счетам, соответственно команды «Исследователь/Бухгалтерия/Книга покупок/Разноска по 60 счету» (76 счету). По данным командам выводится список не закрытых документов по соответствующим счетам и здесь можно либо по выбранным, либо по всем сформировать проводки закрытия.



Для автоматизации следующего шага, создана автоматическая операция «Формирование НДС оплаченного», операция работает со счетами, которые указаны в конфигурациях книги покупок. Данная автоматическая операция предполагает, проведение предыдущего шага и на основании его, производит пропорциональный перевод НДС.

Реализация

Данный раздел предназначен для формировании проводок по реализации, для методики учета «реализация по оплате». Используя указанные в данном разделе автоматические операции, будет сформирован кредит 90.01 счета, дебет 90.02 в разрезе себестоимости ТМЦ реализованных в текущем месяце, дебет 90.03 на сумму начисленного НДС, кредит 97.01 на сумму начисленного НДС по 60 дням. Данный набор операций предполагает, что проводки по отгрузке ТМЦ и услуг используются следующие на отпускную стоимость 62 Расчеты с покупателями 98.01 (специально введенный субсчет, который

отражает отгруженную продукцию в отпускных ценах) и на покупную стоимость 45.01 (Товары отгруженные) - 41.01 (Товары на складе) для отгруженных ТМЦ. Порядок выполнения операций:

01. Разноска по покупателям. Конфигурация разноски по счетам, предназначенная для закрытия документов оплаты от покупателей на документы отгрузки покупателям (товаров и услуг). По 62 счету с группировкой по клиенту и документу.

01. Формирование реализации. Автоматическая операция, формирующая кредит 90.01 счета, дебет 90.02 в разрезе себестоимости покупных ТМЦ, реализованных за период, дебет 90.03 счета на сумму начисленного НДС. При начислении НДС проверяется, не был ли он начислен ранее по истечении 60 дней.

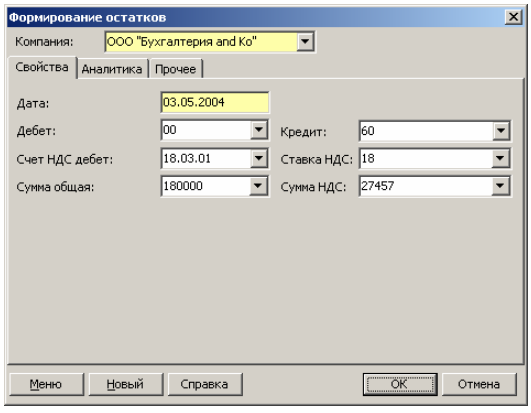
02. Формирование НДС 60 дней. Автоматическая операция, формирует проводку 97.01 (НДС 60 дней) – 68.02.01 (Налог на добавленную стоимость) на сумму начисленного НДС по документам, по которым имеется сальдо на конец периода, и дата которых <= Дата конца периода – 60. По 97.01 ведется аналитический учет по Клиент, Документ, Тип НДС.

03. Зачет НДС. Автоматическая операция формирует проводку 68.02.01 (Налог на добавленную стоимость) – 18.xx.02 (НДС уплаченный) на сумму дебетового сальдо по 18.xx.02, но не более чем кредитовое сальдо по 68.02.01. Данная операция приведена в примере создания автоматических операций.

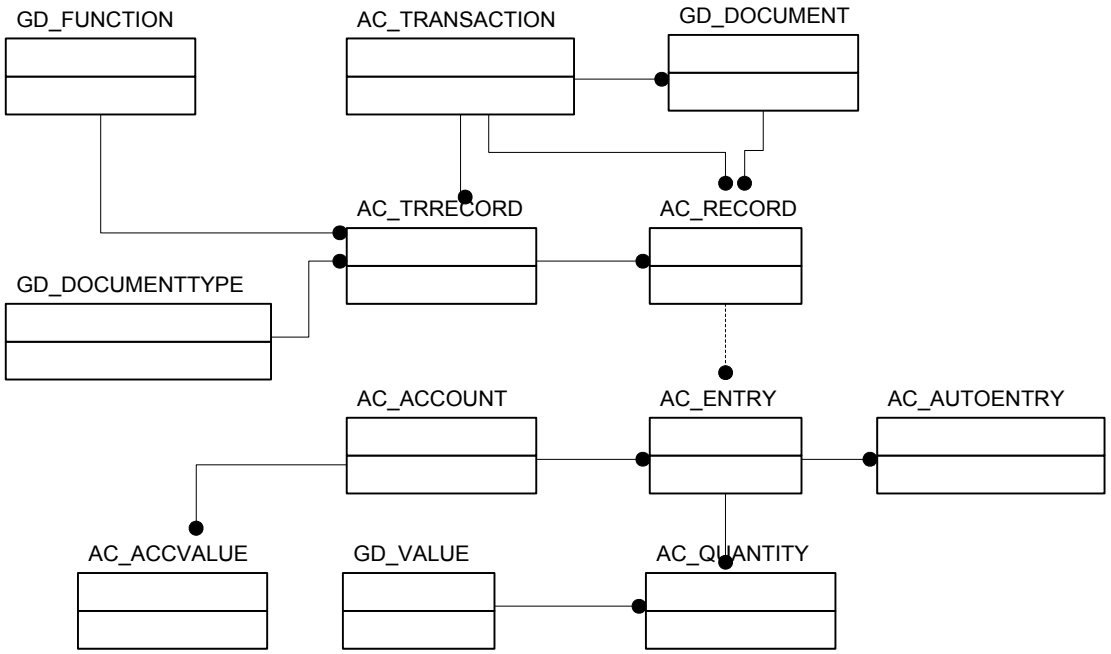
04. Финансовый результат. Автоматическая операция формирует проводку 90.09 – 99 на сумму итоговой прибыли за месяц или 99 – 90.09 на сумму убытка. Операция формируется после всех предыдущих автоматических операций и после расчета всех налогов.

Внесение начальных остатков.

После всей настройки программы, следующим этапом идет внесение начальных остатков по регистрам учета (счета, субсчета, аналитика, количественные показатели). В общем случае остатки могут вноситься в журнале хозяйственных операций, как ручные проводки, где по дебету или по кредиту указывается необходимый счет (субсчет) а в корреспонденции указывается счет «00», заполняется необходимая аналитика и если есть количественные показатели. Однако для корректного ввода остатков по счетам расчета с поставщиками (покупателями), особенного когда необходимо знать не только долги по ним, но и полученный (начисленный) НДС (например для корректного построения книги покупок) рекомендуется использовать отдельный режим «Формирование остатков». В данном режиме указывается счет по дебету или кредиту, при необходимости можно указать счет НДС по дебету, ставка НДС, сумма по счету (дебет, кредит), автоматически будет рассчитана сумма НДС. На закладке аналитика, выбирается необходимая аналитика по счету (счету НДС). При этом аналитике документ будет присвоена ссылка на данный документ, причем для счета и счета НДС это будет одна и та же аналитика.



Структуры данных используемые для хранения бухгалтерской информации в программе



Домены

DACCOUNTACTIVITY – для определения типа счета (Активный, Пассивный, Активно-Пассивный)

```
CREATE DOMAIN daccountactivity
  AS VARCHAR(1)
  CHECK ((VALUE IS NULL) OR (VALUE = 'A') OR (VALUE = 'P') OR (VALUE =
'B'));
```

DCHARTOFACCOUNTPART – для определения типа записи в таблице плана счетов (План счетов, Раздел, Счет, Субсчет)

```
CREATE DOMAIN dchartofaccountpart
  AS VARCHAR(1)
  CHECK ((VALUE = 'C') OR (VALUE = 'F') OR (VALUE = 'A') OR (VALUE = 'S'));
```

DACCOUNTPART – Тип части проводки (дебет или кредит)

```
CREATE DOMAIN daccountpart
  AS VARCHAR(1)
  CHECK ((VALUE = 'D') OR (VALUE = 'C'));
```

Таблицы

AC_ACCOUNT

План счетов. Таблица структуры интервальное дерево. Хранится информация о счетах, субсчетах, разделах и собственно планах счетов. При добавлении аналитики в таблицу добавляются поля с тем же названием, что и в таблице AC_ENTRY, но с типом «DBOOLEAN» и служат они для установки флага по счетам и субсчетам, используется данная аналитика или нет. Для активно-пассивного счета обязательно заполнение поля «ANALITICALFIELD», которое требуется для расчета расширенного сальдо.

Наименование поля	Тип поля	На что ссылается	Комментарий
ID	Dintkey		Индентификатор записи
PARENT	DPARENT	AC_ACCOUNT	Ссылка на родительскую

			ветку
LB	DLB		Левая граница
RB	DRB		Права граница
NAME	DTEXT180		Расшифровка счета
ALIAS	DACCOUNTALIAS		Номер счета
ANALYTICALFIELD	DFOREIGNKEY	AT_RELATION_FIELDS	Поле главной аналитики (для активно-пассивных счетов)
ACTIVITY	DACCOUNTACTIVITY		Тип активности счета A – Активный P – Пассивный B – Активно-пассивный
ACCOUNTTYPE	DCHARTOFACCOUNTPART		Тип записи C – План счетов F – Раздел A – Счет S – Субсчет
MULTYCURR	DBOOLEAN DEFAULT 0		Признак многовалютного счета (0 – нет, 1 – да)
OFFBALANCE	DBOOLEAN DEFAULT 0		Признак забалансового счета (0 – нет, 1 – да)
AFULL	DSECURITY		Дескриптор безопастности Полный доступ
ACHAG	DSECURITY		Просмотр и изменение
AVIEW	DSECURITY		Только просмотр
FULLNAME	COMPUTED BY (IIF(ALIAS IS NULL, '', ALIAS || ' ') || IIF(NAME IS NULL, '', NAME))		Полное наименование (Номер счета + Расшифровка)
DESCRIPTION	DBLOBTEXT80_1251		Описание счета
DISABLED	DBOOLEAN DEFAULT 0		Признак отключенного счета (0 – нет, 1

			– да)
RESERVED	DINTEGER		Зарезервировано

AC_ACCVALUE

Единицы измерения по счетам. Таблица множества содержит пересечения счетов и единиц измерения

Наименование поля	Тип поля	На что ссылается	Комментарий
ID	DINTKEY		Индентификатор записи
ACCOUNTKEY	DMASTERKEY	AC_ACCOUNT	Ссылка на счет по которому Устанавливается количественный показатель
VALUEKEY	DINTKEY	GD_VALUE	Ссылка на единицу измерения

AC_COMPANYACCOUNT

План счетов для организации. Таблица множества содержит пересечения планов счетов и рабочих организаций. Если план счетов не указан не для одной организации он не доступен для работы с ним.

Наименование поля	Тип поля	На что ссылается	Комментарий
COMPANYKEY	DINTKEY	GD_OURCOMPANY	Ссылка на рабочую организацию
ACCOUNTKEY	DINTKEY	AC_ACCOUNT	Ссылка на план-счетов
ISACTIVE	DBOOLEAN DEFAULT 0		Признак активности плана счетов для организации (1 – Да, 0 – Нет)
RESERVED	DINTEGER		Зарезервировано

AC_TRANSACTION

Список типовых операций. Таблица структуры интервальное дерево. Для текущей организации отображаются только те операции, у которых поле «companykey» не задано или совпадает с идентификатором рабочей организации.

Наименование поля	Тип поля	На что ссылается	Комментарий
ID	DINTKEY		Индентификатор записи
PARENT	DPARENT	AC_TRANSACTION	Ссылка на родительскую ветвь
LB	DLB		Левая граница
RB	DRB		Правая граница
NAME	DNAME		Наименование операции
DESCRIPTION	DTEXT180		Описание операции
COMPANYKEY	DFOREIGNKEY	GD_COMPANY	Ссылка на организацию, для которой используется данная операция

AFULL, ACHAG, AVIEW	DSECURITY		Дескрипторы безопасности
DISABLED	DBOOLEAN DEFAULT 0		Отключено
RESERVED	DINTEGER		Зарезервировано
AUTOTRANSACTION	DBOOLEAN		Признак автоматической операции

AC_TRRECORD

Заголовок типовой проводки. Хранит функцию которая формирует проводки по документам. При формировании проводок доступны только те счета, которые относятся к выбранному плану счетов.

Наименование поля	Тип поля	На что ссылается	Комментарий
ID	DINTKEY		Индентификатор записи
TRANSACTIONKEY	DMASTERKEY	AC_TRANSACTION	Ссылка на типовую операцию
DESCRIPTION	DTEXT180		Описание проводки
ISSAVENULL	DBOOLEAN		Признак, сохранять проводку с нулевой суммой или нет.
ACCOUNTKEY	DFOREIGNKEY	AC_ACCOUNT	Ссылка на план счетов. При формировании типовой проводки будут доступны счета только из этого плана счетов
AFULL, ACHAG, AVIEW	DSECURITY		Дескрипторы безопасности
DISABLED	DBOOLEAN DEFAULT 0		Отключено
RESERVED	DINTEGER		Зарезервировано
FUNCTIONKEY	DFOREIGNKEY	GD_FUNCTION	Ссылка на функцию, которая формирует проводку
DOCUMENTTYPEKEY	DFOREIGNKEY	GD_DOCUMENTTYPE	Ссылка на типовой документ, по которому будет формироваться проводка
FUNCTIONTEMPLATE	DBLOB80		Шаблон функции (в формате конструктора функций)
DOCUMENTPART	DTEXT10		Часть документа (шапка или позиция) по которой будет формироваться проводка

AC_RECORD

Заголовок проводки. Хранит общую информацию по проводке.

Наименование поля	Тип поля	На что ссылается	Комментарий
ID	DINTKEY		Индентификатор записи
TRRECORDKEY	DINTKEY	AC_TRRECORD	Ссылка на типовую проводку, на основании которой была создана данная проводка
TRANSACTIONKEY	DINTKEY	AC_TRANSACTION	Ссылка на типовую операцию, к которой относится данная проводка
RECORDDATE	DDATE NOT NULL		Дата проводки
DESCRIPTION	DTEXT180		Описание проводки
DOCUMENTKEY	DMASTERKEY	GD_DOCUMENT	Ссылка на документ по которому создана проводка (шапка или позиция)
MASTERDOCKEY	DINTKEY	GD_DOCUMENT	Ссылка на шапку документа, по которому создана проводка
COMPANYKEY	DINTKEY	GD_OURCOMPANY	Ссылка на рабочую компанию, для которой создана данная проводка
DEBITNCU	DCURRENCY		Сумма проводки по дебету в национальной денежной единице
CREDITNCU	DCURRENCY		Сумма проводки по кредиту в национальной денежной единице
DEBITCURR	DCURRENCY		Сумма проводки по дебету в валюте
CREDITCURR	DCURRENCY		Сумма проводки по кредиту в валюте
Delayed	dboolean DEFAULT 0		Признак отложенной проводки (Не испотзуется)
INCORRECT	DBOOLEAN DEFAULT 0		Признак не корректности проводки (Если сумма по дебету <> сумме по кредиту)
AFULL, ACHAG, AVIEW	DSECURITY		Дескрипторы безопасности
DISABLED	DBOOLEAN DEFAULT 0		Отключено
RESERVED	DINTEGER		Зарезервировано

AC_ENTRY – Строки проводки

Наименование поля	Тип поля	На что ссылается	Комментарий
ID	DINTKEY		Индентификатор записи
RECORDKEY	DMASTERKEY	AC_RECORD	Ссылка на заголовок проводки
ENTRYDATE	DDATE		Дата проводки = Дате проводки в заголовке проводки добавлена для скорости при построении отчетов
ACCOUNTKEY	DINTKEY	AC_ACCOUNT	Ссылка на балансовый счет
ACCOUNTPART	DACCOUNTPART		Тип части проводки (дебет или кредит)
DEBITNCU	DCURRENCY		Сумма по дебету в НДЕ
DEBITCURR	DCURRENCY		Сумма по дебету в вал
DEBITEQ	DCURRENCY		Сумма по дебету в экв
CREDITNCU	DCURRENCY		Сумма по кредиту в НДЕ
CREDITCURR	DCURRENCY		Сумма по кредиту в вал
CREDITEQ	DCURRENCY		Сумма по кредиту в экв
CURRKEY	DINTKEY	GD_CURR	Ссылка на валюту проводки
DISABLED	DBOOLEAN DEFAULT 0		Отключено
RESERVED	DINTEGER		Зарезервировано
ISSIMPLE	DBOOLEAN_NOTNULL		Признак простой части проводки (1 – да, 0 нет)

AC_QUANTITY – Количественные показатели по проводкам

Наименование поля	Тип поля	На что ссылается	Комментарий
ID	DINTKEY		Идентификатор записи
ENTRYKEY	DMASTERKEY	AC_ENTRY	Ссылка на позицию проводки, по которой хранится количественный показатель
VALUEKEY	DINTKEY	GD_VALUE	Ссылка на единицу измерения
QUANTITY	DCURRENCY		Количество

AC_AUTOENTRY – Ссылка на проводки сформированные автоматически

Наименование поля	Тип поля	На что ссылается	Комментарий
ID	DINTKEY		Идентификатор записи
ENTRYKEY	DINTKEY	AC_ENTRY	Ссылка на позицию проводки
TRRECORDKEY	DINTKEY	AC_TRRECORD	Ссылка на типовую проводку
BEGINDATE, ENDDATE	DDATE_NOTNULL NOT NULL		Период за который формировались проводки
CREDITACCOUNT	DINTKEY	AC_ACCOUNT	Ссылка на балансовый счет (счет по кредиту проводки)
DEBITACCOUNT	DINTKEY	AC_ACCOUNT	Ссылка на балансовый счет (счет по дебету проводки)

Структуры данных используемые для стандартных бухгалтерских отчетов

Таблицы

AC_ACCT_CONFIG – Настройки карты счета, журнал-ордер, оборотная ведомость, главная книга

Наименование поля	Тип поля	На что ссылается	Комментарий
ID	DINTKEY		Идентификатор записи
NAME	DTEXT32 NOT NULL		Наименование конфигурации
CLASSNAME	DTEXT60 NOT NULL		Класс, отвечающий за данную конфигурацию
CONFIG	DBLOB		Конфигурация
SHOWINEXPLORER	DBOOLEAN		Отображать в исследователе
FOLDER	DFOREIGNKEY	GD_COMMAND	Папка, в которой отображать
IMAGEINDEX	DINTEGER		Номер картинки, которая будет отображаться в исследователе

AC_LEDGER_ACCOUNTS – Выбранные счета для журнала ордера, карты счета, используется хранимыми процедурами для получения информацию о наборе счетов, по которым строятся данные отчеты

Наименование	Тип поля	На что	Комментарий

поля		ссылается	
ACCOUNTKEY	DFOREIGNKEY NOT NULL	AC_ACCOUNT	Ссылка на счет из плана счетов
SQLHANDLE	DINTEGER_NOTNULL NOT NULL		Идентификатора текущего (выполняемого) отчета

AC_LEDGER_ENTRIES – Список проводок, удовлетворяющий установленным дополнительным ограничениям по аналитике, в случае если журнал ордер строится по дате и установлены ограничения по аналитике

Наименование поля	Тип поля	На что ссылается	Комментарий
ENTRYKEY	DINTKEY	AC_ENTRY	Ссылка на позицию проводки
SQLHANDLE	DINTEGER_NOTNULL NOT NULL		Идентификатора текущего (выполняемого) отчета

Хранимые процедуры

AC_CIRCULATIONLIST – Процедура для построения оборотной ведомости

```
CREATE PROCEDURE AC_CIRCULATIONLIST (
    DATEBEGIN DATE, /* Дата начала периода */
    DATEEND DATE,   /* Дата окончания периода */
    COMPANYKEY INTEGER, /* Идентификатор рабочей компании */
    ALLHOLDINGCOMPANIES INTEGER, /* Признак включать все компании холдинга
*/
    ACCOUNTKEY INTEGER, /* План счетов */
    INGROUP INTEGER, /* Права доступа */
    CURRKEY INTEGER) /* Идентификатор валюты */
RETURNS (
    ALIAS VARCHAR(20), /* Номер счета */
    NAME VARCHAR(180), /* Наименование счета */
    ID INTEGER,
    NCU_BEGIN_DEBIT NUMERIC(15,4), /* Сальдо на начало дебетовое */
    NCU_BEGIN_CREDIT NUMERIC(15,4),/* Сальдо на начало кредитовое */
    NCU_DEBIT NUMERIC(15,4),       /* Оборот дебетовый */
    NCU_CREDIT NUMERIC(15,4),      /* Оборот кредитовый */
    NCU_END_DEBIT NUMERIC(15,4),   /* Сальдо на конец дебетовое */
    NCU_END_CREDIT NUMERIC(15,4),  /* Сальдо на конец кредитовое */
CURR_BEGIN_DEBIT NUMERIC(15,4),/*Сальдо на начало дебетовое вал */
CURR_BEGIN_CREDIT NUMERIC(15,4),/*Сальдо на начало кредитовое вал */
    CURR_DEBIT NUMERIC(15,4), /* Оборот дебетовый валютный */
    CURR_CREDIT NUMERIC(15,4), /* Оборот кредитовый валютный */
    CURR_END_DEBIT NUMERIC(15,4), /* Сальдо на конец дебетовое вал */
    CURR_END_CREDIT NUMERIC(15,4))/* Сальдо на конец кредитовое вал */
```

AC_Q_CIRCULATION – Процедура для построения оборотной ведомости с количественными показателями

```
CREATE PROCEDURE AC_Q_CIRCULATION (
    VALUEKEY INTEGER, /* Идентификатор ед.измерения(кол.показатель) */
    DATEBEGIN DATE, /* Дата начала периода */
    DATEEND DATE, /* Дата окончания периода */
    COMPANYKEY INTEGER, /* Идентификатор рабочей компании */
```

```
    ALLHOLDINGCOMPANIES INTEGER, /* Признак включать все компании холдинга
*/
    ACCOUNTKEY INTEGER, /* идентификатор счета из плана счетов */
    INGROUP INTEGER, /* права доступа */
    CURRKEY INTEGER) /* Идентификатор валюты */
RETURNS (
    ID INTEGER,
    DEBITBEGIN NUMERIC(15,4), /* Сальдо на начало дебетовое */
    CREDITBEGIN NUMERIC(15,4),/* Сальдо на конец кредитовое */
    DEBIT NUMERIC(15,4),  /* Оборот дебетовый */
    CREDIT NUMERIC(15,4), /* Оборот кредитовый */
    DEBITEND NUMERIC(15,4),  /* Сальдо на конец дебетовое */
    CREDITEND NUMERIC(15,4)) /* Сальдо на конец кредитовое */
```

AC_ACCOUNTEXSALDO – Процедура для расчета расширенного сальдо в оборотной ведомости. Тело данной процедуры формируется автоматически при добавлении нового вида аналитики.

```
CREATE PROCEDURE AC_ACCOUNTEXSALDO (
    DATEEND DATE, /* Дата на которую рассчитывается сальдо */
    ACCOUNTKEY INTEGER, /* Идентификатор счет из плана счетов, по которому
рассчитывается сальдо */
    FIELDNAME VARCHAR(60), /* Поле, отвечающее за главную аналитику */
    COMPANYKEY INTEGER, /* Идентификатор рабочей компании */
    ALLHOLDINGCOMPANIES INTEGER, /* Признак включать все компании холдинга
*/
    INGROUP INTEGER, /* Права доступа */
    CURRKEY INTEGER) /* Идентификатор валюты */
RETURNS
    (DEBITSALDO NUMERIC(15,4),
    CREDITSALDO NUMERIC(15,4),
    CURRDEBITSALDO NUMERIC(15,4),
    CURRCREDITSALDO NUMERIC(15,4))
```

AC_GETSIMPLEENTRY – Процедура возвращает сумму проводки по заданной строке проводки и корреспондирующему счету. Используется при построении журнала-ордера, главной книги, оборотной ведомости

```
CREATE PROCEDURE AC_GETSIMPLEENTRY (
    ENTRYKEY INTEGER,  /* Идентификатор строки проводки */
    ACORRACCOUNTKEY INTEGER) /* Корреспондирующий счет к счету в строке
проводки */
RETURNS (
    ID INTEGER, /* Идентификатор строки проводки = entrykey */
    DEBIT NUMERIC(15,4), /* Сумма по дебету */
    CREDIT NUMERIC(15,4), /* Сумма по кредитку */
    DEBITCURR NUMERIC(15,4), /* Сумма по дебету в вал. */
    CREDITCURR NUMERIC(15,4)) /* Сумма по кредиту в вал. */
```

AC_L_S – Процедура возвращает список сальдо по датам. Используется при построении журнала ордера, где в качестве группировки используется дата

```
CREATE PROCEDURE AC_L_S (
    ABEGINENTRYDATE DATE, /* Дата начала периода */
    AENDENTRYDATE DATE,   /* Дата окончания периода */
    SQLHANDLE INTEGER,    /* Идентификатор отчета */
    COMPANYKEY INTEGER,   /* Идентификатор рабочей компании */
    ALLHOLDINGCOMPANIES INTEGER, /* Признак включать все компании холдинга
*/
    INGROUP INTEGER, /* Права доступа */
    CURRKEY INTEGER) /* Идентификатор валюты */
RETURNS (
```

```
    ENTRYDATE DATE, /* Дата строки отчета */
    DEBITNCUBEGIN NUMERIC(15,4), /* Сальдо дебетовое на ENTRYDATE-1 */
    CREDITNCUBEGIN NUMERIC(15,4),/* Сальдо кредитовое на ENTRYDATE-1*/
    DEBITNCUEND NUMERIC(15,4),  /* Сальдо дебетовое на ENTRYDATE */
    CREDITNCUEND NUMERIC(15,4), /* Сальдо кредитовое на ENTRYDATE */
    DEBITCURRBEGIN NUMERIC(15,4),/*Сальдо дебетовое вал ENTRYDATE-1 */
    CREDITCURRBEGIN NUMERIC(15,4),/*Сальдо кредитовое вал ENTRYDATE-1*/
    DEBITCURREND NUMERIC(15,4), /* Сальдо дебетовое вал ENTRYDATE */
    CREDITCURREND NUMERIC(15,4), /* Сальдо кредитовое вал ENTRYDATE */
    FORCESHOW INTEGER) /* Признак отсутствия движения на дату */
```

AC_L_S1 – Процедура возвращает список сальдо по укрупненным датам (Месяц, квартал, год). Используется при построении журнала ордера, где в качестве группировки используется месяц, квартал, год

```
CREATE PROCEDURE AC_L_S1 (
    ABEGINENTRYDATE DATE, /* Дата начала периода */
    AENDENTRYDATE DATE,   /* Дата окончания периода */
    SQLHANDLE INTEGER,    /* Идентификатор отчета */
    COMPANYKEY INTEGER,   /* Идентификатор рабочей компании */
    ALLHOLDINGCOMPANIES INTEGER, /* Признак включать все компании холдинга
*/
    INGROUP INTEGER,      /* Права доступа */
    PARAM INTEGER,        /* Тип временного показателя 1 – Месяц, 2 – Год,
4 – Квартал */
    CURRKEY INTEGER)      /* Идентификатор валюты */
RETURNS (
    DATEPARAM INTEGER,    /* Номер месяца (квартала, года) */
    DEBITNCUBEGIN NUMERIC(15,4),/* Сальдо дебетовое на начало DATEPARAM */
    CREDITNCUBEGIN NUMERIC(15,4),/* Сальдо кредитовое на начало */
    DEBITNCUEND NUMERIC(15,4), /* Сальдо конечное дебетовое */
    CREDITNCUEND NUMERIC(15,4), /* Сальдо конечное  кредитовое */
    DEBITCURRBEGIN NUMERIC(15,4),/*Сальдо дебетовое вал на начало */
    CREDITCURRBEGIN NUMERIC(15,4),/*Сальдо кредитовое вал на начало */
    DEBITCURREND NUMERIC(15,4), /*Сальдо конечное дебетовое вал */
    CREDITCURREND NUMERIC(15,4), /*Сальдо конечное кредитовое вал */
    FORCESHOW INTEGER) /* Признак отсутствия движения на дату */
```

AC_Q_S – Процедура возвращает список сальдо и обороты по датам по количественному показателю. Используется при построении журнала ордера по количественному показателю, где в качестве группировки используется дата

```
CREATE PROCEDURE AC_Q_S (
    VALUEKEY INTEGER,   /* Идентификатор ед.изм. */
    ABEGINENTRYDATE DATE, /* Дата начала периода */
    AENDENTRYDATE DATE,   /* Дата окончания периода */
    SQLHANDLE INTEGER,    /* Идентификатор отчета */
    COMPANYKEY INTEGER,   /* Идентификатор рабочей компании */
    ALLHOLDINGCOMPANIES INTEGER, /* Признак включения всех компаний
холдинга */
    INGROUP INTEGER, /* Права доступа */
    CURRKEY INTEGER) /* Идентификатор валюты */
RETURNS (
    ENTRYDATE DATE,  /* Дата позиции отчета */
    DEBITBEGIN NUMERIC(15,4), /* Сальдо дебетовое на ENTRYDATE – 1 */
    CREDITBEGIN NUMERIC(15,4), /* Сальдо кредитовое на ENTRYDATE – 1*/
    DEBIT NUMERIC(15,4), /* Оборот дебетовый за ENTRYDATE */
    CREDIT NUMERIC(15,4), /* Оборот кредитовый за ENTRYDATE */
    DEBITEND NUMERIC(15,4), /* Сальдо дебетовое на ENTRYDATE */
    CREDITEND NUMERIC(15,4)) /* Сальдо кредитовое на ENTRYDATE */
```

AC_Q_S1 – Процедура возвращает список сальдо по количественному показателю по укрупненным датам (Месяц, квартал, год). Используется при построении журнала ордера, где в качестве группировки используется месяц, квартал, год.

```
CREATE PROCEDURE AC_Q_S1 (
    VALUEKEY INTEGER,     /* Идентификатор единицы измерения */
    ABEGINENTRYDATE DATE, /* Дата начала периода */
    AENDENTRYDATE DATE,   /* Дата окончания периода */
    SQLHANDLE INTEGER,    /* Идентификатор отчета */
    COMPANYKEY INTEGER,  /* Идентификатор рабочей компании */
    ALLHOLDINGCOMPANIES INTEGER, /* Признак включать все компании холдинга
*/
    INGROUP INTEGER, /* Права доступа */
    PARAM INTEGER,   /* Тип временного показателя 1 – Месяц, 2 – Год, 4 –
Квартал */
    CURRKEY INTEGER) /* Идентификатор валюты */
RETURNS (
    DATEPARAM INTEGER, /* Номер месяца (квартала, года) */
    DEBITBEGIN NUMERIC(15,4), /* Сальдо дебетовое на начало */
    CREDITBEGIN NUMERIC(15,4), /* Сальдо кредитовое на начало */
    DEBIT NUMERIC(15,4), /* Оборот дебетовый */
    CREDIT NUMERIC(15,4), /* Оборот кредитовый */
    DEBITEND NUMERIC(15,4), /* Сальдо конечное дебетовое */
    CREDITEND NUMERIC(15,4)) /* Сальдо конечное кредитовое */
```

AC_L_Q – Возвращает значение количественного показателя по проводке

```
CREATE PROCEDURE AC_L_Q (
    ENTRYKEY INTEGER,  /* Идентификатор строки проводки */
    VALUEKEY INTEGER,  /* Идентификатор единицы измерения */
    ACCOUNTKEY INTEGER, /* Идентификатор счета */
    AACCOUNTPART VARCHAR(1)) /* Часть проводки D – Дебетовая, C –
Кредитовая, по которой необходимо получить значение количественного
показателя */
RETURNS (
    DEBITQUANTITY NUMERIC(15,4), /* Количество по дебету */
    CREDITQUANTITY NUMERIC(15,4)) /* Количество по кредиту */
```

AC_E_L_S – Процедура возвращает список сальдо по датам. Используется при построении журнала ордера, где в качестве группировки используется дата и задано хотя бы одно ограничение по аналитике.

```
CREATE PROCEDURE AC_E_L_S (
    ABEGINENTRYDATE DATE, /* Дата начала построения отчета */
    SALDOBEGIN NUMERIC(18,4), /* Сальдо на начало */
    SALDOBEGINCURR NUMERIC(18,4), /* Сальдо на начало в вал */
    SQLHANDLE INTEGER, /* Идентификатор текущего отчета */
    CURRKEY INTEGER)   /* Идентификатор валюты */
RETURNS (
    ENTRYDATE DATE, /* Дата строки отчета */
    DEBITNCUBEGIN NUMERIC(15,4), /* Сальдо дебетовое на ENTRYDATE-1 */
    CREDITNCUBEGIN NUMERIC(15,4),/* Сальдо кредитовое на ENTRYDATE-1*/
    DEBITNCUEND NUMERIC(15,4),  /* Сальдо дебетовое на ENTRYDATE */
    CREDITNCUEND NUMERIC(15,4), /* Сальдо кредитовое на ENTRYDATE */
    DEBITCURRBEGIN NUMERIC(15,4),/*Сальдо дебетовое вал ENTRYDATE-1 */
    CREDITCURRBEGIN NUMERIC(15,4),/*Сальдо кредитовое вал ENTRYDATE-1*/
    DEBITCURREND NUMERIC(15,4), /* Сальдо дебетовое вал ENTRYDATE */
    CREDITCURREND NUMERIC(15,4), /* Сальдо кредитовое вал ENTRYDATE */
    FORCESHOW INTEGER) /* Признак отсутствия движения на дату */
```

AC_E_L_S1 – Процедура возвращает список сальдо по укрупненным датам (Месяц, квартал, год). Используется при построении журнала ордера, где в качестве группировки используется месяц, квартал, год и задано хотя бы одно ограничение по аналитике

```
CREATE PROCEDURE AC_E_L_S1 (
    ABEGINENTRYDATE DATE, /* Дата начала построения отчета */
    SALDOBEGIN NUMERIC(18,4), /* Сальдо на начало */
    SALDOBEGINCURR NUMERIC(18,4), /* Сальдо на начало в вал */
    SQLHANDLE INTEGER, /* Идентификатор текущего отчета */
    PARAM INTEGER,   /* Тип временного показателя 1 – Месяц, 2 – Год, 4 –
Квартал */

    CURRKEY INTEGER)   /* Идентификатор валюты */
RETURNS (
    DATEPARAM INTEGER,    /* Номер месяца (квартала, года) */
    DEBITNCUBEGIN NUMERIC(15,4),/* Сальдо дебетовое на начало  */
    CREDITNCUBEGIN NUMERIC(15,4),/* Сальдо кредитовое на начало */
    DEBITNCUEND NUMERIC(15,4), /* Сальдо конечное дебетовое */
    CREDITNCUEND NUMERIC(15,4), /* Сальдо конечное  кредитовое */
    DEBITCURRBEGIN NUMERIC(15,4),/*Сальдо дебетовое вал на начало */
    CREDITCURRBEGIN NUMERIC(15,4),/*Сальдо кредитовое вал на начало */
    DEBITCURREND NUMERIC(15,4), /*Сальдо конечное дебетовое вал */
    CREDITCURREND NUMERIC(15,4), /*Сальдо конечное кредитовое вал */
    FORCESHOW INTEGER) /* Признак отсутствия движения на дату */
```

AC_E_Q_S – Процедура возвращает список сальдо по датам по количественному показателю. Используется при построении журнала ордера по количественному показателю, где в качестве группировки используется дата и задано хотя бы одно ограничение по аналитике

```
CREATE PROCEDURE AC_E_Q_S (
    VALUEKEY INTEGER,         /* Идентификатор единицы измерения. */
    ABEGINENTRYDATE DATE,     /* Дата начала периода */
    SALDOBEGIN NUMERIC(15,4), /* Сальдо на начало */
    SQLHANDLE INTEGER,        /* Идентификатор отчета */
    CURRKEY INTEGER)          /* Идентификатор валюты */
RETURNS (
    ENTRYDATE DATE,  /* Дата позиции отчета */
    DEBITBEGIN NUMERIC(15,4), /* Сальдо дебетовое на ENTRYDATE – 1 */
    CREDITBEGIN NUMERIC(15,4), /* Сальдо кредитовое на ENTRYDATE – 1*/
    DEBIT NUMERIC(15,4), /* Оборот дебетовый за ENTRYDATE */
    CREDIT NUMERIC(15,4), /* Оборот кредитовый за ENTRYDATE */
    DEBITEND NUMERIC(15,4), /* Сальдо дебетовое на ENTRYDATE */
    CREDITEND NUMERIC(15,4)) /* Сальдо кредитовое на ENTRYDATE */
```

AC_E_Q_S1 – Процедура возвращает список сальдо по количественному показателю по укрупненным датам (Месяц, квартал, год). Используется при построении журнала ордера, где в качестве группировки используется месяц, квартал, год и задано хотя бы одно ограничение по аналитике

```
CREATE PROCEDURE AC_E_Q_S1 (
    VALUEKEY INTEGER,         /* Идентификатор единицы измерения. */
    ABEGINENTRYDATE DATE,     /* Дата начала периода */
    SALDOBEGIN NUMERIC(15,4), /* Сальдо на начало */
    SQLHANDLE INTEGER,        /* Идентификатор отчета */
    PARAM INTEGER,   /* Тип временного показателя 1 – Месяц, 2 – Год, 4 –
Квартал */
    CURRKEY INTEGER)          /* Идентификатор валюты */
RETURNS (
    DATEPARAM INTEGER, /* Номер месяца (квартала, года) */
    DEBITBEGIN NUMERIC(15,4), /* Сальдо дебетовое на начало */
    CREDITBEGIN NUMERIC(15,4), /* Сальдо кредитовое на начало */
    DEBIT NUMERIC(15,4), /* Оборот дебетовый */
```

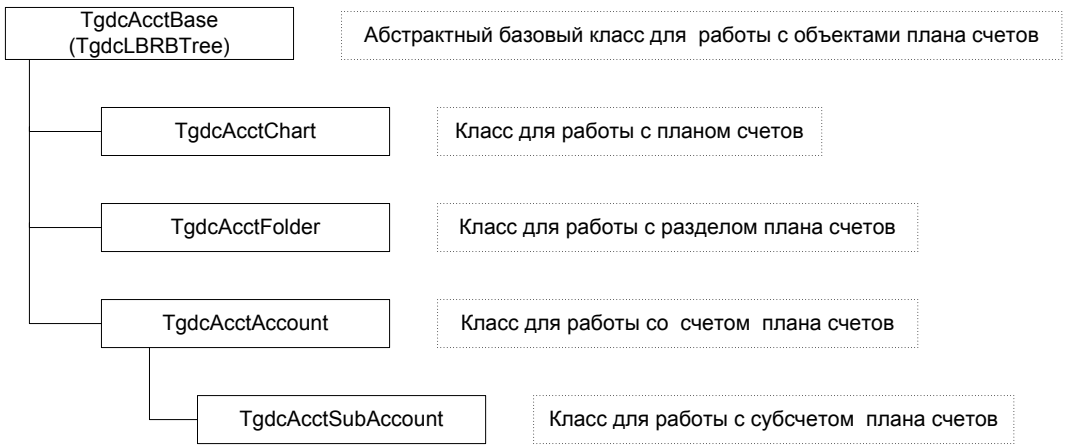
```
    CREDIT NUMERIC(15,4), /* Оборот кредитовый */
    DEBITEND NUMERIC(15,4), /* Сальдо конечное дебетовое */
    CREDITEND NUMERIC(15,4)) /* Сальдо конечное кредитовое */
```

Объекты и формы, используемые для работы с бухгалтерскими данными:

Бизнес объекты

Объекты бухгалтерии, какими бизнес объектами они описываются

План счетов



Для просмотра плана счетов, с разделами, счетами и субсчетами используется следующий класс формы просмотра

Tgdc_frmAcctAccount (Tgdc_frmMDVTree).

Для редактирования и добавления, следующие классы диалоговых окон:

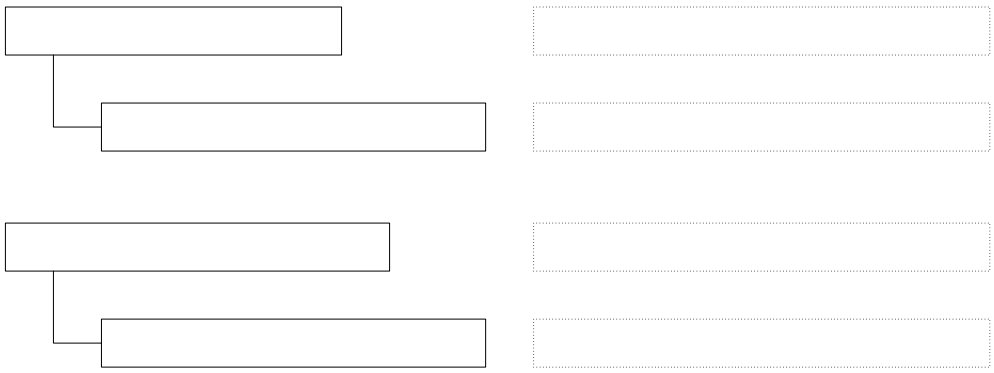
Tgdc_dlgAcctChart (Tgdc_dlgG) – План счетов,

Tgdc_dlgAcctFolder (Tgdc_dlgG) – Раздел плана счетов,

Tgdc_dlgAcctAccount(Tgdc_dlgAcctBaseAccount-Tgdc_dlgTRPC) – Счет,

Tgdc_dlgAcctSubAccount(Tgdc_dlgAcctBaseAccount-Tgdc_dlgTRPC) – Субсчет.

Типовые операции



Для просмотра типовых операций используется следующий класс формы просмотра

Tgdc_frmAcctTransaction (Tgdc_frmMDVTree)

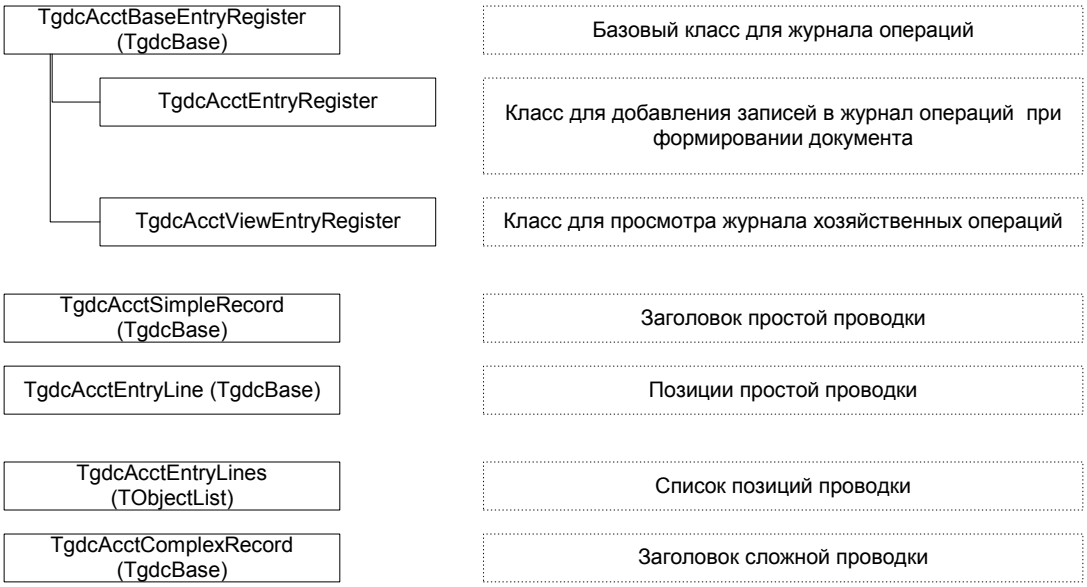
Для редактирования и добавления используются следующие классы диалоговых окон

Tgdc_dlgAcctTransaction (Tgdc_dlgTR) – Типовая операция

Tgdc_dlgAcctTrEntry (Tgdc_dlgTR) – Типовая проводка

Журнал хозяйственных операций;

Для работы с журналом хозяйственных операций используется несколько различных классов, часть из них предназначена для отображения журнала и формирования проводок из документов, другая часть для формирования проводок из скрпитов и автоматических операций.




```
TgdcAcctBaseEntryRegister
    …
    property RecordKey: Integer    // Идентификатор заголовка проводки
    property Document: TgdcDocument // Объект для связи с документом,
создавшим проводку
    property Description: String   // Описание проводки
    property gdcQuantity: TgdcAcctQuantity // Объект для связи с
количественными показателями по проводке.

TgdcAcctEntryRegister – используется в классе TgdcDocument для формирования
проводки, объект данного класса передается в функцию формирования проводки
      …
procedure CreateEntry(const EmptyEntry: Boolean = False); // Процедура
создания проводки. Вызывается из TgdcDocument при добавлении или изменении
документа.

TgdcAcctViewEntryRegister – класс для просмотра журнала хозяйственных
операций. Для редактирования и добавления проводок из данного режима
используется класс TgdcAcctComplexRecord

TgdcAcctEntryLine – класс используется только вместе с
классом TgdcAcctSimpleRecord для определения позиции проводки
      …
    property AccountPart: String // Часть проводки D- Дебет/C- Кредит
    property RecordKey: Integer  // Идентификатор заголовка проводки
    property gdcQuantity: TgdcAcctQuantity // Объект для связи с
количественными показателями по проводке.

TgdcAcctSimpleRecord – класс используется при формировании проводок в
скриптах и автоматических операциях
      …
```

```
    property DebitEntryLine: TgdcAcctEntryLine // Позиция по дебету
проводки
    property CreditEntryLine: TgdcAcctEntryLine // Позиция по кредиту
проводки
    property Document: TgdcDocument // Документ по проводке
    property TransactionKey: Integer // Типовая операция по проводке

TAcctEntryLines = class(TObjectList) – класс используется для хранения
позиций проводки используется в классе TgdcAcctComplexRecord

  private
    function GetLines(Index: Integer): TgdcAcctEntryLine;
  public
    property Lines[Index: Integer]: TgdcAcctEntryLine read GetLines;
default;
  end;

TgdcAcctComplexRecord – класс используется при добавлении и редактировании
проводок в журнале хозяйственных операций
      …
    property EntryLines: TAcctEntryLines // список позиций проводки
    property Document: TgdcDocument // Документ привязанный к проводке
    property TransactionKey: Integer // Идентификатор типовой операции
```

Организация складского учета на Гедымине

Общие сведения

Складской учет на системе Гедымин строится на основании специального вида документов – складские документы, которые используют заложенные алгоритмы для учета товарно-материальных ценностей (ТМЦ). Учет ведется как качественный, так и количественный. По каждому ТМЦ определяется набор признаков, которые описывают их свойства, в разрезе всех установленных признаков ведется количественный учет по местам хранения (учета). Признаки делятся на постоянные (описывающие ТМЦ как физическую сущность) и переменные (связанные с документальным движением ТМЦ).

Справочник ТМЦ. (GD_GOOD)

Справочник товарно-материальных ценностей хранит информацию о постоянных признаках товара, а так же те переменные признаки товара, которые не зависят от документооборота предприятия. К постоянным признакам товара относятся: наименование, шифр, краткое наименование, описание товара, вес единицы товара, единица измерения и т.п. К переменным относятся ставки налогов на соответствующие товары, дополнительные единицы измерения.

Карточка ТМЦ. (INV_CARD)

Карточка ТМЦ хранит информацию о всех признаках ТМЦ, которые могут возникнуть в результате документооборота предприятия. В карточке ТМЦ обычно хранят те признаки, которые могут быть доступны в любом документе или в произвольном отчете. К признакам, которые хранятся в карточке, относятся: все виды цен на товары, поставщик, производитель, страна, виды упаковок, наценки и другие. Карточка ТМЦ создается при создании документа, и может быть изменена только при редактировании данного документа. Изменение любого признака товара в любом другом документе ведет обязательно к созданию новой карточки, в которую переносятся все неизмененные признаки. Каждая карточка знает, от какой карточки она произошла и хранит ссылку на документ, в котором появилась, документ в котором появилась самый первый родитель данной карточки, дата возникновения первого родителя.

Движение ТМЦ. (INV_MOVEMENT)

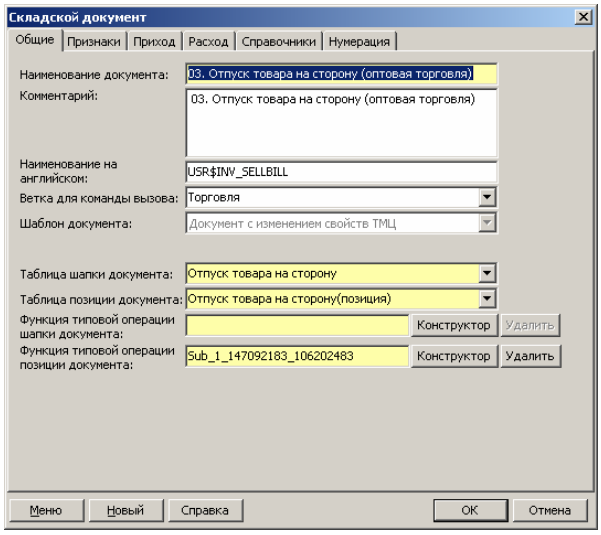
Все движение ТМЦ происходит под средством движения карточек. В системе существует единая таблица движения, которая хранит информацию, какая карточка ушла, какая пришла, с какого места, на какое, когда и по какому документу. Таблица движения формируется только из документа и без документа существовать не может. Таблица движения – основная таблица для получения всей отчетности по складскому учету, а также получения информации по остаткам, в любом разрезе на произвольную дату.

Остатки ТМЦ. (INV_BALANCE)

Как было описано ранее, основным источником для получения информации по остаткам ТМЦ является движение ТМЦ. Однако для ускорения получения информации по текущим остаткам предприятия, а также для отслеживания совместного выбора одной и той же позиции в один момент времени, в системе введена таблица хранения текущих остатков. Эта таблица обновляется автоматически по мере заноса информации в таблицу движения ТМЦ.

Настройка складских документов

Настройка складских документов производится в разделе «Исследователь/Сервис/Типовые документы», где при добавлении документа необходимо выбрать «Добавить складской документ».



При создании документа, указывается его наименование, комментарий, где можно описать, для чего предназначен документ, наименование на английском языке – для названия таблиц входящих в состав документа, таблицы добавляются в строках «Таблица шапки документа» и «Таблица позиции документа». Ветка для вызова – ветка в исследователе, где будет вызов данного документа, шаблон документа. Новые складские документы создаются на основании шаблонов, каждый шаблон определяет вид формирования движения и возможности изменения признаков ТМЦ.

Существует 4 шаблона:

Обычный складской документ

Документ может перемещать ТМЦ с одного места на другое, не изменяя признаков товара (например, внутреннее перемещение) или создавая абсолютно новые признаки, т.е. регистрируя новое поступление товара (например, накладная на получение ТМЦ).
P.S. Не рекомендуется использовать данный тип документа, кроме как для документов получения ТМЦ.

Документ с изменениями свойств

Документ может перемещать ТМЦ с одного места на другое и изменять его признаки. Этот тип документа может работать только с зарегистрированными ТМЦ.

Инвентаризационный документ

Как видно из названия документ используется для создания документов инвентаризации, он не меняет и не создает ни каких признаки ТМЦ, по данному документу происходит или списание ТМЦ в никуда, или появление ТМЦ из ниоткуда.

Документ трансформации.

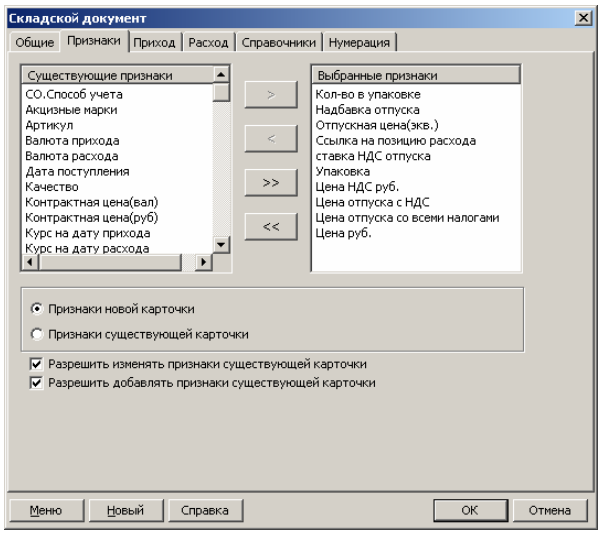
Документ представляет как бы два обычных складских документов, по одному из которых происходит списание ТМЦ, а по второму приход нового ТМЦ. Используется для создания документов комплектации, переработки, когда из нескольких ТМЦ возникает несколько других ТМЦ.

Признаки в документах.

После выбора шаблона документа, определяется, с какими признаками он будет работать. Для документа могут быть определены два вида признаков:

Признаки новой карточки – это те признаки, которые в данном документе могут быть определены. Колонки в документе соответствующие данным признакам доступны пользователю для ввода данных. Наименование колонок в документе идет по правилу «TO_ + Название поля».

Признаки существующей карточки – это те признаки, которые идентифицируют товар при выборе из остатков, по этим признакам система ищет карточки, которые будут перемещаться или изменятся. Название колонок для этих признаков идет с префиксом «FROM_ + Название поля». Изменение этих признаков в документе приводит к тому, что система будет искать новый набор карточек. Если набор данных с новыми признаками не будет найден, то будет или выдано предупреждение о не возможности такого изменения или будет создана новая карточка, которая будет занесена с отрицательным остатком.



Существующие признаки – окно со списком всех признаков, которые определены по ТМЦ. Выбранные признаки – набор признаков, определенных для данного документа.

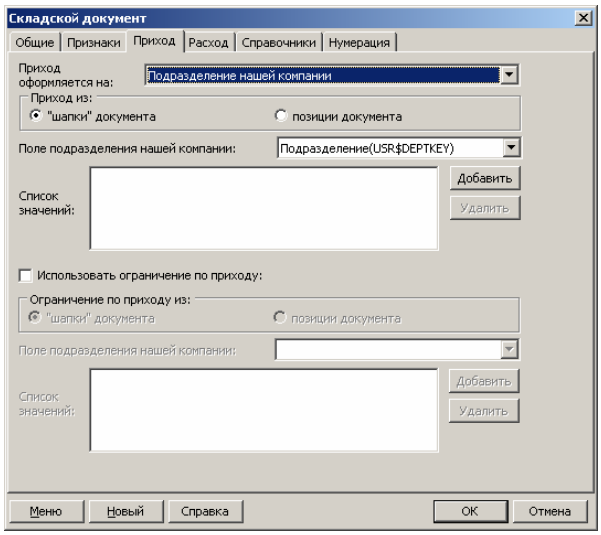
Разрешить изменять признаки существующей карточки – установленный переключатель позволяет при редактировании и добавлении документа, изменять признаки существующей карточки, при этом для изменения будут доступны только те значения, которые по данному товару были определены ранее.

Разрешить добавлять признаки существующей карточки – установленный переключатель позволяет при редактировании и добавлении документа, вводить произвольные значения в признаки существующей карточки. Используется в расходных документах, в которых снят контроль остатков.

Для признаков существующей карточки можно установить можно ли их изменять (т.е. при редактировании позиции документа, выбирать другой признак из существующих значений) и можно ли их добавлять (т.е. вводить произвольное значение признака).

Приход

Для каждого складского документа необходимо указать, куда и как будет оформляться приход, какое поле документа отвечает за место прихода.



В системе предусматривается 7 видов, куда можно оформлять приход

Наша компания – учет ведется без разбиения на подразделения, все остатки хранятся на нашем предприятии. Используется для приходных документов. (Не рекомендуется использовать).

Подразделение нашей компании – остатки хранятся в разрезе наших подразделений. При оформлении документа необходимо указать выбрать данное подразделение. Используется для приходных документов.

Сотрудника нашей компании – остатки хранятся в резрезе материально-ответственных лиц (сотрудников компании). При оформлении документа необходимо указать сотрудника. Используется для приходных документов.

Клиента – при выборе, куда перемещаются товары – список сторонних организаций. Используется для расходных документов.

Подразделение клиента – отпуск товара идет на конкретное подразделение клиента. Обязательно должно быть установлено ограничение по приходу, в котором указывается клиент, подразделение которого будут доступны. Используется для расходных документов.

Сотрудника клиента – отпуск товара идет на конкретного сотрудника клиента. Обязательно должно быть установлено ограничение по приходу, в котором указывается клиент, сотрудники которого будут доступны. Используется для расходных документов.

Физическое лицо – отпуск товара идет на физическое лицо. Используется для расходных документов.

Приход из – (шапки, позиции) – поле, которое отвечает за место хранения, находится либо в шапке документа, тогда один документ осуществляет приход только в одно место хранения, или в позиции документа, тогда для каждой позиции (товара) можно (и нужно) указать свое место хранение

Если необходимо указать конкретные места, на которые можно оформлять приход, заполняется список значений. При выборе списка значений будут доступны справочники в зависимости от типа места.

Иногда для удобства работы при разветленной структуре, необходимо указать сначала главное место, а затем уже выбирать из списка подчиненных значений. Для этого устанавливается «Ограничение по приходу», указывается поле, которое отвечает за главное место. Можно также задать фиксированный набор главных мест, добавив их в список значений.

Расход

Для каждого складского документа необходимо указать, откуда будет оформляться расход, какое поле отвечает за место расхода. Аналогично приходу, имеется 7 видов, откуда можно оформлять расход

Наша компания – учет ведется без разбиения на подразделения, все остатки хранятся на нашем предприятии. Используется для расходных документов. (Не рекомендуется использовать).

Подразделение нашей компании – остатки хранятся в разрезе наших подразделений. При оформлении документа необходимо выбрать данное подразделение. Используется для расходных документов.

Сотрудника нашей компании – остатки хранятся в резрезе материально-ответственных лиц (сотрудников компании). При оформлении документа не обходимо указать сотрудника. Используется для расходных документов.

Клиента – при выборе, откуда поступают товары - список сторонних организаций. Используется для приходных документов.

Подразделение клиента – приход товара идет с конкретного подразделения клиента. Обязательно должно быть установлено ограничение по расходу, в котором указывается клиент, подразделения которого будут доступны. Используется для приходных документов.

Сотрудника клиента – приход товара идет с конкретного сотрудника клиента. Обязательно должно быть установлено ограничение по расходу, в котором указывается клиент, сотрудники которого будут доступны. Используется для приходных документов.

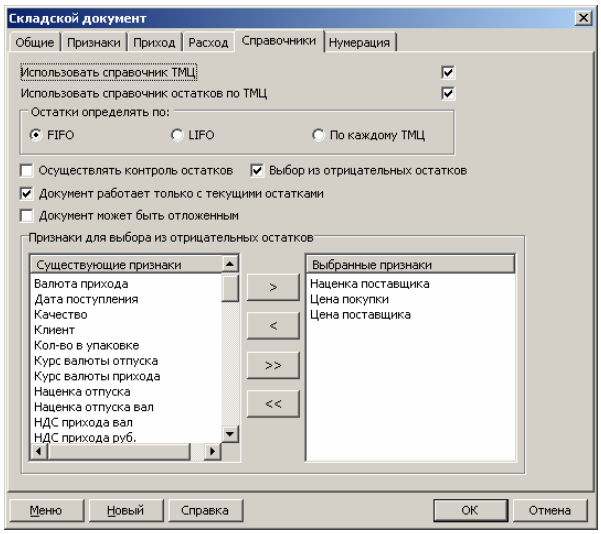
Физическое лицо – приход товара идет с физического лица. Используется для приходных документов.

Аналогично приходу можно указать конкретный список мест откуда идет расход, а также установить ограничения.

Для документов, которые не изменяют местоположение ТМЦ приход и расход указываются одинаковые.

Справочники

Для окончательной настройки складского документа необходимо настроить как будут выбираться товары, идет контроль остатков, вид контроля остатков, методика списания карточек ТМЦ.



Использовать справочник ТМЦ – установка данного флага означает, что в документе можно будет выбирать товары из справочника ТМЦ без остатков. Флаг устанавливается для доукментов, отвечающих за новое поступление ТМЦ. Может устанавливаться и в расходных документах в случае работы без прихода.

Использовать справочник остатков ТМЦ – установка данного флага означает, что в документе можно будет выбирать товары из списка остатков (текущих или на дату документа). Используется только в расходных документах.

Остатки определять по FIFO, LIFO, по каждому ТМЦ – определяется внутренний механизм списания карточек. Надо отметить, что это механизм внутренний и не совсем зависит от методики учета на предприятии. Если предприятие использует методику учета FIFO, LIFO – то эти механизмы совпадают, если предприятие использует другую методику (скажем по учетным ценам или по средним ценам), в документе все равно надо указать FIFO или LIFO, а сам учет настраивается использованием и расчетом признаков (учетная цена напрмер).

Осуществлять контроль остатков – при установленном флаге, по позиции выбранной из остатков нельзя списать больше чем есть. Надо отметить, что если установлено использовать Справочник ТМЦ, контроль остатков надо снять. Если необходимо, что бы только при выборе из справочника программа не контролировала остатки, то необходимо в флаг установить и уже в настройке обрабатывать данную ситуацию.

Выбор из отрицательных остатков – только для приходных документов, позволяет приходовать товары, ранее отпущенные без прихода.

Документ работает только с текущими остатками – флаг имеет смысл только при установленном флаге «Осуществлять контроль остатков». При установленном флаге программа контролирует остатки только на текущую дату, в противном случае контроль идет еще и на дату формирования документа.

Документ может быть отложенным – при установке данного флага, в документе появляется флаг Отложенный. Отложенный документ – документ, по которому не производится движение ТМЦ, и не формируются проводки.

Признаки для выбора отрицательных остатков – Аналогично признакам существующей карточки, только применительно к отрицательным остаткам. Для выбора доступны, только те признаки, которые выбраны как признаки новой карточки.

Настройка складских остатков

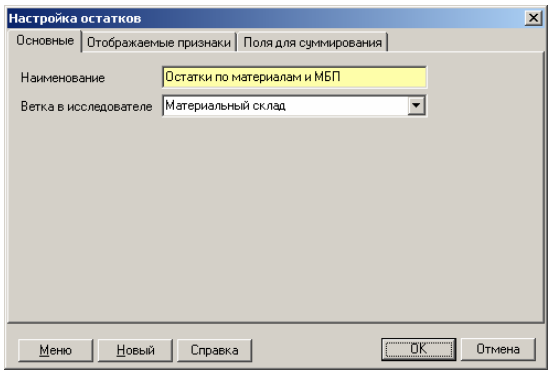
Для создания форм визуального просмотра остатков в различных разрезах, служит раздел «настройка складских остатков». Данный раздел позволяет создать столько форм просмотра остатков сколько их необходимо для целей предприятия.

При добавлении новой настройки необходимо указать название данной настройки (так же она будет называться в «исследователе»), указать ветку в исследователе, откуда форма остатков с данной настройкой будет вызываться.

На закладке «Отображаемые признаки», указываются, какие признаки карточки товара и какие поля из справочника товара будут отображаться при просмотре остатков, соответственно в данном разрезе остатки будут сгруппированы. Если на этой закладке ни чего не выбрать, то остатки будут выводиться в виде:

Наименование | Ед. изм. | Количество.

На закладке «Поля для суммирования» выбираются поля отвечающие, за какие-то цены или т.п. по которым нам необходимо видеть отдельные колонки с суммами.



Структуры данных, используемые для хранения данных о складском движении.

GD_GOODGROUP. Справочник товарных групп.

Наименование поля	Тип поля	На что ссылается	Комментарий
ID	DINTKEY		Идентификатор записи
PARENT	DPARENT	GD_GOODGROUP	Ссылка на родительскую запись
LB	DLB		Правая граница
RB	DRB		Левая граница
NAME	DNAME		Наименование группы
ALIAS	DNULLALIAS		Шифр группы
DESCRIPTION	DTEXT180		Описание группы
DISABLED	DBOOLEAN DEFAULT 0		Признак отключенного счета (0 – нет, 1 – да)
RESERVED	DINTEGER		Зарезервировано
AFULL, ACHAG, AVIEW	DSECURITY		Дескрипторы безопасности

GD_VALUE. Справочник единиц измерения.

Наименование поля	Тип поля	На что ссылается	Комментарий

ID	DINTKEY		Идентификатор записи
NAME	DNAME		Наименование единицы измерения
DESCRIPTION	DTEXT80		Описание
GOODKEY	DFOREIGNKEY	GD_GOOD	Товар сопоставимый единице измерения (ед. изм - бутылка – товар Бутылка)
ISPACK	DBOOLEAN		Признак товар – используется для упаковки

GD_TAX. Справочник налогов

Наименование поля	Тип поля	На что ссылается	Комментарий
ID	DINTKEY		Идентификатор записи
NAME	DNAME		Наименование налога
SHOT	DTEXT20		Краткое наименоване
RATE	DTAX		Ставка налога по умолчанию

GD_TNVD. Справочник кодов ТНВД

Наименование поля	Тип поля	На что ссылается	Комментарий
ID	DINTKEY		Идентификатор записи
NAME	DNAME		Наименование
DESCRIPTION	DTEXT180		Описание

GD_GOOD. Справочник ТМЦ

Наименование поля	Тип поля	На что ссылается	Комментарий
ID	DINTKEY		Идентификатор записи
GROUPKEY	DMASTERKEY	GD_GOODGROUP	Принадлежность к группе
NAME	DNAME		Наименование
ALIAS	DNULLALIAS		Шифр товара
SHORTNAME	DTEXT40		Краткое наименование
DESCRIPTION	DTEXT180		Описание
BARCODE	DBARCODE		Штрих код

VALUEKEY	DINTKEY	GD_VALUE	Базовая единица измерения
TNVDKEY	DFOREIGNKEY	GD_TNVD	Ссылка на код ТНВД
DESCIPLINE	DACCOUNTINGDISCIPLINE		Вид учетной политики для данного ТМЦ F – FIFO, L – LIFO
ISASSEMBLY	DBOOLEAN		Признак является ли данный ТМЦ комплектом (1- Да, 0 – Нет)
EDITORKEY	DFOREIGNKEY		Кто создал (изменил) запись
EDITIONDATE	TIMESTAMP		Время и дата создания (изменения) записи
DISABLED	DBOOLEAN DEFAULT 0		Признак отключенного счета (0 – нет, 1 – да)
RESERVED	DINTEGER		Зарезервировано
AFULL, ACHAG, AVIEW	DSECURITY		Дескрипторы безопасности

GD_GOODTAX. Налоги по товару.

Наименование поля	Тип поля	На что ссылается	Комментарий
GOODKEY	DINTKEY		Идентификатор записи

GD_GOODVALUE. Дополнительные единицы измерения по товару.

Наименование поля	Тип поля	На что ссылается	Комментарий
GOODKEY	DINTKEY	GD_GOOD	Ссылка на товар, по которому задается набор дополнительных единиц измерения
VALUEKEY	DINTKEY	GD_VALUE	Ссылка на единицу измерения
SCALE	DPERCENT		Коэффициент пересчета из базовой единицы измерения в основную. (Сколько в 1-й базовой – дополнительных). Если дополнительная единица используется для упаковки, то сколько в 1-й дополнительной – базовых.

DISCOUNT	DCURRENCY		Скидка на товар для данной единице измерения
DECDIGIT	DDECDIGITS		Количество десятичных

GD_GOODSET. Комплектация товаров.

Наименование поля	Тип поля	На что ссылается	Комментарий
GOODKEY	DINTKEY	GD_GOOD	Ссылка на товар, который является комплектом
SETKEY	DINTKEY	GD_GOOD	Ссылка на товар, входящий в комплект
GOODCOUNT	DPOSITIVE		Количество в товара входящего в комплект

INV_CARD. Складская карточка ТМЦ.

Наименование поля	Тип поля	На что ссылается	Комментарий
ID	DINTKEY		Идентификатор записи
PARENT	DFOREIGNKEY	INV_CARD	Ссылка на родительскую карточку
GOODKEY	DINTKEY	GD_GOOD	Ссылка на товар, по которому ведется карточка
DOCUMENTKEY	DINTKEY	GD_DOCUMENT	Ссылка на документ, который создал карточку
FIRSTDOCUMENTKEY	DINTKEY	GD_DOCUMENT	Ссылка на документ, который создал первого прародителя данной карточки
FIRSTDATE	DDATE NOT NULL		Дата возникновения первого прародителя данной карточки
COMPANYKEY	DINTKEY	GD_OURCOMPANY	Ссылка на компанию, для которой создана карточка
RESERVED	DINTEGER		Зарезервировано
…	Пользовательские поля, которые хранят все переменные свойства товара		

Триггера в таблице inv_card

Перед добавлением. Для присвоения уникального идентификатора.

```
CREATE TRIGGER INV_BI_CARD FOR INV_CARD
ACTIVE BEFORE INSERT POSITION 0
```

```
AS
BEGIN
  IF (NEW.id IS NULL) THEN
    NEW.id = GEN_ID(gd_g_unique, 1) + GEN_ID(gd_g_offset, 0);
END
```

Перед изменением. Триггер отслеживает изменение родителя в карточке и если он изменился, подставляет значения FIRSTDOCUMENTKEY и FIRSTDATE из нового родителя, и их же присваивает всем своим дочерним карточкам.

```
CREATE TRIGGER INV_BU_CARD FOR INV_CARD
ACTIVE BEFORE UPDATE POSITION 0
AS
  DECLARE VARIABLE firstdocumentkey INTEGER;
  DECLARE VARIABLE firstdate DATE;
BEGIN

  IF (OLD.parent <> NEW.parent) THEN
  BEGIN
    SELECT firstdocumentkey, firstdate FROM inv_card
    WHERE id = NEW.parent
    INTO :firstdocumentkey, :firstdate;

    NEW.firstdocumentkey = :firstdocumentkey;
    NEW.firstdate = :firstdate;
  END

  IF ((OLD.firstdocumentkey <> NEW.firstdocumentkey) OR
       (OLD.firstdate <> NEW.firstdate)) THEN
    UPDATE inv_card SET
      firstdocumentkey = NEW.firstdocumentkey,
      firstdate = NEW.firstdate
    WHERE
      parent = NEW.id;

END
```

После добавления пользовательских полей, в таблице INV_CARD добавляется еще один (или несколько) триггеров, которые отслеживают, что бы изменения данных признаков попало во все дочерние карточки, причем только в те, в которых эти признаки совпадают с изменяемой карточкой.

INV_MOVEMENT. Движение товарно-материальных ценностей

Наименование поля	Тип поля	На что ссылается	Комментарий
ID	DINTKEY		Идентификатор записи
MOVEMENTKEY	DINTKEY		Идентификатор движения одной карточки
MOVEMENTDATE	DDATE NOT NULL		Дата движения
DOCUMENTKEY	DINTKEY	GD_DOCUMENT	Ссылка на документ, который создал данное движение
CONTACTKEY	DINTKEY	GD_CONTACT	Ссылка на контрагента данной операции (на кого поступает ТМЦ или с кого списывается ТМЦ)

CARDKEY	DINTKEY	INV_CARD	Ссылка на карточку, которая перемещается данным движением
DEBIT	DCURRENCY		Кол-во прихода
CREDIT	DCURRENCY		Кол-во расхода
DISABLED	DBOOLEAN DEFAULT 0		Признак отключенного счета (0 – нет, 1 – да)
RESERVED	DINTEGER		Зарезервировано

Триггера в таблице INV_MOVEMENT

Перед добавлением. Триггер для контроля остатков

```
CREATE TRIGGER INV_BI_MOVEMENT FOR INV_MOVEMENT
ACTIVE BEFORE INSERT POSITION 0
AS
  DECLARE VARIABLE balance NUMERIC(15, 4);
BEGIN
/* Присваиваем уникальный идентификатор */
  IF (NEW.id IS NULL) THEN
    NEW.id = GEN_ID(gd_g_unique, 1) + GEN_ID(gd_g_offset, 0);
/* Поля DEBIT, CREDIT – не должны быть пустыми */
  IF (NEW.debit IS NULL) THEN
    NEW.debit = 0;
  IF (NEW.credit IS NULL) THEN
    NEW.credit = 0;
/* Проверка, что бы движение не привело к отрицательным остаткам*/
  IF (NEW.credit > 0) THEN
  BEGIN
    SELECT balance FROM inv_balance
    WHERE
      contactkey = NEW.contactkey
       AND cardkey = NEW.cardkey
    INTO :balance;
    IF (:balance IS NULL) THEN
      balance = 0;
    IF ((:balance > 0) AND (:balance < NEW.credit)) THEN
    BEGIN
      EXCEPTION INV_E_INVALIDMOVEMENT;
    END
  END
END
```

После добавления. Триггер для изменения остатков в таблице текущие остатки (INV_BALANCE).

```
CREATE TRIGGER INV_AI_MOVEMENT FOR INV_MOVEMENT
ACTIVE AFTER INSERT POSITION 0
AS
  DECLARE VARIABLE NEWBALANCE NUMERIC(10, 4);
BEGIN
  /* если запись с остатком есть - изменяем ее, если нет - создаем */
  IF (EXISTS(
    SELECT BALANCE
    FROM INV_BALANCE
    WHERE CARDKEY = NEW.CARDKEY AND CONTACTKEY = NEW.CONTACTKEY)
  )
  THEN BEGIN
    UPDATE INV_BALANCE
    SET
```

```
      BALANCE = BALANCE + (NEW.DEBIT - NEW.CREDIT)
    WHERE
      CARDKEY = NEW.CARDKEY AND CONTACTKEY = NEW.CONTACTKEY;
  END ELSE BEGIN
    INSERT INTO INV_BALANCE
      (CARDKEY, CONTACTKEY, BALANCE)
    VALUES
      (NEW.CARDKEY, NEW.CONTACTKEY, NEW.DEBIT - NEW.CREDIT);
  END
END
```

Перед изменением. Триггер для контроля остатков.

```
CREATE TRIGGER INV_BU_MOVEMENT FOR INV_MOVEMENT
ACTIVE BEFORE UPDATE POSITION 0
AS
 DECLARE VARIABLE balance NUMERIC(15, 4);
BEGIN
/* В движении не может поменяться документ */
  IF (NEW.documentkey <> OLD.documentkey) THEN
    EXCEPTION inv_e_movementchange;
/* Проверка на остатки */
  IF((NEW.disabled = 1) OR
    (NEW.contactkey <> OLD.contactkey) OR
    (NEW.cardkey <> OLD.cardkey))
  THEN BEGIN
/* Если изменилась карточка или место */
    IF (OLD.debit > 0) THEN
    BEGIN
/* Если это был приход, то надо проверить, не станет ли остаток по
   старой карточке (или на старом месте) отрицательным */
      SELECT balance FROM inv_balance
      WHERE
        contactkey = OLD.contactkey
        AND cardkey = OLD.cardkey
      INTO :balance;
      IF (:balance IS NULL) THEN
        balance = 0;
      IF (:balance < OLD.debit) THEN
      BEGIN
        EXCEPTION INV_E_INVALIDMOVEMENT;
      END
    END
  END
  ELSE
  BEGIN
    IF (OLD.debit > NEW.debit) THEN
    BEGIN
/* Если уменьшился приход, проверяем, что бы хватало остатков */
      SELECT balance FROM inv_balance
      WHERE
        contactkey = OLD.contactkey
        AND cardkey = OLD.cardkey
      INTO :balance;
      IF (:balance IS NULL) THEN
        balance = 0;
      IF ((:balance > 0) AND (:balance < OLD.debit - NEW.debit)) THEN
      BEGIN
        EXCEPTION INV_E_INVALIDMOVEMENT;
      END
    END
    ELSE
```

```
      IF (NEW.credit > OLD.credit) THEN
      BEGIN
/* Если увеличился расход, проверяем, что бы хватало остатков */
        SELECT balance FROM inv_balance
        WHERE
          contactkey = OLD.contactkey
          AND cardkey = OLD.cardkey
        INTO :balance;
        IF (:balance IS NULL) THEN
          balance = 0;
        IF ((:balance > 0) AND
           (:balance < NEW.credit - OLD.credit))
        THEN
          EXCEPTION INV_E_INVALIDMOVEMENT;
       END
  END
END
```

После изменения. Триггер для изменения остатков в таблице текущие остатки (INV_BALANCE).

```
CREATE TRIGGER INV_AU_MOVEMENT FOR INV_MOVEMENT
ACTIVE AFTER UPDATE POSITION 0
AS
  DECLARE VARIABLE existscontaccard INTEGER;
BEGIN
  /* если запись с остатком есть - изменяем ее, если нет - создаем */
  existscontaccard = 1; /* Есть ли старая карточка в остатках*/
/* Данный флаг используется для корректной обработки случая объединения
контактов */
  IF (OLD.disabled = 0) THEN
  BEGIN
    IF (EXISTS(
      SELECT BALANCE
      FROM INV_BALANCE
      WHERE CARDKEY = OLD.CARDKEY AND CONTACTKEY = OLD.CONTACTKEY))
    THEN BEGIN
      UPDATE INV_BALANCE
      SET
        BALANCE = BALANCE - (OLD.DEBIT - OLD.CREDIT)
      WHERE
        CARDKEY = OLD.CARDKEY AND CONTACTKEY = OLD.CONTACTKEY;
    END
    ELSE
      existscontaccard = 0;
  END

  IF (NEW.disabled = 0) THEN
  BEGIN
    IF (
       (NEW.contactkey <> OLD.contactkey) AND
       (NEW.cardkey = OLD.cardkey) AND
       (NEW.debit = OLD.debit) AND
       (NEW.credit = OLD.credit) AND
       (OLD.disabled = 0) AND
       (existscontaccard = 0)
       )
    THEN
/* Если изменился контакт, и ничего более и карточки нет в остатках по
старому месту, то ничего не делаем */
      existscontaccard = 0;
    ELSE
    IF (EXISTS(
```

```
        SELECT BALANCE
        FROM INV_BALANCE
        WHERE CARDKEY = NEW.CARDKEY AND CONTACTKEY = NEW.CONTACTKEY))
    THEN BEGIN
      UPDATE INV_BALANCE
      SET
        BALANCE = BALANCE + (NEW.DEBIT - NEW.CREDIT)
      WHERE
        CARDKEY = NEW.CARDKEY AND CONTACTKEY = NEW.CONTACTKEY;
    END ELSE BEGIN
        INSERT INTO INV_BALANCE
          (CARDKEY, CONTACTKEY, BALANCE)
        VALUES
          (NEW.CARDKEY, NEW.CONTACTKEY, NEW.DEBIT - NEW.CREDIT);
    END
  END
END
```

INV_BALANCE. Текущие остатки товарно-материальных ценностей

Наименование поля	Тип поля	На что ссылается	Комментарий
CARDKEY	DINTKEY	INV_CARD	Ссылка на карточку
CONTACTKEY	DINTKEY	GD_CONTACT	Ссылка на контакт
BALANCE	DCURRENCY		Остаток
RESERVED	DRESERVED		Зарезервировано

Данная таблица заполняется только триггерами из таблицы inv_movement.

Объекты и формы, используемые для работы со складом

Справочник ТМЦ

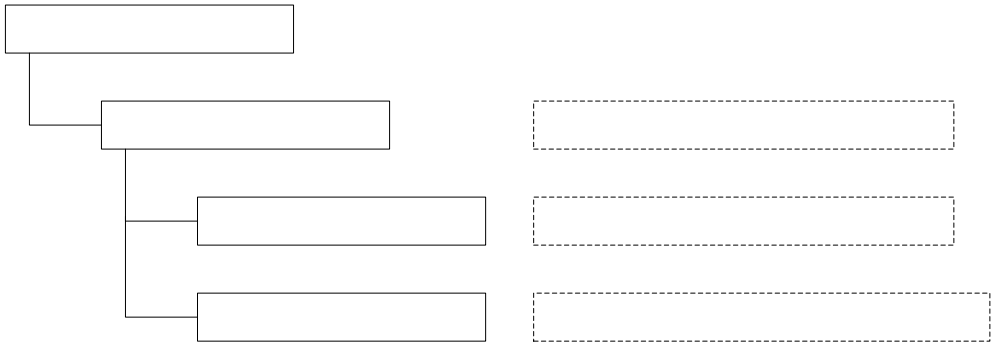
TgdcGood – класс для работы с ТМЦ.

TgdcGoodGroup – класс для работы с группами ТМЦ

TgdcValue – класс для работы с единицами измерения

TgdcTax – класс для работы с налогами по ТМЦ

Складской документ



```
TgdcInvBaseDocument = class(TgdcDocument)
….
  Public
// Тип источника
```

```
    property MovementSource: TgdcInvMovementContactOption;
// Тип получателя
    property MovementTarget: TgdcInvMovementContactOption;
// Наименование таблицы шапки
    property RelationName: String;
// Наименование таблицы позиции
    property RelationLineName: String;
// Тип складского документа
    property RelationType: TgdcInvRelationType read GetRelationType;
….
  end;

TgdcInvDocument = class(TgdcInvBaseDocument)
…
End;

  TgdcInvDocumentLine = class(TgdcInvBaseDocument)
…
  Public
// Установка значений для полей (isFrom = True) существуюшей карточки //
(FROM_ поля),  и (isTo = True) новой картчоки (TO_ поля) на
// основании карточки  указанной в поле FROMCARDKEY
    procedure SetFeatures(isFrom, isTo: Boolean);
// Тип выводимой части для документа трансформации:
//      0 – Приход, 1 – Расход, 2 - Все
    property ViewMovementPart: TgdcInvMovementPart;
// Признак контроля остатков
    property ControlRemains: Boolean;
// Объект отвечающий за дивжение
    property Movement: TgdcInvMovement;
// Признак, что документ работает только с текущими остатками
    property LiveTimeRemains: Boolean;
// Признак, что документ выбирает остатки из отрицательныъх карточек
    property isMinusRemains: Boolean;
// Признак, что заполнение полей идет из режима выбора остатков
    property isSetFeaturesFromRemains: Boolean;
…
  end;
```

Классы для работы со складскими документами, являются абстрактными, объекты данных классов обязательно требуют установленного значения свойства SubType. Для складских документов (также как для любых других пользовательских документов) Subtype – это РУИД соответствующего документ.

Движение ТМЦ

TgdcInvMovement. Данный класс отвечает за формирование движения по складским документам. Объекты данного класса используются исключительно внутри класса TgdcInvDocumentLine. Отдельное создание и использование объектов данного класса не имеет ни какого смысла. Для настройки представляет интерес свойство Movement класса TgdcInvDocumentLine.

```
  TgdcInvMovement = class(TgdcBase)
…
  Public
// Указывает следует ли производить предварительный поиск в остатках //
текущей позиции перед ее сохранением
   property IsGetRemains: Boolean;
…
  end;
```

Остатки. ТМЦ

TgdcInvBaseRemains – базовый класс для работы с остатками ТМЦ

TgdcInvRemains – класс предназначен для работы со всеми остатками ТМЦ.

TgdcInvGoodReamains – класс предназначен для работы с остатками по конкретному ТМЦ.

```
  TgdcInvBaseRemains = class(TgdcBase)
  public
// Набор свойств карточки необходимый для отображения остатаков
    property ViewFeatures: TStringList;
// Набор свойств карточки по которым необходимо осуществлять суммирование
    property SumFeatures: TStringList;
// Набор свойств товара необходимый для отображения остатаков
    property GoodViewFeatures: TStringList;
// Набор свойств товара по которым необходимо осуществлять
// суммирование
    property GoodSumFeatures: TStringList;
// Дата на которую формируются остатки
    property RemainsDate: TDateTime;
// Выводить текущие остатки
    property CurrentRemains: Boolean;
  end;

  TgdcInvRemains = class(TgdcInvBaseRemains)
  public
// Код товара по которому выводятся остатки (елси -1 то по всем ТМЦ)
    property GoodKey: Integer;
// Код группы по которой выводятся остатки (если -1 то по всем группам)
    property GroupKey: Integer;
// Документ для которого будут выбираться остатки
    property gdcDocumentLine: TgdcDocument;
  end;

  TgdcInvGoodRemains = class(TgdcInvRemains)
  …
  End
```

Хранилище

В любой программной системе существуют десятки параметров, определяющих ее функционирование. Наверное, вы уже знакомы с реестром системы Windows — специальной базой данных, где в иерархическом виде хранятся параметры как самой операционной системы, так и программ сторонних разработчиков. Хранилище[18] — это аналог реестра Windows, используемый платформой Гедымин для хранения различных параметров. Естественно, возникает вопрос: зачем понадобилось создавать по сути параллельный механизм? Делалось это из двух соображений. Во-первых, часть параметров являются «привязанными» к конкретному пользователю платформы. Естественно, с какого бы компьютера пользователь не вошел в систему, она должна загрузиться именно с его параметрами. Сделать это можно только, если параметры будут храниться централизованно в базе данных, а не в реестре конкретного компьютера. Во-вторых, хранилище используется для хранения экранных форм, которые являются неотъемлемой частью прикладных настроек. При загрузке новой версии настройки все формы должны обновиться. Очевидно, что если бы они были разбросаны по многим компьютерам в сети, простой процесс установки новой версии приложения превратился бы в сущий ад.

Хранилище имеет древовидную многоуровневую иерархическую структуру, образованную разделами и входящими в них подразделами. Раздел имеет имя, которое должно быть уникальным среди имен всех разделов, входящих в один головной (родительский) раздел. Путем к разделу или его полным именем называется строка, состоящая из последовательного перечисления имен родительских разделов, разделенных символом наклонной черты, а так же, собственно, имени раздела. Пример полного имени раздела: \Options\Inventory\Equivalent. Одиночный символ наклонной черты ссылается на корневой раздел хранилища. Каждый раздел может иметь произвольное число параметров. Параметр имеет уникальное в данном разделе имя и может быть одного из следующих типов:

- Целое число;
- Дробное число;
- Строка;
- Дата и время;
- Булевский тип;
- Двоичные данные;

Подобно тому, как реестр операционной системы Windows имеет несколько предопределенных корневых ветвей (CLASSES_ROOT, CURRENT_USER, LOCAL_MACHINE и т.д.) хранилище платформы Гедымин так же подразделяется на несколько частей. Точне будет сказать: в платформе Гедымин одновременно присутствуют несколько хранилищ, при этом каждое имеет свое предназначение.

Глобальное хранилище

Глобальное хранилище (доступ к нему осуществляется через объект GlobalStorage) предназначено для хранения параметров платформы, общих для всех пользователей и всех рабочих организаций. В частности, в корневом разделе глобального хранилища находятся:

Имя раздела	Описание
DFM	В данном разделе содержатся: 1. Стандартные экранные формы платформы, настроенные дополнительно под учетной записью Администратора (т.е. настроенные для всех пользователей системы). 2. Экранные формы, созданные для пользовательских подтипов бизнес объектов. Данный раздел имеет вложенные подразделы, по одному для каждого класса

[18] Очевидно, такое название — хранилище — было избрано разработчиками для того, чтобы не путать систему хранения параметров платформы Гедымин с реестром операционной системы Windows.

	экранных форм. Каждый вложенный подраздел имеет один или несколько параметров, по числу пользовательских подтипов. Если для данного класса не определены пользовательские подтипы, то форма хранится в единственном параметре, имеющем имя default.
NewForm	Список произвольных пользовательских форм. Каждая форма располагается в отдельном вложенном подразделе, который имеет до пяти параметров, три из которых присутствуют у форм всех типов: • имя класса, который имеет данная форма, • данные формы, • и внутренний тип. Два параметра, присутствуют только у форм, связанных с бизнес объектом: • тип бизнес объекта • и его подтип.
Options	Глобальные, т.е. единые для всех пользователей, параметры платформы. Данный раздел имеет несколько вложеных подразделов.
RegionalSettings	Региональные установки. Формат числа, даты и время и т.п.
SubTypes	Список определенных пользователем подтипов для каждого класса платформы.

Пользовательское хранилище

Пользовательское хранилище (объект UserStorage) предназначено для хранения параметров платформы, привязанных к конкретному пользователю. На верхнем уровне располагаются разделы с настройками экранных форм. Каждый такой раздел имеет имя состоящее из имени экранной формы и присоединенного к нему, в скобках, имени класса этой формы. Например: dlgInvDocument147004729_45137928(TdlgInvDocument). Кроме этого, на верхнем уровне присутствуют следующие системные разделы:

Имя раздела	Описание
DFM	Стандартные экранные формы платформы, а так же формы, созданные для пользовательских подтипов, настроенные дополнительно под конкретного пользователя. Данный раздел имеет вложенные подразделы, по одному для каждого класса экранных форм. Каждый вложенный подраздел имеет один или несколько параметров, по числу пользовательских подтипов. Если для данного класса не определены пользовательские подтипы, то форма хранится в единственном параметре, имеющем имя default.
GDC	Параметры бизнес классов. Для каждого бизнес класса в данном разделе имеется соответствующий подраздел.
Options	Параметры системы, привязанные к конкретному пользователю.

Хранилище компании

Параметры платформы, привязанные к рабочей организации. Хранилище доступно через объект CompanyStorage. Одним из самых распространенных использований данного хранилища, является сохранение выбранного пользователем рассчетного счета компании в формах просмотра и диалоговых окнах.

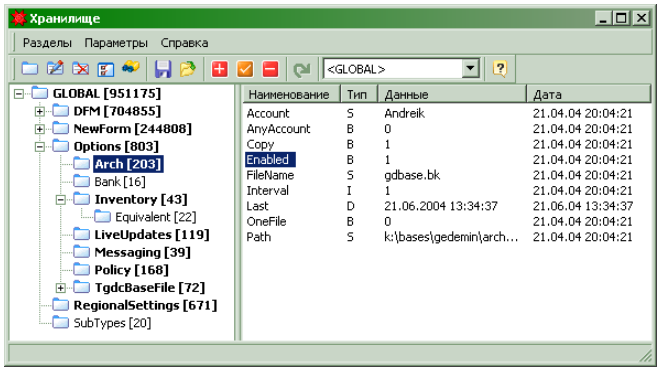
Хранилище рабочего стола

Хранилище параметров, привязанных к конкретному рабочему столу. Хранилище доступно через объект DesktopStorage.

Работа с хранилищем

Для работы с Хранилищем необходимо открыть окно просмотра и редактирования Хранилища из подраздела Сервис Исследователя платформы.

Ниже приведен общий вид окна:



Как видно, окно имеет следующие основные части:

- Главное меню, для вызова команд работы с хранилищем;
- Панель инструментов, для вызова команд и для выбора текущего хранилища; Выпадающий список для выбора хранилища находится в правой части панели. В нем находятся имена всех пользователей системы. При выборе конкретного имени в окно загружается хранилище этого пользователя. Кроме этого в списке находятся три мета-символа: <GLOBAL>, <COMPANY> и <DESKTOP> для загрузки глобального, хранилища компании и хранилища рабочего стола соответственно;
- Дерево разделов хранилища;
- Список параметров, содержащихся в выбранном разделе;
- Строку состояния, где выводится путь к текущему, выбранному разделу.

После имени раздела, в квадратных скобках, приводится размер блока оперативной памяти в байтах, занимаемого этим разделом. Если раздел имеет хотя бы один параметр или вложенный подраздел, то его имя выводится жирным шрифтом. Если раздел или параметр входит хотя бы в одну настройку, то его имя выводится синим цветом.

Имя корневого раздела хранилища, выводимое в дереве, отражает имя хранилища и, если это пользовательское хранилище или хранилище рабочей компании, имя соответствующего объекта[19].

При работе с деревом разделов доступны следующие команды:

- Создать новый раздел;
- Переименовать существующий раздел;
- Удалить раздел;
- Просмотреть свойства раздела; На экран будет выведено окно с такой полезной информацией как количество параметров, подразделов, дата создания или последнего изменения данного раздела;
- Поиск; Искать можно среди названий разделов, названий параметров и их значений. После ввода строки поиска и запуска в нижней части окна откроется список найденных объектов. При щелчке мыши по объекту из списка соответствующий раздел будет выбран в дереве разделов;
- Сохранить в файл. Все данные хранилища будут сохранены в указанный файл, после чего их можно будут загрузить на другой базе данных. Рекомендуется делать архивную копию хранилища перед загрузкой новых настроек.
- Загрузить из файла. Загружает данные хранилища из файла.

---

[19] Выводится в скобках, после имени хранилища.

- Добавить в настройку. Добавляет выбранный раздел в указанную настройку;
- Обновить. Перечитывает хранилище из базы данных и обновляет дерево и список параметров.

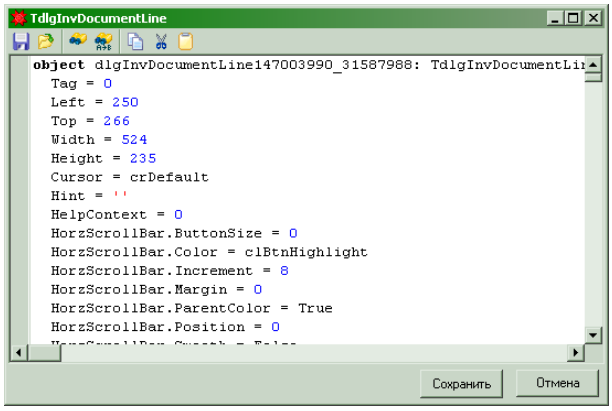
Справа от дерева разделов выводится список параметров текущего, выбранного раздела. Для каждого параметра указывается его имя, Тип данных, Размер, и Дата последнего изменения или создания данного параметра.

При работе со списком параметров доступны следующие команды:

- Создать новый параметр. При этом необходимо указать имя параметра, его тип и значение;
- Удалить параметр;
- Переименовать параметр;
- Изменить значение параметра;
- Добавить выбранный параметр в указанную настройку.

Изменить тип параметра после его создания невозможно. Если был создан параметр с неверным типом, необходимо удалить его и создать новый параметр с нужным типом.

Редактирование параметра, содержащего двоичные данные, осуществляется в отдельном окне.



Организация данных хранилища

Выше уже отмечалось, что для работы с хранилищем используются специальные объекты. Возникает вопрос: если данные хранилища находятся непосредственно в базе данных, то почему для доступа к ним не может быть использован бизнес-объект? Дело в том, что содержание данных хранилища в базе, в чистом реляционном виде, когда каждый раздел или параметр — это отдельная запись, связанная отношениями с другими записями, невозможно по той причине, что извлечение данных будет занимать весьма значительное время[20]. Не стоит забывать, что

Метаданные хранилища

GD_COMPANYSTORAGE

Для каждой записи в таблице GD_OURCOMPANY создается одна запись в таблице GD_COMPANYSTORAGE, где сохраняются данные хранилища рабочей организации.

PK	FK	Поле Домен Тип данных	NN	Описание
		COMPANYKEY DINTKEY INTEGER		Уникальный идентификатор записи он же ссылка на таблицу GD_OURCOMPANY.

[20] Только настройки одного экранного элемента — таблицы — могут состоять из нескольких сотен параметров.

		DATA DBLOB4096 BLOB		Данные пользовательского хранилища.
		MODIFIED DTIMESTAMP TIMESTAMP		Дата и время изменения данных.

GD_GLOBALSTORAGE

Данные глобального хранилища находятся в таблице GD_GLOBALSTORAGE, которая содержит всегда только одну запись.

PK	FK	Поле Домен Тип данных	NN	Описание
		ID DINTKEY INTEGER		Уникальный идентификатор записи.
		DATA DBLOB4096 BLOB		Данные глобального хранилища.
		MODIFIED DTIMESTAMP TIMESTAMP		Дата и время последней записи в таблицу.

GD_USERSTORAGE

Для каждой учетной записи пользователя в таблице GD_USERSTORAGE создается одна запись, которая содержит данные пользовательского хранилища.

PK	FK	Поле Домен Тип данных	NN	Описание
		USERKEY DINTKEY INTEGER		Уникальный идентификатор записи он же ссылка на таблицу GD_USER.
		DATA DBLOB4096 BLOB		Данные пользовательского хранилища.
		MODIFIED DTIMESTAMP TIMESTAMP		Дата и время изменения данных.

GD_DESKTOP

Данные хранилища рабочего стола находятся в записи с данными о рабочем столе в таблице GD_DESKTOP.

Доступ к хранилищу из макросов

Для работы с хранилищем из макросов используются три глобальных объекта GlobalStorage, UserStorage и CompanyStorage. Заметьте, что объект для доступа к хранилищу рабочего стола отсутствует. Все три объекта поддерживают следующие основные методы и свойства:

Методы

BuildComponentPath

```
function BuildComponentPath(C: TComponent; Context: String): String;
```

Clear

```
procedure Clear;
```

Очищает содержимое всего хранилища. Будьте осторожны с использованием данного метода, особенно при работе с глобальным хранилищем. Восстановить данные хранилища будет невозможно если у вас нет резервной копии базы данных или, если предворительно вы не сохранили данные хранилища в файл.

CloseFolder

```
procedure CloseFolder(gsStorageFolder: TgsStorageFolder; SyncWithDatabase:
Boolean);
```

Закрывает раздел, открытый ранее с помощью метода OpenFolder. В зависимости от значения параметра SyncWithDatabase будет происходить синхронизация содержимого хранилища с базой данных или нет.

FolderExists

```
function FolderExists(APath: String): Boolean;
```

Возвращает Истину, если указанный раздел существует и Ложь в противном случае.

LoadFromDatabase

```
procedure LoadFromDatabase;
```

Загружает данные хранилища из базы данных. Обычно нет необходимости использовать этот метод непосредственно.

LoadFromFile

```
procedure LoadFromFile(AFileName: String; AFileFormat: Variant);
```

Загружает хранилище целиком из файла. AFileFormat может принимать следующие значения: 0 — двоичный формат. 1 — текстовый формат.

LoadFromStream

```
procedure LoadFromStream(S: TStream);
```

Загружает хранилище целиком из переданного потока. Поток должен содержать данные хранилища в двоичном формате.

LoadFromStream2

```
procedure LoadFromStream2(S: TStringStream);
```

Загружает хранилище целиком из переданного потока. Поток должен содержать данные хранилища в текстовом формате.

OpenFolder

ReadBoolean

ReadCurrency

ReadDateTime

ReadInteger

ReadStream

ReadString

SaveToDatabase

SaveToFile

SaveToStream

SaveToStream2

ValueExists

WriteBoolean

WriteCurrency

WriteDateTime

WriteInteger

WriteStream

WriteString

Примеры работы с хранилищем

Восстановление хранилища

Настройки

Совместная разработка

Создать простую прикладную настройку по силам одному человеку, но рано или поздно вы приступите к решению задач, которые потребуют одновременного участия нескольких разработчиков. Ниже приводится список правил, следование которым позволит существенно снизить трудозатраты при совместной разработке, избежать путаницы и непроизвольных потерь макросов и данных.

1. Первое правило относится не только к проектам на платформе Гедымин, а является общим для любого проекта по созданию программного обеспечения. Перед началом кодирования вы должны иметь четкое представление о том, что должно получиться на выходе. Совместно с заказчиком составьте подробное техническое задание. Исходя из него, разработайте принципиальную схему будущей программы. Из каких модулей она будет состоять? Какие структуры данных потребуются? Как будет осуществляться взаимодействие между модулями? Как будет организован пользовательский интерфейс? Какие выходные формы потребуются?

   Имея на руках подробную документацию, вы будете в состоянии разбить весь объем работ на отдельные задания и распределить их среди членов команды. Назначьте регулярное совещание для того, чтобы отслеживать ход выполнения проекта.

2. Выберите префикс для своего проекта. Используйте этот префикс в наименованиях объектов базы данных, макросов, отчетов и т.п.

3. Если в разработке принимает участие несколько человек, то важно избежать путаницы, которая может возникнуть, когда два разработчика присвоят своим объектам одинаковые наименования. Как правило, в большом проекте, каждый из программистов трудится над своей подсистемой. В таком случае, можно предложить использовать свой префикс для каждой из подсистем. Таким образом, имя объекта будет иметь два префикса: сначала — префикс проекта, затем — подсистемы. Например, если для разрабатываемого проекта был выбран префикс X_, а для подсистемы — Y_, то функция AddParam, логически относящаяся к указанной подсистеме, получит полное имя X_Y_AddParam.

4. Каждый разработчик должен работать со своим экземпляром базы данных.

5. Объекты, разрабатываемые одним разработчиком, должны группироваться в одну или несколько настроек. Эта настройка (настройки) может зависеть от других настроек, в том числе и созданных другими разработчиками, но не должна включать объекты, над которыми работают другие разработчики.

6. Рекомендуется имена файлов настроек, относящихся к одному проекту, начинать с имени этого проекта. Например, «Зарплата и Кадры. Метаданные.gsf», «Зарплата и Кадры. Макросы.gsf». Зачем? Когда на базе данных загружено несколько проектов, список настроек в соответствующем окне может содержать десятки и даже сотни позиций. Правильное именование файлов, позволит легко сгруппировать настройки, относящиеся к одному проекту, просто упорядочив их список по имени.

7. Для упрощения процесса загрузки настроек на эталонную базу данных следует создать пакет и указать связи между зависимыми настройками.

8. По выполнении определенного объема работ, все разработчики должны сформировать свои настройки и сохранить их на диске. Рекомендуется использовать систему контроля версий, например, Borland StarTeam, для хранения истории изменения настроек.

9. Следует постоянно контролировать, чтобы, во-первых, в настройки попадали все необходимые объекты, а во-вторых, чтобы не возникало конфликтов, между объектами, созданными разными разработчиками. Для осуществления контроля рекомендуется периодически выполнять следующее:

   a. Загрузить на чистую, эталонную базу данных последние версии всех настроек;

   b. Заменить полученной таким образом базой рабочие базы у всех разработчиков. Перед заменой, каждый разработчик должен создать архивную копию своей базы!

c. Если в процессе дальнейшей работы выяснится, что какой-то объект не попал в настройку, то следует восстановить базу из архивной копии, включить объект в настройку, переформировать ее и сохранить в файле. Повторить все действия, начиная с пункта a).

10. Если вам необходимо передать базу данных клиенту или к вашей команде присоединился еще один разработчик, у вас может возникнуть соблазн скопировать одну из существующих рабочих баз. Никогда не делайте этого! Возьмите чистый эталон и загрузите на него последние версии ваших настроек.
11. Каждая база должна иметь свой уникальный идентификатор — DBID. Если вам необходимо получить копию базы, то добиться присвоения нового уникального кода можно:
    a. Создав архивную копию базы и выполнив затем восстановление из архива с установленным флагом «Присваивать уникальный ИД базы»;
    b. Подключиться к базе данных, открыть окно «SQL редактора» и с помощью команды «SET GENERATOR GD_G_DBID TO 0» присвоить значение 0 генератору. Новый уникальный идентификатор будет присвоен базе данных при следующем подключении к ней.
12. После того, как вы начнете, бета-тестирование проекта, часто будет возникать ситуация, когда изменения, сделанные на клиентской базе данных, необходимо перенести в настройки проекта. В таком случае рекомендуется поступать следующим образом:
    a. Макросы, отчеты и структуры данных, измененные или созданные на клиентской базе, включаются в отдельную временную настройку;
    b. Эта настройка формируется и сохраняется в файле;
    c. На той базе данных, на которой ведется разработка, настройка загружается и активизируется;
    d. Объекты, перенесенные с помощью временной настройки, включаются в настройки проекта, после чего она удаляется из списка;

Безопасность

Подсистема безопасности платформы Гедымин предоставляет развитые средства разграничения доступа к данным и функциям системы, а так же позволяет организовать аудит действий пользователя и вести журнал совершенных изменений данных.

Следует заметить, что хотя абсолютно устойчивых к несанкционированному доступу систем не существует, грамотный администратор всегда может создать программно-аппаратный комплекс, затраты на взлом которого будут многократно превышать стоимость данных, полученных в результате такого взлома, т.е. сделают его экономически нецелесообразным.

Для начала, мы рассмотрим средства разграничения доступа, предоставляемые сервером Interbase, поскольку, во-первых, на них частично базируется подсистема безопасности платформы Гедымин, во-вторых, знание механизмов разграничения доступа предоставляемых стандартом SQL может пригодиться разработчику при организации прямого обращения к базе данных из сторонних программ. Если вы уверены в своих знаниях SQL, то можете лишь бегло ознакомиться с содержимым следующих параграфов, либо сразу перейти к главе «Разграничение прав доступа на платформе Гедымин».

Средства разграничения доступа сервера Interbase

Сервер Interbase поддерживает механизмы определенные стандартом SQL-92 (включая роли) для разграничения прав пользователей для доступа к данным в базе.

Только пользователь, для которого создана учетная запись может осуществить подключение к серверу и базе данных. При этом необходимо указать имя учетной записи и пароль.

Учетные записи могут быть созданы при помощи:

- утилиты командной строки gsec;
- графических утилит для работы с базами данных: IBExpert, QuickDesk, IBConsole и др.;
- вызовов соответствующих функций API Interbase (напрямую или используя промежуточные классы, например, из библиотеки Delphi компонент Interbase Express (сокращенно IBX).

Interbase использует единый список пользователей для всех баз данных, находящихся на одном физическом сервере. Информация об учетных записях хранится в отдельной базе данных, в файле с именем isc4.gdb[21], находящемся в корневом каталоге сервера[22].

Хотя имя пользователя и его пароль могут иметь длинну до 32-х символов, значимыми являются только первые 8, т.е. пароли «VeryLongPassword» и «VeryLong» для сервера идентичны. Для строки пароля различается регистр символов.

Учетная запись SYSDBA

Сразу после установки сервера Interbase существует только одна учетная запись с именем SYSDBA и паролем masterkey. Эта учетная запись обладает максимальными правами и позволяет выполнять любые действия как над данными в базе, так и над структурой базы данных. Учитывая «всемогущество» записи SYSDBA важно изменить ее пароль сразу после установки сервера.

Предотвращение несанкционированного доступа

Существует достаточно простой способ получения доступа к информации в базе. Для этого достаточно скопировать файл базы данных на другой компьютер, проинсталлировать там сервер Interbase и подключиться к этой базе используя учетную запись SYSDBA и пароль по умолчанию. Вот почему важно исключить возможность несанкционированного доступа пользователей к файлу базы данных.

---

[21] В зависимости от клона сервера Interbase (Firebird, Yaffil) и его версии имя файла с информацией об учетных записях может различаться. При необходимости, обратитесь к соответствующей документации по работе с сервером базы данных.

[22] Для сервера Yaffil, как правило, полный путь к базе данных с учетными записями выглядит следующим образом: c:\Program Files\Yaffil\isc4.gdb.

Если используется операционная система Windows 2000 Server[23], то сделать это достаточно просто, установив соответствующие права доступа на директорию, где находится файл базы данных или непосредственно на сам файл[24]. Однако, не следует забывать, что если будет похищен компьютер, являющийся сервером или винчестер, на котором располагается база данных, то злоумышленник всегда сможет подключить винчестер в другой компьютер, на котором установлена операционная система Windows 2000 Server и получить полный доступ к файлу базы данных. Противодействием здесь может быть:

- Размещение сервера в отдельном, хорошо защищенном и охраняемом помещении;
- Хранение файла базы данных на съемном носителе информации, который по окончании рабочего дня изымается и помещается в безопасное место;
- Шифрование файла базы данных средствами операционной системы[25]. Даже если злоумышленник получит доступ к такому файлу, на другом компьютере он не сможет прочитать его.

Очевидно, что следование одной или нескольким из перечисленных выше рекомендаций способно существенно снизить вероятность взлома базы данных и утечки конфеденциальной информации.

Назначение прав доступа на таблицы, поля и процедуры

В SQL права доступа разграничиваются на уровне таблицы или представления при помощи привелегий: списка операций, которые пользователь может выполнять над данной таблицей или представлением. Команда GRANT позволяет назначить привелегии на доступ к таблице или представлению указанным пользователям, роли или объектам, таким как хранимые процедуры и триггеры. С помощью GRANT так же можно предоставлять права пользователям или хранимым процедурам на выполнение процедур (привелегия EXECUTE), или позволять пользователям использовать определенную роль. Для удаления привелегий, ранее назначенных с помощью команды GRANT, служит команда REVOKE.

Права доступа по умолчанию

Создаваемые таблицы, представления и хранимые процедуры изначально защищены от несанкционированного доступа. Только создатель (называемый так же владельцем) или SYSDBA имеют доступ к этим объектам. Они же могут использовать команду GRANT для того, чтобы открыть доступ другим учетным записям или ролям.

Доступные привелегии

Ниже приводится список привелегий, которые могут быть назначены с помощью команды GRANT или отозваны с помощью команды REVOKE:

Привелегия	Доступ
ALL	Чтение, вставка, изменение и удаление данных, а так же ссылка на первичный ключ.
SELECT	Чтение данных.
INSERT	Добавление новых данных.
UPDATE	Изменение существующих данных.
DELETE	Удаление данных.
EXECUTE	Выполнение хранимой процедуры.

[23] В равной степени сказанное можно отнести и к операционным системам Windows NT 4.0, Windows 2003 Server и Windows XP. Установка прав доступа на файл возможна только в файловой системе NTFS.

[24] Достаточно предоставить права на данный файл (директорию) учетной записи SYSTEM и глобальной пользовательской группе Администраторы. Кроме файла базы данных необходимо защитить файл с информацией об учетных записях isc4.gdb.

[25] Доступно в Windows 2000 Server и более новых версиях этой операционной системы при использовании файловой системы NTFS.

REFERENCES	Ссылка на первичный ключ.
роль	Все привелегии, назначенные роли.

С помощью ключевого слова ALL можно назначить привелегии SELECT, INSERT, UPDATE, DELETE и REFERENCES. ALL не позволяет назначить привелегию EXECUTE, равно как и предоставить право на использование роли. Привелегии SELECT, INSERT, UPDATE, DELETE и REFERENCES могут назначаться как по-одной, так и в произвольной коомбинации. Следует заметить, что команда, которая назначает или отзывает привелегию EXECUTE или дает пользователю право на использование указанной роли не может назначать или отзывать никакие другие привелегии.

Роли

SQL позволяет назначать права группам пользователей через использование механизма ролей. Использование роли подразумевает выполнение четырех шагов:

1. Создание роли с помощью команды CREATE ROLE;
2. Назначение привелегий созданной роли с помощью команды GRANT <привелегия> TO <роль>;
3. Назначение роли определенному пользователю с помощью команды GRANT <роль> TO <пользователь>;
4. Указание роли совместно с именем учетной записи и паролем при подключении к базе данных;

Назначение прав доступа на таблицу

Права доступа могут быть назначены как целиком на конкретную таблицу или представление, так и на отдельные их колонки. Для того, чтобы назначить права необходимо задать следующие параметры:

- Привеления доступа (или их список);
- Таблица или представление;
- Имя пользователя, которому даются права доступа.

Привелегией может быть одно или несколько значений из множества: SELECT, INSERT, UPDATE, DELETE и REFERENCES. В качестве привелегии может быть так же указано имя роли. В этом случае указанному пользователю будут предоставлены все права, присвоенные данной роли.

Формальная спецификация команды GRANT

```
GRANT{
< privileges> ON [TABLE] { tablename | viewname}
TO { <object>|<userlist> | GROUP UNIX_group}
|<role_granted> TO {PUBLIC | < role_grantee_list>}};
< privileges> = {ALL [PRIVILEGES] | < privilege_list>}
< privilege_list>={
SELECT
| DELETE
| INSERT
| UPDATE [( col [, col …])]
| REFERENCES [(col [, col …])]
[, < privilege_list>…]}}
<object> ={
PROCEDURE procname
| TRIGGER trigname
|VIEWviewname
| PUBLIC
[, <object> …]}
<userlist> ={
[USER] username
| rolename
| UNIX_user}
[, <userlist>…]
[WITH GRANT OPTION]
< role_granted> = rolename [, rolename …]
```

```
< role_grantee_list> = [USER] username [, [USER] username …]
[WITH ADMIN OPTION]
```

Следующая команда предоставляет привелегию SELECT на таблице GD_CONTACT пользователю Alex:

```
GRANT SELECT ON GD_CONTACT TO ALEX;
```

Следующий пример присваивает привелегию REFERENCES на таблице GD_CONTACT пользователю Julia. После чего, Julia сможет создавать внешние ссылки (FOREIGN KEY) на первичный ключ таблицы GD_CONTACT, даже если у нее нет прав на эту таблицу.

```
GRANT REFERENCES ON GD_CONTACT(ID) TO Julia;
```

Назначение прав доступа на отдельные колонки таблицы

В дополнение к назначению привелегий доступа для всей таблицы вы можете назначать привелегии на отдельные колонки. Для этого необходимо сразу за наименованием привелегии указать в круглых скобках список имен колонок разделенных запятыми. Например, следующая команда предоставляет права на обновление значений в полях PHONE и FAX таблицы GD_CONTACT для пользователя Alex:

```
GRANT UPDATE (phone, fax) ON GD_CONTACT TO Alex;
```

С помощью команды GRANT вы можете только добавлять права на отдельные колонки к тем правам, которые были установлены на уровне таблицы. Для отзыва прав следует использовать команду REVOKE.

Назначение прав доступа для процедур или триггеров

Может сложиться такая ситуация, когда процедра, триггер или представление нуждаются в доступе к таблице, которая имеет другого владельца. Для предоставления прав, поместите ключевое слово PROCEDURE перед наименованием процедуры в команде GRANT. Аналогично, с помощью ключевых слов TRIGGER или VIEW, вы можете назначить права для триггера или представления.

Следующая команда предоставляет доступ процедуре MONEY_TRANSFER к таблице ACCOUNTS:

```
GRANT INSERT ON ACCOUNTS TO PROCEDURE MONEY_TRANSFER;
```

Обратите внимание, что если у некоторого пользователя есть права на выполнения процедуры, а у этой процедуры, в свою очередь, есть права на доступ к некоторой таблице, то пользователь сможет выполнять процедуру, даже если у него не будет прав на эту таблицу.

Множественные привелегии

В данном разделе обсуждается возможность назначения нескольких привелегий в одной команде GRANT и возможность назначения прав доступа нескольким пользователям или объектам одновременно.

Предоставление нескольких привелегий одновременно

Для того, чтобы предоставить пользователю несколько привелегий одновременно перечислите их в команде GRANT, разделив с помощью запятых:

```
GRANT INSERT, UPDATE, DELETE ON GD_CONTACT TO Alex;
```

Предоставление всех привелегий

Для того, чтобы предоставить привелегии SELECT, INSERT, UPDATE, DELETE и REFERENCES за один раз, используйте ключевое слово ALL в команде GRANT:

```
GRANT ALL ON GD_CONTACT TO Alex;
```

Назначение привелегии EXECUTE требует выполнения отдельной команды GRANT.

Предоставление привелегий нескольким пользователям

Можно назначить привелегии нескольким пользователям одновременно или даже всем пользователем системы. В первом случае, список имен пользователей, разделенных запятыми, должен быть указан в команде GRANT. Во втором — следует использовать ключевое слово PUBLIC при назначении привелегий.

Следующая команда предоставит права доступа на вставку записей в таблицу GD_CONTACT пользователям Alex, Julia и Petr:

```
GRANT INSERT ON GD_CONTACT TO Alex, Julia, Petr;
```

Следующая команда предоставит все привелегии доступа на таблицу GD_CONTACT для всех пользователей:

```
GRANT ALL ON GD_CONTACT TO PUBLIC;
```

Предоставление привелегий списку процедур

Для назначения привелегий доступа сразу нескольким процедурам следует перечислить их имена в команде GRANT после ключевого слова PROCEDURE:

```
GRANT INSERT, UPDATE ON ACCOUNTS TO PROCEDURE MONEY_TRANSFER, ACCT_MAINT;
```

Использование ролей для предоставления привелегий

Назначение прав доступа группе пользователей при помощи роли требует выполнения четырех последовательных шагов:

1. Создать роль при помощи команды CREATE ROLE.
    a. CREATE ROLE <имя роли>;
2. Назначить одну или несколько привелегий созданной роли используя команду GRANT.
    a. GRANT <список привелегий> ON <имя таблицы> TO <имя роли>;
3. Использовать команду GRANT еще раз для назначения роли одному или нескольким пользователям.
    a. GRANT <имя роли> TO <список пользователей>;
4. При подключении к базе данных указать имя роли наравне с именем пользователя и паролем.
    a. CONNECT ‘database’ USER ‘username’ PASSWORD ‘password’ ROLE ‘rolename’;

Для отзыва привелегий роли или исключения роли из списка ролей назначенных пользователям следует использовать команду REVOKE.

Предоставление привелегий роли

Синтаксис команды предоставления привелегий роли аналогичен синтаксису предоставления привелегий пользователю или группе пользователей:

```
GRANT <privileges> ON [TABLE] {tablename | viewname}
TO rolename;
<privileges> = ALL [PRIVILEGES] | <privilege_list>
<privilege_list> = {
SELECT
| DELETE
| INSERT
| UPDATE [(col [, col …])]
| REFERENCES [(col [, col …])]
}
[, <privilege_list> …]
```

Назначение роли пользователям

После того, как роль создана и для нее назначены права доступа на некоторые таблицы или представления, можно присвоить эту роль одному или нескольким пользователям. Синтаксис команды следующий:

```
GRANT {rolename [, rolename …]} TO {PUBLIC
| {[USER] username [, [USER] username …]} }[WITH ADMIN OPTION];
```

Если указать WITH ADMIN OPTION, то пользователь получит так же право назначать данную роль другим пользователям сервера базы данных.

Следующий пример иллюстрирует использование ролей. В нем сначала создается роль R, которая затем наделяется всеми правами на таблицу GD_CONTACT, после чего правом использовать данную роль наделяется пользователь Alex, который, в результате получает права SELECT, INSERT, UPDATE, DELETE и REFERENCES на таблицу GD_CONTACT.

```
CREATE ROLE R;
GRANT ALL ON GD_CONTACT TO R;
GRANT R TO Alex;
```

Предоставление пользователям прав на назначение привелегий

Изначально, только создатель таблицы или SYSDBA имеют права на назначение привелегий доступа к этой таблице другим пользователям. Использование секции WITH ADMIN OPTION в команде GRANT позволяет передать право на назначение привелегий доступа другим пользователям.

Пример вызова команды:

```
GRANT SELECT ON GD_CONTACT TO Alex WITH ADMIN OPTION;
```

Мы будем называть команду по назначению прав доступа некоторому пользователю авторизацией. Соответственно, секция WITH ADMIN OPTION в команде GRANT позволяет присвоить некоторому пользователю право на авторизацию.

WITH ADMIN OPTION нельзя использовать при присваивании прав процедуре.

Передача прав с помощью секции WITH ADMIN OPTION может аккумулироваться. Так пользователь Alex может получить право на передачу привелегии SELECT от отдного пользователя и — на INSERT от другого.

«Подводные течения» при назначении прав доступа

Учитывая, что назначенные пользователю права доступа аккумулируются, следует быть осторожным с предоставлением прав на авторизацию. Рассмотрим такой пример: пользователь Alex имеет права доступа на таблицу GD_CONTACT с возможностью авторизации. Пользователь Julia так же имеет права на GD_CONTACT с возможностью авторизации. Оба пользователя присваивают права пользователю Petr. Позже, Julia отзывает свои права и думает, что Petr теперь не имеет прав на доступ к таблице GD_CONTACT. Однако это не так. Права, выданные Alex-ом никто не отзывал и, следовательно, пользователь Petr сохраняет доступ к таблице GD_CONTACT.

Если произошла путаница у кого какие права на некоторую таблицу, то стоит от имени владельца этой таблицы отозвать права ото всех пользователей и затем назначить их по-новой.

Предоставление привелегий на выполнение процедуры

Для того, чтобы использовать процедуру, пользователь или другая процедура должны обладать привелегией EXECUTE, выданной с помощью команды GRANT следующего синтаксиса:

```
GRANT EXECUTE ON PROCEDURE procname TO {<object> | <userlist>}
<object> = {
PROCEDURE procname
| TRIGGER trigname
| VIEW viewname
```

```
| PUBLIC
}
[, <object> …]
<userlist> = {
[USER] username
| rolename
| UNIX_user
}
[, <userlist> …]
[WITH GRANT OPTION]
```

Необходимо предоставлять права на вызов процедуры другой процедуре или тригеру, если у них разные владельцы.

Следующая команда наделяет правом выполнения процедуры PROC пользователя Alex:

```
GRANT EXECUTE ON PROC TO Alex;
```

Отзыв привелегий доступа

Синтаксис вызова команды REVOKE похож на вызов команды GRANT. В качестве параметров требуется указать:

- Привелегию или список привелегий, которые отзываются;
- Имя таблицы или представления, на которые были выданы привелении доступа;
- Имя пользователя (список пользователей) или объект, от которого отзывается привелегия.

```
REVOKE <privileges> ON [TABLE] {tablename | viewname}
FROM {<object> | <userlist> | GROUP UNIX_group};
<privileges> = ALL [PRIVILEGES] | <privilege_list>
<privilege_list> = {
SELECT
| DELETE
| INSERT
| UPDATE [(col [, col …])]
| REFERENCES [(col [, col …])]
}
[, <privilege_list> …]
<object> = {
PROCEDURE procname
| TRIGGER trigname
| VIEW viewname
| PUBLIC
}
[, <object> …]
<userlist> = [USER] username [, [USER] username …]
```

Следующая команда отзывает права на просмотр таблицы GD_CONTACT у пользователя Alex:

```
REVOKE SELECT ON GD_CONTACT FROM Alex;
```

Ограничения команды REVOKE

Команда REVOKE подчиняется следующим правилам:

- Привелегии могут быть отозваны только тем пользователем, который их выдал;
- Другие привелегии, выданные другими пользователями останутся не тронутыми при выполнении команды REVOKE;

- Отзыв некоторой привелегии от пользователя A, которому ранее было дано право на назначение привелегий (авторизацию), автоматически отзывает эту привелегию ото всех пользователей, которым она была передана пользователем A;
- Привелегии выданные PUBLIC могут быть отозваны только у PUBLIC.

Отзыв множества привелегий

Одной командой REVOKE можно отозвать сразу несколько привелегий. Для этого их необходимо перечислить, разделив запятыми. Например:

```
REVOKE INSERT, UPDATE ON GD_CONTACT FROM Alex;
```

Отзыв всех привелегий

Используя ключевое слово ALL можно отозвать сразу все привелегии (кроме привелегии EXECUTE). Например:

```
REVOKE ALL ON GD_CONTACT FROM Alex;
```

Отзыв привелегий для списка пользователей

Как и при вызове команды GRANT, REVOKE позволяет указать список пользователей или ключевое слово PUBLIC, которое будет означать отзыв прав сразу у всех пользователей системы:

```
REVOKE ALL ON GD_CONTACT FROM PUBLIC;
```

После отзыва прав у всех пользователей только владелец таблицы сохранит полный контроль над ней.

Отзыв привелегий роли

Если вы назначили некоторые привелегии роли или назначили роль некоторым пользователям, вы можете использовать команду REVOKE для того, чтобы отозвать привелегии или назначенную роль.

Забрать привелегии у роли можно с помощью команды:

```
REVOKE <привелегии> ON <таблица> FROM <имя_роли>;
```

Отзыв роли для пользователей

Запретить пользователю использовать некоторую роль можно с помощью команды:

```
REVOKE <роль> FROM <пользователь или PUBLIC>;
```

Если удалить роль с помощью команды DROP ROLE, то все права, переданные пользователям посредством этой роли, будут отозваны.

Отзыв привелегий на выполнение процедуры

С помощью команды REVOKE можно отозвать право на выполнение процедуры (EXECUTE). Синтаксис команды следующий:

```
<object> = {
PROCEDURE procname
| TRIGGER trigname
| VIEW viewname
| PUBLIC
}
[, <object> …]
<userlist> = [USER] username [, [USER] username …]
```

Отзыв привелегий для объектов

При отзыве прав доступа у таких объектов, как процедуры, триггеры или представления следут перечислить их в соответствующей части команды REVOKE, предворив ключевым словом PROCEDURE, TRIGGER или VIEW.

Отзыв привелегий на назначение прав доступа

Права на авторизацию можно отозвать с помощью команды REVOKE следующего синтаксиса:

```
REVOKE GRANT OPTION FOR privilege [, privilege …] ON table
FROM user;
```

Например, следующая команда запретит пользователю Alex назначать любые права доступа на таблицу GD_CONTACT другим пользователям системы:

```
REVOKE GRANT OPTION FOR ALL ON GD_CONTACT FROM Alex;
```

Разграничение прав доступа на платформе Гедымин

Платформа Гедымин использует свою оригинальную систему разграничения прав доступа, которая частично базируется на механизмах сервера Interbase, и позволяет осуществлять:

- Аутентификацию пользователя;
- Разграничение прав доступа к функциям системы;
- Разграничение прав доступа к бизнес классам;
- Разграничение прав доступа к отдельным бизнес объектам (записям в данных конкретного бизнес класса);
- Аудит действий пользователя;
- Аудит изменений данных;
- Блокировку изменения данных в указанном временном периоде.

Разграничение прав доступа осуществляется на уровне групп пользователей. В свойствах каждого пользователя присутствует поле INGROUP типа INTEGER, которое хранит битовую маску групп, участником которых он является. Все группы пронумерованы. Наличие установленного бита в определенной позиции маски означает, что пользователь является членом группы с номером, соответствующим номеру позиции. Например, значение 45 поля INGROUP, в двоичном исчислении представляется как 101101, что означает, что пользователь входит в группы с номерами: 1, 3, 4, 6. Нумерация битов начинается с единицы и осуществляется справа налево. Поскольку, в настоящее время[26] тип данных INTEGER представляет собой 32-х битовое целое число, то количество групп ограничено 32-мя. Первые шесть групп с номерами от 1 до 6 являются системными и присутствуют изначально в эталонной базе данных. Остальные 26 групп могут быть созданы и использованы пользователем (настройщиком) по своему усмотрению.

Аутентификация пользователя

Для подключения к системе Гедымин надо иметь учетную запись и знать ее пароль. В отличие от сервера Interbase, все учетные записи Гедымина хранятся в самой базе данных в таблице GD_USER. Таким образом, существенно облегчается процесс переноса файла базы с одного сервера на другой, что особенно актуально при развертывании распределенной базы данных.

При создании учетной записи в системе Гедымин автоматически создается учетная запись на сервере Interbase. Ее имя и пароль генерируются системой. Этой учетной записи предоставляется право на использование роли ADMINISTRATOR, которая изначально определена в каждой базе данных Гедымина. Роли ADMINISTRATOR предоставлены полные права на доступ ко всем таблицам и процедурам. При подключении к серверу базы данных используется роль ADMINISTRATOR и учетная

[26] В будущем, при переходе на 64-битные вычисления, количество битов в типе INTEGER наверняка возрастет.

запись сервера Interbase, сопоставленная учетной записи платформы Гедымин. В процессе работы (за исключением момента аутентификации) Гедымин не использует средства SQL для разграничения доступа пользователей к таблицам и колонкам базы данных. Вместо этого используются внутренние механизмы платформы.

Важным элементом механизма безопасности платформы Гедымин является учетная запись сервера Interbase с именем STARTUSER и паролем startuser. Поскольку учетные записи Interbase не передаются вместе с базой данных, а хранятся в отдельной базе на сервере, то STARTUSER создается инсталлятором в процессе установки серверной части платформы. Если по какой-либо причине используется ручная установка или база данных размещается на компьютере, где уже установлен и сконфигурирован сервер, например, для работы с другими приложениями, то данную учетную запись необходимо создать вручную, либо с помощью утилиты командной строки gsec, либо с помощью любой доступной графической надстройки.

Единственным правом, которым обладает учетная запись STARTUSER является право выполнения процедуры аутентификации GD_P_SEC_LOGINUSER. Таким образом, хотя имя и пароль этой учетной записи известны всем, подключение под ней не представляет никакой опасности[27].

В базе данных присутствует процедура с именем GD_P_SEC_LOGINUSER, которая отвечает за аутентификацию пользователя. На вход этой процедуры передается наименование учетной записи платформы Гедымин и введенный пользователем пароль. Процедура возвращает код завершения (успех, или номер ошибки), а так же имя учетной записи Interbase, ее пароль и дополнительную информацию.

Рассмотрим подробно, как происходит загрузка платформы и ее подключение к серверу базы данных.

1. Запускается файл GEDEMIN.EXE;
2. Определяется имя сервера и путь к файлу базы данных:
    a. Если среди параметров командной строки присутствует ключ /sn, то его значение используется в качестве имени сервера и пути к файлу базы данных;
    b. Если в командной строке имя сервера и путь к базе данных не заданы, то из системного реестра считывается информация о последнем подключении;
3. С использованием имени сервера и пути к файлу базы данных осуществляется попытка подключиться используя учетную запись STARTUSER и ее пароль;
4. Если возникла ошибка, то проверяется ее код:
    a. Если сервер не сконфигурирован, нет учетной записи STARTUSER, то Гедымин предлагает ее создать. При этом запрашивается пароль учетной записи SYSDBA. После успешного создания учетной записи STARTUSER попытка подключения повторяется.
    b. Если указанной базы данных нет на сервере, то выдается предупреждающее сообщение и открывается окно выбора базы данных.
    c. При возникновении других ошибок соответствующие сообщения выводятся на экран.
5. Если подключение под учетной записью STARTUSER прошло успешно, то:
    a. если в параметрах командной строки заданы имя пользователя и его пароль, то эти значения используются для аутентификации пользователя. Диалоговое окно запроса пароля на экран не выводится.
    b. если в параметрах командной строки не заданы ни имя пользователя, ни его пароль или задан только один параметр, то из системного реестра считывается имя

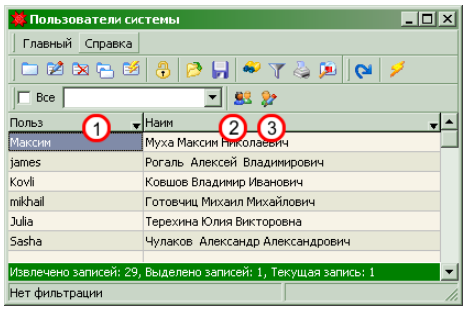
[27] Теоретически учетная запись STARTUSER может быть использована для подбора пароля методом перебора, но учитывая, что выполнение хранимой процедуры даже на производительном сервере занимает достаточный промежуток времени (порядка миллисекунды), а использование восьмисимвольного пароля, состоящего из цифр и букв английского алфавита дает порядка $36^8 \approx 10^{13}$ вариантов, то перебор всего диапазона займет более 300 лет.

учетной записи Гедымина, под которой последний раз было осуществлено подключение и на экран выводится окно для ввода пароля;

6. Если необходимо, на экран выводится диалоговое окно для ввода имени пользователя и пароля. В этот момент подключение к базе данных под учетной записью STATUSER по-прежнему активно.
    a. Если в течение трех минут пользователь не вводит имя пользователя и пароль, то диалоговое окно и подключение к базе данных закрываются. Пользователю предлагается либо завершить выполнение программы, либо продолжить загрузку платформы без подключения к базе данных.
    b. В процессе работы в диалоговом окне пользователь может выбрать другую базу данных из выпадающего списка, либо зарегистрировать новую базу данных. в любом случае, активное подключение закрывается и повторяются шаги 3-5.
7. Используя активное подключение вызывается процедура GD_P_SEC_LOGINUSER на вход которой передаются: имя учетной записи Гедымин и введенный пользователем пароль;
8. Процедура проверяет введенную информацию. По заданному имени пользователя в таблице GD_USER отыскивается запись:
    a. Если запись существует, и введенный пользователем пароль совпадает с тем, который хранится в записи, то возвращается имя пользователя Interbase, его пароль, а так же вспомогательная информация.
    b. Если переданы неверные данные или подключение невозможно, то возвращается соответствующий код ошибки. Например, такой учетной записи нет в базе данных, или запись заблокирована или введен неверный пароль и т.п.
9. Закрывается активное подключение к базе данных.
10. Происходит подлючение к базе данных с использованием учетной записи Interbase, ее пароля и роли ADMINISTRATOR.
    a. Если такой учетной записи Interbase нет на сервере, то, возможно, база данных была перенесена на другой сервер или сервер был переустановлен, т.е. файл с паролями isc4.gdb был заменен чистым файлом. Запрашиваем у пользователя пароль SYSDBA и пытаемся пересоздать учетную запись. Если получилось, отражаем изменения в таблице GD_USER.
    b. Если подключение прошло успешно, то на этом процедура аутентификации завершена. Осуществляется загрузка платформы.

Просмотр списка учетных записей

Список учетных записей вызывается командой Пользователи из раздела «Сервис»—«Администратор» Исследователя системы.



Обратите внимание, что на панели инструментов, кроме набора стандартных команд так же присутствуют:

1. фильтр, для отбора пользователей, входящих в определенную группу,
2. команда просмотра списка групп, в которые входит выбранная учетная запись,
3. команда пересоздания учетных записей Interbase для всех учетных записей Гедымин.

Действие, выполняемое последней командой нуждается в пояснении. Выше, мы уже говорили о том, что каждой учетной записи Гедымин соответствует учетная запись сервера Interbase, под которой происходит подключение к базе данных. Поскольку, список пользователей сервера Interbase хранится отдельно от базы данных в файле isc4.gdb, то при переносе базы на другой сервер (или при полной переустановке сервера) необходимо для каждой учетной записи платформы Гедымин создать соответствующую учетную запись сервера Interbase. Проще всего это сделать войдя в систему под учетной записью Administrator (она использует учетную запись SYSDBA, которая, как правило, всегда присутствует на сервере Interbase), открыть окно Пользователи и выполнить команду «Пересоздать всех пользователей Interbase».

Создание новой учетной записи

Для создания новой учетной записи необходимо вызвать соответствующую команду и заполнить поля в диалоговом окне.

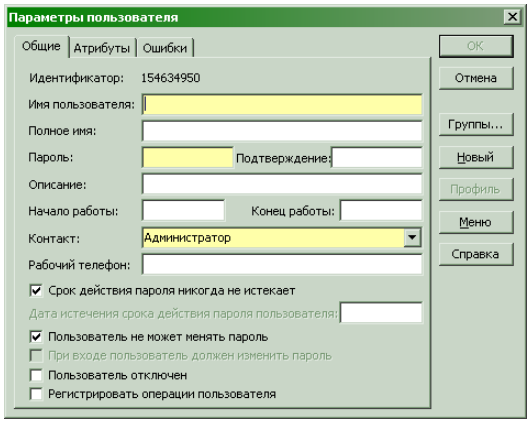


Рис. 95

Поля, доступные при создании новой учетной записи:

Поле	Комментарий
Имя пользователя	Наименование учетной записи пользователя платформы Гедымин.
Полное имя	Полное имя пользователя.
Пароль	Пароль, который будет запрашиваться при входе в систему.
Подтверждение пароля	Повторный ввод пароля, для исключения ошибки.
Описание	Произвольный комментарий.
Начало работы	Начало временного интервала, в течение которого пользователю разрешено входить в систему.
Конец работы	Окончание временного интервала, в течение которого пользователю разрешено входить в систему. Если оба поля пустые, то ограничений на время входа не налагается.
Контакт	Связь учетной записи с записью в списке контактов (людей). Важно для каждой учетной записи завести отдельную запись в справочнике людей (сотрудников предприятия). При создании или изменение записей в базе данных регистрируется кто произвел операцию. Причем фиксируется не ссылка на учетную запись[28], а ссылка на контакт, который связан с этой учетной записью. Поэтому, если несколько учетных записей ссылаются на одного и того же человека, то в последствии может возникнуть путаница с определением

[28] Очевидно, что учетная запись — информация не постоянная. Сегодня человек работает на предприятии, а завтра уволился и системный администратор удалил его учетную запись из базы.

	кто и что изменял.
Рабочий телефон	Поле только для чтения. Заполняется автоматически телефонным номером, взятым из записи контакта, связанного с данной учетной записью.
Срок действия пароля никогда не истекает	Если отмечено, то пароль действует всегда, пока существует данная учетная запись. Если не отмечено, то необходимо указать дату, до которой действует введенный пароль. При очередном входе в систему, если срок действия пароля истек, пользователю будет предложено ввести новый пароль.
Дата истечения срока действия пароля пользователя	Указывается дата, до которой действует пароль пользователя. Поле доступно только если флажок Срок действия пароля никогда не истекает.
Пользователь не может менять пароль	Если флажок установлен, то пользователь не может самостоятельно изменить свой пароль.
При входе пользователь должен изменить пароль	Если флажок установлен, то при входе пользователь должен ввести новый пароль.
Пользователь отключен	Если флажок установлен, то данный пользователь не может войти в систему.
Регистрировать операции пользователя	Если флажок установлен и включен аудит действий пользователя, то все операции (действия) совершенные пользователем регистрируются в системном журнале.

В правой части окна находится кнопка «Группы…» с помощью которой можно просмотреть список групп, в которые входит данный пользователь, изменить его.

Просмотр/изменение списка групп пользователя

Находясь в окне «Группы пользователя» можно просмотреть список групп, в которые входит выбранный пользователь, либо изменить его.

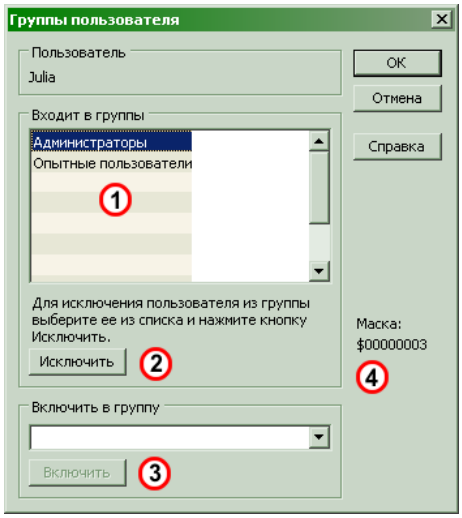


Рис. 96

Для того, чтобы исключить пользователя из группы, следует выбрать нужную группу в списке (1) и выбрать кнопку «Исключить» (2).

Для того, чтобы включить пользователя в группу, следует выбрать нужную группу из выпадающего списка внизу окна и выбрать кнопку «Включить» (3).

В правой части окна выводится шестнадцатиричное (4) число, соответствующее битовой маске групп, в которые входит пользователь.

Системные группы пользователей

Ниже приводится список из шести системных групп пользователей, имеющих предопределенное значение. Системную группу нельзя удалить.

Администраторы

Имеют максимальные права доступа ко всем функциям и данным системы. Единственное, чего не может сделать пользователь, входящий в группу Администраторы — это изменять структуру базы данных и, соответственно, активизировать настройки. Для этого предназначена учетная запись Administrator. Идентификатор группы 1.

Опытные пользователи

Пользователи с расширенными полномочиями. Могут изменять отчеты в режиме просмотра, перед отправкой их на печать. Идентификатор группы 2.

Пользователи

Простые пользователи системы. Идентификатор группы 3.

Операторы архива

Операторы архивного копирования. На равне с администраторами имееют право создавать архивные копии базы данных и производить восстановление из архива. Идентификатор группы 4.

Операторы печати

Операторы печати имеют право изменять отчеты в режиме просмотра перед отправкой их на печать. Идентификатор группы 5.

Гости

Гости системы имеют наименьшие полномочия.

Разграничение доступа к функциям системы

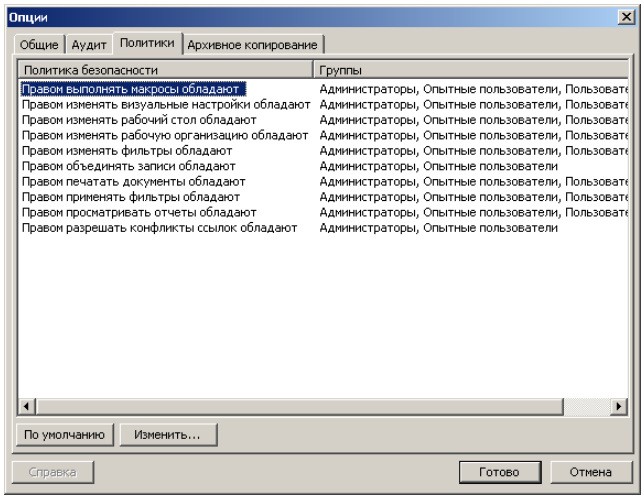
Доступ к некоторым функциям системы контролируется непосредственно в коде платформы. Например, открывать редактор скрипт-объектов или SQL редактор могут только участники группы Администраторы. Создавать архивные копии базы данных — Администраторы и Операторы архива и т.д. Более подробно, с предназначением каждой из системных групп можно ознакомиться в разделе «Системные группы пользователей».

Управлять доступом к остальным функциям можно с помощью политик групповой безопасности.

Политики групповой безопасности

Подобно тому как в операционной системе Windows можно назначить права как на отдельные файлы и папки, так и на функции операционной системы с помощью политик групповой безопасности, в Гедымине, так же можно разграничивать доступ к основным функциям платформы. Для этого предназначены политики групповой безопасности или просто политики безопасности.

Для просмотра списка политик необходимо открыть окно Опции в меню Сервис главного окна программы и перейти на вкладку Политики.



Как видно на рисунку вкладка содержит список доступных политик и список групп пользователей, на которых распространяется конкретная политика безопасности. Внизу списка располагаются две кнопки:

- По умолчанию — позволяет вернуть установки для данной политики к стандартным значениям, поставляемым вместе с изначальной базой данных;
- Изменить… — открывает окно, в котором можно указать на какие группы пользователей системы распространяется выбранная политика;

Доступны следующие политики групповой безопасности:

Правом выполнять макросы обладают

Только пользователи, входящие в указанные группы будут обладать правом на выполнение макросов из меню Макросы на форме просмотра.

В процессе настройки платформы может возникнуть ситуация, когда определенной группе пользователей необходимо дать доступ на выполнение только одного-двух макросов и запретить доступ ко всем остальным. Сделать это можно двумя способами:

1. Назначить права доступа для конкретных макросов в окне редактора скрипт объектов. В этом случае политику групповой безопасности задействовать не нужно;
2. Используя политику групповой безопасности запретить группе доступ к макросам в меню Макросы. Открыть на редактирование нужную форму и разместить дополнительную кнопку (кнопки) на панели инструментов для вызова нужных макросов. Присвоив событию OnClick этих кнопок следующий код:

Код…

Очевидно, что если первый способ более быстрый, то последний более гибкий и предоставляет гораздо больше возможностей. Например, в коде можно установить проверку не на группу пользователей, а на конкретную учетную запись, разрешив или запретив ей доступ на выполнение макроса.

Правом изменять визуальные настройки обладают

Данная политика определяет какие группы пользователей имеют право на:

- Изменение внешнего вида формы в дизайнере (вызывается по коомбинации клавиш Ctrl-Alt-E);
- Изменение визуальных настроек таблицы (соответствующий мастер вызывается по нажатию на F10 в таблице);
- Копированию настроек формы от другого пользователя системы.

Правом изменять рабочий стол обладают

Данная политика определяет какие группы пользователей имеют право на изменение рабочего стола, т.е. на закрытие или сворачивание форм, входящих в рабочий стол, создание новых рабочих столов либо удаление существующих.

Правом изменять рабочую организацию обладают

Как следует из названия политики, только указанные группы пользователей будут обладать правом выбрать рабочую организацию из выпадающего списка на главной форме Гедымина.

Правом изменять фильтры обладают

Только указанные группы пользователей будут иметь возможность создавать, удалять или изменять фильтры.

Правом объединять записи обладают

Только указанные группы пользователей будут обладать правом объединения записей.

Правом печатать документы обладают

Только указанные группы пользователей будут обладать правом отправлять документы на печать или экспортировать их в другие программы.

Правом применять фильтры обладают

Только указанные группы пользователей будут обладать правом выбрать и применить фильтр.

Правом просматривать отчеты обладают

Только указанные группы пользователей будут обладать правами на просмотр отчетов перед отправкой их на печать.

Правом разрешать конфликты ссылок обладают

Только указанные группы пользователей будут обладать правом разрешать конфликты ссылок в базе данных. Конфликты ссылок возникают, когда удаляется запись и в базе данных присутствуют другие записи, ссылающиеся на удаляемую.

Правом работать с эквивалентом

Только пользователи указанных групп будут иметь возможность просматривать и изменять значения сумм в эквиваленте в документах складского учета.

Разграничение доступа к бизнес классам

С помощью Исследователя системы можно разграничить уровни доступа для конкретного бизнес класса или группы классов. Для этого необходимо установить курсор на наименование класса и с помощью правой кнопки мыши открыть контекстное меню.

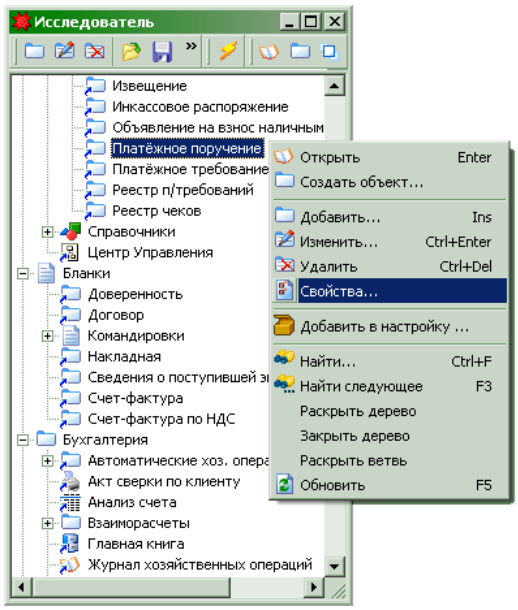


Рис. 97

В нем следует выбрать команду Свойства, как показано на рисунке выше. На экране появится диалоговое окно «Свойство объекта». Для установки прав доступа следует перейти на вкладку «Доступ».

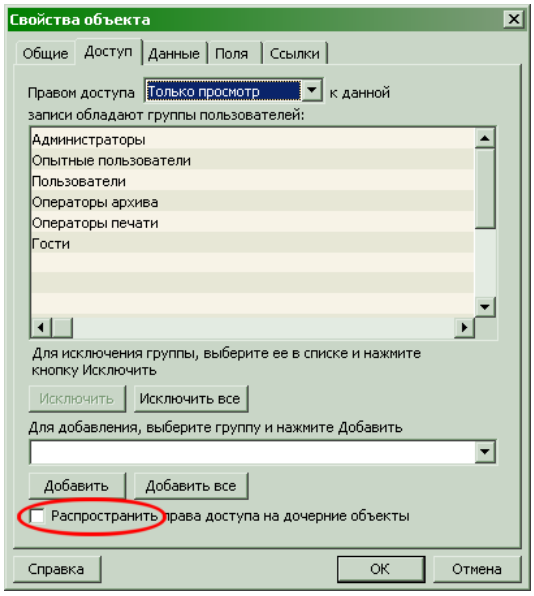


Рис. 98

В верхней части вкладки находится выпадающий список с тремя уровнями доступа:

- Только просмотр;
- Просмотр и изменение;
- Полный доступ;

В таблице, которая занимает основную часть вкладки отображается список групп пользователей обладающих указанным правом. Чтобы исключить группу из списка, следует установить на нее курсор и нажать кнопку «Исключить», которая располагается снизу таблицы. С помощью команды «Исключить все» можно быстро очистить список, оставив в нем только группу Администраторы, которая всегда обладает максимальными полномочиями.

Добавление группы в список осуществляется с помощью выпадающего списка групп и кнопок «Добавить» и «Добавить все». Первая кнопка добавляет выбранную группу, а вторая — все группы пользователей.

Обратите внимание на флаг «Распространить права доступа на дочерние объекты» в нижней части вкладки. Если флаг установлен, то права будут применены не только к выбранному объекту, но и ко всем вложенным объектам и папкам в Исследователе. Таким образом можно быстро настроить права доступа для группы классов. Например, вместо того, чтобы по отдельности устанавливать идентичные права доступа на «Платежное поручение», «Платежное требование», «Приходный кассовый ордер» и т.п. Можно установить курсор на ветку «Банк и касса» и настроить права доступа для нее с распространением на все дочерние объекты.

Команды, для которых у текущего пользователя нет вообще никакого доступа, в Исследователе не отображаются. Если у пользователя нет доступа на некоторую папку, но есть хотя бы минимальные права на входящие в эту папку команды, то они будут отображены в Исследователе, однако древовидная структура при этом будет нарушена. Обычно, такие команды будут выведены просто списком на самом верхнем уровне ирерархии. По этой причине, после настройки прав доступа в Исследователе, следует зайти в Гедымин под пользовательской учетной записью и убедиться, что окно Исследователя отображает корректную информацию.

Назначение прав доступа на отдельные поля

Разграничение доступа к данным

Настраивая доступ в Исследователе мы разрешаем или запрещаем определенные операции для всех записей одного класса. Но, что если для определенной группы требуется открыть доступ к части записей и закрыть к остальным? Гедымин предоставляет такую возможность, т.е. позволяет разграничивать права доступа на уровне отдельных записей в наборе данных.

Дескрипторы безопасности

Каким образом обеспечивается разграничение доступа для отдельных записей? Как известно, для каждого бизнес-класса существует таблица, которая содержит его данные. Если для этого класса поддерживается разграничение доступа на уровне записей, то в таблице будет присутствовать одно, два или три специальных поля, т.н. дескрипторы безопасности. Дескриптор безопасности — это целое число, содержащее битовую маску групп пользователей, обладающих определенным правом. Каким? Это зависит от наименования дескриптора безопасности:

- AFULL — Полный доступ;
- ACHAG — Доступ на просмотр и изменение данных (удаление и изменение прав доступа запрещены);
- AVIEW — Доступ только на просмотр данных.

Для того, чтобы скрыть от пользователя записи на просмотр которых у него нет прав, в SQL запрос для извлечения данных бизнес-класса добавляется следующее условие:

```
WHERE
  G_SEC_TEST ( Z.AVIEW,  - 1 )  <>  0
```

Второй параметр функции G_SEC_TEST — это целое число, представляющее битовую маску групп, в которые входит текущий пользователь. Так, например, если пользователь является только членом группы с идентификатором 3 («Пользователи»), то в качестве параметра функции будет передаваться значение 2^(3-1)=2^2=4. Если текущий пользователь входит в группу «Администраторы», то в качестве параметра будет передаваться значение -1 (все 32 бита установлены).

Если заглянуть внутрь функции G_SEC_TEST, то мы увидим две очень простые операции:

```
G_SEC_TEST(X, Y) = (X OR 1) AND Y
```

Если с двоичным умножением (AND) все понятно — мы пытаемся определить входит ли пользователь хотя бы в одну группу из тех, которым разрешен доступ. То присутствие здесь оператора OR требует пояснения. Ранее, мы отмечали, что члены группы «Администраторы» всегда имеют доступ ко всем данным, поэтому перед выполнением побитового умножения, следует установить бит, соответствующий группе «Администраторы». Теперь, даже если доступ установлен только для группы «Пользователи», а текущий пользователь входит только в группу «Администраторы», функция G_SEC_TEST вернет значение отличное от 0 и доступ к записи будет открыт.

Инициализация дескрипторов безопасности

При создании новой записи дескрипторам безопасности присваиваются значения, установленные для этого класса в Исследователе. Инициализация полей происходит в методе OnNewRecord. Перекрыв данный метод, можно присвоить дескрипторам другие значения. Например, чтобы разрешить просмотр записи только пользователям из группы, в которую входит пользователь, создавший ее, следует после вызова наследованного метода вставить следующую строку:

```
…
Self.FieldByName('aview').AsInteger = IBLogin.InGroup
…
```

Если добавить такую строку к коду метода OnNewRecord документа «Счет-фактура» и разделить менеджеров на группы по региональному признаку, например, «Минск», «Брест», «Могилев» и т.д., то менеджеры каждого города будут видеть только свои счета.

Почему в Гедымине не используются представления для разграничения прав доступа?

У разработчика, хорошо знакомого с возможностями языка SQL может возникнуть вопрос: почему для разграничения доступа на уровне отдельных колонок и на уровне записей в Гедымине не используются представления (view), как это рекомендовано стандартом SQL?

Ответов на такой вопрос может быть несколько:

1. Если использовать представления для разграничения прав доступа на колоноки таблицы, то каждый раз, когда администратор захочет разграничить права для той или иной группы придется создавать отдельное представление.
2. Как правило, данные бизнес объекта хранятся не в одной, а в нескольких таблицах. Если представление основано на запросе из нескольких таблиц, то оно не может обновляться (read-only). И для модификации данных придется либо создавать отдельные триггеры, либо реализовывать иные нетривиальные механизмы.

Изменение прав доступа с помощью SQL

Мы добавили новую группу пользователей и хотим открыть ей доступ ко всем документам. Можно, конечно, заходить в каждый из документов, выбирать все записи в списке и устанавливать для них права, а можно выполнить задание гораздо быстрее. Для этого следует напрямую подкорректировать значения дескрипторов безопасности в таблице GD_DOCUMENT. С помощью функций BIN_AND и BIN_OR мы можем снимать или устанавливать нужные биты.

Пусть идентификатор добавленной группы равен 22. Битовая маска с одним установленным двадцать вторым битом соответствует целому значению 2^(22-1)=2^21=2097152.

Перейдем в редактор SQL и выполним следующую команду:

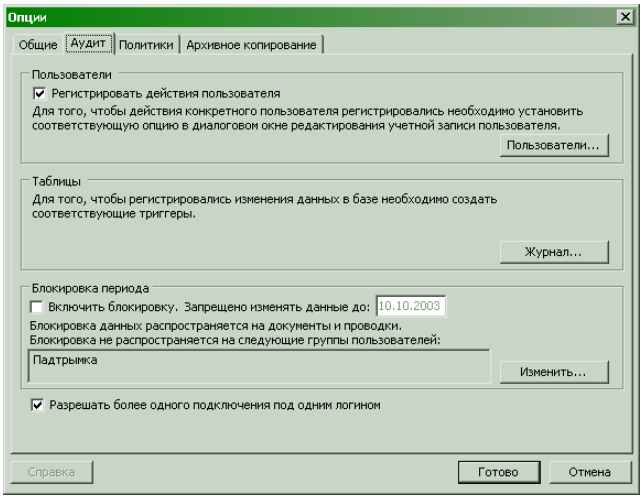
```
UPDATE GD_DOCUMENT SET AVIEW=BIN_OR(AVIEW, 2097152), ACHAG=BIN_OR(ACHAG,
2097152), AFULL=BIN_OR(AFULL, 2097152)
```

Если позже мы передумаем, то отозвать право доступа у группы можно с помощью команды:

```
UPDATE GD_DOCUMENT SET AVIEW=BIN_AND(AVIEW, BIN_XOR(2097152, 0)),
ACHAG=BIN_AND(ACHAG, BIN_XOR(2097152, 0)), AFULL=BIN_AND(AFULL,
BIN_XOR(2097152, 0))
```

Аудит действий пользователя

Механизмы разграничения прав доступа позволяют не дать пользователю доступ к определенным данным или право на выполнение определенных операций. Но что если и права вроде настроены правильно, а в системе происходит что-то неладное? Нужные документы «исчезают» или «самопроизвольно» изменяются и все пользователи отрицают свою причастность к этому? Помочь может журнал событий, в котором регистрируются операции, выполненные пользователями и изменения данных в базе. Обе опции выключены по умолчанию. Для того, что бы включить аудит действий пользователя необходимо: 1) в главном окне Гедымина выбрать меню Сервис и в нем команду Опции, в появившемся окне перейти на вкладку Аудит и установить флажок Регистрировать действия пользователя; 2) перейти в список пользователей системы, нажав кнопку Пользователи, открыть на редактирование каждую учетную запись, для которой мы хотим активизировать аудит и установить флажок «Регистрировать действия пользователя».



После этого, такие выполненные действия как: вход пользователя в систему, открытие и закрытие окон, создание, изменение, удаление объектов будут регистрироваться в журнале.

Просмотр журнала событий

Открыть журнал операций для просмотра можно через исследователь системы. Команда «Просмотр событий» находится в разделе «Сервис»—«Администратор».

Очистка журнала событий

Регистрация событий из макросов

Запись дополнительных событий из макросов производится с помощью метода AddEvent объекта IBLogin.

Аудит изменения данных

Регистрация изменений данных может быть полезна в следующих случаях:

- Необходимо установить контроль за действиями пользователей;
- Необходимо найти логическую ошибку в коде программы (настройки), которая проявляется в формировании неверных данных;
- Необходимо выполнить профилирование и оптимизацию программного кода, выявить наиболее часто выполняющиеся, неоптимальные участки;
- Необходимо отслеживать изменения в данных, производимые внешними программами.

Включение регистрации изменений

Для того, чтобы включить регистрацию изменений необходимо открыть Просмотр событий в разделе Администратор Исследователя. Далее в контекстном меню выбрать команду Создать триггеры.

Система предложит создать триггеры на все таблицы базы данных или на часть таблиц. Если выбрано второе, то откроется окно для выбора необходимых таблиц из списка.

Для каждой таблицы, изменения в данных которой будут фиксироваться, будут созданы три триггера: после вставки записи, после изменения записи и после удаления записи.

Триггеры будут заносить данные в таблицу GD_JOURNAL. При этом поле SOURCE будет заполнено именем таблицы, а поле DATA будет содержать наименование операции: "Вставка записи", "Удаление записи", "Изменение записи". Для изменения записи дополнительно будет занесено какие поля изменились и каким образом. Изменения полей фиксируются для полей всех типов, кроме БЛОБ полей.

Если таблица, является главной таблицей бизнес-объекта, т.е. имеет поле ID, то для такой таблицы в GD_JOURNAL, в поле OBJECTID будет заносится идентификатор записи (объекта) данные которого были изменены.

Триггеры не создаются для следующих таблиц:

- системных таблиц (имя начинается с RDB$);
- таблиц репликации (имя начинается с RPL);
- таблицы GD_JOURNAL;

Выключение регистрации изменений

Для того, чтобы отключить регистрацию изменений данных необходимо зайтив Просмотр событий в разделе Администратор Исследователя и выбрать в контекстном меню команду "Удалить триггеры".

Изменение структуры базы и регистрация

Перед изменением структуры базы данных необходимо отключить регистрацию изменений данных.

Блокировка периода

Блокировка периода распространяется на следующие таблицы:

- ac_entry — бухгалтерские проводки;
- gd_document — документы;
- inv_card — карточки складского учета;
- inv_movement — складское движение.

На каждую из этих таблиц в базе данных созданы по три триггера для отслеживания операций вставки, изменения и удаления записи. Триггеры не активны, если блокировка периода выключена и активизируются при ее включении.

В своем коде каждый триггер использует значения двух генераторов: gd_g_block и gd_g_block_group. Первый генератор содержит приведенную к целому числу дату, до кототорой блокируются данные, а второй — битовую маску групп пользователей, на которых не распространяется блокировка.

Триггер блокировки периода

Поскольку, все двенадцать триггеров однотипны, мы приведем здесь только один из них для того, чтобы проиллюстрировать как работает механизм блокировки периода.

```
CREATE TRIGGER AC_BI_ENTRY_BLOCK FOR AC_ENTRY
  INACTIVE
  BEFORE INSERT
  POSITION 28017
AS
  DECLARE VARIABLE IG INTEGER;
BEGIN
  IF (CAST(NEW.entrydate AS INTEGER) < GEN_ID(gd_g_block, 0)) THEN
  BEGIN
    IF (GEN_ID(gd_g_block_group, 0) = 0) THEN
    BEGIN
      EXCEPTION gd_e_block;
    END ELSE
    BEGIN
      SELECT ingroup FROM gd_user WHERE ibname = CURRENT_USER
        INTO :IG;
      IF (BIN_AND(GEN_ID(gd_g_block_group, 0), :IG) = 0) THEN
      BEGIN
        EXCEPTION gd_e_block;
      END
    END
  END
END
```

Обратите внимание на большой номер позиции триггера — 28017 — который гарантирует, что данный триггер будут выполняться после других триггеров изменяющих данные записи. Может возникнуть

вопрос, почему бы в качестве номера позиции триггера не использовать максимально допустимое сервером Interbase число? Сделано это специально для того, чтобы оставить пространство для триггеров (например, участвующих в механизме репликации), которые должны выполняться еще позже, чем триггер контроля блокировки периода.

Алгоритм, реализованный в триггере можно описать следующим образом: сначала проверяется дата записи. Если она попадает в заблокированный период, то проверим заданы ли группы пользователей, на которых блокировка не распространяется. Если такие группы не заданы, то генерируется исключение. Если группы заданы, то из таблицы GD_USER извлечем битовую маску групп, которым принадлежит текущий пользователь. Если ни одна из групп пользователя не входит во множество групп, на которых не распространяется блокировка, то генерируем исключение.

Обеспечение безопасности в случае локальной установки

При локальной инсталляции платформы Гедымин используется встроенный сервер Interbase, который допускает подключение под любой учетной записью. Точнее будет сказать, что при подключении к встроенному серверу учетная запись не проверяется вообще и любой подключившийся пользователь обладает максимальными правами доступа.

Прочие вопросы

Гедымин вместо Explorer

Права доступа и перенос данных

Разграничение прав на уровне пользователей

Метаданные подсистемы безопасности

Таблицы

GD_USER

Таблица используется для хранения списка учетных записей платформы Гедымин.

PK	FK	Поле Домен Тип данных	NN	Описание
1		ID DINTKEY INTEGER	☒	Первичный ключ записи.
		NAME DUSERNAME VARCHAR(20)	☒	Имя учетной записи.
		PASSW DPASSWORD VARCHAR(20)	☒	Пароль учетной записи. Хранится в незашифрованном виде. В версии 2.0 планируется хранить MD5 хэш значение.
		INGROUP DINTEGER INTEGER, DEFAULT 1	☒	Битовая маска групп, в которые входит данная учетная запись.
		FULLNAME DTEXT180 VARCHAR(180)		Полное имя пользователя.
		DESCRIPTION DTEXT180 VARCHAR(180)		Произвольный комментарий.

		IBNAME DUSERNAME VARCHAR(20)	☒	Имя пользователя Interbase. Хранится в незашифрованном виде. В версии 2.0 планируется шифровать это значение.
		IBPASSWORD DPASSWORD VARCHAR(20)	☒	Пароль пользователя Interbase. Хранится в незашифрованном виде. В версии 2.0 планируется шифровать это значение.
	FF	CONTACTKEY DINTKEY INTEGER	☒	Ссылка на таблицу GD_CONTACT. Ссылка на запись контакт, сопоставленную с данной учетной записью.
		EXTERNALKEY DFOREIGNKEY INTEGER		Вспомогательное поле. Может использоваться в случае эксплуатирования платформы Гедымин совместно с внешними программами. В этом случае в поле будет прописан ключ учетной записи во внешней программе, сопоставленной с учетной записью Гедымин.
		DISABLED DBOOLEAN INTEGER, DEFAULT 0		Запись отключена.
		LOCKEDOUT DBOOLEAN INTEGER, DEFAULT 0		Запись заблокирована.
		MUSTCHANGE DBOOLEAN INTEGER, DEFAULT 0		Пользователь обязан изменить пароль при следующем входе в систему.
		CANTCHANGEPASSW DBOOLEAN INTEGER, DEFAULT 1		Пользователь не может менять пароль.
		PASSWNEVEREXP DBOOLEAN INTEGER, DEFAULT 1		Срок действия пароля никогда не истекает.
		EXPDATE DDATE DATE		Дата истечения срока действия пароля.
		WORKSTART DTIME TIME		Начало временно интервала, в котором пользователю разрешен вход в систему.
		WORKEND DTIME TIME		Окончание временного интервала, в котором пользователю разрешен вход в систему.
		ALLOWAUDIT DALLOWAUDIT INTEGER		Разрешен аудит действий пользователя.
		EDITIONDATE DEDITIONDATE TIMESTAMP		Дата и время последнего изменения записи.
	FF	EDITORKEY DFOREIGNKEY INTEGER		Кто изменял запись. Ссылка на таблицу GD_CONTACT.
		ICON DINTEGER INTEGER		Вспомогательное поле. В настоящий момент не используется.

		RESERVED DINTEGER INTEGER		Зарезервировано.

GD_USERGROUP

Содержит список групп пользователей.

PK	FK	Поле Домен Тип данных	NN	Описание
1		ID DINTKEY INTEGER	☒	Уникальный идентификатор группы. Обратите внимание, что это поле не заполняется с помощью генератора gd_g_unique, как остальные первичные ключи в базе данных, а содержит целое число в диапазоне 1-32.
		NAME DNAME VARCHAR(60)	☒	Наименование группы пользователей.
		DESCRIPTION DTEXT180 VARCHAR(180)		Описание группы пользователей.
		DISABLED DBOOLEAN INTEGER		Группа отключена. Пользователи, входящие в отключенную группу не смогут войти в систему.
		ICON DINTEGER INTEGER		Поле не используется.
		RESERVED DINTEGER INTEGER		Поле зарезервировано для использования настройщиком или для будущих версий системы.

GD_JOURNAL

Содержит журнал (лог) действий пользователя и изменений данных.

PK	FK	Поле Домен Тип данных	NN	Описание
1		ID DINTKEY INTEGER	☒	Идентификатор записи.
	F	CONTACTKEY DFOREIGNKEY INTEGER		Ссылка на контакт пользователя, под которым была произведена операция.
		OPERATIONDATE DTIMESTAMP_NOTNULL TIMESTAMP	☒	Дата и время операции.
		SOURCE DTEXT40 VARCHAR(40)		Наименование источника изменения. Это может быть некоторая строка, например, «Client library», полное имя класса бизнес-объекта, имя таблицы и т.п.
		OBJECTID DFOREIGNKEY INTEGER		Идентификатор бизнес-объекта, если применимо, -1 — в противном случае.

		DATA DBLOBTEXT80_1251 BLOB		Произвольные данные, характеризующие произведенную операцию.

Сложные сценарии разграничения прав доступа

Как мы уже говорили выше, Гедымин позволяет заблокировать изменение документов, а так же бухгалтерских проводок и складского движения, оформленных до указанной даты. При этом, выборочно, могут быть заданы группы пользователей, на которых блокировка не распространяется. Встроенная функциональность системы не позволяет заблокировать период только для документов определенного типа. Попробуем преодолеть это ограничение. Рассмотрим следующую задачу:

Задача

На некотором предприятии, занимающемся оптовой торговлей товарами народного потребления, пользователи системы разделены на три группы:

- Руководство,
- Бухгалтерия,
- Менеджеры.

Предприятие использует стандартные настройки из поставки «Оптово-розничный склад». В документообороте участвуют следующие документы:

- Счет-фактура,
- Накладная на отпуск товара на сторону.

Конечно, наш пример достаточно условен. Реальное предприятие использует гораздо больше видов документов, но наша задача — научиться разграничивать права доступа и для ее выполнения будет вполне достаточно и счетов-фактур с накладными.

Необходимо разграничить права доступа следующим образом:

- Руководство имеет право только просматривать все документы;
- Бухгалтерия имеет полный доступ ко всем документам;
- Менеджеры имеют полный доступ к документам в текущем месяце. В более ранних периодах они имеют только доступ на изменение к счетам-фактурам и не имеют никакого доступа к накладным.

Решение

Блокировать изменение документов можно как на уровне базы данных, так и на уровне Гедымина. В первом случае необходимо создать тригеры на таблицы документов, проводок и складских остатков и складского движения, которые будут вызываться перед выполнением операций изменения, добавления или удаления данных и, при нарушении определенного условия, блокировать их. Во втором случае, следует перекрыть методы у бизнес-класса документа, для того, чтобы предотвратить нежелательное изменение данных. Каждый способ имеет свои положительные и отрицательные стороны. Реализация блокировки на уровне базы данных более трудоемкая задача, так как требует создания двенадцати тригеров: по одному тригеру для каждой из трех операций изменения данных для четырех таблиц, содержащих документы, бухгалтерские проводки складские остатки и складское движение[29]. Кроме этого, в случае реализации блокировки только на уровне базы данных, нет никакой возможности взаимодействия с пользователем до того, как он приступит к редактированию документа или в процессе этого редактирования. Т.е. пользователь сможет открыть диалоговое окно, отредактировать данные и только в момент сохранения их на экран будет выведено сообщение о том, что данный документ заблокирован и не может быть изменен. Сильной стороной блокировки на уровне базы данных является то, что данные будут защищены вне зависимости от того, как они изменяются: через методы бизнес-

[29] Стоит заметить, что в версии сервера Firebird 2.0 будут предусмотрены т.н. универсальные тригеры, вызываемые на все операции изменения для заданной таблицы. В этом случае, очевидно, количество необходимых тригеров для решения поставленной задачи может быть уменьшено с 12 до 4.

объекта, с помощью выполнения SQL запросов в макросах Гедымина или через внешнюю программу, такую, например, как IBExpert или EMS Interbase Manager. При реализации блокировки на уровне Гедымина мы сможем предупредить пользователя о том, что он может только просматривать данные и запретить нажатие кнопки «Ок» в диалоговом окне. Но, с другой стороны, не сможем запретить изменение данных напрямую из макроса или внешней программы.

Для решения данной задачи мы выберем смешанный вариант, реализовав код блокировки как в тригерах в базе данных, так и в коде макросов системы Гедымин.

Внимательно ознакомившись с условиями задачи мы можем увидеть, что разграничение прав доступа для групп Руководство (только просмотр) и Бухгалтерия (полный доступ) не требут дополнительного программирования и может быть реализовано имеющимися средствами платформы.

Разграничение прав для групп Руководство и Бухгалтерия

Откроем Исследователь системы. Найдем в нем раздел Торговля и установив курсор на документ, именуемый как «03. Отпуск товара на сторону (оптовая торговля)».

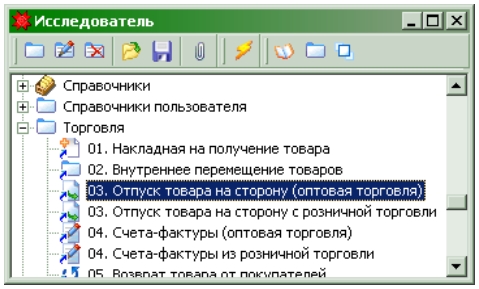


Рис. 99 Исследователь системы.

Нажмем правую кнопку мыши и из появившегося меню выберем команду «Свойства…». В появившемся диалоговом окне перейдем на вкладку «Доступ» и настроим права следующим образом:

- Только просмотр — Администраторы, Руководство, Бухгалтерия и Менеджеры;
- Просмотр и изменение — Администраторы, Бухгалтерия, Менеджеры;
- Полный доступ — Администраторы, Бухгалтерия и Менеджеры.

Расстановку прав доступа рекомендуется производить именно в таком порядке: от менее сильных прав к более сильным.

Аналогично настроим права доступа для документа «04. Счета-фактуры (оптовая торговля)».

Ограничение доступа для группы Менеджеры

Займемся теперь менеджерами. Задачу разграничения прав доступа для этой группы разобьем на следующие подзадачи:

- Запретить добавление и удаление счетов-фактур за пределами текущего месяца;
- Запретить добавление, изменение и удаление накладных за пределами текущего месяца;
- Запретить просмотр накладных за пределами текущего месяца.

Первые два пункта из нашего списка можно решить, наложив ограничения в тригерах на таблице GD_DOCUMENT.

Прежде чем перейти к созданию тригеров определим идентификаторы типов документов и групп пользователей. Для получения идентификаторов типов перейдем в Исследователь и в разделе «Сервис» выберем команду «Типовые документы». В древовидном списке, в левой части окна найдем раздел «Оптово-розничный склад». В правой части, установим курсор на позицию «04. Счета-фактуры (оптовая торговля)» как показано на следующем рисунке.

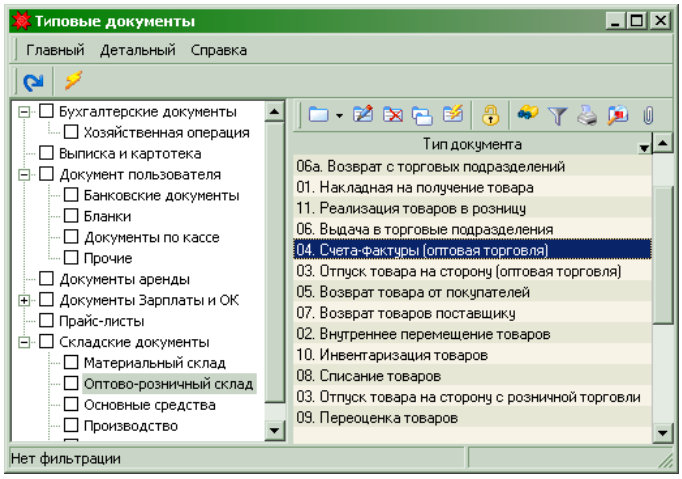


Рис. 100 Типы документов.

Нажмем правую кнопку мыши и выберем команду «Свойства…». Выпишем значение идентификатора записи. Аналогичную операцию проделаем для документа «03. Отпуск товара на сторону (оптовая торговля)». У нас получились следующие значения:

- Счет-фактура — 147043282;
- Накладная — 147043395.

Обратите внимание, что значения идентификаторов на каждой базе данных будут отличаться от приведенных выше! Поэтому, выполняя наш пример на своей базе данных вы получите другие значения. Неизменными от базы к базе будут РУИДы этих записей, но об их использовании мы поговорим отдельно.

Для получения идентификаторов групп пользователей перейдем в раздел «Исследователь»—«Сервис»—«Администратор» и выберем команду «Группы пользователей». Так же, как и для документов, воспользуемся командой «Свойства…» из контекстно-зависимого меню. Наши значения идентификаторов:

- Руководство — 7;
- Менеджеры — 9;

Как и в случае с идентификаторами типов документов, значения полученные на вашем экземпляре базы данных скорее всего будут отличаться от полученных нами.

Собственно, использовать непосредственно идентификаторы групп в тригерах мы не сможем. Нам нужна битовая маска, которую мы вычислим следующим образом:

$2^{7-1}+2^{9-1}=2^6+2^8=64+256=320$

В рассматриваемой нами задаче необходимо запретить менеджерам и представителям группы руководство вставку и удаление счетов-фактур и накладных за пределами текущего месяца. Кроме этого, руководству запрещается изменение счетов и накладных за пределами текущего месяца, а менеджерам запрещается изменение только накладных, но разрешается изменение счетов.

Создадим три тригера на таблице GD_DOCUMENT, которые будут вызываться, соответственно, перед вставкой, изменением и удалением записи. Если дата документа выходит за пределы текущего месяца и документ является счетом-фактурой или накладной и пользователь входит хотя бы в одну из «запрещенных» групп, то сгенерируем исключение (EXCEPTION). Обратите внимание, что мы не будем создавать своего объекта исключения, а воспользуемся GD_E_BLOCK — объектом присуствующим в эталонной конфигурации базы данных.

Для создания тригера необходимо открыть Исследователь системы. Далее последовательно откроем раздел «Сервис» и, находящийся в нем раздел «Атрибуты». Вызовем команду «Таблицы». На экране откроется список таблиц базы данных. Найдем в нем GD_DOCUMENT и откроем данную таблицу на редактирование в диалоговом окне. Перейдем на вкладку «Триггеры». В верхней части вкладки установим курор на позицию «Before Insert» и нажмем кнопку «Добавить».

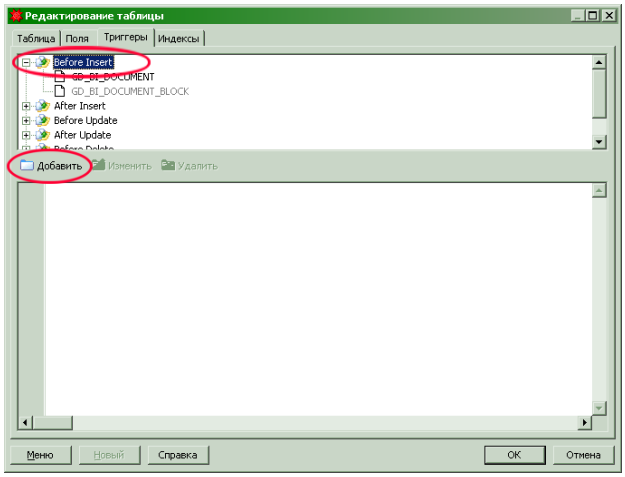


Рис. 101 Добавление триггера.

На экране откроется окно создания нового триггера.

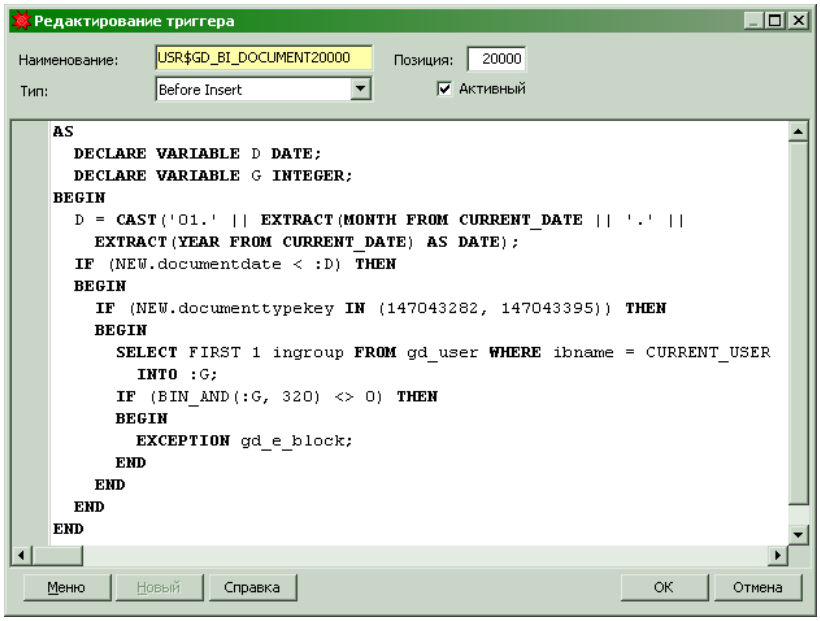


Рис. 102 Создание тригера.

Введем текст триггера и нажмем «Ок». Вышеуказанные операции повторим последовательно для создания триггеров перед изменением и перед удалением записи. Ниже приводятся исходные коды всех трех триггеров. Будьте внимательны! Они не идентичны. В триггере перед обновлением записи мы проверяем пользователя только на вхождению в группу Руководство (битовая маска 256), а в триггере перед удалением мы используем мета-переменную OLD, а не NEW.

Триггер Before Insert

```
AS
  DECLARE VARIABLE D DATE;
  DECLARE VARIABLE G INTEGER;
BEGIN
  D = CAST('01.' || EXTRACT(MONTH FROM CURRENT_DATE) || '.' ||
    EXTRACT(YEAR FROM CURRENT_DATE) AS DATE);
  IF (NEW.documentdate < :D) THEN
  BEGIN
    IF (NEW.documenttypekey IN (147043282, 147043395)) THEN
    BEGIN
      SELECT FIRST 1 ingroup FROM gd_user WHERE ibname = CURRENT_USER
        INTO :G;
```

```
      IF (BIN_AND(:G, 320) <> 0) THEN
      BEGIN
        EXCEPTION gd_e_block;
      END
    END
  END
END
```

Триггер Before Update

```
AS
  DECLARE VARIABLE D DATE;
  DECLARE VARIABLE G INTEGER;
BEGIN
  D = CAST('01.' || EXTRACT(MONTH FROM CURRENT_DATE) || '.' ||
    EXTRACT(YEAR FROM CURRENT_DATE) AS DATE);
  IF (NEW.documentdate < :D) THEN
  BEGIN
    IF (NEW.documenttypekey IN (147043282, 147043395)) THEN
    BEGIN
      SELECT FIRST 1 ingroup FROM gd_user WHERE ibname = CURRENT_USER
        INTO :G;
      IF (BIN_AND(:G, 64) <> 0) THEN
      BEGIN
        EXCEPTION gd_e_block;
      END
      IF ((BIN_AND(:G, 256) <> 0)
        AND (NEW.documenttypekey = 147043395)) THEN
      BEGIN
        EXCEPTION gd_e_block;
      END
    END
  END
END
```

Триггер Before Delete

```
AS
  DECLARE VARIABLE D DATE;
  DECLARE VARIABLE G INTEGER;
BEGIN
  D = CAST('01.' || EXTRACT(MONTH FROM CURRENT_DATE) || '.' ||
    EXTRACT(YEAR FROM CURRENT_DATE) AS DATE);
  IF (OLD.documentdate < :D) THEN
  BEGIN
    IF (OLD.documenttypekey IN (147043282, 147043395)) THEN
    BEGIN
      SELECT FIRST 1 ingroup FROM gd_user WHERE ibname = CURRENT_USER
        INTO :G;
      IF (BIN_AND(:G, 320) <> 0) THEN
      BEGIN
        EXCEPTION gd_e_block;
      END
    END
  END
END
```

Создание триггеров в базе данных произойдет только после того, как будет закрыто по кнопке «Ок» окно редактирования таблицы. При этом на экран будет выведено окно с ходом выполнения соответствующих SQL команд.

Осталось только скрыть от менеджеров накладные за пределами текущего месяца. Для этого, при формировании запроса для извлечения из базы данных списка накладных мы будем проверять входит ли текущий пользователь в группу «Менеджеры» и если да, то добавлять в часть WHERE запроса условие для отбора записей только после первого числа текущего месяца.

Реализуем вышеописанный алгоритм:

1. Откроем форму просмотра накладных;
2. Переведем ее в режим дизайнера нажав одновременно Ctrl-Alt-E на клавиатуре;
3. Найдем на форме компонент gdcInvDocument и выделим его;
4. В Инспекторе объектов перейдем на вкладку События;
5. Найдем в списке событие OnGetWhereClause и создадим обработчик для него;
6. В сгенерированный системой код обработчика добавим свои строки, как показано на рисунке ниже.

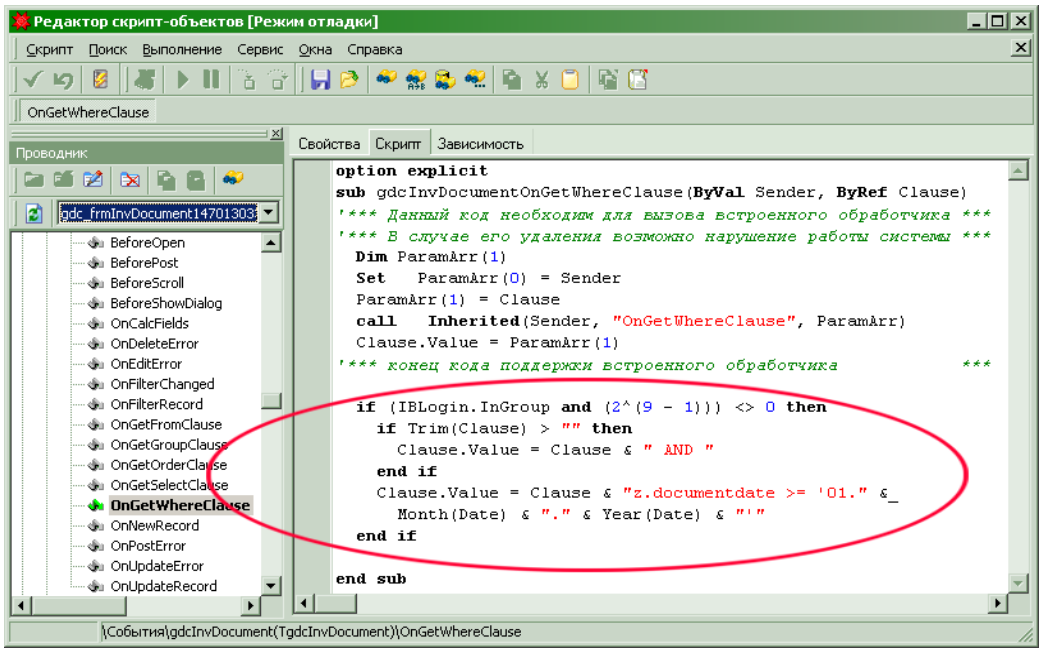


Рис. 103 Определение обработчика события.

Полный текст обработчика события приведен ниже. Добавленый нами код достаточно тривиален. Мы проверяем не установлен ли в битовом наборе групп в которые входит текущий пользователь бит, соответствующий группе «Менеджеры», если установлен, то добавляем к условиям секции WHERE ограничение на дату документа. При этом, если секция WHERE уже содержала некоторые условия, то присоединим новое условие через логический оператор AND.

```
option explicit
sub gdcInvDocumentOnGetWhereClause(ByVal Sender, ByRef Clause)
'*** Данный код необходим для вызова встроенного обработчика ***
'*** В случае его удаления возможно нарушение работы системы ***
  Dim ParamArr(1)
  Set   ParamArr(0) = Sender
  ParamArr(1) = Clause
  call   Inherited(Sender, "OnGetWhereClause", ParamArr)
  Clause.Value = ParamArr(1)
'*** конец кода поддержки встроенного обработчика            ***

  if (IBLogin.InGroup and 2^(9 - 1)) <> 0 then
    if Trim(Clause) > "" then
      Clause = Clause & " AND "
    end if
    Clause = Clause & "z.documentdate >= '01." &_
```

```
      Month(Date) & "." & Year(Date) & "'"
  end if
end sub
```

Тестирование

Осталось протестировать, созданный нами код. Для этого следует по-очередно войти в Гедымин под учетными записями Руководителя, Бухгалтера и Менеджера и попробовать создать, изменить и удалить документ как в рамках текущего месяца и за более ранние даты. Если все было сделано правильно, то при попытке выполнения недозволенной операции на экран должно быть выдано сообщение о возникновении исключения.

Третье приложение

Цель главы открыть перед разработчиком все аспекты системы Гедымин. Поэтому возьмемся за написание склада.

Постановка задачи

Разработка настройки автоматизации для документооборота оптово-розничной торговли. В самом начале нам необходимо определить объекты учеты, необходимые документы, используемые атрибуты. Основа документооборота оптово-розничной торговли составляют складские документы (документы осуществляющие перемещение и учет товаров), основные объекты учета – товары и контрагенты.

Атрибуты необходимы для учета товара

Фиксированные атрибуты (справочник ТМЦ)

Наименование ТМЦ	
Шифр ТМЦ	
Штрих код	
Ед. изм.	
Наценочная группа	
Вес товара	
Ед. изм. Отпуска	
Изображение	

Изменяемые атрибуты (карточка ТМЦ)

Счет	
Ссылка на позицию прихода	
ставка НДС прихода	
Позиция счета	
НДС прихода руб.	
Сертификат(старые)	
Контрактная цена(вал)	
Контрактная цена(руб)	
Цена учета руб	
Цена покупки(вал)	
Цена покупная(экв.)	
Цена покупная	
Цена(вал)	
Цена отпуска(экв.)	
Цена отпуска со всеми налогами	
Цена стеклопосуды	
Цена руб.	
Цена НДС(вал)	

Цена НДС руб.	
Цена налога на топливо входящий	
Оптовая цена(вал)	
Оптовая цена(экв.)	
Оптовая цена руб	
Цена поставщика	
Розничная цена(экв.)	
Розничная цена	
Цена налога с продаж входящий	
Цена с торговой надбавкой	
Цена с НДС и торг.надбавкой	
Цена отпуска с НДС(вал)	
Цена отпуска с НДС	
Валюта прихода	
Там.сбор.вал.часть(вал)	
Там.сбор вал.часть(руб)	
Там.сбор руб.часть(руб)	
Таможенные пошлины(вал)	
Таможенные пошлины(руб)	
Таможенный НДС(руб)	
Таможенное разрешение	
Дата поступления	
Срок годности	
Артикул	
Удостоверение ГТР	
Фиксированная цена	
Ссылка на позицию расхода	
Валюта расхода	
Условия поставки	
Оптовая надбавка	
Курс на дату расхода	
Страна происхождения	
Накладные расходы рублевые	
Накладные расходы(вал)	
Накладные расходы(руб)	
Упаковка	
Надбавка поставщика	

Надбавка отпуска	
Производитель	
Поставщик	
Качество	
Кол-во в упаковке	
Ссылка на заявку	
Курс на дату прихода	
Ставка НДС отпуска	
Акцизные марки	
Статистическая декларация	
Ставка налога с продаж	
Торговая надбавка	
Процент торг.надб.от покуп.цены	
Торг.надб.от отповой цены	
Трансп.расходы(вал)	
Трансп.расходы(руб)	
Транс.расх. по терр.РБ	
Сертификат	

Перечисление документов и форм, которые нам понадобятся

Складские документы

01. Накладная на получение товара

По этому документу на подразделении нашей организации регистрируется товар в количестве и его цене. В документе производится расчет оптовой и розничной цены. По данному документу возникает задолженность нашей организации перед поставщиком товара.

01.Накладная на получение импортного товара

Аналогично предыдущему документу с добавлением валютных цен, с возможностью указания таможенных расходов и алгоритмом распределения этих расходов по позициям накладной с включением их в стоимость товара.

02. Внутреннее перемещение товаров

По данному документу производится перемещение ТМЦ с одного подразделения нашего предприятия на другое, при этом ни какие другие атрибуты за исключением оптовой цены и балансового счета не изменяются

03. Отпуск товара на сторону

Оптовая торговля в рублях

По данному документу производится отпуск товара сторонней организации, при этом уменьшается количество товара на нашем подразделении и возникает задолженность организации перед нами. В данном документе должна быть возможность автоматического заполнения цены отпуска на основании сформированных цен в приходе или цен по выбранному прайсу, при этом пользователь должен иметь возможность изменять цену самостоятельно.

Розничная торговля в рублях

Аналогично предыдущему документу, но с фиксорованным алгоритмом формирования цены. Т.к. отпуск идет из розничной торговли, значит цена должна быть использована розничная с соответствующими расчетами налогов и округлениями.

Оптовая торговля на экспорт

Аналогично отпуску товара в оптовой торговли, но с валютными ценами, возникновением валютной задолженности, с возможностью отпуска без начисления НДС.

04. Счета-фактуры валютные(оптовая торговля)

04. Счета-фактуры из розничной торговли

04. Счета-фактуры (оптовая торговля)

05. Возврат товара от покупателей

06. Выдача в торговые подразделения

06а. Возврат с торговых подразделений

07. Возврат товаров поставщику

08. Списание товаров

09. Переоценка товаров

10. Инвентаризация товаров

11. Реализация товаров в розницу

Документы пользователя

Доверенность

Договор

Изменение надбавок

Прайс-лист

Прейскурант фиксированных цен

Оптовый прайс-лист

Справочники

Автомобили

Марки автомобилей

Сертификаты

Таможенное разрешение

Типы складов

Группы надбавок

Удостоверение ГТР

Условия поставки

Условия оплаты

Форма оплаты

Цели приобретения

Справочник качества продукции

Статистическая декларация

Типы транспортировки

Типы формирования цен

Проектирование базы данных (проектирование документов);

Проектирование пользовательского интерфейса;

Кодирование;

Как осуществляется отладка кода в системе Гедымин;

Запуск приложения:

Ввод остатков;

Повседневная эксплуатация;

Получение выходных данных;

Сопряжение с внешними программами

Вряд ли сегодня найдется предприятие, применяющее для автоматизации только один программный продукт. В реальной практике можно встретить такие случаи, когда бухгалтерия использует одну программу, отдел сбыта — другую, а руководство использует Microsoft Excel для анализа и обработки управленческой информации. Современная система должна как предоставлять доступ к своим данным и объектам внешним программам, так и уметь воспользоваться функциями внешних программ, извлекать данные из внешних источников.

В настоящей главе мы на конкретных примерах рассмотрим как с помощью Гедымина можно импортировать данные из текстовых файлов, работать с данными в формате XML, подключаться к COM серверам, а также к внешним базам данных с использованием технологии ADO.

Импорт данных из текстовых файлов

Хотя мы живем в эпоху XML, DCOM, ADO и прочих технологий, призванных максимально упростить передачу данных от одного приложения к другому, тем не менее, еще находятся программы, единственный способ взаимодействия с которыми — чтение или запись данных в текстовые файлы. Пример такой программы — электронная платежная система банк-клиент, широко распространенная в Беларуси. Клиентская часть системы соединяется с сервером в банке по модему и принимает информацию о движении денег по счетам, курсам валют и т.п. Информация записывается в текстовые файлы, имеющие определенную структуру.

Пусть перед нами стоит задача извлечь полученную информацию из текстового файла и поместить ее в базу данных. Решить задачу можно, что называется, «в лоб», написав макрос, который будет считывать текст из файла, анализировать его, извлекать данные и помещать их в базу. Однако, такой подход не самый удачный. Во-первых, банки любят периодически изменять формат выходного файла, в результате чего придется частично или полностью переписывать созданный макрос. Во-вторых, у каждого банка может быть свой формат данных. В-третьих, написать корректный парсер текста, так чтобы он аккуратно обрабатывал все возможные ошибки в данных, несоответствие формата файла и т.п. не так уж и просто. Решение данной задачи потребует как высокой квалификации разработчика, так и большого количества времени. К счастью, в Гедымин уже встроен инструментарий для автоматизации процедур импорта данных из текстовых файлов.

Работает механизм импорта следующим образом:

1. существует глобальный объект Converter, который позволяет считать с диска текстовый файл, содержащий информацию, разобрать его и поместить данные в специальную внутреннюю структуру, откуда, впоследствии, они могут быть записаны в базу или обработаны произвольным образом в макросе;
2. глобальному объекту Converter передается шаблон (своего рода инструкция на специальном языке), который определяет структуру текстового файла с данными;
3. для разбора текста и извлечения данных вызывается метод StartConvert, в который передается имя текстового файла. Данный метод разбирает текст, извлекает из него данные и помещает их во внутреннюю структуру;
4. после окончания разбора, данные из внутренней структуры перекачиваются в базу с помощью бизнес-объектов или компонент доступа к базе данных таких как: TIBSQL, TIBDataSet.

Схематически весь процесс можно изобразить следующим образом:

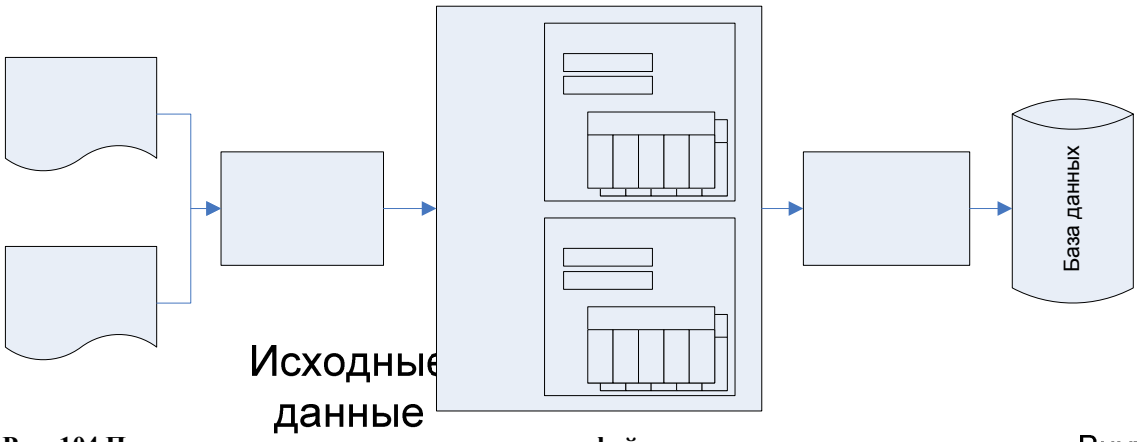


Рис. 104 Процесс импорта данных из текстового файла.

Рассмотрим компоненты процесса более подробно:

Шаблон

Шаблон — это последовательность инструкций на специальном языке, которая описывает структуру текстового файла с данными. Предполагается, что файл с данными организован следующим образом:

1. Начало данных помечено определенной последовательностью символов или метасимволов[30]. До начала данных может располагаться произвольный текст;
2. Данные можно структурировать (разбить) на области (Areas). Областей может быть одна или несколько.
3. Начало области задается последовательностью символов или метасимволом.
4. Каждая область имеет свое имя.
5. Внутри области может располагаться произвольное количество полей и таблиц.
6. Каждое поле имеет свое имя. Поле задается путем:
    a. указания его начала и указанием длины поля в символах,
    b. или указанием начала и указанием множества символов, которые может содержать данное поле,
    c. или указанием начала и указанием последовательности символов, которая завершает поле (терминатор поля)[31].
7. Началом поля считается первый левый символ, если указана ориентация поля слева-направо или крайний правый символ, если указана ориентация справа-налево. Если ориентация поля не указана непосредственно, то, по-умолчанию, принимается ориентация слева направо.
8. Начало поля задается:
    a. Либо указанием координаты первого символа поля по горизонтали относительно начала строки и по вертикали относительно текущей строки,
    b. Либо с помощью маркера.
9. Маркер — это последовательность символов или метасимвол. Маркер отыскивается в исходном тексте и курсор устанавливается на первый символ маркера. Дополнительно, при определении маркера можно указать:
    a. Что необходимо пропустить следующие за маркером N символов. В этом случае курсор устанавливается на N + 1 символ за маркером;

---

[30] Поддерживаются следующие метасимволы: BOF — начало файла, EOF — конец файла, BOL — начало строки, EOL — конец строки, EL — пустая строка.

[31] При разборе текстовых данных, сама указанная последовательность символов, завершающая поле в это поле не включается.

b. Что необходимо пропускать следующие символы, пока они соответствуют заданному символу или последовательности символов;

c. Что необходимо пропускать следующие за меркером символы, пока не встретится символ, входящий в заданное множество символов[32].

10. Каждая таблица имеет свое имя. Начало таблицы может задаваться определенной последовательностью символов или метасимволом.
11. Таблица содержит одинаковые по своей структуре записи (ноль, одну или несколько). Структура записи определяется перечислением полей[33].
12. Окончание таблицы может задаваться определенной последовательностью символов или метасимволом;
13. Окончание области может задаваться определенной последовательностью символов или метасимволом;
14. Окончание данных может задаваться определенной последовательностью символов или метасимволом;
15. Если окончание таблицы или области не задано явно, то текущая таблица или область считаются законченными, если за ними начинается следуюшая таблица или область.
16. Если окончание данных не задано явно с помощью последовательности символов или метасимволом, то конец файла считается концом данных.

Схематически, внутреннюю организацию файла с данными можно представить следующим образом:

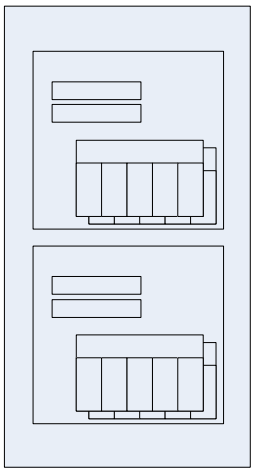


Рис. 105 Внутренняя организация текстового файла с данными.

Параметры обработки текста

Кроме описания структуры текстового файла с данными шаблон может содержать следующие параметры, определяющие как будет идти обработка текста:

- CodePage — Допустимые значения Oem или ANSI. Определяет кодовую страницу, в которой представлен текст в файле. Пример: CodePage: Oem. Кодировка OEM применяется для файлов созданных в DOS, ANSI — во Windows.
- TrimRight — Допустимые значения Yes или No. Определяет будут ли удаляться концевые пробелы из строк файла. Пример: TrimRight: Yes.
- TrimLeft — Допустимые значения Yes или No. Определяет будут ли удаляться начальные пробелы из строк файла. Пример: TrimLeft: No.

Начало данных

[32] Множество символов задается строкой. Например, множество цифровых символов можно задать строкой “0123456789”. Порядок символов в строке значения не имеет.

Область 1

[33] Одна запись таблицы может занимать произвольное количество строк в исходном текстовом файле.

Поле 1

Поле ...

Таблица

Таблица

- CaseSensitive — Допустимые значения Yes или No. Определяет будет ли учитываться регистр символов при распознавании меток начала области, таблицы, маркеров и т.п. Пример: CaseSensitive: Yes.

Каждый параметр должен располагаться на отдельной строке в файле шаблона.

Комментарии в тексте шаблона

В тексте шаблона допускается использование комментариев. Символ коментария — двойная косая черта — "//". Как и в Delphi, символ коментария помечает весь текст, следующий за ним, до конца строки как, собственно, коментарий.

Пример:

// Пример комментария.

Формальная спецификация шаблона

Ниже приводится формальная спецификация шаблона:

```
//Учитываем кодовую страницу в файле данных, по умолчанию OEM
CodePage: {ANSI| OEM}

//Указывает на то, удалять концевые пробелы или нет, по умолчанию Yes
TrimRight: {Yes|No}

// Указывает на то, удалять начальные пробелы или нет, по умолчанию No
TrimLeft: {Yes|No}

//Учитывать или не учитывать регистр при поиске маркеров, по умолчанию Yes
CaseSensitive: {Yes|No}

// Системные символы
// BOF -- пачатак файлу
// EOF -- канец файлу
// EOL -- канец радка
// BOL -- пачатак радка
// EL  -- пусты радок

  File           = BEGINFILE :<Chars>
                     [[<Area>][<Area>]...]
                   ENDFILE :<Chars>

  Chars          = { <System_Symbol> | <Literal_String> }
  System_Symbol  = { BOF | EOF | EOL | BOL | EL }
  Literal_String = "any chars"

  Area           = BEGINAREA :<Chars>
                     Name: <Literal_String>
                     [[<AreaItem>][<AreaItem>]...]
                   ENDAREA :<Chars>

  AreaItem       = { <Field> | <Table> }

  Marker         = BEGINMARKER
                     [[TEXT: <Literal_String>]
                     [WHOLEWORD: {Yes|No}]]
                     [SKIPCHARS: {EL|Literal_String}] |
                     [SKIPUNTILCHARS: <Chars>] |
                     [SKIPNEXTCHARS: int N]
                   ENDMARKER

  Field          = BEGINFIELD
```

```
                    NAME: <Literal_String>
                    SIZE: int Size
                    [MARKER: <Marker>]
                    [POSX: int X]
                    [POSY: int Y]
                    [NODATA: int]
                    [TERMINATOR: <Literal_String>]
                    [TRIMCHARS: <Literal_String>]
                    [LEGALCHARS: <Literal_String>]
                    [ALIGNMENT: <Alignment>]
                    [HEIGHT: int Height]
                  ENDFIELD

UserField       = BEGINUSERFIELD
                    NAME: <Literal_String>
                    SIZE: int Size
                  ENDUSERFIELD

Alignment       = { LEFT | RIGHT }

Table           = BEGINTABLE :<Chars>
                    NAME: <Literal_String>
                    [<Marker>]
                    [
                      [NECESSARILY: int ColumnIndex] |
                      [<RecordLabel>] |
                      [ENDRECORD: <Chars> ]
                    ]
                    [[Field][Field]...]
                  ENDTABLE :<Chars>

RecordLabel     = BEGINLABEL
                    POSX: int X
                    MASK: <Chars>
                  ENDLABEL
```

Пояснения по формальной спецификации шаблона

Данные

BEGINFILE :<Chars>	Задает последовательность символов, по которой в файле будет определяться начало данных.
[[<Area>][<Area>]...]	Файл содержит произвольное количество областей.
ENDFILE :<Chars>	Задает последовательность символов, по которой в файле будет определяться окончание данных.

Область

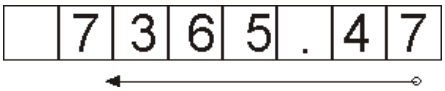
BEGINAREA :<Chars>	Задает последовательность символов, по которой в файле будет определяться начало области.
Name: <Literal_String>	Наименование области. Обязательный параметр.
[[<AreaItem>][<AreaItem>]...]	Область содержит произвольное количество полей и таблиц.
ENDAREA :<Chars>	Задает последовательность символов, по которой в файле будет определяться окончание области.

Маркер

BEGINMARKER	Начало определения маркера.
TEXT: <Literal_String>	Необязательный параметр. Текст для поиска в файле. Указатель становится на первый символ за текстом. На поиск влияет параметр WHOLEWORD, который указывает должен ли быть маркер отдельным словом в тексте, а так же установка CASESENSITIVE.
WHOLEWORD: {Yes|No}]	Необязательный параметр, используется в паре с параметром TEXT. См. выше.
SKIPCHARS: {EL|Literal_String}	Пропустить указанные символы. Параметр не обязательный. Если указан параметр TEXT, то указатель пропустит все указанные символы, следующие за заданной строкой, если параметр TEXT не указан, то указатель пропустит все символы начиная с текущей позиции.
SKIPUNTILCHARS: <Chars>	Необязательный параметр. Указатель перемещается, пока не встретятся указанные символы.
SKIPNEXTCHARS: int N	Необязательный параметр. Пропустить указанное количество символов.
	Все три параметра могут использоваться как по-отдельности, так и одновременно. Во втором случае, указанный в данной таблице порядок их появления в шаблоне должен быть соблюден.
ENDMARKER	Конец определения маркера.

Поле

BEGINFIELD	Начало определения поля.
NAME: <Literal_String>	Обязательный параметр. Имя поля.
SIZE: int Size	Обязательный параметр. Длина поля. При считывании поля из файла берется не больше символов, чем указанная длина поля. Может прочитаться и меньше, если указаны правила ограничения, о которых см. ниже.
MARKER: <Marker>	Необязательный параметр. Позволяет указать маркер, относительно которого будут считываться данные поля.
POSX: int X	Смещение данных поля в файле по оси X, относительно текущего положения указателя. Параметр не обязательный.
POSY: int Y	Смещение данных поля в файле по оси У, относительно текущего положения указателя. Параметр не обязательный.
NODATA: int	Необязательный параметр. Указывает на сколько позиций по границе выравнивания поля не должно быть значимых символов (т.е. на данных позициях могут располагаться пробелы или системные символы). Используется при описании взаимоисключающих полей.
TERMINATOR: <Literal_String>	Необязательный параметр. Последовательность символов, которая будет считаться границей поля. Поле будет считываться до тех пор, пока не встретится указанная последовательность символов или не будет считано количество символов, равное размеру поля.
TRIMCHARS: <Literal_String>	Необязательный параметр. Задает множество символов, которые будут удалены из поля после того, как оно будет прочитано.
LEGALCHARS: <Literal_String>	Необязательный параметр. Задает множество символов, которые могут встречаться в поле. Поле будет считываться из файла, пока встречаются символы, входящие в указанное множество или пока не будет считано максимальное количество символов.
ALIGNMENT: <Alignment>	Выравнивание поля в файле с данными. Слева-направо или справа-налево. По-умолчанию параметр имеет значение: слева-направо.

	Данный параметр определяет в какую сторону будет происходить движение указателя при считывании данных поля. Слева-направо:  Справа-налево: 
HEIGHT: int Height	Необязательный параметр. Высота поля. При использовании полей с варьируемой высотой (HEIGHT > 1) и при переходе их на другую строку (POSY > 1) необходимо указать обязательный столбец какого-либо поля в первой строке записи, с которым не будет пересекаться никакое другое поле. Если такой столбец указать невозможно, то считывание будет проходить некорректно.
ENDFIELD	Конец определения поля.
Пользовательское поле	
BEGINUSERFIELD	Начало определния пользовательского поля.
NAME: <Literal_String>	Обязательный параметр. Наименование поля.
SIZE: int Size	Размер поля.
ENDUSERFIELD	Конец определения пользовательского поля.
Таблица	
BEGINTABLE :<Chars>	Начало таблицы в данных помечено определенной последовательностью символов или метасимволом.
NAME: <Literal_String>	Наименование таблицы. Параметр обязательный.
[<Marker>]	Необязательный маркер, для перемещения курсора на начало таблицы.
[NECESSARILY: int ColumnIndex]	Необязательный параметр. Содержит номер обязательного столбца в первой строке записи. Т.е. в данном столбце в первой строке записи таблицы должны находиться какие-либо данные. Используется для определения первой строки любой записи таблицы.
[<RecordLabel>]	Метка записи. Параметр необязательный. Используется в таблицах, каждая запись которых занимает более одной строки в исходном файле.
[ENDRECORD: <Chars>]	Последовательность символом, которой помечается конец записи. Параметр необязательный. Применяется, в основном, при определении таблиц, каждая запись которых занимает более одной строки в исходном файле.
[[Field][Field]...]	Определение полей, составляющих запись таблицы.
ENDTABLE :<Chars>	Конец таблицы должен быть помечен определенной последовательностью символов или метасимволом.
Метка	
BEGINLABEL	Начало определения метки записи.
POSX: int X	Позиция в строке, с которой начинается запись.

MASK: <Chars>	Последовательность символов, определяющая начало записи.
ENDLABEL	Конец определения метки записи.

Глобальный объект Converter

Глобальный объект Converter предназначен для извлечения данных из текстовых файлов, имеющих определенную структуру.

Методы

- procedure LoadFromTempFile(ATempName: String)
  Загружает шаблон из текстового файла, заданного именем ATempName.
- procedure LoadTemplateFromStream(S: TStream)
  Загружает шаблон из потока, переданного параметром S.
- procedure StartConvert(ATextName: String)
  Разбирает переданный текстовый файл (конвертирует его данные во внутреннюю структуру). Имя файла передается через параметр ATextName. Перед вызовом процедуры в объект с помощью методов LoadFromTempFile или LoadTemplateFromStream должен быть загружен шаблон.

Свойства

- Database
  Объект, содержащий данные, извлеченные из текстового файла. Подробное описание см. ниже.

Объект Database

Объект Database представляет собой внутреннюю структуру, куда помещаются данные из влеченные из текстового файла после его разбора в соответствии с шаблоном. Объект содержит всего два свойства: массив областей и количество областей в массиве.

Свойства

- Areas(Index: Integer)
  Массив объектов, областей с данными.
- AreasCount: Integer
  Количество объектов в массиве Areas.

Объект Area

Свойства

- AreaFields: TClientDataSet
  Поля области. Хранятся в наборе данных, который содержит только одну запись. Обращение к каждому полю возможно по имени, как к обычному полю датасета: Converter.Database.Areas(0).AreaFields('FieldName').AsString.
- Name: String
  Наименование области. Задается в шаблоне.
- TableCount: Integer
  Количество таблиц в области.
- Tables(Index: Integer)
  Массив таблиц, содержащихся в области. См. объект Table ниже.

Объект Table

Свойства

- Name: String
  Наименование таблицы. Задается в шаблоне.

- Records: TClientDataset
  Записи, находящиеся в таблице. Представлены ввиде обычного TClientDataset.

Пример импорта

Поясним сказанное на конкретном примере. Пусть на некотором предприятии установлена система банк-клиент, которая ежедневно принимает выписки по рассчетному счету. Стоит задача импортировать эти выписки из текстового файла и записывать их в базу данных.

Образец выписки

Ниже приводится образец файла с выпиской, формируемого системой банк-клиент:

```
BEG
ЗДЧ   :7
ОТ    :153001369
КОМУ  :159051369
ДАТА  :971010
ИСХ   :0710100033000001
ТИП   : 199 /9/ОТВЕТ НА ЗАПРОС

:20: 10/10/91 11:52
:79: /9/ОТВЕТ НА КЛИЕНТСКИЙ ЗАПРОС



Лицевой по счету N 3012-006540016   за  10/10/91
ООО "ПРОЕКТ"
                                              Дебет / Кредит
Входящий остаток за 09/18/97                     13281945.39

01 369 3012-006210012      1                     100.00
01 369 3012-006210012      2 00100               200.00

Всего док-тов:   2     Обороты Дт:               300.00
                       Обороты Кт:                        0.00

Исходящий остаток за 10/10/97                    13281645.39
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

Состояние 99812 / 3012005640016 (К1) на 10/10/91

Срок акц  МФО    Счет КТ     N док              Сумма
09/22/97  720 3012100270089   3401          3533000.00
09/22/97  253 3012111111138    225          5154000.00
09/22/97  239 3012200250012 111501           760100.00
09/23/97  357 3012000160011   2552          1939000.00

Остаток  по 99812 за 10/10/97             11386100.00
Состояние 99814 / 3012005640016 (К2) на 10/10/91

Срок опл  МФО    Счет КТ     N док   Нпл  пеня       Сумма + пеня
07/18/97  715 3430210570035    557     0  0.00        2628268000.00
07/31/97  715 3782210260019    574     0  0.00            325000.00
07/21/97  386 3012003070010    198     0  0.00           2266300.00
07/21/97  730 3012212110014     91     0  0.00           3587400.00
07/23/97  807 3012000001037 507026     0  0.00           8453000.00
07/25/97  777 2103216630027  27747     0  0.00           4369000.00
07/28/97  357 3012009550026     35     0  0.00           9902600.00
07/29/97  777 2103216630027  28237     0  0.00           4512000.00
07/30/97  226 3012000785006 129592     0  0.00            961000.00
07/23/97  239 3012200250012 111501     0  0.00           2367400.00
```

```
07/30/97  764 3012206450017    103     0  0.00            300000.00
07/31/97  239 3012200250012 111501     0  0.00           1000200.00
08/20/97  720 3012100270063   3091     0  0.00         118528000.00
08/22/97  720 3012100270089   3368     0  0.00           3503000.00
08/22/97  239 3012200250012 111501     0  0.00           1941300.00
08/27/97  239 3012200250012 111501     0  0.00           2433100.00

Всего к оплате                                        2792717300.00
Остаток  по 99814/3012005640016(K2)  за 10/10/91     2792717300.00
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

Код банка 153001369     Время-11:52
Г.МИНСК ЦА БЕЛПРОМСТРОЙБАНКА


------------------------------------------------------------


END
```

Шаблон выписки

Построим шаблон, для извлечения данных из выписки:

```
// все концевые пробелы в каждой строке выписки следует удалить
TrimRight: Yes

// начальные пробелы не трогаем, оставляем как есть
TrimLeft:No

// выписка формируется и сохраняется в файле в DOS кодировке
CodePage: Oem

// поиск маркеров в тексте выписки будет производиться с учетом
// регистра
CaseSensitive:Yes

// Данные в файле начинаются после слова BEG
BeginFile: "BEG"

  // файл содержит выписки
  // каждая выписка начинается со слова "Лицевой"
  BeginArea: "Лицевой"

    Name: "Account Area"

    // выписка содержит два поля и таблицы, содержащие
    // записи о движении средств по счету и список документов
    // помещенных на картотеку

    // начнем с полей:

    // поле - расчетный счет, по которому сформирована выписка
    BeginField

      // расчетный счет располагается в тексте выписки
      // за символом N и следующими за этим символом
      // пробелами
      // расчетный счет имеет длину 15 символов
      // непосредственно за расчетным счетом располагается пробел
      BeginMarker
```

```
      Text: "N"
      SkipChars: " "
    EndMarker
    Name: "Account"
    Terminator: " "
    Size: 15
  EndField

  // поле - дата на которую сформирована выписка
  BeginField

    // дата располагается после предлога "за" и следующими
    // за ним пробелами
    // поле Дата может содержать цифры и знаки "." и "/",
    // как разделители частей даты
    // длина поля 12 символов
    BeginMarker
      Text: "за"
      SkipChars: " "
    EndMarker
    Name: "Date"
    LegalChars: "0123456789./"
    Size: 12
  EndField

  // таблица, содержащая записи о движении средств по
  // счету, начинается со строки "Входящий остаток"
  BeginTable: "Входящий остаток"
    Name: "StatementLines"

    // после подстроки, определяющей начало таблицы,
    // пропустим все символы до конца строки
    BeginMarker
      SkipUntilChars: EL
    EndMarker

    // поле - тип операции
    BeginField

      // поле занимает первые два символа строки
    BeginMarker
        Text: BOL
      EndMarker
      Name: "OpType"
      PosX: 2
      Size: 2
      Alignment: Right
    EndField

    // поле - код банка
    BeginField
      // поле начинается с 4-го символа от начала строки
      // и имеет длину 3 символа
      // обратите внимание, что так как в описании данного поля
      // нет никакой информации о перемещении курсора (маркере)
      // то положение курсора отсчитывается от маркера,
      // установленного предыдущим полем. т.е. от начала строки
      Name: "BankCode"
      PosX: 4
      Size: 3
    EndField
```

```
// поле - номер счета
BeginField
  // поле начинается с 8 символа о начала строки
  // обратите внимание, как формируются данные этого поля:
  // сначала считываются все символы, пока они входят
  // во множество "0123456789-", потом из считанных
  // данных убираются символы, входящие во множество "-"
  // (в данном случае - это один символ)
  Name: "Account"
  PosX: 8
  LegalChars: "0123456789-"
  TrimChars: "-"
  Size: 15
EndField

// поле - номер документа
BeginField
  // обратите внимание, что поле выровнено по правому краю
  // соответственно и мы указываем выравнивание вправо и,
  // в этом случае, PosX задает позицию крайнего правого
  // символа. В номер мы включим все символы, начиная от
  // 28 позиции, двигаясь влево, пока не встретится символ
  // пробела
  Name: "DocNumber"
  PosX: 28
  Terminator: " "
  Alignment: Right
  Size: 7
EndField

// сума по дебету
BeginField
  // поле имеет выравнивание вправо. крайний правый символ
  // занимает позицию 53. длина поля 15 символов.
  // число может быть записано с разделителями тысяч - пробелами,
  // поэтому мы указываем терминатор считываемых данных не один
  // символ, а строку из двух пробелов. после считывания данных
  // удалим из них все пробелы (т.е. все разделители тысяч),
  // чтобы можно было легко преобразовать строку в число.
  // обратите внимание на использование параметра nodata, равного
  // 8 символам. дело в том, что колонки Дебет и Кредит могут
  // пересекаться, если сумма по кредиту слишком большая.
  // т.е. число по Кредиту может залезть на 53-ю, 52-ю, 51-ю и т.д.
  // позиции. Для того, чтобы отличить, где данные принадлежат
  // полю Дебет, а где - Кредит, мы указываем, что после поля
  // Дебет должно идти хотябы 8 пробелов или строка должна
  // заканчиваться.
  Name: "Debet"
  PosX: 53
  Alignment: Right
  TrimChars: " "
  Terminator: "  "
  Size: 15
  Nodata: 8
EndField

// поле - сумма по Кредиту
BeginField
  Name: "Credit"
  PosX: 61
```

```
      Alignment: Right
      TrimChars: " "
      Terminator: "  "
      Size: 25
    EndField

  // таблица заканчивается пустой строкой
  EndTable: EL

  // после таблицы с информацией о движении средств по счету
  // может находиться таблица со списком документов, помещенных
  // на картотеку К1. таблица (если она есть) начинается
  // со строки "(К1) на"
  BeginTable: "(К1) на"
    Name: "K1"

    // пропускаем пустые строки, которые могут идти после начала
    // таблицы и строку заголовок таблицы.
    BeginMarker
      SkipUntilChars: "Сумма"
    EndMarker

    // поле - дата документа. Длина 10 символов, начиная с первого
    // символа строки.
    BeginField
      Name: "Term"
      PosX: 1
      Size: 10
      Alignment: Left
    EndField

    // поле - код МФО. 5 символов вправо от тринадцатой позиции
    // от начала строки.
    BeginField
      Name: "MFO"
      PosX: 13
      Size: 5
      Alignment: Right
    EndField

    // поле - номер счета
    BeginField
      Name: "Account"
      PosX: 27
      LegalChars: "0123456789-"
      Terminator: " "
      Size: 15
      Alignment: Right
    EndField

    // поле - номер документа
    BeginField
      Name: "DocNumber"
      PosX: 34
      Terminator: " "
      Alignment: Right
      Size: 7
    EndField

    // поле - сумма
    BeginField
```

```
      Name: "Summa"
      PosX: 53
      Alignment: Right
      TrimChars: " "
      Terminator: "  "
      Size: 18
    EndField

  // таблица заканчивается пустой строкой
  EndTable: EL

  // таблица с картотекой К2 начинается со строки "(К2) на"
  BeginTable: "(К2) на"
    Name: "К2"

    // пропускаем заголовок таблицы
    BeginMarker
      SkipUntilChars: "Сумма + пеня"
    EndMarker

    // поле - дата документа
    BeginField
      Name: "Term"
      PosX: 1
      Size: 10
      Alignment: Left
    EndField

    // поле - код МФО
    BeginField
      Name: "MFO"
      PosX: 13
      Size: 5
      Alignment: Right
    EndField

    // поле - номер счета
    BeginField
      Name: "Account"
      PosX: 27
      LegalChars: "0123456789-"
      Terminator: " "
      Size: 15
      Alignment: Right
    EndField

    // поле - номер документа
    BeginField
      Name: "DocNumber"
      PosX: 34
      Terminator: " "
      Alignment: Right
      Size: 7
    EndField

    // поле - Np1
    BeginField
      Name: "Npl"
      Size: 5
      PosX: 40
      Terminator: " "
```

```
        Alignment: Right
      EndField

      // поле -- пеня
      BeginField
        Name: "Penya"
        Size: 5
        PosX: 46
        Alignment: Right
      EndField

      // поле -- Сумма
      BeginField
        Name: "Summa"
        PosX: 67
        Alignment: Right
        TrimChars: " "
        Terminator: "  "
        Size: 20
      EndField

    // таблица заканчивается пустой строкой
    EndTable: EL

  // укажем строку, которой помечено окончание области
  EndArea: "--------------------"

// данные в файле оканчиваются строкой "END"
EndFile: "END"
```

Проверка шаблона

Проверим, насколько правильно сформирован шаблон для распознавания файла выписки. Пусть сама выписка сохранена в файле vyp.txt, а шаблон — в файле vyp.tem. Откроем Гедымин, в главном окне выберем пункт меню Сервис и в нем команду «Просмотр шаблона документов…». В появившемся окне введем полное имя файла с шаблоном и полное имя файла с выпиской и нажмем кнопку «Считать». Если шаблон не содержит ошибок и файл с выпиской содержит корректные данные, соответствующие шаблону, то текстовые данные будут считаны, разобраны и отображены в окне:

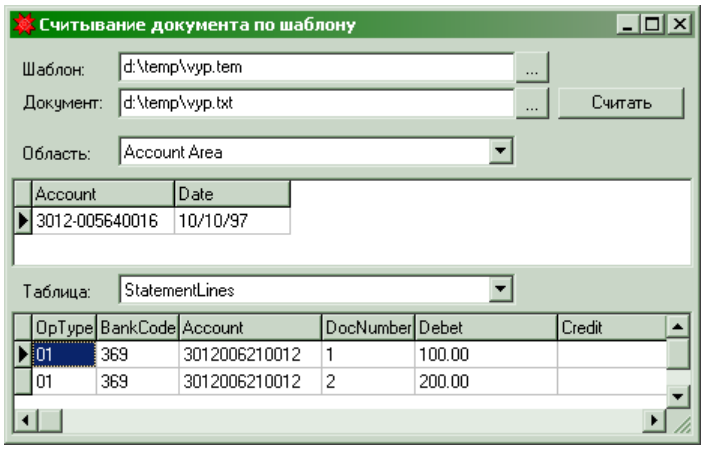


Рис. 106 Окно "Считывание документа по шаблону".

Используя данное окно можно просмотреть список областей, распознанных в указанном файле. Для каждой области можно ознакомиться со списком переменных и списком, содержащихся в ней таблиц.

Если шаблон содержит ошибки, то при нажатии на кнопку «Считать» на экран будет выведено соответствующее сообщение с пояснением ошибки и номером неверной строки.

Макрос импорта

После того, как составлен шаблон для разбора файла с выпиской, необходимо написать макрос, который будет извлеченные из текстового файла данные помещать в базу. Полнофункциональные макросы для импорта выписок в форматах различных банков входят в состав настройки «Банк и касса». При желании, вы можете ознакомиться с ними. Поскольку цель данной главы скорее состоит в том, чтобы предоставить общую информацию о принципах работы механизма импорта, нежели в том, чтобы вдаваться в конкретные подробности мы приведем очень простой пример. Создадим макрос, который будет загружать данные из текстового файла с выпиской, суммировать обороты по дебету и кредиту и выводить их на экран.

```
option explicit
sub ImportVyp

  ' загружаем в конвертер шаблон из файла
  ' обратите внимание! в вашем случае путь и имя файла
  ' могут быть другими.
  Converter.LoadFromTempFile("d:\temp\vyp.tem")

  ' разбираем файл с выпиской
  Converter.StartConvert("d:\temp\vyp.txt")

  dim I, D, C, K, DB

  D = 0
  C = 0

  set DB = Converter.Database

  ' пробегаемся по всем областям в файле
  for I = 0 to DB.AreasCount - 1

    ' в каждой области просматриваем все таблицы
    ' нас интересуют только те, которые называются "StatementLines"
    for K = 0 to DB.Areas(I).TableCount - 1
      if DB.Areas(I).Tables(K).Name = "StatementLines" then
        dim T
        set T = DB.Areas(I).Tables(K).Records

        ' пробегаем от первой до последней запис и суммируем значения
        ' по дебету и кредиту
        ' обратите внимание, что мы не обрабатываем незаполненные поля
        ' и как мы подготавливаем строковое значение (а при импорте
        ' из текстового файла все значения строковые) для конвертации
        ' в число. Поскольку в файле всегда используется разделитель
        ' целой и дробной части числа точка, мы заменяем точку на
        ' разделитель, установленный в нашей системе.
        ' если бы мы не сделали этого и на нашем компьютере
        ' была бы установлена запятая, то при выполнении данного
        ' макроса возникла бы ошибка
        T.First
        while not T.EOF
          if T.FieldByName("Debet").AsString > "" then
            D = D + CDbl(Replace(T.FieldByName("Debet").AsString, ".",_
              Application.DecimalSeparatorSys))
          end if

          if T.FieldByName("Credit").AsString > "" then
            C = C + CDbl(Replace(T.FieldByName("Credit").AsString, ".",_
              Application.DecimalSeparatorSys))
          end if
```

```
          T.Next
        wend
      end if
    next
  next

  MsgBox "Оборот по дебету: " & D & vbCrLf &_
    "Оборот по кредиту: " & C, vbOkOnly, "Импорт выписки"

end sub
```

Результат выполнения макроса:

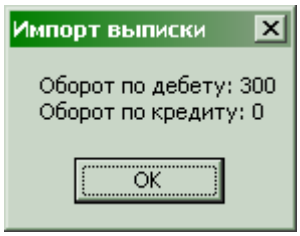


Рис. 107 Результат выполнения макроса.

Отправка электронной почты

```
Set iMsg = CreateObject("CDO.Message")
Set iConf = CreateObject("CDO.Configuration")
set Fields = iConf.Fields
  Fields("http://schemas.microsoft.com/cdo/configuration/smtpserver") =
"mail.tut.by" ' имя SMTP сервера
  Fields("http://schemas.microsoft.com/cdo/configuration/smtpserverport") =
2525 ' порт сервера для отправки сообщений
  Fields("http://schemas.microsoft.com/cdo/configuration/sendusing")  = 2 '
cdoSendUsingPort  ' CdoSendUsing enum value =  2
  Fields("http://schemas.microsoft.com/cdo/configuration/smtpaccountname")
= "KLN"
  Fields("http://schemas.microsoft.com/cdo/configuration/sendemailaddress")
= """KLN"" <KLN@tut.by>"
  Fields("http://schemas.microsoft.com/cdo/configuration/smtpauthenticate")
= 1 'cdoBasic
  Fields("http://schemas.microsoft.com/cdo/configuration/sendusername") =
"KLN"
  Fields("http://schemas.microsoft.com/cdo/configuration/sendpassword")
= "password"
  Fields.Update

' Using the CDO instance
With iMsg
     .Configuration = iConf
     ' Set who the email is going to
     .To = "KLN@TUT.BY"
     ' Set the subject
     .Subject = "Test"
     ' Set the Text of the message - I could have used the HTMLBody
property to send
     ' HTML e-mail
     .TextBody = "THIS is the test!"
     ' Send the mail
     .AddAttachment "C:\ORDER.XLS"
     .Send
     On Error Resume Next
End With
```

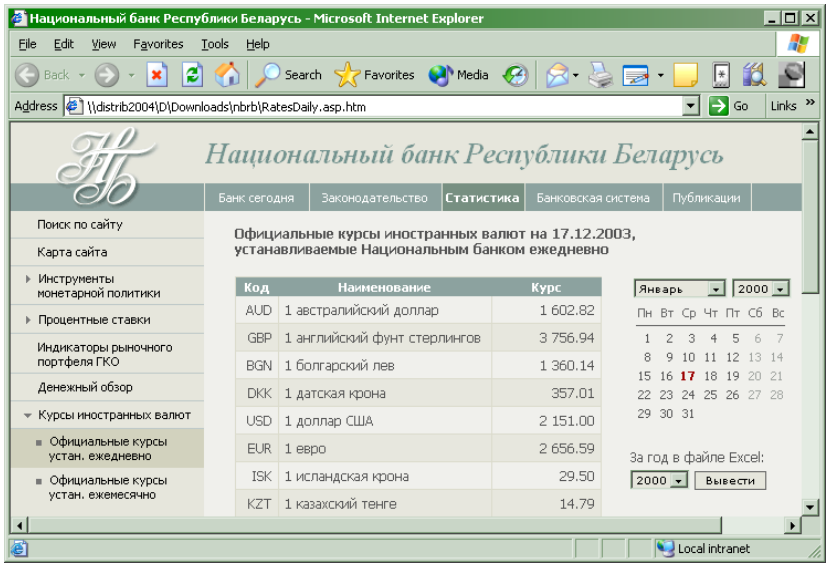
```
set iMsg = Nothing
set iConf = Nothing
```

Импорт данных с веб сайта

Часто возникает необходимость в извлечении информации с определенного веб сайта и помещении ее в базу данных. Рассмотрим пример организации импорта актуальных курсов валют с сайта Национального Банка Республики Беларусь.

Информация о курсах доступна по адресу[34]:
http://www.nbrb.by/statistics/rates/RatesDaily.asp?fromDate=YYYY-MM-DD, где YYYY, MM и DD — это год, месяц и день, на который будут отображены курсы валют.

Нам придется извлекать данные вручную, непосредственно из кода HTML.



Ниже приводится текст макроса снабженный подробными комментариями.

Макрос импорта курсов валют

```
option explicit

' системные требования: Windows XP SP1, Windows 2000 SP3
' Windows 2003 Server
'
' Процедура Curr_LoadRates загружает курсы валют с сайта НБ РБ.
' Список валют, для которых будут запрашиваться курсы задается
' в словаре CurrCode.
'
sub Curr_LoadRates

  Dim Creator
  Set Creator = New TCreator

  ' словарь CurrCode нужен нам для двух целей:
  ' во-первых, в нем задается список валют, курсы
  ' которых мы будем загружать с сайта
  ' во-вторых, в нем мы устанавливаем соответствие
  ' между кодом (буквенной аббревиатурой) валюты
  ' используемой на сайте (Key) и кодом валюты
  ' используемым в нашей базе данных (Item). Очевидно,
```

[34] URL действовал на момент написания документации.

```
' что они могут как совпадать, так и различаться
' например, доллар США на сайте может обозначаться
' как USD, а у нас в базе он будет проходить как
' USD NBRB, что означает курс доллара США, установленный
' Национальным банком Республики Беларусь.
Dim CurrCode
Set CurrCode = CreateObject("Scripting.Dictionary")
CurrCode.Add "USD", "USD"
CurrCode.Add "EUR", "EUR"
CurrCode.Add "RUB", "RUB"
CurrCode.Add "PLN", "PLN"
CurrCode.Add "UAH", "UAH"
CurrCode.Add "LTL", "LTL"
CurrCode.Add "LVL", "LVL"
CurrCode.Add "GBP", "GBP"

' идентификатор записи валюты, относительно которой
' задаются курсы валют на сайте
' в данном случае -- это наш родной белорусский рубль
Dim BaseCurrID
BaseCurrID = gdcBaseManager.GetIDByRUIDString("200010_17")

' даты: с какой и по какую загружать курсы валют с сайта
' по какую -- возьмем текущую системную дату
' с какой -- определим следующим образом: будем искать в нашей
' базе для каждой из заданных валют самую последнюю
' дату курса. из всех найденных дат возьмем наиболее раньнюю.
Dim FromDate, ToDate
ToDate = Date
FromDate = ToDate

Dim q, qFind, Tr
Set Tr = Creator.GetObject(Application, "TIBTransaction", "")
Set q = Creator.GetObject(Application, "TIBSQL", "")
Set qFind = Creator.GetObject(Application, "TIBSQL", "")

Set Tr.DefaultDatabase = gdcBaseManager.Database
Set q.Transaction = Tr
Set qFind.Transaction = Tr

Tr.StartTransaction

' уберем коды валют которых нет нашей базе или
' которые встречаются более одного раза
q.SQL.Text =_
  "SELECT COUNT(c.code) " &_
  "FROM gd_curr c  " &_
  "WHERE c.code = :C "

Dim I
For Each I in CurrCode
  q.ParamByName("C").AsString = CurrCode.Item(I)
  q.ExecQuery
  if q.Fields(0).AsInteger <> 1 then _
    CurrCode.Remove(I)
  q.Close
Next

q.SQL.Text =_
  "SELECT MAX(r.fordate) " &_
  "FROM gd_currrate r JOIN gd_curr c ON r.fromcurr = c.id " &_
```

```
    "WHERE c.code = :FC and r.tocurr = :TC"
  q.ParamByName("TC").AsInteger = BaseCurrID

  For Each I in CurrCode
    q.ParamByName("FC").AsString = CurrCode.Item(I)
    q.ExecQuery
    if Int(q.Fields(0).AsDateTime) < FromDate then _
      FromDate = Int(q.Fields(0).AsDateTime)
    q.Close
  Next

  ' проверим, может в базе уже есть все курсы
  if ToDate < FromDate then _
    exit sub

  ' если интервал слишком большой -- ограничим его
  if (ToDate - FromDate) > (365 * 1) then _
    FromDate = ToDate - 365 * 1

  ' подготовим запрос для вставки курса валюты
  q.SQL.Text = _
    "INSERT INTO gd_currrate (fromcurr, tocurr, fordate, coeff) " &_
    "SELECT ID, :TC, :FD, :R FROM gd_curr WHERE code = :FC "
  q.ParamByName("TC").AsInteger = BaseCurrID

  ' подготовим запрос для поиска курса валюты на указанную дату
  qFind.SQL.Text =_
    "SELECT * " &_
    "FROM gd_currrate r JOIN gd_curr c ON r.fromcurr = c.id " &_
    "WHERE c.code = :FC and r.tocurr = :TC AND r.fordate = :FD"
  qFind.ParamByName("TC").AsInteger = BaseCurrID

  Dim strResult
  Dim WinHttpReq
  Dim strURL

  ' если объект не удается создать (например, неподходящая
  ' версия Windows), то просто завершаем выполнение
  On Error Resume Next
  Set WinHttpReq = CreateObject("WinHttp.WinHttpRequest.5.1")
  if Err.Number <> 0 then _
    Exit Sub
  On Error GoTo 0

  Dim D, K, J, E, SS, Mo, Da
  For D = FromDate to ToDate Step 1
    qFind.Close
    qFind.ParamByName("FD").AsDateTime = D

    q.ParamByName("FD").AsDateTime = D

    if Month(D) < 10 then
      Mo = "0" & Month(D)
    else
      Mo = Month(D)
    end if

    if Day(D) < 10 then
      Da = "0" & Day(D)
    else
      Da = Day(D)
```

```
    end if

    strURL = "http://www.nbrb.by/statistics/rates/RatesDaily.asp?fromDate="
&_
      Year(D) & "-" & Mo & "-" & Da

    'strUrl = "http://localhost/nbrb/ratesdaily.asp.htm"

    ' в случае если сайт не доступен -- просто
    ' заверши процедуру
    On Error Resume Next

    WinHttpReq.Open "GET", strURL, false
    WinHttpReq.Send
    strResult = WinHttpReq.ResponseText

    if Err.Number <> 0 then _
      Exit Sub

    On Error GoTo 0

    For Each I in CurrCode
      ' для каждой валюты проверим: нет ли в базе курса на
      ' указанную дату. если есть, то пропускаем эту валюту
      qFind.ParamByName("FC").AsString = CurrCode.Item(I)
      qFind.ExecQuery

      if qFind.EOF then
        K = InStr(strResult, ">" & I & "</td>")

        if K > 0 then
          For J = 1 to 4
            K = InStr(K + 1, strResult, ">")
            if K = 0 then _
              Exit For
          Next

          if K > 0 then
            K = K + 1
            E = InStr(K, strResult, "<")

            if E > 0 then
              SS = Replace(Mid(strResult, K, E - K), " ", "")
              SS = Replace(SS, " ", "")
              SS = Replace(SS, Chr(160), "")

              ' на сайте используется точка в качестве десятичного
              ' разделителя. Заменим ее на разделитель, использующийся в
нашей
              ' системе
              SS = Replace(SS, ".", Application.DecimalSeparatorSys)

              if SS > "" then
                ' в случае если строку не удается перевести в число
                ' или возникнет ошибка при добавлении записи --
                ' проигнорируем их
                On Error Resume Next

                q.ParamByName("R").AsCurrency = CCur(SS)
                q.ParamByName("FC").AsString = CurrCode.Item(I)
```

```
                if Err.Number = 0 then _
                  q.ExecQuery

                On Error GoTo 0
              end if
            end if
          end if
        end if
      end if

      qFind.Close
    Next

  Next

  ' подтверждаем запись в базу данных, только если все прошло успешно
  ' и не возникало ошибок
  Tr.Commit

end sub
```

Автоматизация процесса загрузки курсов валют

Для полной автоматизации загрузки курсов валют не хватает только вызова написанного нами макроса по расписанию, например, ежедневно в 9:00.

Решить данную задачу можно с помощью стандартных средств операцинной системы Windows: встроенного интерпретатора языка VBScript и планировщика задач.

Для начала, создадим скрипт, который будет загружать Гедымин как COM сервер, подключаться к базе данных и выполнять скрипт-функцию загрузки курсов валют.

Текст скрипта приводится ниже:

```
Dim G, ID, User, Passw, Database
Set G = CreateObject("Gedemin.gsGedeminApplication")

User = "Andreik"
Passw = "1"
Database = "server:k:\bases\gedemin\gdbase.gdb"

if not G.LoginSilent(User, Passw, Database) then
  Set G = CreateObject("Gedemin.gsGedeminApplication")
  if not G.LoginSilent User, Passw, Database then _
    Quit
end if

ID = G.gdcBaseManager.GetIDByRUIDString("154674334_287654914")

G.System.ExecuteScriptFunction ID, NULL
```

Рассмотрим, шаг за шагом, что делает данный скрипт:

1. Сразу после объявления переменных идет создание COM объекта автоматизации с ProgID “Gedemin.gsGedeminApplication”;
2. Далее заполняются переменные с именем пользователя системы Гедымин, его паролем и полным путем к базе данных;
3. Следующие пять строк — это попытка подключиться к базе данных. Для этого используется функция LoginSilent, в которую передаются: имя пользователя, пароль и полный путь к базе данных. Функция вернет булевское значение: успех или неудача. Обратите внимание, что в случае неудачи повторно создается COM объект и повторно осуществляется попытка

подключения к базе данных. Дело в том, что внутренняя архитектура платформы не позволяет иметь более одного подключения к базе данных[35]. Если на момент выполнения нашего скрипта Гедымин уже был загружен в память, то функция CreateObject вернет объект, созданный в рамках данного загруженного экземпляра. Но поскольку одно подключение к базе данных уже существует, то функция LoginSilent вернет False. В этом случае будет предпринята попытка повторного создания объекта с помощью функции CreateObject. При повторном создании объекта операционная система загрузит отдельный экземпляр gedemin.exe и повторный вызов LoginSilent будет осуществляться уже в рамках этого экземпляра.

4. Вызов скрипт-функции осуществляется с помощью метода ExecuteScriptFunction объекта System. На вход метода необходимо передать идентификатор скрипт-функции и массив параметров. В нашем случае, поскольку функция не требует параметров, мы передаем NULL.

5. Обратите внимание, как мы получаем идентификатор функции по ее РУИДу с помощью метода GetIDByRUIDString объекта gdcBaseManager. РУИД скрипт-функции можно узнать на вкладке Свойства макроса в редакторе скрипт-объектов.

Настройка планировщика задач

Сохраним набранный скрипт в файле currrate.vbs, например, на рабочем столе Windows. Теперь вызовем команду добавления запланированного задания: Start->Settings->Control Panel->Scheduled Tasks->Add Scheduled Task.

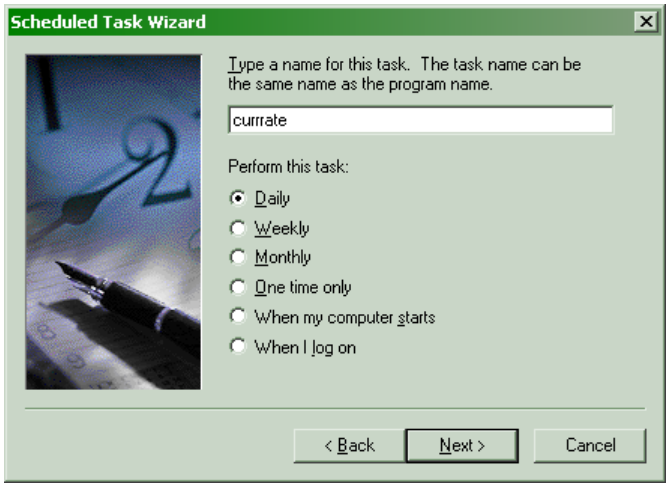


Рис. 108 Добавление запланированного задания.

Рис. 109 Добавление запланированного задания.

[35] Здесь идет речь имеено о подключении платформы к базе данных. Данное ограничение не распространяется на подключения организованные разработчиком из макросов с помощью создания компонентов TIBDatabase. Таких подключений может быть сколько угодно.

Использование Гедымина как COM сервера

Заключение

Контрольные вопросы

- С помощью какого глобального объекта осуществляется чтение и распознование данных, хранящихся в текстовых файлах определенной структуры?

Распределенные базы данных

До сих пор мы рассматривали два режима работы системы Гедымин: локальный и сетевой. В локальном режиме исполняемый файл gedemin.exe и файл базы данных должны находиться на одном компьютере. В этом режиме используется так называемый встроенный (embedded) сервер, который не допускает более одного подключения к базе данных. В сетевом режиме работы произвольное количество пользователей может одновременно подключаться к одной базе данных. При этом на компьютере, где находится файл базы данных, должен быть установлен сервер Interbase, а на каждом пользовательском компьютере должна быть установлена клиентская часть для доступа к серверу. Подробно процедура установки сервера базы данных и программного комплекса Гедымин в автоматическом и ручном режиме уже рассматривалась ранее в данной книге. В определенных случаях работа всех пользователей с одной базой данных невозможна. Например, компания может иметь два офиса в разных городах, между которыми нет возможности установить постоянную связь, или директор компании хочет иметь возможность доступа к информации со своего ноутбука, находясь в пути, или нагрузка на сервер столь велика, что часть пользователей не может выполнять свои запросы без риска остановить работу других пользователей. Решением любой из вышеперечисленных проблем может быть использование нескольких экземпляров базы данных, расположенных на разных компьютерах, между которыми организован обмен информацией, т.н. репликация.

Рисунок, иллюстрирующий репликацию.

Теоретические основы

Репликацией называется процесс переноса изменений данных, сделанных на одной базе, в произвольное количество других баз данных. Обычно, базы данных, между которыми осуществляется репликация, имеют одинаковую структуру, однако, в общем случае, возможен перенос данных между базами, различными по своей структуре. Репликация может быть полной, когда переносятся абсолютно все изменения и частичной — переносятся только изменения, удовлетворяющие определенному критерию. Например, между офисами, расположенными в разных городах, может передаваться информация о заключенных сделках с клиентами компании, в то же время сведения о затратах каждого офиса передаваться не будут. Различают одно- и двунаправленную репликацию. В первом случае изменения всегда передаются только в одном направлении от главной базы данных[36] к второстепенной[37]. Как правило, в случае однонаправленной репликации принимающая (второстепенная) база данных используется только для извлечения данных. Например, это может быть база данных, расположенная на веб-сервере, полностью или частично дублирующая информацию главной базы данных, находящейся в офисе компании. Репликация может быть синхронной, когда сделанное изменение одновременно распространяется по всем базам данных и асинхронной. В последнем случае, между совершением изменения в данных одной базы и переносом его в другие базы может пройти достаточно значительное количество времени. Различают симметричную и ассиметричную репликацию. В первом случае одна из баз данных является главной, а остальные — второстепенными. Изменения из одной второстепенной базы данных в другую могут попасть только через главную базу данных, т.е. второстепенные базы данных не могут обмениваться изменениями между собой напрямую. В случае ассиметричной репликации, данные между базами могут перемещаться в произвольном порядке. Изменения, сделанные на одной базе данных и передаваемые на другую в процессе репликации, называются репликой.

Натуральные и суррогатные первичные ключи

В реляционной базе данных каждая таблица должна иметь поле (или группу полей) которые однозначно идентифицируют любую запись в этой таблице. Если такое поле (поля) используются в ограничении PRIMARY KEY, созданном для данной таблицы, то оно (они) называются первичным ключом таблицы. Первичным ключом может быть одно из полей записи, уникальность которого гарантируется природой хранимых в нем данных. Например, в таблице, содержащей список сотрудников предприятия, натуральным первичным ключом может являться поле, содержащее персональный номер сотрудника, указанный в паспорте. Натуральные первичные ключи имеют определенный ряд недостатков. Во-первых, в связи с изменением внешних условий атрибут, выбранный в качестве натурального ключа, может утратить свою уникальность. Например, если база данных содержит список городов некоторой

[36] Часто называемой Издателем (Publisher).

[37] Часто называемой Подписчиком (Subscriber).

страны, то изначально разработчик может выбрать в качестве натурального ключа телефонный код города. Однако, в последствии, в результате реформы телефонной связи некоторые коды могут быть объединены, по одному коду для нескольких городов, после чего код города уже не будет однозначно определять город, что приведет к нарушению требования уникальности первичного ключа. Вторая проблема, связанная с использованием натуральных первичных ключей, заключается в неоправданном увеличении размера базы данных, в случае, если поле первичный ключ имеет достаточно большую длину. Например, в рассмотренном нами выше случае с идентификацией сотрудника по его паспортному персональному номеру, поле первичный ключ — это строка из 32-х символов и каждое поле-ссылка на данную таблицу будет так же иметь длину 32 символа. И наконец, третья проблема — это изначальная разнотипность натуральных ключей. Это могут быть строки, целые и дробные числа, даты, время и даже булевские значения. Необходимость поддержки всех возможных типов данных способна существенно усложнить код компонентов программы, которые отвечают за автоматическое формирование SQL запросов.

Вместо использования натурального первичного ключа разработчик базы данных может добавить в таблицу поле заполняемое автоматически, которое будет однозначно идентифицировать каждую запись в этой таблице. Как правило, используется целочисленное поле, заполняемое в тригере при вставке записи. Для заполнения поля используется очередное значение генератора. Такое поле называется суррогатным первичным ключем. Очевидно, что при использовании целочисленного суррогатного первичного ключа решаются все обозначенные выше проблемы. Конечно, может возникнуть проблема исчерпания целочисленного диапазона, но для абсолютного большинства практических задач два с лишним миллиарда[38] записей в одной таблице — более чем достаточно.

Кандидат на первичный ключ

Кроме первичного ключа в таблице могут быть другие поля однозначно идентифицирующие каждую запись. Такие поля называются кандидатами на первичный ключ таблицы. Как правило, на такие поля накладывается ограничение уникальности или для них создается уникальный индекс. Ниже будет показано, как наличие в таблице кандидатов на первичный ключ может помочь избежать конфликтов при изменении данных в распределенных базах.

Репликация данных на платформе Гедымин

Для репликации баз данных платформы Гедымин используется утилита ИБРепликатор. Она позволяет организовать полную или частичную, асинхронную, двунаправленную, симметричную репликацию между произвольным количеством баз данных.

Установка утилиты репликации

ИБРепликатор должен быть установлен на сервер базы данных или на любой компьютер, подключенный к локальной сети предприятия. В последнем случае серверный диск, на котором находится файл базы данных, должен быть подключен на компьютере, где устанавливается утилита репликации, как сетевой диск под той же буквой[39]. У пользователя должны быть полные права доступа к каталогу и файлу базы данных на сервере.

Для установки ИБРепликатора необходимо загрузить инстолятор (поставляется на установочном диске платформы Гедымин), указать путь для размещения файлов и запустить процедуру копирования. После завершения программы установки в созданной папке (по-умолчанию, это папка Program Files\Golden Software\IBReplicator) будут размещены два файла: собственно, сам репликатор IBReplicator.EXE и файл локализации пользовательского интерфейса IBReplicator.RUS[40], кроме этого будет создана программная группа IBReplicator, куда будет помещена пиктограмма для вызова утилиты.

---

[38] Если использовать только положительные числа, как это делается в Гедымине. При использовании же всего диапазона 32-х битных целых чисел количество возможных значений равно 4 294 967 296.

[39] Например, если на сервере полный путь к файлу базы данных выглядит как k:\bases\gdbase.fdb, то и на сетевом компьютере, на котором устанавливается ИБРепликатор, он должен выглядеть точно так же.

[40] Без файла IBReplicator.RUS, или в случае, если в операционной системе установлен язык отличный от русского будет использоваться англоязычный интерфейс пользователя.

Настройка схемы репликации

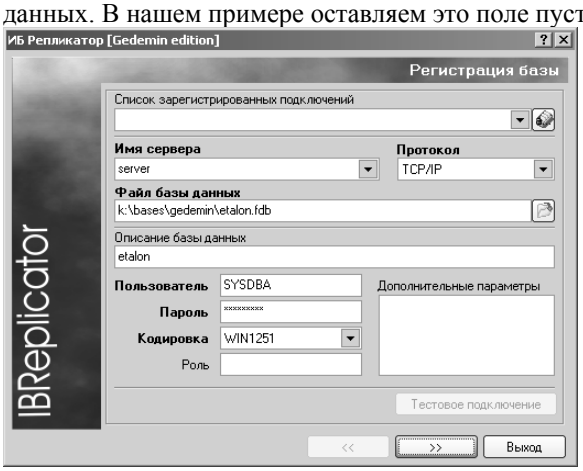
Прежде чем начать использовать репликацию, необходимо указать в каких таблицах будут отслеживаться изменения, сколько экземпляров базы данных будет создано, как будут разрешаться конфликтные ситуации и т.п. Совокупность вышеуказанных параметров называется схемой репликации. Одна база данных может иметь только одну схему репликации. Данные о схеме сохраняются в самой базе данных, в таблицах, имеющих префикс RPL$. Эти таблицы создаются утилитой автоматически, при подготовке базы данных к репликации.

Для настройки схемы репликации запускаем утилиту ИБРепликатор и выполняем следующую последовательность действий:

Создаем подключение к базе данных

Для этого заполняем поля:

1. Имя сервера (указывается имя компьютера, на котором установлен сервер базы данных. Можно указать localhost, если ИБРепликатор и сервер базы данных находятся на одном компьютере),
2. Протокол (протокол подключения к серверу базы данных, рекомендуется установить TCP/IP),
3. Файл базы данных (указывается размещение файла базы данных на сервере, например, d:\bases\gdbase.fdb),
4. Описание базы данных (указывается произвольное наименование данного подключения. По этому имени, в дальнейшем, можно будет выбрать данное подключение из Списка зарегистрированных подключений, расположенного вверху окна),
5. Пользователь (указывается пользователь сервера Interbase для подключения к базе данных. Рекомендуется использовать учетную запись администратора базы данных SYSDBA, так как создание схемы репликации требует достаточных прав для создания метаданных),
6. Пароль (указывается пароль для введенной учетной записи[41]),
7. Кодировка (указывается кодовая таблица подключения к базе данных. Как правило, базы данных на платформе Гедымин создаются в кириллической кодировке Win1251),
8. Роль для подключения к базе данных (поле может быть оставлено пустым, особенно если используется учетная запись SYSDBA),
9. Дополнительные параметры (указываются дополнительные параметры подключения к базе данных. В нашем примере оставляем это поле пустым).



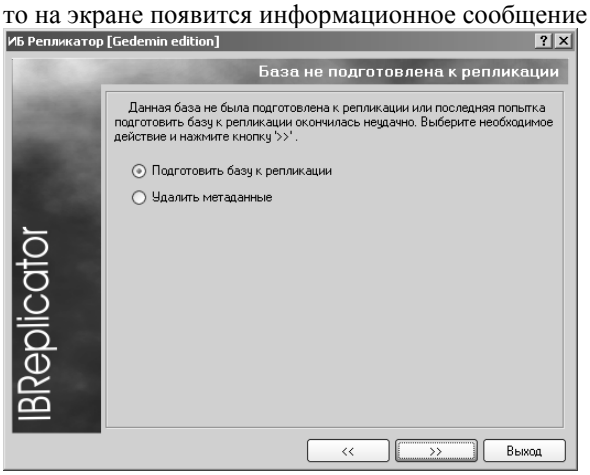
10. По завершении заполнения всех полей окна нажимаем кнопку Дальше[42].

---

[41] Пароль по-умолчанию для учетной записи SYSDBA — masterkey. Настоятельно рекомендуется изменить этот пароль при установке или конфигурировании сервера.

[42] Кнопка Дальше расположена слева от кнопки Выход внизу окна. На ней изображена двойная стрелка вправо. Слева от кнопки Дальше располагается кнопка Назад, которая позволяет вернуться на предыдущий экран.

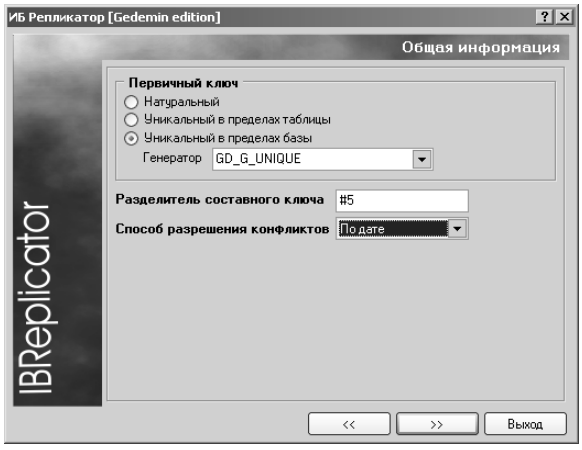
11. Если к базе данных подключены другие пользователи, то на экран будет выведен их список. Необходимо дождаться, пока они не закроют свои соединения. Настраивать схему репликации можно только в монопольном (однопользовательском) режиме. Если активных соединений нет, то на экране появится информационное сообщение следующего вида:



Выбираем действие «Подготовить базу к репликации» и нажимаем кнопку Дальше.

Общая информация о структуре базы

Следующий экран предназначен для указания общих параметров структуры базы данных и схемы репликации:



В разделе первичный ключ указывается формат первичного ключа, используемого в базе данных. Утилита репликации позволяет работать с базами данных, в которых применяются натуральные, а так же суррогатные целочисленные первичные ключи, уникальные на уровне отдельной таблицы или для всей базы данных. Уникальность ключа должна обеспечиваться генератором. Если ключ уникален в пределах всей базы данных, то используется только один генератор и его следует указать, в противном случае необходимо будет указать наименование генератора для каждой таблицы на следующем экране программы. Если используется редакция ИБРепликатора для платформы Гедымин, то данный раздел недоступен для выбора. Базы Гедымина используют суррогатные целочисленные ключи, уникальные в пределах всей базы, что достигается использованием единственного генератора GD_G_UNIQUE. Применение ИБРепликатора для баз произвольного типа будет подробно рассмотрено в следующих главах. Символ-разделитель составного ключа используется для разделения полей составного первичного ключа при сохранении их в виде одной строки. Значение заданное по-умолчанию — #5 — означает, что будет использоваться единичный символ с ASCII кодом 5. Последнее поле в этом окне задает способ разрешения конфликтных ситуаций. Конфликты возникают, если одна и та же запись была изменена на нескольких базах. Возможны следующие механизмы автоматического разрешения конфликтных ситуаций: По дате (сохранена будет наиболее поздняя версия записи. Если дата и время изменения записи совпадают, то изменения, передаваемые в файле реплики, затрут изменения, произведенные в базе данных, на которую передаются изменения), По приоритету (данным, поступающим из базы данных с более высоким приоритетом, будет отдаваться предпочтение. Если приоритет баз одинаков, будет анализироваться время изменения записей. Более поздние изменения затрут более ранние. Если время изменения записей окажется равным, то предпочтение будет отдано

записи, передаваемой в файле реплики), По статусу (изменения, произведенные в главной базе данных, всегда будут затирать изменения, произведенные во второстепенных базах).

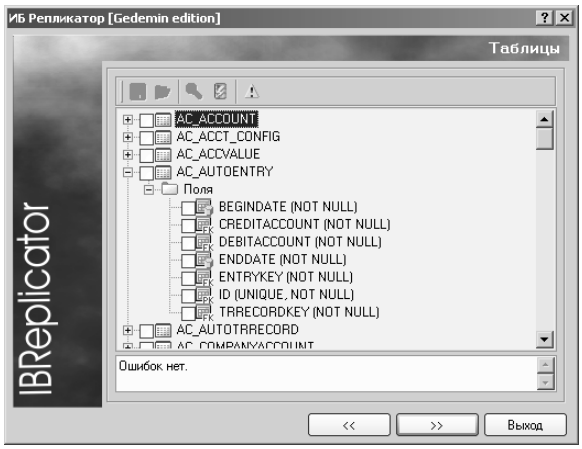
После заполнения всех полей нажимаем кнопку Дальше.

Выбор таблиц и полей

Следующий экран предназначен для указания таблиц и полей, данные которых будут реплицироваться, а так же для настройки первичного ключа репликатора для каждой из таблиц.

Первичный ключ репликатора используется для идентификации записи при переносе ее между базами данных. Первичным ключом репликатора для таблицы может быть или ее первичный ключ или любой из кандидатов на первичный ключ таблицы, т.е. поле по которому создан уникальный индекс.

В настоящее време утилита ИБРепликатор поддерживает только простые ключи, состоящие из одного поля.



Основную часть окна занимает древовидный список, содержащий все таблицы, находящиеся в базе данных, и все поля для каждой таблицы. Слева от наименования каждого поля выводится графический значок, показывающий его тип:


Простое поле таблицы.

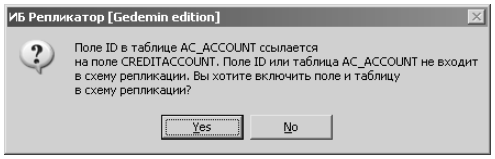

Ссылка на другую таблицу (FOREIGN KEY).


Первичный ключ таблицы (PRIMARY KEY).


Первичный ключ репликатора для таблицы.

Настраивать схему репликации можно в ручном или в автоматическом режиме. В первом случае необходимо отметить каждую таблицу, которая должна войти в схему репликации. При выделении таблицы, все ее поля так же выделяются. Если действие репликации не должно распространятся на определенное поле, то с него необходимо снять пометку.

При включении в схему репликации поля, являющегося ссылкой на другую таблицу (т.е. если для этого поля создан внешний ключ — FOREIGN KEY), репликатор предложит включить эту таблицу в схему репликации, на экран будет выведен соответствующий вопрос:



Если у включаемой в схему таблицы нет других уникальных полей кроме первичного ключа, то это поле будет автоматически помечено, как первичный ключ репликатора. Если же другое уникальное поле (кандидат на первичный ключ) присутствует, и в таблице используется суррогатный первичный ключ, то в качестве первичного ключа репликатора будет выбрано это уникальное поле. Если таких уникальных полей несколько, то первичным ключем репликатора будет выбран первичный ключ таблицы. При необходимости установить в качестве первичного ключа репликатора одно из уникальных полей это можно сделать вручную, установив на поле курсор и нажав соответствующую кнопку на панели инструментов вверху окна.

При выборе автоматического режима настройки схемы (соответствующая кнопка находится на панели инструментов вверху окна) в схему репликации будут включены все таблицы и все поля из базы данных. Для каждой таблицы будет установлен первичный ключ репликатора согласно правилу, указанному выше.

При необходимости, сгенерированную автоматически схему можно подправить. При использовании репликатора с базой данных платформы Гедымин некоторые таблицы НЕ ДОЛЖНЫ попадать в схему репликации. К таковым относятся:

1. Все таблицы, имеющие префикс AT_. Эти таблицы являются внутренними, вспомогательными таблицами системы Гедымин. Они содержат информацию о структуре базы данных[43].
2. FLT_LASTFILTER.

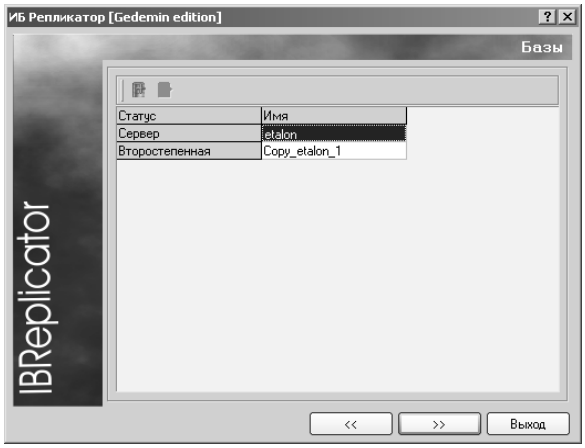
В большинстве случаев не рекомендуется включать в схему репликации таблицы, содержащие данные хранилищ платформы:

1. GD_GLOBALSTORAGE;
2. GD_USERSTORAGE;
3. GD_COMPANYSTORAGE;

Если для таблицы включенной в схему репликации не указан первичный ключ репликатора или не выбрано ни одно из полей, то пиктограмма такой таблицы будет помечена желтым треугольником с восклицательным знаком и предупреждение будет выведено внизу экрана в отдельном поле. Пока все проблемы не устранены перейти на следующий этап нельзя. Если все нормально — нажимаем кнопку Дальше.

Создание второстепенных баз данных

Следующий экран предназначен для заполнения списка второстепенных баз данных. Используя кнопки на панели инструментов можно добавлять или удалять записи из списка.

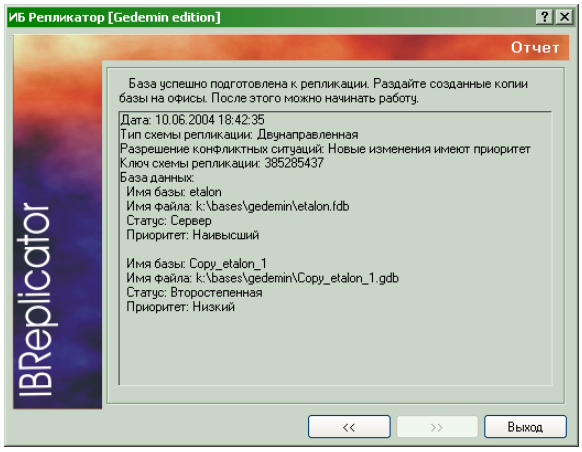


По завершении ввода необходимо нажать кнопку Дальше.

Подготовка баз данных

Если все данные внесены корректно, то начнется процесс подготовки базы данных к репликации. В ходе процесса будут созданы необходимые таблицы и тригеры, подготовлены копии второстепенных баз данных. По завершении на экран будет выведена следующая информация:

[43] Может возникнуть такая ситуация, когда разработчик настройки создаст таблицу, которая будет содержать ссылку на одну из AT_ таблиц. При включении такой таблицы в схему репликации соответствующая AT_ таблица так же будет включена. В этом случае можно порекомендовать исключить из схемы репликации все поля соответствующей AT_ таблицы, за исключением первичного ключа.

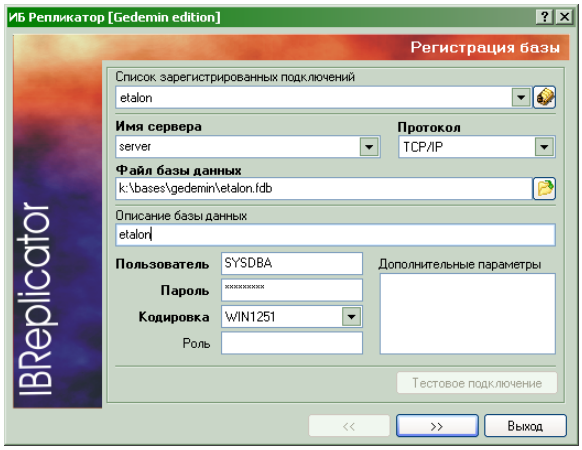


Осталось перенести второстепенные базы данных на удаленные компьютеры и можно начинать работу.

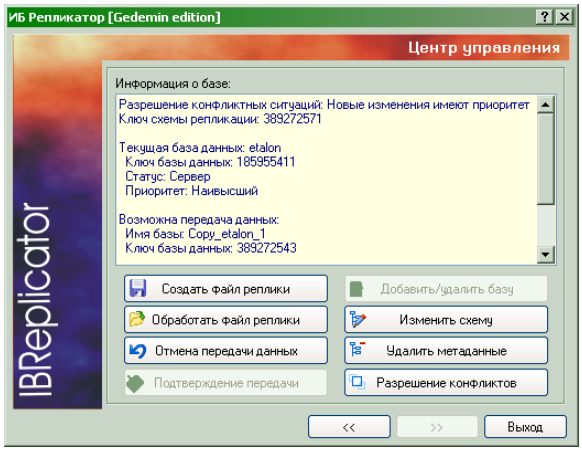
Репликация данных

Частоту, с которой будет происходить обмен репликами между удаленными базами данных, администратор определяет самостоятельно исходя из требований задачи, интенсивности обновления данных, пропускной способности и доступности канала связи.

Для создания реплики и переноса ее на второстепенные базы данных необходимо запустить утилиту ИБРепликатор и подключиться к главной базе данных. Вместо того, чтобы вручную заполнять поля: сервер, имя файла базы данных и т.п. достаточно выбрать ранее сформированное подключение из выпадающего списка вверху окна.



Для продолжения нажимаем кнопку Дальше. На экран будет выведена информация о состоянии базы данных и список возможных действий:



Создать файл реплики

Обработать файл реплики

Отмена передачи данных

Изменить схему репликации

Удалить метаданные

Разрешение конфликтов

Как работает репликатор

Основная идея, на которой построено функционирование репликатора, состоит в том, чтобы регистрировать все изменения, произошедшие в базе данных и сохранять их в специальной таблице RPL$LOG в том порядке, в котором они были совершены. Поскольку, хранить все данные каждой измененной записи не представляется возможным, то в таблицу записывается только первичный ключ записи, до и после произошедшего изменения, а так же вид изменения: вставка, обновление или удаление записи. Для регистрации изменений, на каждую таблицу, включенную в схему репликации, создается три тригера (по одному на каждый вид изменения).

В качестве иллюстрации приведем тригеры, созданные для регистрации изменений в таблице AC_ACCOUNT:

```
CREATE TRIGGER RPL$BI_AC_ACCOUNT FOR AC_ACCOUNT
  ACTIVE
  BEFORE INSERT
  POSITION 32767
AS
BEGIN
  INSERT INTO RPL$LOG(RELATIONKEY, REPLTYPE, OLDKEY, NEWKEY)
    VALUES(377502950, 'I', NEW.ID, NEW.ID);
END

CREATE TRIGGER RPL$BU_AC_ACCOUNT FOR AC_ACCOUNT
  ACTIVE
  BEFORE UPDATE
  POSITION 32767
AS
BEGIN
  INSERT INTO RPL$LOG(RELATIONKEY, REPLTYPE, OLDKEY, NEWKEY)
     VALUES(377502950, 'U', OLD.ID, NEW.ID);
END

CREATE TRIGGER RPL$BD_AC_ACCOUNT FOR AC_ACCOUNT
  ACTIVE
  BEFORE DELETE
  POSITION 32767
AS
BEGIN
  INSERT INTO RPL$LOG(RELATIONKEY, REPLTYPE, OLDKEY, NEWKEY)
    VALUES(377502950, 'D', OLD.ID, OLD.ID);
END
```

Обратите внимание, что тригеры имеют максимально возможный номер позиции, так как они должны выполняться после всех остальных тригеров для того, чтобы корректно сохранить значение первичного ключа записи.

При создании файла реплики, сначала, список изменений сканируется и из него удаляется избыточная информация. Например, если для определенной записи зафиксировано изменение, а потом она была удалена, то запись об изменении удаляется и в логе остается только информация об удалении записи. Аналогично обрабатывается информация о нескольких последовательных изменениях одной и той же записи или о вставке записи и ее последующем изменении и т.п. Затем, анализируется таблица RPL$LOGHIST, которая содержит историю передачи изменений из данной базы данных на другие базы, участвующие в схеме репликации. Те записи, которые не передавались на указанную второстепенную базу данных, или, передача которых не была подтверждена, включаются в файл реплики. Соответствующие записи о включении этих изменний в файл реплики делаются в таблице RPL$LOGHIST. В таблице RPL$REPLICATIONDB запоминается номер последнего файла реплики, переданного на другую базу данных.

При приеме файла реплики сначала проверяется не были ли пропущены предыдущие реплики. Если да, то применять данную реплику нельзя. Необходимо вернуться на базу, где она была сформирована, откатить все последние передачи данных, файлы которых были утеряны, и сформировать новую реплику, которая будет содержать все, до сих пор не переданные записи. Если реплика является очередной по счету, то данные из нее записываются в базу. Процесс записи данных состоит из нескольких стадий. Сначала, отключаются все тригеры для таблиц, участвующих в схеме репликации. Список отключенных тригеров сохраняется, так что если в процессе применения реплики произойдет сбой, то при следующей обработке базы данных репликатором ее можно будет вернуть в первоначальное состояние. Затем, находятся все циклические ссылки между любыми двумя таблицами в базе и внешние ключи, образующие эти ссылки отключаются. Список деактивированных внешних ключей сохраняется. Записи, поступившие в файле реплики сгруппированы по имени таблицы и упорядочены по порядку их изменения. Они начинают добавляться одна за одной в базу данных. Если в базе уже существует запись с идентификатором как у передаваемой, то в зависимости от способа разрешения конфликтных ситуаций установленном в данной схеме репликации (по дате изменения, по статусу или приоритету базы) существующая запись будет либо оставлена неизменной, либо будет замещена записью пришедшей в файле реплики. Если некоторую запись не удается добавить в базу, то она пропускается. Процесс обработки списка продолжается циклически до тех пор пока все записи не будут обработаны или пока на очередной итерации не будет обработано ни одной записи. Оставшиеся в списке записи (если таковые имеются) требуют ручного разрешения конфликтов, о чем будет уведомлен пользователь. На экран будет выведено соответствующее окно.

Утилита поддерживает только двунаправленную репликацию. После того, как одна или несколько реплик поступили и были обработаны, необходимо сформировать ответную реплику и передать ее на исходную базу. Ответная реплика будет содержать изменения данных, произошедшие в этой базе, а также дополнительную информацию, подтверждающую успешную передачу изменений. Изменения, которые были успешно переданы на все второстепенные базы данных удаляются из файла RPL$LOG и RPL$LOGHIST.

Использование РУИД

Сопоставление РУИД

Ограничения репликатора

Существуют определенные ограничения при использовании утилиты ИБРепликатор. В настоящий момент времени ведется работа над версией 2.0 в которой большинство из нижеперечисленных ограничений будут сняты.

Ограничение на длину первичного ключа

Длина строкового представления первичного ключа записи не должна превышать 255 символов. Если первичный ключ состоит из нескольких полей, то его строковое представление — это конкатенция значений образующих его полей, разделенных символом-разделителем, указанным при определении схемы репликации. По-умолчанию, используется единичный символ с ASCII кодом 5.

Только двунаправленная симметричная репликация

В настоящий момент времени утилита репликации поддерживает только двунаправленную симметричную репликацию. Это означает, что данные обязательно должны передаваться как от главной базы к второстепенной, так и в обратном направлении. Кроме того, невозможен прямой обмен репликами между второстепенными базами минуя главную.

Циклические ссылки между более чем двумя таблицами

Репликатор не позволяет передавать изменения, если в базе данных существуют циклические ссылки между более чем двумя таблицами. Например, в базе данных существует таблица A, которая содержит запись с идентификатором A1. Запись A1 содержит ссылку на запись с идентификатором B1 в таблице B. Запись B1 содержит ссылку на запись C1 в таблице C. Запись C1 содержит ссылку на запись A1 в таблице A. При возникновении таких циклических ссылок в базе данных и попадании их в реплику произойдет конфликт в момент применения этой реплики на удаленной базе данных.

Целочисленные ключи только INTEGER

Утилита репликации поддерживает только 32—х битные целочисленные ключи. Поддержка типов данных SMALLINT и LARGEINT планируется в следующей версии.

Метаданные репликатора

Домены

Наименование	Тип данных	Размер	Not null	Примечание
DRPLBLOB	BLOB			Segment length: 80
DRPLDESCRIPTION	VARCHAR	255	*	Char Set: WIN1251
DRPLINTEGER	INTEGER			
DRPLINTEGER_NOT_NULL	INTEGER		*	
DRPLKEYSET	VARCHAR	255	*	Char Set: WIN1251
DRPLNAME	VARCHAR	31	*	Char Set: WIN1251

Генератор

Генератор RPL$G_SEQ используется для сохранения порядка изменений, произошедших в базе данных.

Таблицы репликатора

RPL$DBSTATE

Данная таблица всегда содержит только одну запись с информацией о состоянии текущей базы данных.

PK	FK	Поле Домен Тип данных	NN	Описание
		DBKEY DRPLINTEGER_NOT_NULL INTEGER	☒	Идентификатор базы данных.
		REPLICATIONID DRPLINTEGER_NOT_NULL INTEGER	☒	Идентификатор схемы репликации.
		ERRORDECISION DRPLINTEGER_NOT_NULL INTEGER	☒	Способ разрешения конфликтов: 0 — по статусу базы (главная, второстепенная); 1 — по приоритету базы; 2 — по времени изменения;

		FK DRPLBLOB BLOB SUB_TYPE 0 SEGMENT SIZE 80	☐	В данном поле сохраняется список отключенных внешних ключей. Если в процессе применения реплики возникнет сбой или компьютер будет выключен, при следующем запуске репликатора состояние внешних ключей будет восстановлено.
		TRIGGERS DRPLBLOB BLOB SUB_TYPE 0 SEGMENT SIZE 80	☐	Хранится список, отключенных в процессе применения реплики, тригеров. Предназначение — аналогично предыдущему полю.
		PRIMEKEYTYPE DRPLINTEGER_NOT_NULL INTEGER	☒	Тип, используемого в базе первичного ключа: 0 — Натуральный первичный ключ; 1 — Целочисленный суррогатный ключ, уникальный в пределах всей базы данных; 2 — Целочисленный суррогатный ключ, уникальный для каждой таблицы.
		GENERATORNAME DRPLNAME VARCHAR(31)	☒	Имя генератора, если тип первичного ключа = 1.
		KEYDIVIDER VARCHAR(5)	☐	Символ-разделитель, используемый для разделения частей сложных первичных ключей, состоящих из нескольких полей, при сохранении их ввиде строки.
		PREPARED DRPLINTEGER_NOT_NULL INTEGER	☒	1 — подготовка базы данных прошла успешно. 0 — в процессе подготовки базы данных к репликации произошли ошибки.

RPL$FIELDS

Таблица предназначена для хранения списка полей, входящих в схему репликации.

PK	FK	Поле Домен Тип данных	NN	Описание
1	F	RELATIONKEY DRPLINTEGER_NOT_NULL INTEGER	☒	Ссылка на таблицу RPL$RELATIONS.
2		FIELDNAME DRPLNAME VARCHAR(31)	☒	Имя поля.

RPL$KEYS

Таблица хранит информацию о первичном ключе репликатора для каждой таблицы, входящей в схему репликации.

PK	FK		NN	Описание
1	F	RELATIONKEY DRPLINTEGER_NOT_NULL INTEGER	☒	Ссылка на таблицу RPL$RELATIONS.
2		KEYNAME DRPLNAME VARCHAR(31)	☒	Имя поля.

RPL$LOG

Список изменений данных, произведенных в базе.

PK	FK	Поле Домен Тип данных	NN	Описание
1		SEQNO DRPLINTEGER_NOT_NULL INTEGER	☒	Порядковый номер изменения.
	F	RELATIONKEY DRPLINTEGER_NOT_NULL INTEGER	☒	Ссылка на таблицу, в которой была изменена запись.
		REPLTYPE CHAR(1)	☐	Тип изменения: I — запись была добавлена; U — запись была изменена; D — запись была удалена;
		OLDKEY DRPLKEYSET VARCHAR(255)	☒	Значение первичного ключа записи до изменения.
		NEWKEY DRPLKEYSET VARCHAR(255)	☒	Значение первичного ключа записи после изменения.
		ACTIONTIME TIMESTAMP	☐	Дата и время изменения.
		DBKEY DRPLINTEGER INTEGER	☐	Идентификатор базы данных, с которой пришли изменения. Поле заполняется только на главной базе и только для изменений пришедших со второстепенных баз данных. Поле необходимо для того, чтобы исключить передачу изменений, сделанных на второстепенной базе, на эту же базу.

RPL$LOGHIST

История передачи изменений в другие базы данных.

PK	FK	Поле Домен Тип данных	NN	Описание
1	F	SEQNO DRPLINTEGER_NOT_NULL INTEGER	☒	Порядковый номер изменения.
2	F	DBKEY DRPLINTEGER_NOT_NULL INTEGER	☒	Идентификатор базы данных, на которую было передано изменение.
		REPLKEY DRPLINTEGER_NOT_NULL INTEGER	☒	Номер файла реплики, с которым было передано изменение.
		TR_COMMIT DRPLINTEGER_NOT_NULL INTEGER	☒	Состояние передачи: -1 — информация не была передана; 0 — информация передана (попала в файл реплики); 1 — информация передана и пришло подтверждение об успешном применении файла реплики;

RPL$MANUAL_LOG

В данной таблице хранится список конфликтов, которые нуждаются в ручном разрешении.

PK	FK	Поле Домен Тип данных	NN	Описание
1		SEQNO DRPLINTEGER_NOT_NULL INTEGER	☒	Порядковый номер изменения.
2	F	DBKEY DRPLINTEGER_NOT_NULL INTEGER	☒	База данных, из которой пришло изменение.
		REPLKEY DRPLINTEGER_NOT_NULL INTEGER	☒	Номер файла реплики.
	F	RELATIONKEY DRPLINTEGER_NOT_NULL INTEGER	☒	Ссылка на таблицу.
		REPLTYPE CHAR(1)	☐	Тип изменения (I, U, D).
		OLDKEY DRPLKEYSET VARCHAR(255)	☒	Первичный ключ записи до того, как произошло изменение.
		NEWKEY DRPLKEYSET VARCHAR(255)	☒	Первичный ключ записи после того, как произошло изменение.
		ACTIONTIME TIMESTAMP	☐	Дата и время изменения.
		CORTEGE DRPLBLOB BLOB SUB_TYPE 0 SEGMENT SIZE 80	☐	Дополнительная информация.
		ERRORCODE DRPLINTEGER INTEGER	☐	Код возникшей ошибки.
		ERRORDESCRIPTION DRPLDESCRIPTION VARCHAR(255)	☒	Описание возникшей ошибки.

RPL$RELATIONS

Список таблиц, участвующих в схеме репликации.

PK	FK	Поле Домен Тип данных	NN	Описание
1		ID DRPLINTEGER_NOT_NULL INTEGER	☒	Идентификатор таблицы.
		RELATION DRPLNAME VARCHAR(31)	☒	Наименование таблицы.
		GENERATORNAME DRPLNAME	☒	Имя генератора, если в базе данных используются суррогатные целочисленные ключи, уникальные для

		VARCHAR(31)		каждой таблицы.

RPL$REPLICATIONDB

Список таблиц, участвующих в схеме репликации.

PK	FK	Поле Домен Тип данных	NN	Описание
1		DBKEY DRPLINTEGER_NOT_NULL INTEGER	☒	Идентификатор базы данных.
		DBSTATE DRPLINTEGER_NOT_NULL INTEGER	☒	Статус базы данных: 0 — главная; 1 — второстепенная;
		PRIORITY DRPLINTEGER_NOT_NULL INTEGER	☒	Приоритет базы данных: 0 — наивысший, 9 — наинизший.
		DBNAME DRPLNAME VARCHAR(31)	☒	Имя базы данных.
		RUIDCONFORMITY DRPLBLOB BLOB SUB_TYPE 0 SEGMENT SIZE 80	☐	Сопоставление РУИДов записей.
		LASTPROCESSREPLKEY DRPLINTEGER_NOT_NULL INTEGER	☒	Идентификатор последнего файла реплики который был передан с базы данных, определяемой текущей записью, и успешно обработан на нашей базе данных.
		LASTPROCESSREPLDATE TIMESTAMP	☐	Дата обработки последнего файла реплики.
		REPLKEY DRPLINTEGER_NOT_NULL INTEGER	☒	Значения счетчика файлов реплики передаваемых с нашей базы данных на базу идентифицируемую текущей записью.
		REPLDATE TIMESTAMP	☐	Дата передачи последнего файла реплики.

GD_RUID

Предназначена для хранения РУИДов записей. Данная таблица будет создаваться только для баз данных с первичнам ключом иникальным в пределах базы и в пределах отношения.

PK	FK	Поле Домен Тип данных	NN	Описание
1		ID DRPLINTEGER_NOT_NULL INTEGER	☒	Идентификатор записи в данной базе данных.
	F	Relationid DRPLINTEGER_NOT_NULL INTEGER		Идентификатор таблицы. Используется только в случае баз данных с суррогатным целочисленным ключем, уникальным в пределах одной таблицы.
		Xid DRPLINTEGER_NOT_NULL		Идентификатор записи в базе данных, где запись была создана.

		INTEGER		
		Dbid DRPLINTEGER_NOT_NULL INTEGER		Идентификатор базы данных, где была создана запись.

Приложения

Рекомендации по конфигурированию сервера и локальной сети предприятия[44]

Для нормальной работы (надежной и комфортной в плане скорости отклика системы) на предприятиях, где в системе Гедымин задействовано более 10 клиентских мест, необходимо обеспечить наличие выделенного сервера, на котором будет выполняться **только** сервер базы данных Interbase/Firebird/Yaffil.

Рекомендуемая конфигурация сервера

1. AMD Athlon 2600 и выше или Pentium 4 и выше. Только не Celeron и не Duron!
2. 2-3 Гб оперативной памяти (желательно на частоте 400 МГц или выше).
3. 2 быстрых винчестера на 7200 или 10000 оборотов с большим внутренним кэшем. Желательно Serial ATA или, еще лучше, Ultra SCSI. Если винчестеры IDE, то подключать их на **разные шлейфы**. Важно, не подключать на один и тот же шлейф винчестер и CD ROM!
4. блок бесперебойного питания, который бы мог обеспечить минимум 40-60 минут работы при отсутствии электроэнергии. Желательно, чтобы при перебоях с электроэнергией сервер выключался позже всех рабочих станций.

Рекомендуемые настройки операционной системы

1. Операционная система Windows 2003 или Windows 2000 Server.
2. Файловая система NTFS.
3. Операционная система, временный каталог и файл подкачки должны располагаться на одном **физическом** жестком диске.
4. Файл базы данных должен располагаться на другом физическом жестком диске.

Рекомендуемые настройки базы данных

1. Для обеспечения максимальной надежности режим принудительной записи должен быть **включен**. Для его активизации необходимо включить соответствующий флаг при восстановлении базы данных из архива из Гедымина.
2. Для обеспечения максимальной производительности необходимо установить следующий параметры при восстановлении базы данных из архива: размер страницы 16384 байт, количество страниц в буфере 32000.
3. В целях повышения производительности рекомендуется раз в неделю архивировать базу данных и восстанавливать ее из архива, заменяя восстановленным файлом рабочий файл базы данных. Очевидно, что в этом момент не должно быть подключенных пользователей. Лучше всего делать это с помощью специального скрипта, который будет запускаться по расписанию в ночной время. В последнем случае можно настроить ежедневное архивирование/разархивирование базы данных.
4. Рекомендуется раз в 7-10 дней перезагружать сервер. Очевидно, что в этом момент не должно быть подключенных пользователей. Лучше всего делать это по расписанию в ночное время.

Прочие настройки

1. Необходимо убедиться, что на сервере установлена версия Yaffil 885 и на каждом клиентском месте установлена библиотека gds32.dll версии 885. Для этого необходимо на каждом рабочем месте загрузить Гедымин и в окне «Справка»-«О программе» проверить соответствующие параметры.

---

[44] Рекомендации приведены по состоянию на январь 2005 г.

Архивное копирование

2. Архивное копирование необходимо производить не реже чем два раза в день. Лучше всего настроить его автоматически, написав скрипт и установив его выполнение по расписанию.
3. Новые архивы не должны затирать старые архивы!
4. Рекомендуется, периодически важные архивы переписывать на съемные носители (компакт-диски или ленту) и хранить отдельно.

Категорически запрещается

1. Ни в коем случае нельзя выключать сервер или принудительно перезагружать его, если в данный момент времени существуют клиентские подключения к базе данных. Перед перезагрузкой необходимо зайти в Гедымин под Администратором и в меню База данных выбрать пункт Подключенные пользователи. Если там только одна запись, то все нормально: нужно закрыть Гедымин и можно перезапускать сервер.
2. Нельзя в целях архивирования копировать непосредственно файл базы данных, если нет уверенности, что в данный момент времени нет активных клиентских подключений к серверу. Необходимо использовать команду архивного копирования Гедымина или утилиту командной строки для создания архивной копии.

Приведенные выше рекомендации действуют для сервера базы данных, работающего в архитектуре «супер».

Метапеременные в SQL запросах

Метапеременная -- это строковое выражение, имеющее определенный формат. Метапаременная может быть указана в любом месте SQL запроса. Перед выполнением запроса система Гедымин вместо метапеременных вставит в текст запроса соответствующие значения.

В системе Гедымин можно использовать следующие метапеременные в SQL запросах:

COMPANYKEY

CONTACTKEY

INGROUP

RUID

HOLDINGLIST

COMPANYKEY

Идентификатор текущей (рабочей) компании.

Синтаксис:

```
<COMPANYKEY/>
```

Пример использования:

данный запрос извлечет из базы данных все банковские счета текущей организации:

```
select * from gd_companyaccount where companykey = <COMPANYKEY/>
```

CONTACTKEY

Идентификатор текущего контакта (т.е. идентификатор контакта, соответствующего учетной записи текущего пользователя).

Синтаксис:

```
<CONTACTKEY/>
```

INGROUP

Битовый набор групп, в которые входит текущий пользователь.

Синтаксис:

```
<INGROUP/>
```

RUID

Возвращает идентификатор записи в текущей базе данных по ее РУИДу.

Синтаксис:

```
<RUID XID="xid_number" DBID ="dbid_number"/>
```

HOLDINGLIST

Возвращает список ключей компаний, входящих в текущую рабочую организацию, если она является холдингом. Список так же содержит ключ самой рабочей организации. Если текущая рабочая организация не является холдингом, то возвращается ее идентификатор.

Синтаксис:

```
<HOLDINGLIST/>
```

Параметры командной строки

Следующие параметры командной строки поддерживаются программой gedemin.exe:

/sn <database>

Задает файл базы данных. Требует задания имени сервера и полного пути к файлу базы данных на сервере.

Пример:

gedemin.exe /sn win2000server:k:\bases\gedemin\gdbase.gdb

/user <user_name>

Задает имя пользователя для входа в систему. Если задать имя пользователя и пароль, то окно входа в систему не будет выводиться на экран.

Пример:

gedemin.exe /user Administrator /password Administrator

/password <password>

Задает пароль пользователя для входа в систему. Если задать имя пользователя и пароль, то окно входа в систему не будет выводиться на экран.

Пример:

gedemin.exe /user Administrator /password Administrator

/unevent

Отключает определенные пользователем обработчики событий.

Пример:

gedemin.exe /unevent

/unmethod

Отключает перекрытые пользователем методы классов системы.

Пример:

gedemin.exe /unmethod

/settingpath или /sp

Задает путь к пакетам настроек. Аналогичен пути, задаваемому в окне 'Установка пекетов'. Используется совместно с параметром /settingfilename.

Пример:

gedemin.exe /SP c:\settings

/settingfilename или /sfn

Задает имя файла, содержащего конечную настройку для автоматической установки в систему. Используется совместно с параметром /settingpath и параметрами, задающими имя и пароль пользователя.

Пример:

gedemin.exe /sfn c:\settings\bank\bank.gsf

/ns

Не выводить заставку при запуске программы.

Пример:

gedemin.exe /ns

Предопределенные названия полей

Название поля	Тип	Комментарий
SUMNCU		Сумма в национальной денежной единице.
SUMCURR		Сумма в валюте.
CURRKEY		Используется в паре с полем SUMCURR. Ссылка на валюту (таблица GD_CURR) в которой выражено значение поля SUMCURR.

Быстрые клавиши

Настоящий список содержит наиболее часто используемые клавиатурные комбинации при работе с системой Гедымин. В каждом конкретном окне могут быть определены свои дополнительные комбинации клавиш, узнать про которые вы можете открыв интересующее вас окно и нажав клавишу F1 (вызов справки).

Некоторые клавиши имеют несколько значений в зависимости от текущего контекста. В этом случае, в приведенной ниже таблице перечисляются все значения с указанием соответствующего контекста.

Клавиша	Функция
F1	Открывает данное руководство пользователя.
F2	при работе с выпадающим списком: Создает новый объект.
F3	при работе с выпадающим списком: Осуществляет поиск согласно введенной строке. при работе с таблицей: Поиск или повторный поиск. при работе с деревом: Поиск или повторный поиск.
F4	при работе с выпадающим списком: Если в списке выбран объект, открывает его на редактирование.
F5	при работе с таблицей:

	Обновляет содержимое таблицы. Перечитывает его из базы данных. при работе с деревом: Обновляет дерево. Перечитывает данные из базы данных.
F6	Переход на следующее окно.
Ctrl-F6	Переход на предыдущее окно.
F7	при работе с выпадающим списком: Осуществляет точный поиск, согласно введенной строке.
F8	при работе с выпадающим списком: Удаляет выбранный объект из базы данных.
F9	при работе с выпадающим списком: Открывает просмотровую форму для выбранного объекта.
F10	при работе с таблицей: Открывает окно настройки таблицы.
F11	при работе с выпадающим списком: Открывает окно Свойства для выбранного объекта.
F12	при работе с выпадающим списком: переключает раскладку клавиатуры
Ctrl-F5	при работе с таблицей: Если на текущий запрос наложен фильтр с параметрами, то повторно запрашивает параметры и переоткрывает запрос. Если параметров нет, то просто перечитывает данные.
Ctrl-F11	Открывает и переходит в окно Исследователя.
Ctrl-Enter	при работе с формой просмотра: Открывает запись на редактирование. при работе с диалоговым окном: Закрывает окно. Аналогично нажатию на кнопку Ок (Готово). Такая комбинация понадобилась для того, чтобы диалоговое окно можно было быстро закрыть, когда курсор находится в таблице.
Ctrl-Del	Удаляет текущую запись.
Enter	Открывает запись на редактирование. Если таблица находится в режиме "Редактирование информации в таблице", то переводит текущее поле в режим редактирования.
Alt	Открывает меню формы.
Ins	Добавляет новую запись.
Ctrl-Alt-F	Открывает панель поиска.
Ctrl-D	Создает копию текущей записи и открывает ее на редактирование.
Ctrl-M	Открывает меню Макросы.
Ctrl-P	Открывает меню Печать.
Ctrl-R	при работе с выпадающим списком: Вызывает окно слияния записей. при работе с формой просмотра:

	Вызывает окно слияния записей.
Ctrl-F	при работе с таблицей: Поиск. при работе с деревом: Поиск.

"Хитрые" клавиши

Если при открытии формы нажать и удерживать клавишу Shift, то настройки формы загружаться не будут. При этом, сохранение настроек для этой формы будет отключено. То есть, загрузившись без настроек, вы не рискуете потерять при закрытии формы то, что было раньше. Если все-таки необходимо, чтобы настройки при закрытии формы сохранились необходимо нажать и удерживать клавишу Shift в момент закрытия.

Если при закрытии формы просмотра нажать и удерживать клавишу Shift, то форма будет не просто закрыта, но и уничтожена. Вся память и все ресурсы, выделенные и использованные данной формой будут освобождены.

Компиляция Гедымина

Исходные коды последней версии платформы Гедымин можно получить с официального сайта компании http://gsbelarus.com. Для компиляции выполняемого модуля вам потребуется персональный компьютер с установленной операционной системой Windows 2000, Windows XP или более поздней версией. Желательно наличие хотя бы 256 Мб оперативной памяти.

Стоит заметить, что для запуска и проверки в действии откомпилированного файла, вам потребуется три составляющие части:

1. собственно, сам выполняемый модуль gedemin.exe;
2. файл базы данных к которому вы будете подключаться;
3. установленный и настроенный сервер базы данных.

Мы рекомендуем, перед компиляцией платформы, скачать с сайта компании один из дистрибутивов и установить его в вашей локальной сети. Таким образом, вы получите настроенный и работающий сервер базы данных и установленный файл базы данных.

Ниже приводится пошаговый процесс сборки исполняемого модуля gedemin.exe из исходных кодов.

1. Установка Delphi 5

На компьютере должен быть установлен компилятор Borland Delphi 5 и (обязательно!) апдейт к нему. Файл динамической библиотеки midas.dll, необходимый для работы Гедымина, устанавливается в процессе инсталяции компилятора.

2. Установка GDS32.DLL

Динамическая библиотека GDS32.DLL должна находиться в каталоге WINDOWS\SYSTEM32 или в любом другом каталоге, перечисленном в перменной окружения PATH. Данная библиотека является клиентской частью сервера базы данных Interbase/Firebird/Yaffil.

3. Отключение компонент IBX

Так как Гедымин использует модифицированные компоненты IBX (Interbase Express), стандартные компоненты, входящие в поставку Delphi следует отключить. Для этого необходимо:

1. открыть Delphi, открыть меню Component и выбрать команду “Install packages…”.
2. В списке “Design packages” следует отыскать позицию “Interbase Data Access Components” и снять галочку напротив нее.
3. Далее, закрываем Delphi и удаляем из каталога “\Program Files\Borland\Delphi5\Bin” файл DCLIB50.BPL.
4. Из каталога “\WINDOWS\SYSTEM32” удаляем файл VCLIB50.BPL.

5. Заходим в каталог «\Program Files\Borland\Delphi5\Source\Vcl» и удаляем все файлы по маске IB*.*.

4. Распаковка архива

На диске d: создаем каталог с именем GOLDEN. Распаковываем в него содержимое архива с исходным кодом платформы.

В принципе, можно использовать любую букву диска и любое имя каталога, но в этом случае, будьте готовы внести соответствующие изменения во все DPK, DPR, BPG файлы.

5. Создание каталогов

По умолчанию, скомпилированные файлы будут размещаться в следующих каталогах:

- d:\golden\bpl — библиотеки компонентов;
- z:\dcu — DCU файлы;
- z:\ — откомпилированный файл gedemin.exe.

С созданием каталога d:\golden\bpl особых проблем возникнуть не должно.

Если у вас нет диска с именем z:, то следует, либо переименовать существующий физический диск, либо использовать утилиту subst. Например, создать папку c:\temp\dcu, открыть системный реестр с помощью утилиты regedit.exe и в ключе «HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run» добавить строковый параметр SUBST с содержимым «SUBST z: c:\temp\dcu». После чего следует перезагрузить компьютер.

6. Компиляция

После того, как необходимая структура каталогов создана, откроем в Delphi группу проектов d:\golden\gedemin\gedemin.bpg, которая содержит следующие файлы:

- SynEdit_D5.bpl — визуальные компоненты редактора программного кода с подсветкой синтаксиса;
- tb2k_d5.bpl — набор визуальных компонентов Toolbar 2000;
- tb2kdsgn_d5.bpl — набор визуальных компонентов Toolbar 2000 (design package);
- fr5.bpl — генератор отчетов FastReport. Внимание! Компоненты FastReport являются платным программным продуктом. Лицензию на их использование необходимо приобретать отдельно. Информацию можно получить на сайте http://fastreport.ru. Только не устанавливайте приобретенный самостоятельно дистрибутив поверх файлов, входящих в проект Гедымин, так как в них были внесены некоторые изменения, которых нет в оригинальной поставке;
- djcl50_2.bpl — библиотека компонентов JCL (JEDI Code Library);
- greference.bpl — библиотека визуальных компонентов компании Golden Software, используемых проектом Гедымин;
- gedemin.exe — исполняемый модуль платформы;
- upgrade.exe — утилита обновления базы данных;
- gudf.dll — библиотека UDF (User Defined Function) функций для сервера базы данных.

Поочередно перестроим (команда Build) каждый проект[45]. Библиотеки компонентов следует установить — команда Install, вызываемая из контекстного меню в дереве проектов.

После компиляции, файл gudf.dll следует перенести из каталога d:\golden\gedemin\gudf в UDF каталог сервера базы данных. Обычно, это каталог c:\program files\yaffil\udf на том компьютере, где установлен сервер.

[45] Важно использовать именно команду Build, а не Compile, так как каждый проект может использовать свои символы условной компиляции.

7. Подключение к базе данных

После выполнения предыдущего шага, в каталоге z:\, будет находиться скомпилированный файл gedemin.exe. Прежде чем запускать его следует установить базу данных из дистрибутива или собрать ее.

Первый вариант — более простой. Для установки базы данных следует скачать с сайта компании Golden Software один из дистрибутивов Гедымина и установить его на компьютере. После этого можно запускать скомпилированный модуль и подключаться к установленной базе данных.

Второй вариант заключается в том, чтобы собрать чистую (эталонную) базу данных. Необходимые SQL скрипты находятся в каталоге d:\golden\gedemin\sql. Там же находится пакетный файл create.bat, с помощью которого вы сможете собрать базу.

Пакетный файл create.bat требует наличия двух утилит: isql.exe и makelbrbtree.exe, в одном с собой каталоге. Первая утилита входит в поставку сервера Interbase/Firebird/Yaffil и является интерфейсом командной строки. Вторая утилита предназначена для сканирования структуры базы данных и создания необходимых триггеров и индексов для таблиц, имеющих структуру интервальных деревьев.

8. Установка настроек

Воспользовавшись одним из имеющихся на сайте дистрибутивов вы получите базу данных с установленными прикладными решениями. Но, если вы создали чистую эталонную базу из SQL скриптов, прикладные настройки необходимо установить вручную. Для этого:

1. скачайте соответствующий архив с сайта компании и распакуйте его в произвольный каталог;
2. запустите gedemin.exe и подключитесь к эталонной базе данных;
3. из меню «Сервис» выберите команду «Установить пакеты настроек…»;
4. отметьте галочками нужные пакеты и нажмите кнопку «Установить».

Структура каталогов

Исходный код проекта Гедымин располагается в следующих каталогах и их подкаталогах:

\golden\comp5\	Некоторые визуальные компоненты.
\golden\setting\	Прикладные настройки.
\golden\gedemin\	Исходный код проекта Гедымин.

Папка \golden\gedemin содержит следующие вложенные папки:

AddressBook	Справочник, клиентов, физических лиц, банков, расчетных счетов.
Attr	Атрибуты.
BugBase	Регистрация ошибок и пожеланий.
ClassTree	В основном не используется, но содержит файл с разборщиком SQL запросов.
Common	Набор вспомогательных форм и модулей.
Component	Визуальные и не визуальные компоненты в том числе бизнес-классы и типовые экранные формы.
Const	Формы и модули бизнес-класса TgdcConst.
Designer	Редактор скрипт-объектов, редактор экранных форм.
DirectoryGood	Файлы, относящиеся к бизнес-классу «Товар».
Doc	Разнообразная документация.
Document	Файлы, относящиеся к бизнес-классам «Документ», «Типовой документ».
FastReport	Библиотека FastReport. Внимание! Код библиотеки предоставляется только для ознакомления. В случае использования, Вы обязаны самостоятельно приобрести лицензию. Код библиотеки изменен, для того, чтобы сделать ее совместимой с Гедымином. Не устанавливайте новые версии в этот каталог!

GAdmin	Файлы, относящиеся к бизнес-классам «Пользователь», «Группа пользователей». Так же, в данном каталоге, находится дата-модуль, содержащий компонент подключения к базе данных.
gdcFile	Файлы, относящиеся к бизнес-классу «Файл».
Gedemin	Проект gedemin.bpg и относящиеся к нему файлы. Главная форма программы.
GReference	Пакет визуальных компонент, используемых в Гедымине.
GUDF	Библиотека UDF для сервера Interbase/Yaffil/Firebird.
IBX	Модернизированный вариант компонентов Interbase Express из поставки Delphi 5.
Images	Картинки, пиктограммки и т.п.
Imports	Импортированные файлы заголовков для Microsoft Script Control и HTML Help.
Inventory	Файлы, относящиеся к подсистеме складского учета.
jcl	Библиотека вспомогательных функций JEDI Code Library (JCL).
Log	Файлы бизнес-класса «Журнал действий пользователя».
Messaging	Файлы, относящиеся к бизнес-классу «Сообщение».
Modem	Библиотека для работы с модемом.
NewTransaction	Файлы подсистемы «Бухгалтерский учет».
Property	Редактор скрипт-объектов.
QueryFilter	Визуальный редактор условий для SQL запросов.
Registration	Защита от несанкционированного использования, регистрация программы.
Report	Подсистема отчетов.
Security	Окно входа в систему (ввода пользователя и пароля).
SQL	Файлы с SQL скриптами для генерации эталонной базы данных.
Storage	Хранилище.
Tax	Экранные формы и файлы, относящиеся к налогам.
ToDo	Документация и технические задания по проекту.
Transaction	Файлы, относящиеся к бизнес-объектам бухгалтерская проводка и хозяйственная операция.
Utility	Вспомогательные утилиты с исходниками.
Wage	Файлы, относящиеся к бизнес-классам подсистемы «Зарплата и Кадры».
zlib	Общедоступная библиотека для упаковки/распаковки данных zlib.

Символы условной компиляции

В следующей таблице приводится список символов условной компиляции, определяя которые можно включать ту или иную функциональность в состав модуля gedemin.exe.

Символ	Значение
DEBUG	Включает отладочную информацию и диагностические сообщения.
SPLASH	Включает вывод заставки при запуске программы.
MESSAGE	Определение данного символа приводит к компиляции кода некоторых бизнес-классов, необходимых для функционирования подсистемы «CRM».
SYNEDIT	Включение в проект кода редактора программного кода с подсветкой

	синтаксиса — SynEdit.
WAGE	Определение данного символа приводит к компиляции кода некоторых бизнес-классов, необходимых для функционирования подсистемы «Зарплата и Кадры».
GEDEMIN	Данный символ должен быть определен при компилировании файла gedemin.exe. Используется для выделения кода, который должен быть откомпилирован при сборке gedemin.exe и не должен компилироваться при сборке пакета компонент — greference.dpk, — или других проектов.
MODEM	Данный символ условной компиляции включает в проект код работы с модемом.
GEDEMIN_LOCK	Активизация кода защиты от копирования.
GED_LOC_RUS	Определение данного символа условной компиляции включает перевод на русский язык некоторых строковых констант, сообщений программы.
LOCALIZATION	При компиляции с данным символом включается код локализации пользовательского интерфейса. Выбор языка пользовательского интерфейса осуществляется с помощью параметров командной строки. Более подробно см. «Настройка ярлыка для запуска Гедымина».

Глоссарий

SDI (Single Document Interface)

Бизнес объект

План счетов

План счетов бухгалтерского учета — это систематизированный перечень применяемых счетов, в котором они группируются по разделам. Каждый счет имеет свое наименование и цифровой код. К ряду счетов предлагаются наименования и цифровые коды субсчетов.

План счетов активный

План счетов базовый

См. План счетов активный.

Позиция документа

Организация рабочая

Организация, по которой в системе ведется учет.

Организация текущая

Одна из рабочих организаций, являющаяся активной в данный момент.

Репликация

Процесс переноса изменений от одной базы данных к другой.

Хранилище

Древовидная, иерархическая, многоуровневая база данных, предназначенная для хранения параметров системы. На платформе Гедымин существуют четыре объекта хранилища: глобальное, пользовательское, хранилище рабочей компании и хранилище рабочего стола.

Хранилище глобальное

Хранилище глобальных (единых для всех пользователей и рабочих организаций) параметров платформы.

Хранилище пользовательское

Хранилище параметров платформы, привязанных к конкретному пользователю.

Хранилище рабочего стола

Хранилище параметров платформы, привязанных к конкретному рабочему столу.

Хранилище рабочей компании

Хранилище параметров платформы, привязанных к рабочей организации.

Шапка документа

Эталонная база данных

Чистая база данных без загруженных настроек.

Список литературы

1. Ковязин А., Востриков С. Мир Interbase. Архитектура, администрирование и разработка приложений баз данных в Interbase/Firebird/Yaffil — М.: КУДИЦ-ОБРАЗ, 2002. — 432 с. ISBN 5-93378-042-1.
2. Тексейра Стив, Пачеко Ксавье. Delphi 5. Руководство разработчика том 1-2. Основные методы и технологии программирования: Пер. с англ.: Уч. пос. — М.: Издательский дом «Вильямс», 2000. — 832с.: ил. — Парал. тит. англ.

Предметный указатель