БОРИС ПАХОМОВ

Interbase и C++Builder

НА ПРИМЕРАХ

ПРОЕКТИРОВАНИЕ БАЗ ДАННЫХ

ОСНОВЫ ЯЗЫКА SQL

ХРАНИМЫЕ ПРОЦЕДУРЫ, ТРИГГЕРЫ, ГЕНЕРАТОРЫ, ИСКЛЮЧЕНИЯ И ПРИВИЛЕГИИ

ПРОГРАММИРОВАНИЕ В СРЕДЕ BORLAND C++ BUILDER

ВЕДЕНИЕ УЧЕТНОЙ КАРТОЧКИ РАБОТНИКА

УЧЕТ ДВИЖЕНИЯ МАТЕРИАЛОВ НА СКЛАДАХ

РАСЧЕТ КРАТЧАЙШЕГО ПУТИ МЕЖДУ ДВУМЯ ТОЧКАМИ СЕТИ

+CD

bhv®

Борис Пахомов

Interbase *и* C++Builder
НА ПРИМЕРАХ

Санкт-Петербург
«БХВ-Петербург»
2006

УДК 681.3.068
ББК 32.973.26-018.1
П12

Пахомов Б. И.

П12 Interbase и C++ Builder на примерах. — СПб.: БХВ-Петербург, 2006. — 288 с.: ил.

ISBN 5-94157-570-X

На практических примерах решения типичных задач по управлению кадрами, учету движения материалов на складах и нахождению оптимального пути между двумя пунктами показан процесс проектирования и программной реализации баз данных с использованием популярной СУБД Interbase и среды разработки Borland C++ Builder. Рассмотрены теоретические основы проектирования баз данных: модель базы данных, идентификация сущностей и атрибутов, создание индексов и набора правил при разработке таблиц и др. Дан обзор инструментальных средств Interbase. Описаны основные элементы СУБД Interbase: таблицы, триггеры, процедуры, исключения, привилегии и др. Уделено внимание составлению различных запросов на языке SQL. Рассмотрены основные компоненты среды Borland C++ Builder при разработке приложений баз данных.

На прилагаемом компакт-диске размещены учебные базы данных и исходные коды программ, описанных в книге.

Для начинающих программистов

УДК 681.3.068
ББК 32.973.26-018.1

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Игорь Шишигин*
Зав. редакцией	*Григорий Добин*
Редактор	*Елена Кашлакова*
Компьютерная верстка	*Натальи Смирновой*
Корректор	*Наталия Першакова*
Дизайн серии	*Инны Тачиной*
Оформление обложки	*Елены Беляевой*
Зав. производством	*Николай Тверских*

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 30.08.06.
Формат $70\times100^{1}/_{16}$. Печать офсетная. Усл. печ. л. 23,22.
Тираж 3000 экз. Заказ №
"БХВ-Петербург", 194354, Санкт-Петербург, ул. Есенина, 5Б.

Санитарно-эпидемиологическое заключение на продукцию № 77.99.02.953.Д.006421.11.04
от 11.11.2004 г. выдано Федеральной службой по надзору
в сфере защиты прав потребителей и благополучия человека.

Отпечатано с готовых диапозитивов
в ГУП "Типография "Наука"
199034, Санкт-Петербург, 9 линия, 12

ISBN 5-94157-570-X

Оглавление

Введение

Предлагаемая читателю книга посвящена проблемам знакомства с одной из современных передовых систем обработки информации — средой Borland C++Builder. В предыдущих книгах почти не рассматривался вопрос работы с системами управления базами данных (СУБД), и этот пробел автор попытался восполнить в данной работе. Кроме этой цели была и другая: показать начинающему, как на практике применить полученные знания. Для этого автором были разработаны программы в среде Borland C++Builder для решения реальных задач современного предприятия: создание и ведение картотеки учета персонала предприятия, на базе которой можно построить решение общей задачи управления кадрами, и создание и ведение компьютерного складского учета, на базе которого можно строить решение более общей задачи по учету движения материальных ценностей на предприятии. В основе обеих задач лежит использование средств современной СУБД Interbase, которая обеспечивает создание и ведение баз данных в сетевом варианте в режиме "клиент-сервер", имеет она и другие полезные возможности. Эта СУБД может использоваться во взаимодействии со средой Borland C++Builder через имеющиеся в ней специальные компоненты.

При создании программных продуктов использовался Borland C++Builder 6 и встроенная в него СУБД Interbase 6. Для нынешнего времени это несколько староватые версии, ведь в 2006 г. вышли в свет Borland C++Builder 2006 и Interbase 8, а Interbase 7 существует с 2002 г. Однако не станем огорчаться, ибо дела обстоят несколько иначе. Наша основная цель — обучить начинающего программиста работе с СУБД. Он должен узнать, что такое база данных, какова ее подробная структура, ее возможности, как строить ее элементы и как ими эффективно пользоваться при решении задач. На эти вопросы успешно отвечает версия 6. Если в версии 7 добавлен, например, новый тип данных BOOLEAN и удален оператор SET TERM, если седьмая версия обеспечивает многопроцессорную работу серверов и возможность получения статистической информации о списках пользователей, подсоединенных к базе данных, о том, сколько времени работает данный пользователь и какие запросы он выполняет, к каким таблицам обращается и т. п., то такие вопросы начинающего мало волнуют — это задачи администратора СУБД, а ему, начинающему, в первую очередь надо узнать, как построить свои базы данных, как их наполнять информацией и все остальное в этом же роде. Версия 6 Interbase срослась с Borland C++Builder 6, широко распространенным сегодня, который для начала вполне может быть удовлетворительным: из него взяты практически те же компоненты, которые использовались бы и в версии 2006 (третья книга автора как раз и посвящена этой

версии), которая только-только поступает на рынок программных продуктов. К тому же, Interbase 6 — продукт бесплатный. Кстати, все написанное работает и в СУБД Firebird, которая также является бесплатной.

Читающему эту книгу надо быть знакомым с основными компонентами Borland C++Builder. Здесь он познакомится только с Interbase: увидит и научится, на что автор надеется, как применять знания по СУБД и Borland C++Builder в решении практических задач предприятия.

Отметим еще один момент. В созданных и прилагаемых к настоящей книге комплексах программ (обработка карточки по учету кадров и обработка данных складского учета) имеются очень похожие обработчики событий, в частности, компонентов ComboBox. И довольно большие по объему. Многие из этих компонентов можно было бы заменить одной функцией с параметрами. Однако этого не сделано с определенной педагогической целью: пусть читатель постоянно натыкается на одни и те же малоразличимые блоки, привыкая к их содержимому. Для начинающего простые блоки, естественно, более понятны, чем усложненные функции, построенные как обобщающие элементы этих блоков. По мере привыкания читателя к повторяющимся блокам у него самого возникнет потребность их обобщить, и тогда он начнет искать способ, как это сделать: в программе по складскому учету в одном случае показано, как это делается.

Тексты обоих комплексов программ, снабженные в наиболее трудных местах комментарием, совместно с двумя базами данных для каждого из них прилагаются на компакт-диске.

И еще одно замечание. Для обслуживания результатов работы по вводу данных и расчету документов предусмотрены выводы результатов в табличном представлении, для чего использовались компоненты TDGrid. Некоторые из этих компонентов наделены свойствами причаливания (Docable). В частности, после щелчка на таком элементе он приобретает свойства перемещаемого и изменяющего свои размеры окна. Это сделано для таких объектов, как справочник материалов, справочник поставщиков-покупателей и др. Дело в том, что при проектировании таких справочников из-за большого количества имеющихся у них полей последними занимается почти вся площадь рабочего стола, и чтобы видеть всю картину в момент отладки, желательно наблюдать и поля ввода, и в целом всю таблицу через элемент TDBGrid. А для него остается небольшое пространство рабочего стола. При проектировании это еще можно пережить, но в реальной эксплуатации такое положение создает неудобства. Поэтому следует создать возможность перетаскивания мышью элемента TDBGrid в более удобное для пользователя место рабочего стола, одновременно давая возможность изменять размеры элемента.

Желаю успеха изучающим этот материал.

Глава 1

Базы данных. Interbase

Interbase — это система управления базой данных (СУБД), имеющая собственную структуру. Предоставляет пользователю возможность не только создавать собственно БД (структуру и элементы), но и наполнять ее данными, модифицировать ее, поддерживать в актуальном состоянии и пользоваться ее содержимым. У нее есть собственный алгоритмический язык. Этот язык — подмножество языка структурированных запросов — SQL. Interbase содержит в себе средства обеспечения собственной безопасности. Хотя Interbase может применяться в качестве автономного программного продукта при работе с базами данных, ее возможности можно использовать и при создании приложений в среде разработки Borland C++Builder, в которой имеется набор компонентов для работы с базами данных этого типа. Interbase обеспечивает работу в системе "клиент-сервер", давая возможность пользователям создавать свои серверы с базами данных, которые обслуживают сеть удаленных клиентов. Чтобы лучше понимать работу компонентов раздела Interbase среды Borland C++Builder, рассмотрим основные положения СУБД Interbase.

Обзор некоторых инструментальных элементов Interbase

IBConsole

Это графический пользовательский интерфейс, представленный отдельным программным продуктом. В его среде можно конфигурировать и поддерживать InterBase-сервер, создавать и вести базы данных на этом сервере, интерактивно работать на языке SQL, поддерживать пользователей и обеспечивать безопасность.

IBConsole запускается под Windows, однако может управлять базами данных на любом InterBase-сервере в локальной сети, в UNIX, Linux и NetWare-системах.

SQL

Interbase поддерживает язык структурированных запросов (SQL), с помощью которого можно работать с базами данных. Осуществлять эти запросы можно с помощью так называемого интерактивного SQL — isql, либо запуская из командной строки консольного режима (запуск интерактивного SQL из командной строки выполняется с помощью специальной утилиты с именем isql), после чего можно либо набирать сами операторы SQL, либо задать имя файла с операторами SQL, который будет исполняться, либо можно воспользоваться средствами IBConsole, в меню которого имеется режим выхода на работу с интерактивным SQL.

При создании базы данных встречаются с понятием диалекта SQL. Диалектов существует несколько. Дело в том, что некоторые элементы языка, такие как разделители объектов (двойные кавычки, например), данные точных типов (не с плавающей точкой, которые, по определению, не являются точными), такие как DECIMAL и NUMERIC, дата и время, по-разному интерпретируются в разных версиях Interbase. Чтобы добиться совместимости с предыдущими версиями, и было введено понятие диалекта SQL. В Interbase 6, который мы здесь рассматриваем, диалекты 1 и 2 обеспечивают совместимость с предыдущими версиями Interbase, диалект 3 — это версия SQL-92, интерпретируемая в Interbase 6.

Приведем перечень некоторых SQL-операторов, подробный смысл большинства из которых определен в последующем материале:

- ALTER DATABASE — добавляет вторичные файлы к текущей базе данных;
- ALTER DOMAIN — изменяет определение домена;
- ALTER EXCEPTION — изменяет текст сообщения, связанного с существующим исключением;
- ALTER INDEX — активирует или деактивирует индекс;
- ALTER PROCEDURE — изменяет определение существующей хранимой процедуры;
- ALTER TABLE — изменяет таблицу путем добавления, удаления или модификации колонок или ограничений на целостность данных;
- ALTER TRIGGER — изменяет существующий триггер;
- COMMIT — делает изменения транзакции неизменными и завершает транзакцию;

- CONNECT — соединяет с одной или большим количеством баз данных;
- CREATE DATABASE — создает новую базу данных;
- CREATE DOMAIN — задает определение колонки, которое становится глобальным в базе данных;
- CREATE EXCEPTION — создает определенную пользователем ошибку и связанное с ней сообщение об ошибке для использования в хранимых процедурах и триггерах;
- CREATE GENERATOR — создает генератор к базе данных;
- CREATE INDEX — создает индекс на одной и более колонках таблицы;
- CREATE PROCEDURE — создает хранимую процедуру, ее входные и выходные параметры и ее действия;
- CREATE ROLE — создает роль (пользователя или группу, обладающих определенными правами доступа. Например, роль "администратор");
- CREATE SHADOW — создает одну и более копий базы данных;
- CREATE TABLE — создает таблицу базы данных;
- CREATE TRIGGER — создает триггер базы данных;
- CREATE VIEW — создает вьюер базы данных;
- DECLARE EXTERNAL FUNCTION — объявляет функцию к базе данных (объявление будет храниться в базе данных), определенную пользователем (user-defined function — UDF): имя, где ее найти, входные параметры, единственное возвращаемое ею значение. Каждая UDF в библиотеке должна быть объявлена один раз для каждой базы данных, где она будет использована;
- DECLARE FILTER — объявляет существующий BLOB-фильтр к базе данных (объявление будет храниться в базе данных): где его найти, его имя, с какими типами данных он работает. BLOB-фильтр — это написанная пользователем программа, которая преобразует данные, хранящиеся в колонках данных типа BLOB (об этом и других типах будет сказано далее) в другой тип данных;
- DECLARE TABLE — объявляет таблицу базы данных;
- DELETE — удаляет строки из таблицы;
- DISCONNECT — отсоединяет приложение от базы данных;
- DROP DATABASE — удаляет базу данных;
- DROP DOMAIN — удаляет домен из базы данных;
- DROP EXCEPTION — удаляет исключение из базы данных;
- DROP EXTERNAL FUNCTION — удаляет объявление UDF из базы данных;

- DROP FILTER — удаляет объявление фильтра из базы данных;
- DROP INDEX — удаляет индекс из базы данных;
- DROP PROCEDURE — удаляет хранимую процедуру из базы данных;
- DROP ROLE — удаляет роль из базы данных;
- DROP SHADOW — удаляет набор копий базы данных;
- DROP TABLE — удаляет таблицу из базы данных;
- DROP TRIGGER — удаляет триггер из базы данных;
- DROP VIEW — удаляет вьюер из базы данных;
- EXECUTE PROCEDURE — вызывает хранимую процедуру;
- GRANT — назначает привилегии пользователям базы данных;
- INSERT — добавляет одну и более строк к таблице;
- REVOKE — отбирает привилегии у пользователей базы данных;
- ROLLBACK — восстанавливает базу данных в ее предыдущее (перед началом транзакции) состояние;
- SELECT — извлекает данные из одной и более таблиц базы данных;
- SET SQL DIALECT — задает диалект SQL для доступа к базе данных;
- SET STATISTICS — пересчитывает избирательность указанного индекса (избирательностью пользуется оптимизатор Interbase для планирования исполнения запроса на выборку);
- SET TRANSACTION — запускает транзакцию, определяя ее взаимодействие с базой данных и другими транзакциями;
- UPDATE — изменяет данные в строке таблицы или вьюера;
- WHENEVER — задает обработку ошибок при исполнении SQL-операторов.

Проектирование баз данных

С помощью базы данных описывают не только информацию о реальных предприятиях и организациях, но и обеспечивают ее обработку, символически представляя реальные объекты своими специальными структурами: таблицами, вьюерами, хранимыми процедурами и другими объектами. Поскольку информация в базе данных организована и хранится в виде определенных объектов, то к таким объектам может быть организован доступ с помощью приложений, создаваемых в различных программных средах и пользовательских интерфейсах, формируемых с помощью средств БД.

Одним из главных факторов в создании базы данных является ее правильное проектирование. Обычно перед помещением в базу данных информация представляется в виде прямоугольных таблиц, состоящих из строк и столбцов (колонок) и отражающих определенные сущности действительности. Например, процесс начисления и выдачи зарплаты работникам предприятия можно отобразить в виде такой прямоугольной таблицы, в которой в качестве столбцов будут характеристики работника и виды начислений и удержаний, даты оплаты, а в качестве строк — конкретные работники. Логическое проектирование БД — это интерактивный процесс, состоящий из расчленения на мелкие, элементарные данные больших структур информации. Этот процесс носит название нормализации (в философском смысле этот процесс носит название анализа). Цель нормализации состоит в определении природных связей между данными в будущей базе данных. Это делается путем разбиения конкретной таблицы на более мелкие таблицы с меньшим количеством столбцов. После такого разбиения лучше видно, каковы на самом деле связи между построенными таблицами (а этот обратный процесс в философии носит название синтеза).

Модель базы данных

При проектировании базы данных важно представлять различие между описанием базы данных (БД) и самой базой данных. Описание БД называют моделью данных. Она формируется на этапе проектирования БД. Модель — это шаблон для создания таблиц и их колонок. Он описывает логическую структуру БД, включая данные и их суть, типы данных, пользовательские операции, связи между объектами-данными, ограничения на целостность данных. Логические структуры, описывающие базу данных, не подвержены воздействиям со стороны соответствующих физических структур, хранимых в БД. *Реляционную базу данных* (базу данных, в которой отношения между данными определены) нетрудно переносить на различные платформы, потому что механизм доступа к базе данных определен моделью базы данных и остается неизменным, где бы база данных ни хранилась. Логическая структура БД также независима от взгляда на нее конечного пользователя. Логическая структура, если пользоваться строительной терминологией, это проект дома, а сама БД — дом, построенный по этому проекту. Поэтому, имея проект "дома", его можно построить и на земле (платформе) Windows, и на земле (платформе) UNIX, и т. д.

При создании базы данных ее можно настраивать на различные информационные потребности пользователя. Это делается с помощью так называемых *вьюеров* (viewers) — программ просмотра данных, которые выводят подмножества данных по заказам конкретных пользователей или их групп. Вьюеры могут использоваться, чтобы спрятать необходимые данные или отфильтровать данные, которые не требуются пользователю.

Цели проектирования

При проектировании базы данных разработчики преследуют определенные цели, которые, в основном, состоят в следующем:

- удовлетворить пользовательским требованиям к базе данных. До ее проектирования надо исследовать требования пользователей к БД, обеспечение пользовательского объема информации, выборки по запросам, расчет необходимых данных, безопасность информации и т. п.;
- обеспечить связность и целостность данных. Не должно быть такой ситуации, когда, например, строка из одной таблицы модифицируется или удаляется, а связанные с ней данные в другой таблице автоматически не претерпевают никаких изменений;
- обеспечить пользователя естественным, легко понимаемым структурированием информации. Хороший дизайн базы данных позволяет легче с ней общаться. Такая БД проще поддерживается и обновляется;
- удовлетворять пользовательским требованиям по эксплуатации БД. Хорошо спроектированная БД дает лучшие результаты по ее эксплуатации. Если таблицы слишком велики или у них слишком много или слишком мало индексов (об индексах и их свойствах — позднее), результатом будет большое время ожидания результатов. Если БД слишком велика и с высоким объемом транзакций (о них тоже поговорим позже), проблемы производительности во многом будут зависеть от качества проектирования (дизайна).

Структура проектирования БД

При проектировании базы данных выполняются следующие конкретные действия:

- определение требований к информации (объем, частота использования, безопасность и т. д.) путем опроса будущих пользователей;
- анализ реальных объектов, которые требуется смоделировать в БД. Перевод управления объектами в управление элементами БД и формирование списка элементов БД (синтез БД);
- решение вопросов идентификации элементов в БД;
- разработка набора правил доступа к каждой таблице (того, как каждая таблица станет наполняться и модифицироваться);
- установка отношений между объектами (таблицами и колонками);
- планирование безопасности БД.

Сбор и анализ данных

Перед проектированием основных объектов базы данных — таблиц и их колонок — необходимо провести анализ реальных данных на концептуальном уровне:

- главные функции и деятельность предприятия. Например, наем работников, морская перевозка продукции, управление запчастями, обработка платежных чеков и т. д.;
- объекты этой деятельности и функции (т. е. на что направлена деятельность). Соединение последовательности деловых операций или транзакций (групп операций, имеющих законченное экономическое действие — например, бухгалтерская проводка) в единую последовательность событий поможет определить все сущности и отношения в операционных актах. Например, когда вы наблюдаете за процессом найма рабочей силы, вы сможете сразу выделить и идентифицировать такие сущности этой деятельности, как должность (JOB), работник (EMPLOYEE) и подразделение (DEPARTMENT);
- характеристики этих объектов. Например, сущность EMPLOYEE может включать такую информацию, как табельный номер работника (EMPLOYEE_ID), имя (FIRST_NAME), фамилию (LAST_NAME), вид работы (JOB), зарплату (SALARY) и т. д.;
- отношения между объектами. Например, как такие сущности, как EMPLOYEE, JOB и DEPARTMENT, соотносятся между собой? Допустим, работник имеет одну должность и входит в одно подразделение, в то время как одно подразделение имеет множество работников и должностей.

Идентификация сущностей и атрибутов

Основываясь на требованиях заказчика (пользователя), которые вы собрали, можно определить сущности и их атрибуты. Сущность — это тип, объект или вещь, которые нуждаются, чтобы их описали в базе данных. Это может быть физический объект или компания, должность или проект. Каждая сущность имеет свойства, которые называют ее атрибутами. Допустим, что проектируется база данных, которая будет содержать сведения о работниках компании, об иерархии подразделений компании, о ее текущих проектах, о клиентах (покупателях) и продажах. Пример в табл. 1.1 показывает, как создать список сущностей и атрибутов для организации данных.

Таблица 1.1. *Сущности и атрибуты данных*

Сущность	Атрибуты сущности
EMPLOYEE	Employee Number (Табельный номер) Last Name (Фамилия) First Name (Имя) Department Number (Код подразделения) Job Code (Код должности) Phone Extension (Телефон) Salary (Зарплата)
DEPARTMENT	Department Number (Код подразделения) Department Name (Наименование) Department Head Name (Имя руководителя) Budget (Бюджет) Location (Месторасположение) Phone Number (Телефон)
PROJECT	Project ID (Код проекта) Project Name (Название проекта) Project Description (Описание проекта) Team Leader (Руководитель) Product (Изделие)

Составляя таким способом списки сущностей и соответствующих атрибутов, удается удалить избыточные записи. Могут быть таблицами сущности из составленного списка? Могут ли быть перемещены колонки из одной группы в другую? Появляется ли один и тот же атрибут в разных сущностях? На такие вопросы надо дать ответ. Каждый атрибут может появляться лишь однажды, и вы должны определить, какая сущность является первичным собственником этого атрибута. Например, атрибут DEPARTMENT HEAD NAME (имя руководителя) может быть удален, так как имя работника (FIRST NAME и LAST NAME) и его табельный номер уже существуют в сущности EMPLOYEE. А для доступа к руководителю подразделения можно воспользоваться табельным номером работника (руководитель в качестве работника имеет свой табельный номер).

Далее описывается, как отобразить составленный вами список в объекты базы данных: сущности — в таблицы, а атрибуты — в колонки.

Проектирование таблиц

В реляционной базе данных объектом базы данных, представляющим отдельную сущность, является таблица, которая является двумерной матрицей строк и столбцов. Каждый столбец матрицы представляет один атрибут. Каждая строка представляет специфический экземпляр сущности. Например, таблица "Работники" (EMPLOYEE) — это сущность. Ее атрибуты — код работника, фамилия, имя, контактный телефон и т. д. Специфические экземпляры сущности — это первый работник, второй работник и т. д. После определения сущностей и атрибутов создается модель данных, которая служит логическим проектом структуры вашей InterBase-базы данных. Модель данных является детальным описанием сущностей базы данных — таблиц, колонок, свойств колонок и отношений между таблицами и колонками. Пример в табл. 1.2 показывает, как сущность EMPLOYEE была преобразована из списка сущности/атрибуты в таблицу/колонки.

Таблица 1.2. *Таблица EMPLOYEE*

EMP_NO	LAST_NAME	FIRST_NAME	DEPT_NO	JOB_CODE	PHONE_EXT	SALARY
24	Smith	John	100	Eng	4968	64000
48	Carter	Catherine	900	Sales	4967	72500
36	Smith	Jane	600	Admin	4800	37500

Каждая строка таблицы EMPLOYEE представляет одного работника. EMP_NO, LAST_NAME, FIRST_NAME, DEPT_NO, JOB_CODE, PHONE_EXT и SALARY — это колонки, которые представляют атрибуты работника. Когда строка добавляется к таблице, элемент строки называется в таблице ячейкой. Вся строка записи называется в таблице просто записью. Колонки еще называют полями таблицы.

Определение неповторяющихся атрибутов

Одной из задач проектирования базы данных является обеспечение уникального определения каждого экземпляра (говоря простым языком, необходимо сделать так, чтобы система могла извлечь из таблицы единственную строку). Одну строку от другой отличают по значениям так называемого *первичного ключа*. Первичный ключ определяет неповторимость: значения, введенные в колонку первичного ключа или во множество колонок, составляющих первичный ключ (а он может состоять и из нескольких атрибутов), являются уникальными для каждой строки. Если вы попытаетесь ввести в первичный ключ значение, которое уже существует в другой строке той же колонки, InterBase прекратит операцию и выдаст ошибку. Например, в таб-

лице EMPLOYEE поле EMP_NO является уникальным атрибутом, который может использоваться в качестве идентификатора каждой строки таблицы. Следовательно, этот реквизит можно взять в качестве первичного ключа этой таблицы. Когда вы выбираете какой-то реквизит на роль первичного ключа, проверьте, действительно ли он уникален, т. е. не встречается более одного раза во всех строках таблицы.

Если не существует единственной колонки, обладающей уникальностью, то следует определить первичный ключ на основе двух или более колонок, которые вместе дают требуемую однозначность.

Вместе с первичным ключом таблицы (PRIMARY KEY) существует и *уникальный ключ* (UNIQUE KEY). В колонку, выбранную в качестве уникального ключа, вводятся уникальные значения. У таблицы может быть только один первичный ключ и любое количество уникальных. Если поле помечено как уникальное, то в него можно вводить для каждой строки только различные данные. Если в некоторую строку данного поля ввести данное, которое уже встречалось в какой-либо другой строке, система выдаст сообщение об ошибке. Так работает Interbase. Первичный ключ и уникальный ключ задаются при создании таблицы.

Создание набора правил при разработке таблицы

При разработке таблицы базы данных требуется создать набор правил для каждой таблицы и колонки, устанавливающих и обеспечивающих целостность данных (об этом мы говорили ранее: связанные данные должны и модифицироваться, и удаляться совместно). Эти правила включают:

- определение типов данных;
- выбор интернациональных наборов символов для интерпретации данных;
- создание колонок, основанных на доменах: например, в разных таблицах имеются колонки с одинаковыми характеристиками. Тогда создают элементы-домены с такими характеристиками и на их основе строят колонки таблиц;
- установка значений по умолчанию и по NULL-статусу;
- определение ограничений на целостность данных;
- определение CHECK-ограничений.
- Далее описывается суть каждой из определенных только что позиций.

Спецификация типов данных

Как только вы выбрали атрибут в качестве колонки таблицы, вы должны определить тип данного для атрибута. Тип данного автоматически определя-

ет, какие операции можно совершать с данными этой колонки и какое дисковое пространство может занимать элемент данного. Основные категории типов данных в SQL таковы:

- ❒ символьный тип;
- ❒ целое число;
- ❒ фиксированное десятичное число и десятичное число с плавающей точкой;
- ❒ типы данных даты и времени;
- ❒ типы BLOB-данных (графические данные, данные мультимедиа и т. п.).

> ***Примечание***
>
> В версии Interbase 7 добавлен тип BOOLEAN.

Выбор интернациональных наборов символов

При создании базы данных следует определить набор символов по умолчанию, который задает:

- ❒ какие символы должны использоваться в колонках с типами данных CHAR, VARCHAR и BLOB-text (текстовая информация типа BLOB);
- ❒ в каком порядке по умолчанию данные колонки будут сортироваться (по убыванию или по возрастанию). Элемент COLLATE оператора CREATE TABLE (Создать таблицу) позволяет пользователям указать свой порядок сортировки для колонок с типами данных CHAR и VARCHAR. Надо выбрать порядок сортировки, поддерживаемый заданным набором символов (например, символы английского алфавита упорядочиваются не так, как русского, потому что у них могут быть различные таблицы кодировки). Выбор набора символов по умолчанию — это шаг разработчика базы данных к ее международному использованию. Например, следующий оператор создает базу данных, которая использует набор символов ISO8859_1:

```
CREATE DATABASE 'employee.gdb'
DEFAULT CHARACTER SET iso8859_1;
```

Вы можете переопределить набор символов, заданный по умолчанию, создав другой набор, когда задаете тип данных колонки: спецификация типа данных для CHAR, VARCHAR или BLOB-text может включать утверждение CHARACTER SET для спецификации собственного набора символов для колонки. Если вы не указали свой набор для колонки, то для нее берется набор, принятый по умолчанию для базы данных. Если набор символов по умолчанию для базы данных позже будет изменен, все колонки, определенные после этого изменения, будут иметь новый набор символов, но существовавшие до изменения колонки новое изменение не затронет. Если вы не

задали набор символов по умолчанию во время создания базы данных, то набор будет установлен в значение NONE. Это означает, что не существует набора символов для колонок. Данные в таком случае хранятся и извлекаются в том порядке, в котором они были введены. Можно загрузить любой набор символов в колонку со статусом NONE, но нельзя загрузить те же данные в другую колонку, для которой был определен другой набор символов. Просто не будет выполнена транслитерация (отображение символов одной таблицы в другую) между исходным и новым набором символов.

Указание доменов

Когда некоторые таблицы в базе данных содержат колонки с одинаковыми характеристиками, то по ним можно создать шаблон элемента базы данных (домен) и хранить его в базе данных, а при создании таблиц делать ссылки на такой домен.

Установка значений по умолчанию и NULL-статуса

Когда определяется колонка таблицы, в операторе определения имеется утверждение DEFAULT, в котором задается значение колонки по умолчанию. Фактически это не значение всей колонки, а ее ячейки: при выполнении над таблицей операторов INSERT (вставить) или UPDATE (обновить) действия осуществляются над строками, а в рамках колонки — над ее ячейками. Если значение колонки во вставляемой строке не будет определено, то вместо него вставится то, которое указано в утверждении DEFAULT. Такой подход позволяет избежать многих неприятностей при работе с таблицей. Вот несколько примеров задания значений по умолчанию при определении колонок:

```
stringfld VARCHAR(10) DEFAULT 'abc'
integerfld INTEGER DEFAULT 1
numfld NUMERIC(15,4) DEFAULT 1.5
datefld1 DATE DEFAULT '5/5/2005'
datefld2 DATE DEFAULT 'TODAY'
userfld VARCHAR(12) DEFAULT USER
```

Если назначить по умолчанию NULL, то в колонку будет вставляться значение NULL (аналог значения "пусто" в C++), если пользователь не введет значение. Если назначить по умолчанию NOT NULL, это заставит систему потребовать от пользователя обязательного ввода значения в данное поле или же определить значение по умолчанию для колонки. Значение NOT NULL должно обязательно назначаться для первичного (PRIMARY KEY) и уникального (UNIQUE KEY) ключей-колонок.

Определение ограничений на целостность данных

Ограничения на целостность данных — это правила, управляющие связями колонка-таблица и таблица-таблица и проверкой данных на достоверность. Они охватывают все транзакции, которые имеют доступ к базе данных и автоматически поддерживаются системой. Ограничения на целостность данных могут быть применены как в целом к таблице, так и к отдельной колонке. Мы видели, что ограничения, налагаемые на первичный или уникальный ключ, гарантируют, что любые два значения в колонке или в наборе колонок не будут совпадать.

Значения данных, которые однозначно определяют строки (первичный ключ) в одной таблице, могут появляться в других таблицах. В этой связи вводится понятие *внешнего ключа* (FOREIGN KEY) — колонки или набора колонок таблицы, которые содержат значения, совпадающие с первичным ключом в другой таблице. С помощью утверждений языка SQL ON UPDATE и ON DELETE определяют, что произойдет с соответствующим внешним ключом, если первичный ключ изменится или будет удален. В листинге 1.1 — пример таблицы "Дополнительные данные" dop_dannie из приложения по обработке данных по персоналу предприятия (*см. главу 2*). Отметим, что здесь и в дальнейшем операторы SQL в листингах могут быть написаны в верхнем или нижнем регистрах, что не имеет существенного значения: компилятор в обоих случаях не выдает ошибки.

Листинг 1.1

```
create table dop_dannie
/*при удалении первичного ключа
таблицы osn_dannie автоматически будет удалена строка с таким же ключом
этой таблицы*/
(
       tn float not null,
       adr_e_mail varchar(25),
       alimenti varchar(100),
       primary key (tn),
             foreign key (tn) references osn_dannie (tn)
       on delete cascade
);
```

Задание CHECK-ограничений

Наряду с применением первичных и уникальных ключей, можно использовать и другой тип проверки данных на достоверность. Это применение

утверждения CHECK, которое действует на данное в момент его ввода. CHECK-ограничение запускает проверку условия, которое должно давать значение "истинно" (TRUE), когда происходит вставка в таблицу или колонку или их обновление. Например, при создании домена, который затем станет использоваться при создании колонки "Покупатель" (custno) в некоторой таблице, введен контроль на ввод значений в такую колонку (значения должны быть обязательно больше 1000) оператором:

```
create domain custno
as integer
check (value > 1000);
```

Установление связей между объектами

Связи между объектами (таблицами и колонками) в базе данных должны определяться на этапе проектирования. Например, связаны ли в компании работники и подразделения? Ответ: связаны. Каким образом? Ответ: один работник может входить только в одно подразделение (связь "один-к-одному"), но одно подразделение имеет много работников (связь "один-ко-многим"). Как связаны проекты и работники? Ответ: один работник может работать в одном и более проектах, и один проект может включать в себя некоторое количество работников (связь "многие-ко-многим"). Каждый из этих типов связей должен быть смоделирован в базе данных. Реляционная модель (модель со связями) представляет связи "один-ко-многим" с помощью пары первичный ключ/внешний ключ. Обратимся к рассмотренной уже таблице EMPLOYEE (см. табл. 1.2) и таблице проектов PROJECT (табл. 1.3).

Таблица 1.3. *Таблица PROJECT*

PROJ_ID	TEAM_LEADER	PROJ_NAME	PROJ_DESC	PRODUCT
DGPII	36	Automap	BLOB	data hardware
VBASE	48	Video database	BLOB	data software
HWRII	24	Translator upgrade	BLOB	data software

Проект может включать много работников, поэтому, чтобы избежать дублирования данных, таблица PROJECT должна ссылаться на информацию о работниках с помощью внешнего ключа. TEAM_LEADER — это внешний ключ, ссылающийся на первичный ключ EMP_NO в таблице EMPLOYEE. Теперь модификация табельного номера в таблице EMPLOYEE не пройдет бесследно для таблицы PROJECT, о чем говорится в следующем разделе.

Принудительное обеспечение целостности данных

Обычные соображения, лежащие в основе задания внешнего ключа, — быть уверенным, что станет поддерживаться целостность данных, когда более одной таблицы ссылается на одни и те же данные, так как строки в одной таблице всегда должны соответствовать строкам в ссылающихся на нее других. Например, обрабатывается движение товаров на складе. В вашей базе данных имеются таблицы "Поставщики", "Покупатели", содержащие коды товаров, и таблица "Ценник", которую сегодня модно называть "Прайс-лист" и которая содержит характеристики товаров: цену, единицу измерения и т. п. Все три таблицы связаны между собой кодом товара. Поэтому, например, неаккуратное изменение какой-либо строки в "Ценнике" может разрушить данные в остальных таблицах, а поддержание режима целостности данных обеспечивает их нормальное функционирование. InterBase принудительно обеспечивает ссылочную целостность следующими путями:

- до добавления внешнего ключа уникальный и первичные ключи, на которые ссылается внешний ключ, должны быть уже определены;
- если информация изменена в одном месте, она должна быть изменена во всех остальных местах, в которых она существует. Например, при создании таблицы БД внешний ключ задается такой последовательностью операторов:

```
foreign key (col [, col …]) references other_table (col [, col …])
[on delete {no action|cascade|set default|set null}]
[on update {no action|cascade|set default|set null}]
```

  Здесь `col` — это колонки, составляющие внешний ключ в данной таблице и первичный ключ в ссылочной таблице, `other_table` — это таблица, на которую ссылается данная таблица.

InterBase выполняет изменение информации автоматически, когда вы применяете режим ON UPDATE к утверждению REFERENCES при определении ограничений для таблицы или ее колонок. Вы можете задать:

- чтобы значение внешнего ключа заменялось на новое значение первичного ключа (режим CASCADE);
- чтобы оно установилось на значение колонки по умолчанию (режим SET DEFAULT);
- или в ноль (режим SET NULL).

Если вы выбираете режим NO ACTION в качестве режима ON UPDATE, вы должны вручную удостовериться, что внешний ключ обновлен после обновления первичного ключа. Например, чтобы изменить значение в колонке

EMP_NO (первичный ключ) таблицы EMPLOYEE, следует обязательно изменить колонку TEAM_LEADER (внешний ключ) таблицы PROJECT.

Нормализация таблицы

После того как таблица, ее колонки и ключи определены, надо посмотреть на проектирование как бы в целом и применить в нем правила нормализации таблицы, чтобы обнаружить логические ошибки. Нормализация — это разбиение больших таблиц на мелкие с целью объединить в группы данные, связанные между собой (детально методы нормализации здесь не рассматриваются). Такие связанные данные как раз и объединяются в таблицу. А польза от их объединения в общую таблицу очевидна: их легче обновлять и удалять (все происходит в одной и той же таблице, а не в разных), снижается вероятность дублирования и ввода несовместимых данных. В общем случае процесс нормализации включает в себя:

- удаление повторяющихся групп данных;
- удаление частично зависимых колонок;
- удаление транзитивно зависимых колонок.

Удаление повторяющихся групп данных

Если для какого-то значения первичного ключа существуют колонки, в которых имеется более одного значения, то такие значения образуют повторяющиеся группы. Посмотрим на таблицу DEPARTMENT в первом варианте (табл. 1.4). Ее первая строка, в которой DEPT_NO = 100, содержит повторяющуюся группу в колонке DEPT_LOCATIONS (Monterey, Santa Cruz, Salinas).

В таблице DEPARTMENT во втором варианте (табл. 1.5) видно, что для каждого значения первичного ключа, равного 100, все колонки, кроме колонки DEPT_LOCATION, содержат повторяющуюся информацию, т. е. имеют место повторяющиеся группы.

Таблица 1.4. *Первый вариант таблицы DEPARTMENT*

DEPT_NO	DEPARTMENT	HEAD_DEPT	BUDGET	DEPT_LOCATIONS
100	Sales	000	1000000	Monterey, Santa Cruz, Salinas
600	Engineering	120	1100000	San Francisco
900	Finance	000	400000	Monterey

Таблица 1.5. *Второй вариант таблицы DEPARTMENT*

DEPT_NO	DEPARTMENT	HEAD_DEPT	BUDGET	DEPT_LOCATION
100	Sales	000	1000000	Monterey
100	Sales	000	1000000	Santa Cruz
600	Engineering	120	1100000	San Francisco
100	Sales	000	1000000	Salinas

Чтобы нормализовать такую таблицу, мы должны удалить атрибут DEPT_LOCATION из таблицы и создать другую таблицу под именем DEPT_LOCATIONS (табл. 1.6). А затем мы можем создать первичный ключ из комбинации атрибутов DEPT_NO и DEPT_LOCATION.

Таблица 1.6. *Таблица DEPT_LOCATIONS*

DEPT_NO	DEPT_LOCATION
100	Monterey
100	Santa Cruz
600	San Francisco
100	Salinas

Теперь уже для каждого места размещения подразделения DEPT_LOCATION имеется отдельная строка, и повторяющиеся группы исчезли. У таблицы DEPARTMENT во втором варианте поля DEPT_NO и DEPT_LOCATION будут представлять собой внешний ключ, ссылающийся на первичный ключ таблицы DEPT_LOCATIONS. Любые изменения в таблице DEPT_LOCATIONS отразятся в таблице DEPARTMENT второго варианта.

Удаление частично зависимых колонок

Другим важным шагом в процессе нормализации является удаление любой неключевой колонки, которая зависит от некоторой части ключа. Такие колонки называют частично зависимыми. Неключевые колонки представляют информацию об объекте, но не определяют его однозначно. Предположим, например, что вам требуется отыскать некоего работника с помощью таблицы PROJECT (табл. 1.7) с составным первичным ключом по колонкам EMP_NO и PROJ_ID.

Таблица 1.7. *Таблица PROJECT (вначале)*

EMP_NO	PROJ_ID	LAST_NAME	PROJ_NAME	PROJ_DESC	PRODUCT
44	DGPII	Smith	Automap	BLOB data	hardware
47	VBASE	Jenner	Video	data- base BLOB data	software
24	HWRII	Stevens	Translator	up- grade BLOB data	software

Сразу возникнет проблема, так как поля PROJ_NAME, PROJ_DESC и PRODUCT — это атрибуты идентификатора проекта (PROJ_ID), но не табельного номера работника (EMP_NO), который таким образом частично зависит от первичного ключа EMP_NO + PROJ_ID. Чтобы нормализовать эту таблицу, надо будет удалить колонку LAST_NAME из таблицы PROJECT и создать другую таблицу под именем EMPLOYEE_PROJECT, которая в качестве первичного ключа имеет комбинацию EMP_NO + PROJ_ID (табл. 1.8).

Таблица 1.8. *Таблица EMPLOYEE_PROJECT*

EMP_NO	PROJ_ID	LAST_NAME
44	DGPII	Smith
47	VBASE	Jenner
24	HWRII	Stevens

А таблица PROJECT примет вид (табл. 1.9):

Таблица 1.9. *Таблица PROJECT (после преобразования)*

EMP_NO	PROJ_ID	PROJ_NAME	PROJ_DESC	PRODUCT
44	DGPII	Automap	BLOB data	hardware
47	VBASE	Video	database BLOB data	software
24	HWRII	Translator	upgrade BLOB data	software

Теперь для каждого работника в проекте будет существовать единственная строка, с ним связанная. Внешним ключом в таблице PROJECT надо объявить EMP_NO + PROJ_ID. Тогда при изменении данных в таблице EMPLOYEE_PROJECT автоматически изменятся данные в таблице PROJECT.

Удаление транзитивно зависимых колонок

Следующим шагом по нормализации таблицы является удаление любой неключевой колонки, которая зависит от другой неключевой колонки. Каждая такая неключевая колонка должна быть характеристикой ключевой колонки. Например, мы хотим добавить в таблицу PROJECT колонки TEAM_LEADER_ID и PHONE_EXT и сделать колонку PROJ_ID первичным ключом (табл. 1.10). Тогда PHONE_EXT становится неключевой колонкой, являясь характеристикой колонки TEAM_LEADER_ID (это ведь телефон, связанный с руководителем).

Таблица 1.10. *Таблица PROJECT ненормализованная*

PROJ_ID	TEAM_LEADER_ID	PHONE_EXT	PROJ_NAME	PROJ_DESC	PRODUCT
DGPII	48	4929	Automap	BLOB data	hardware
VBASE	36	4967	Video	database BLOB data	software
HWRII	24	4668	Translator	upgrade BLOB data	software

Чтобы нормализовать эту таблицу (табл. 1.11), мы должны удалить колонку PHONE_EXT, заменить TEAM_LEADER_ID на TEAM_LEADER и сделать TEAM_LEADER внешним ключом, ссылающимся на первичный ключ EMP_NO в таблице EMPLOYEE (табл. 1.12).

Таблица 1.11. *Таблица PROJECT нормализованная*

PROJ_ID	TEAM_LEADER	PROJ_NAME	PROJ_DESC	PRODUCT
DGPII	48	Automap	BLOB data	hardware
VBASE	36	Video	database BLOB data	software
HWRII	24	Translator	upgrade BLOB data	software

Таблица 1.12. *Таблица EMPLOYEE для ссылки по внешнему ключу*

EMP_NO	LAST_NAME	FIRST_NAME	DEPT_NO	JOB_CODE	PHONE_EXT	SALARY
24	Smith	John	100	Eng	4968	64000
48	Carter	Catherine	900	Sales	4967	72500
36	Smith	Jane	600	Admin	4800	37500

Выбор индексов

Индекс (указатель) — это механизм, используемый для ускорения извлечения записей таблицы в ответ на некоторый запрос при некоторых условиях поиска. Это как поиск по указателю в книге: индекс (указатель) содержит номера страниц, связанных с данным термином, что позволяет быстро найти нужную страницу. Индекс базы данных служит логическим указателем на физическое размещение (адрес) строки в таблице. Индекс хранит каждое значение проиндексированной колонки (колонки, по значениям которой предполагается вести поиск в таблице) или колонок с указателями на все дисковые блоки, содержащие строки с таким значением. Когда выполняется запрос, среда InterBase сначала проверяет, существует ли какой-либо индекс для данной таблицы. Затем определяется, будет ли более эффективным просмотреть всю таблицу или использовать существующий индекс, чтобы выполнить запрос. Если среда обработки решит, что надо использовать индекс, она ищет индекс, чтобы найти требуемые ключевые значения, по которым выбирает указатели для поиска строки таблицы, содержащей эти значения. Извлечение данных является быстрым, потому что значения индекса упорядочены и сами они относительно невелики. Это позволяет системе быстро отыскивать ключевые значения. Как только в индексе ключевое значение найдено, система выбирает связанный с этим значением указатель на физическое размещение данных. Использование индексов обычно требует меньшего количества страниц выборки, чем последовательный просмотр каждой строки таблицы. Индекс может быть определен на одной или нескольких колонках таблицы.

Стоит ли проводить индексацию и когда

Время последовательного просмотра строк таблицы для поиска нужной строки пропорционально количеству строк. Индексирование таблицы хотя бы по одной колонке значительно сокращает время поиска. Но есть и проблемы. Во-первых, главная помеха здесь в том, что индексы (а это отдельные файлы) потребляют дополнительное дисковое пространство. Во-вторых, вставка, удаление и обновление данных происходят дольше на индексированных, чем на неиндексированных колонках. Дело в том, что индекс должен каждый раз обновляться, когда изменяется данное в колонке, являющейся индексом, или когда строка таблицы меняет свои координаты: добавляется новая строка или удаляется существующая. Поэтому надо быть аккуратным, задавая сложные составные индексы. Однако преимущества индексов перевешивают их недостатки в процессе извлечения данных.

Создать индексированную колонку в таблице можно, когда:

- условия поиска часто используют эту колонку;
- JOIN-условия часто используют эту колонку (с помощью утверждения JOIN можно извлекать данные из двух и более таблиц при одном операторе SELECT);
- ORDER BY-утверждения оператора SELECT часто используют эту колонку для сортировки данных.

Нельзя создавать индексы для:

- колонок, на которые редко идет ссылка в условиях поиска;
- для часто обновляемых неключевых колонок;
- для колонок, имеющих незначительное количество возможных значений.

Создание индексов

Индексы создаются либо с использованием оператора CREATE INDEX, либо автоматически системой как часть оператора CREATE TABLE. InterBase допускает создание до 64 индексов в таблице. Чтобы создать индекс, надо иметь авторизованный доступ к таблице.

> ***Примечание***
>
> Чтобы увидеть все индексы, определенные в текущей базе данных, надо выполнить isql-команду SHOW INDEX. Чтобы увидеть все индексы, определенные в таблице, надо выполнить команду SHOW INDEX имя_таблицы. Чтобы увидеть информацию о конкретном индексе, надо выполнить команду SHOW INDEX имя_индекса. Но для выполнения всех перечисленных действий можно воспользоваться и утилитой IBConsole (об этом — далее).

InterBase автоматически генерирует индексы, когда таблицы определяются с использованием утверждений PRIMARY KEY, FOREIGN KEY и UNIQUE. Индексы по PRIMARY KEY и FOREIGN KEY сохраняют целостность информации.

Использование CREATE INDEX

Этот оператор создает индекс по одной и более колонкам таблицы. Индексирование одной колонки обеспечивает поиск по данным этой колонки, в то время как индексирование по совокупности колонок обеспечивает комплексный поиск по конъюнкции данных проиндексированных колонок.

Задаваемые режимы:

- порядок сортировки для индекса;
- разрешаются ли в колонке дублированные значения (могут ли встречаться в колонке одинаковые значения). Последнее требование крайне важно при создании приложений. Например, вы хотите, чтобы таблица, отражающая социальные льготы работников, была упорядочена по табельным номерам. Так как у одного работника может быть более одной льготы, то в такой таблице появятся строки с одинаковыми табельными номерами. Без возможности применения отмеченного выше режима такого положения добиться не удастся.

Для более быстрого ответа на запросы, которые требуют отсортированных данных, надо использовать порядок индексов, совпадающий с утверждением запроса ORDER BY оператора SELECT. Надо использовать индекс и для колонок, которые указываются в утверждении WHERE того же оператора, чтобы ускорить поиск. Для улучшения работы индекса следует использовать утверждение SET STATISTICS, чтобы пересчитать избирательность индекса или перестроить индекс за счет его деактивизации, а затем активизировать его последовательным вызовом оператора ALTER INDEX.

Синтаксис оператора CREATE INDEX таков:

```
CREATE [UNIQUE] [ASC[ENDING] | DESC[ENDING]]
INDEX index ON table (col [, col ...]);
```

Здесь:

- `index` — имя создаваемого индекса, которое должно быть уникальным в БД, в которой он создается;
- `table` — имя таблицы, для которой создается данный индекс;
- `col [, col ...])` — имена полей таблицы, по которым организуется данный индекс.

Например, для таблицы `newtable` (*см. главу 4*) создан индекс `indnewtable`,

```
/*  index definitions for indnewtable */
CREATE INDEX indnewtable ON newtable(iz);
```

который позволяет сортировать таблицу по полю `iz`.

Предотвращение появления дублированных значений

Когда задано утверждение UNIQUE, то при вводе данных в столбец с таким атрибутом не окажется ни одной строки таблицы, у которой бы по индексу совпало какое-то значение с другой строкой. Система проверяет дублирование значений сразу при создании индекса, и каждый раз при вставке или обновлении строки. InterBase автоматически создает UNIQUE-индекс для

колонки, определенной как PRIMARY KEY. UNIQUE-индексы имеют смысл только тогда, когда единственность, уникальность является самой характеристикой данного. Например, нет необходимости задавать уникальным индекс по колонке LAST_NAME (фамилия), так как могут встречаться одинаковые фамилии у разных людей. Но когда колонка содержит, например, налоговый код человека, здесь наличие уникальности обязательно.

Чтобы определить индекс, который не позволяет появление дублированных значений, надо включать утверждение UNIQUE в оператор CREATE INDEX.

Задание порядка сортировки индекса

Задание порядка сортировки индекса определяется утверждениями ASCENDING (от меньшего к большему — по возрастанию значения индекса) и DESCENDING (от большего значения к меньшему — по убыванию значения индекса). По умолчанию принято утверждение ASCENDING. Например:

```
create descending index desc_x on salary_history (change_date);
```

Чтобы извлечь индексированные данные из такой таблицы в убывающем порядке, надо использовать утверждение

```
order by change_date descending
```

в операторе SELECT. Если требуется использовать оба направления сортировки (по возрастанию и убыванию) на какой-то колонке, следует на этой колонке определить два индекса: один — по возрастанию, другой — по убыванию. Следующий пример это иллюстрирует:

```
create ascending index ascend_x on salary_history (change_date);
create descending index desc_x on salary_history (change_date);
```

Когда используют многоколоночный индекс

Основным поводом для использования многоколоночного индекса является необходимость ускорения исполнения запросов, которые часто обращаются к одному и тому же множеству колонок. Нет нужды создавать запрос с точным списком колонок, который определен в индексе, в утверждении ORDER BY оператора SELECT, так как InterBase станет использовать подмножество компонентов многоколоночного индекса, чтобы оптимизировать запрос, но при условии, если:

- подмножество колонок, использованное в утверждении ORDER BY, начинается с первой колонки в многоколоночном индексе. Например, если список индексированных колонок содержит колонки A1, A2, A3, то запрос, в котором используются колонки A1, A2, будет оптимизирован при использовании индекса, а запрос, использующий индексы A2, A3, — нет;

- порядок, в котором запрос имеет доступ к колонкам в утверждении ORDER BY, совпадает с порядком колонок, определенным в индексе (запрос не стал бы оптимизироваться, если бы его колонки были в списке в порядке A2, A1).

Улучшение работы индексов

Индексы могут разбалансироваться после многократных изменений в базе данных. Когда это происходит, работа с БД может быть улучшена следующими методами:

- надо перестроить индекс, используя оператор ALTER INDEX;
- надо пересчитать избирательность индекса, используя оператор SET STATISTICS;
- надо удалить и пересоздать индекс операторами DROP INDEX и CREATE INDEX;
- надо скопировать и восстановить базу данных утилитой Interbase gbak.

Использование ALTER INDEX

Оператор ALTER INDEX деактивирует и реактивирует индекс. Эти операции полезны при разбалансировании индекса. Чтобы перестроить индекс, надо сначала использовать ALTER INDEX INACTIVE, чтобы деактивировать индекс, а затем — ALTER INDEX ACTIVE, чтобы реактивировать индекс. Этот метод воссоздает сбалансированный индекс. Можно перестроить индекс копированием и восстановлением базы данных с помощью утилиты gbak, которая хранит только определения индекса, а не структуру данных и поэтому, когда вы восстанавливаете базу данных, gbak перестраивает индексы.

> ***Примечание***
>
> Перед вставкой в базу данных большого количества строк деактивируйте (отключите) индексы таблицы, а затем после вставки реактивируйте (подключите) их, чтобы их перестроить. В противном случае InterBase станет последовательно обновлять индекс каждый раз, когда вставляется одна строка, что займет много времени.

Синтаксис ALTER INDEX таков:

```
alter index имя индекса {active | inactive};
```

Например, следующие операторы деактивируют и реактивируют индекс, чтобы его перестроить:

```
alter index budgetx inactive;
alter index budgetx active;
```

При изменении индекса существуют ограничения:

- операцию изменения индекса может выполнять только его создатель или пользователь с именем SYSDBA, или пользователь с привилегиями по операционной системе;
- изменяться может только тот индекс, который в этот момент не используется в базе данных;
- нельзя изменять индекс, определенный как UNIQUE, PRIMARY KEY или FOREIGN KEY. Если все же такие индексы надо перестроить, следует использовать оператор ALTER TABLE (т. е. работать на уровне таблицы);
- нельзя использовать ALTER INDEX для добавления или удаления индексированных колонок или ключей. Для этого надо использовать DROP INDEX, а за ним CREATE INDEX.

Использование оператора SET STATISTICS

Для таблиц, в которых количество дублированных значений радикально возрастает или убывает, периодический перерасчет индексов может улучшить их работу. Оператор SET STATISTICS пересчитывает избирательность индекса. Избирательность индекса — это расчет, выполняемый оптимизатором InterBase на базе количества различных строк в таблице на этапе доступа к таблице. Таблица сохраняется в памяти, где и происходит работа с ней оптимизатора — создается оптимальный план извлечения данных для конкретного запроса.

Синтаксис оператора SET STATISTICS таков:

```
set statistics index имя индекса;
```

Следующий оператор перерасчитывает избирательность для индекса:

```
set statistics index minsalx;
```

При применении SET STATISTICS существуют ограничения:

- при использовании оператора вы должны иметь права создателя индекса или быть SYSDBA (администратором баз данных);
- SET STATISTICS не перестраивает индекс. Чтобы его перестроить, надо применить оператор ALTER INDEX.

Использование оператора DROP INDEX

DROP INDEX удаляет определенный пользователем индекс из базы данных. Системно определенные индексы, определенные с помощью операторов UNIQUE, PRIMARY KEY и FOREIGN KEY, не могут быть удалены. Чтобы модифицировать индекс, сначала его надо удалить оператором DROP

INDEX, а затем создать заново оператором CREATE INDEX, используя то же имя, но уже с требуемыми характеристиками.

Синтаксис DROP INDEX таков:

```
drop index имя индекса;
```

Следующий оператор удаляет индекс:

```
drop index minsalx;
```

Существуют следующие ограничения на удаление индекса:

- при использовании оператора вы должны иметь права создателя индекса или быть SYSDBA (администратором баз данных);
- используемый индекс не может быть удален;
- нельзя удалить индекс, определенный как UNIQUE, PRIMARY KEY или FOREIGN KEY. Если все же надо удалять такие индексы, следует использовать оператор ALTER TABLE (т. е. работать на уровне таблицы).

Выбор индексов

В момент разработки таблицы следует определить, какие индексы необходимы. Основным мотивом при этом является тот, что при большем числе различных индексов извлечение необходимых данных можно сделать более быстрым, но обновление и хранение — более медленным. Необходимо избегать создания разных индексов на базе одной и той же колонки. Например, если вы извлекаете записи работников на основе имени или табельного номера, надо определить свой индекс для каждой из этих колонок.

Когда вы тестируете свой проект с целью оптимизации выбора индексов, запомните, что размеры таблиц значительно воздействуют на эффект извлечения записей. Если ожидается таблица размером от 10 до 100 тыс. записей, то нельзя тестировать проект на таблице от 10 до 100 записей, так как будут получены неверные временные данные. Другой фактор — это размер страницы: когда открывается база данных (оператором CONNECT), последний может устанавливать количество кэш-буферов, предназначенных для оперативной обработки данных базы данных этого соединения. Когда программа устанавливает соединение с базой данных, InterBase выделяет системную память для использования ее в качестве частного буфера. Буферы используются для хранения доступных страниц базы данных, чтобы увеличить скорость доступа к элементам БД. Количество буферов, назначенных для программы, определяет, к скольким страницам одновременно может идти доступ в пуле памяти. Буферы остаются назначенными до тех пор, пока программа не закончит работу с базой данных. Увеличивая размер страницы, вы можете разместить больше записей на одной странице и, соответственно, снизите количество страниц, используемых индексами. Если один из ваших индексов имеет более четырех уровней глубины (состоит более, чем

из четырех колонок), вы должны значительно увеличить размер страницы. Если индексы созданы на слишком часто изменяемых данных (которые регулярно удаляются и восстанавливаются) и имеют менее трех уровней глубины, вы должны значительно снизить размер страницы базы данных. В общем случае вы должны выбирать размер страницы больше, чем ваша самая большая запись, хотя InterBase сократит размеры записей, которые содержат строковые данные или числовые данные со значениями 0 или NULL. Если ваши записи обладают такими характеристиками, можно ожидать, что удастся хранить записи на страницах, которые на 20% меньше полного размера записи. С другой стороны, если ваши записи несжимаемы, вы должны добавить 5% к размеру записи.

Увеличение размера кэша

Когда InterBase читает страницу базы данных с диска, он сохраняет ее в своем кэше, являющемся набором буферов, зарезервированных для управления страницами. Обычно по умолчанию размер кэша равен 2048 буферам. Если ваше приложение включает объединение пяти или более таблиц, InterBase автоматически увеличивает размер кэша. Для ручного увеличения размера кэша используется утилита gfix:

```
gfix -buffers n database_name
```

где `n` — новое количество буферов, а `database_name` — имя базы данных, для которой изменяется размер кэша.

> ***Примечание***
>
> Если вы чувствуете, что производительность вашего программного приложения ограничена размерами диска, вы можете создать многофайловую базу данных и распределить ее между различными дисками.

Описание таблиц на языке SQL

Таблицы в существующей базе данных можно создавать с помощью оператора CREATE TABLE, используя утилиту isql или IBConsole.

Синтаксис CREATE TABLE таков:

```
CREATE TABLE table [EXTERNAL [FILE] 'filespec']
(<col_def> [, <col_def> | <tconstraint> …]);
<col_def> = col {<datatype> | COMPUTED [BY] (<expr>) | domain}
[DEFAULT {literal | NULL | USER}]
[NOT NULL]
[<col_constraint>]
```

```
[COLLATE collation]
<datatype> =
{SMALLINT | INTEGER | FLOAT | DOUBLE PRECISION}[<array_dim>]
| (DATE | TIME | TIMESTAMP}[<array_dim>]
| {DECIMAL | NUMERIC} [(precision [, scale])] [<array_dim>]
| {CHAR | CHARACTER | CHARACTER VARYING | VARCHAR} [(int)]
[<array_dim>] [CHARACTER SET charname]
| {NCHAR | NATIONAL CHARACTER | NATIONAL CHAR}
[VARYING] [(int)] [<array_dim>]
| BLOB [SUB_TYPE {int | subtype_name}] [SEGMENT SIZE int]
[CHARACTER SET charname]
| BLOB [(seglen [, subtype])]
<array_dim> = [[x:]y [, [x:]y …]]
<expr> = SQL-выражение, результатом которого является единственное
значение.
<col_constraint> = [CONSTRAINT constraint]
{ UNIQUE
| PRIMARY KEY
| REFERENCES other_table [(other_col [, other_col …])]
[ON DELETE {NO ACTION|CASCADE|SET DEFAULT|SET NULL}]
[ON UPDATE {NO ACTION|CASCADE|SET DEFAULT|SET NULL}]
| CHECK (<search_condition>)}
<tconstraint> = [CONSTRAINT constraint]
{{PRIMARY KEY | UNIQUE} (col [, col …])
| FOREIGN KEY (col [, col …])
REFERENCES other_table [(other_col [, other_col …])]
[ON DELETE {NO ACTION|CASCADE|SET DEFAULT|SET NULL}]
 [ON UPDATE {NO ACTION|CASCADE|SET DEFAULT|SET NULL}]
| CHECK (<search_condition>)}
<search_condition> = <val> <operator> {<val> | <select_one>
| <val> [NOT] BETWEEN <val> AND <val>
| <val> [NOT] LIKE <val> [ESCAPE <val>]
| <val> [NOT] IN (<val> [, <val> …] | <select_list>)
| <val> IS [NOT] NULL
| <val> {>= | <=}
| <val> [NOT] {= | < | >}
```

```
| {ALL | SOME | ANY} (<select_list>)
| EXISTS (<select_expr>)
| SINGULAR (<select_expr>)
| <val> [NOT] CONTAINING <val>
| <val> [NOT] STARTING [WITH] <val>
| (<search_condition>)
| NOT <search_condition>
| <search_condition> OR <search_condition>
| <search_condition> AND <search_condition>
<val> = { col [<array_dim>] | :variable
| <constant> | <expr> | <function>
| udf ([<val> [, <val> …]])
| NULL | USER | RDB$DB_KEY | ? }
[COLLATE collation]
<constant> = num | 'string' | charsetname 'string'
<function> = COUNT (* | [ALL] <val> | DISTINCT <val>)
| SUM ([ALL] <val> | DISTINCT <val>)
| AVG ([ALL] <val> | DISTINCT <val>)
| MAX ([ALL] <val> | DISTINCT <val>)
| MIN ([ALL] <val> | DISTINCT <val>)
| CAST (<val> AS <datatype>)
| UPPER (<val>)
| GEN_ID (generator, <val>)
<operator> = {= | < | > | <= | >= | !< | !> | <> | !=}
<select_one> = SELECT над одним полем; возвращает точно одно значение.
<select_list> = SELECT над одним полем; возвращает ноль или много значений.
<select_expr> = SELECT над списком значений; возвращает ноль или много
значений.
```

Суть многих аргументов описания разъяснялась ранее при рассмотрении вопроса создания таблиц. Строгое же описание аргументов приводится в табл. 1.13.

Таблица 1.13. *Описание аргументов*

Аргумент	Описание
table	Имя таблицы. Должно быть уникальным среди имен таблиц и процедур в БД

Таблица 1.13 *(продолжение)*

Аргумент	Описание
EXTERNAL [FILE] 'filespec'	Объявляет, что данные для создания таблицы находятся во внешнем (по отношению к БД) файле. filespec — это полная срецификация файла (с путем)
col	Имя колонки таблицы. Должно быть уникальным среди имен столбцов таблицы
datatype	Тип данных колонки. Типы берутся те, что определены в SQL
COMPUTED [BY] (expr)	Выражение указывает, что значение данного колонки рассчитывается выражением expr в момент исполнения приложения и таким образом не занимает память в БД. expr — это арифметическое выражение, действительное для типов данных, примененных в нем. Любые колонки, участвующие в выражении, должны быть определены до их использования в выражении. expr не работает с колонками типа BLOB. expr должно возвращать единственное значение и не может возвращать массив
domain	имя существующего домена
DEFAULT	ЗАДАЕТ значение по умолчанию в колонке, которое станет ей присваиваться, если никакое другое значение не будет в нее введено. Возможными значениями могут быть: • литерал — строка символов, числовое значение или дата; • NULL — вводится нулевое значение; • USER — вводится имя текущего пользователя. Колонка должна быть совместима по типу вводимых в нее по умолчанию данных. Наборы по умолчанию для колонок перекрывают наборы по умолчанию для доменов
CONSTRAINT constraint	Имя таблицы или колонки-ограничителя. Должно быть уникальным внутри таблицы
constraint_def	Задает род колонки-ограничителя. Действительными значениями будут UNIQUE, PRIMARY KEY, CHECK и REFERENCES
REFERENCES	Задает положение, в соответствии с которым значения колонки происходят от значений колонки другой таблицы. Если не указать имени колонки, InterBase рассматривает в качестве имени имя колонки той таблицы, из которой происходит ссылка
ON DELETE | ON UPDATE	Выражение используется с утверждением REFERENCES: изменяет внешний ключ всякий раз, когда меняется первичный ключ, на который ссылается внешний ключ.

Таблица 1.13 *(окончание)*

Аргумент	Описание
	Действительными значениями являются: • [Default] NO ACTION — не меняется внешний ключ. Может послужить причиной при изменении первичного ключа выдачи ошибки из-за нарушения целостности данных; • CASCADE — для выражения ON DELETE удаляет соответствующий внешний ключ. Для ON UPDATE заменяет соответствующий внешний ключ up на новое значение первичного ключа; • SET NULL — устанавливает все колонки соответствующего внешнего ключа в NULL; • SET DEFAULT — каждая колонка соответствующего внешнего ключа устанавливается в свое значение по умолчанию. Если значение по умолчанию для внешнего ключа не обнаруживается в первичном ключе, то попытка удаления или обновления первичного ключа оканчивается неудачей
CHECK search_condition	Обеспечивает контроль ввода по заданному условию search_condition. Если введенное данное не соответствует условию, попытка ввода заканчивается неудачей
COLLATE collation	Устанавливает сортировку для колонки. Типами данных в колонке могут быть CHAR, VARCHAR и BLOB text

Перечислим важные замечания к оператору CREATE TABLE.

- При объявлении массивов используются квадратные скобки для описания размерностей. Например, следующий оператор создает двумерный массив размером 5×5 из символьных переменных длиной в 6 символов:

```
my_array varchar(6)[5,5];
```

> ***Примечание***
>
> В приложениях, написанных на C и C++ и на языке isql, в конце оператора должна присутствовать точка с запятой.

- Следует использовать двоеточие для задания массива с начальным элементом, номер которого отличен от 1. Следующий пример создает массив целых чисел, нумерация элементов которого начинается с 10 и кончается 20:

```
my_array integer[10:20];
```

- В isql val не может быть переменной.
- Нельзя задавать элемент COLLATE для колонок типа BLOB (графические изображения нельзя сортировать).

Типы данных, используемые в SQL

Типы данных, используемые в SQL, приведены в табл. 1.14.

Таблица 1.14. *Описание типов данных*

Тип	Размер памяти	Описание
BLOB	До 64 Кбайт	Здесь размещаются bitmap-изображения, векторная графика, звуковые файлы, видеофайлы, другие документы мультимедиа
CHAR	До 32 Кбайт	Данное с фиксированным количеством символов
VARCHAR	До 32 Кбайт	Данное с переменным количеством символов
SHORT	2 байта	Целое число типа Integer
LONG	4 байта	Целое число типа Integer
FLOAT	4 байта	Число с плавающей точкой с точностью до 7 знаков
DOUBLE	8 байтов	Число с плавающей точкой с точностью до 15 знаков
DECIMAL (precision, scale)	2, 4, 8 байтов	Десятичное число, в котором precision — количество цифр всего в числе (от 1 до 18), а scale — количество цифр в дробной части (от 0 до 18) и должно не превосходить precision. Например, DECIMAL(10, 3) — это число точно формата ppppppp.sss
TIME	4 байта	Время дня — беззнаковое целое с шагом в 0,0001 секунды. Начало отсчета — полночь
DATE	4 байта	Календарная дата
TIMESTAMP	8 байтов	Дата и время дня
ARRAY	Поэлементно	Массив данных различных типов

Обратим внимание на тип данных ARRAY. InterBase позволяет создавать массивы данных различных типов. Использование массива дает возможность собирать элементы данного в одну колонку. InterBase может выполнять операции над целым массивом, работая с ним как с отдельным элементом, но может работать и с подмножеством элементов массива.

Использовать массив выгодно, когда:

- элементы данного образуют набор данных одного типа;

- ❒ весь набор данных в одной колонке базы данных должен быть представлен и обрабатываем как одно целое;
- ❒ каждый элемент должен быть идентифицирован и доступ к нему должен быть индивидуален.

Элементы данного в массиве называются элементами массива. Массив может содержать элементы любого типа InterBase-данных, кроме типа BLOB. Кроме того, не допускается, чтобы был массив массивов. Все элементы одного массива должны быть одного типа (этим, собственно, и определяется массив как структура). Массивы определяются операторами SQL CREATE DOMAIN или CREATE TABLE. Например, вот как определяется обычная строка символов и одномерный символьный массив из четырех элементов по десять символов в каждом:

```
exec sql
create table table1
(name char(10), name_arr char(10)[4]);
```

Размер массива должен быть указан в квадратных скобках и следовать за описанием типа данного.

Существуют и многомерные массивы. InterBase поддерживает многомерные массивы (размерности от 1 до 16). Например, следующий оператор определяет три целочисленных многомерных массива:

```
exec sql
create table table1
(int_arr2 integer[4,5],
int_arr3 integer[4,5,6],
int_arr4 integer[4,5,6,7]);
```

В этом примере INT_ARR2 состоит из четырех строк по 5 элементов в каждой строке (всего 20 элементов), INT_ARR3 имеет три измерения и содержит уже 120 элементов, а INT_ARR4 — 840 элементов.

Важно отметить, что InterBase располагает многомерные массивы в порядке возрастания строк. Этого требуют некоторые языки программирования (например, Fortran).

> ***Примечание***
>
> В приведенных примерах точка с запятой (;) имеет командный смысл: не разделитель операторов, как в некоторых языках программирования, а запуск на выполнение цепочки команд. Набираешь строками операторы, и последним — точку с запятой.

Преобразование типов данных

При выполнении арифметических операций следует использовать совместимые типы данных. Если же требуется выполнять операции над смешанными типами данных или рабочий язык программирования использует тип данного, не поддерживаемый InterBase, тогда следует выполнить предварительное преобразование типов данных. InterBase либо автоматически (диалект 1) преобразует данное к эквивалентному типу (неявное преобразование типов), либо требуется самому явно с использованием функции CAST() (диалект 3) выполнить преобразование типов. Однако следует помнить, что нельзя преобразовать BLOB- или Array-данные в любой другой тип данных, так же как любой тип данных нельзя преобразовать в тип BLOB или Array.

Неявные преобразования

InterBase поддерживает некоторые типы неявных преобразований типов данных. Например, тип TIMESTAMP неявно всегда будет переведен в целую дату, а смешанное выражение из чисел типа integer, numeric или float неявно преобразует число типа integer в выражении в подходящий тип. Однако диалект 3 InterBase отличается от диалекта 1 в следующем: в диалекте 3 неявно преобразование string-to-integer не поддерживается. Например, выражение

```
3 + '1' = 4
```

InterBase диалекта 1 автоматически преобразует символ "1" в integer, в то время как на этом выражении InterBase диалекта 3 выдаст ошибку. Здесь потребуется явное преобразование типов:

```
3 + CAST('1' AS INT)
```

В следующем примере ошибка выдастся в любом случае:

```
3 + 'a' = 4
```

так как InterBase не может преобразовать "a" в integer.

Явные преобразования

Когда InterBase не в состоянии выполнить неявное преобразование типов данных в выражениях, требуется выполнить это явным образом с использованием функции CAST(). Эту функцию применяют внутри оператора SELECT. Обычно CAST() используется в элементе WHERE при сравнении данных различных типов. Синтаксис при этом таков:

```
CAST (value | NULL AS datatype)
```

Например, только что мы видели преобразование символьной единицы в ее числовой формат:

```
CAST('1' AS INT)
```

CAST() используется для преобразования следующих типов данных:

- DATE, TIME, TIMESTAMP в CHARACTER;
- CHARACTER в DATE, TIME, TIMESTAMP;
- TIMESTAMP в TIME,DATE;
- TIME, DATE в TIMESTAMP.

Например,

```
... where hire_date = (cast(interview_date as date);
```

Здесь данное INTERVIEW_DATE имеет тип CHAR, а HIRE_DATE имеет тип DATE.

А в следующем примере происходит обратное преобразование:

```
... where cast(hire_date as char) = interview_date;
```

Преобразование числовых типов данных в символьные требует минимальной длины для данных символьного типа, как это указано в табл. 1.15.

Таблица 1.15. *Минимальная длина данного для преобразуемого символьного типа*

Тип данного	Минимальная длина для преобразуемого символьного типа
Decimal	20 (преобразуемое десятичное число отобразится как минимум в 20 символов)
Double	22
Float	13
Integer	11
Numeric	22
Smallint	6

Определение ограничений для контроля данных

При рассмотрении вопроса создания таблиц БД мы немного касались вопроса контроля ввода данных с помощью задания определенных ограничений путем введения в описание колонки таблицы утверждения CHECK. Ар-

гументом этого утверждения является условие поиска (search condition). Использование утверждения CHECK заставляет проверяться условие перед тем, как данное будет введено или обновлено. Условие поиска проверяет, попадает ли введенное значение внутрь некоторого возможного диапазона или совпадает с одним из значений из списка значений. Условие поиска может также сравнивать вводимое значение со значением из других колонок.

CHECK-контроль гарантирует целостность данных только в тех случаях, когда сравниваемые величины находятся в одной строке, которая либо вставляется, либо удаляется. Синтаксис создания CHECK-ограничений такой:

```
CHECK (<search condition>);
<search_condition> =
<val> <operator> {<val> | (<select_one>)}
| <val> [NOT] BETWEEN <val> AND <val>
| <val> [NOT] LIKE <val> [ESCAPE <val>]
| <val> [NOT] IN (<val> [, <val> ...] | <select_list>)
| <val> IS [NOT] NULL
| <val> {[NOT] {= | < | >} | >= | <=}
{ALL | SOME | ANY} (<select_list>)
| EXISTS (<select_expr>)
| SINGULAR (<select_expr>)
| <val> [NOT] CONTAINING <val>
| <val> [NOT] STARTING [WITH] <val>
| (<search_condition>)
| NOT <search_condition>
| <search_condition> OR <search_condition>
| <search_condition> AND <search_condition>
```

Здесь:

- `<val>` — вводимое и сравниваемое значение;
- `<operator>` — {= | < | > | <= | >= | !< | !> | <> | !=};
- `<select_one>` — оператор SELECT, выбирающий значение из одной колонки и возвращающий точно одно значение (оно и станет участвовать в операции контроля);
- `<select_list>` — оператор SELECT, выбирающий значение из одной колонки и возвращающий множество ее значений (употребляется при контроле на входимость в заданное множество значений, определяемое оператором select контролируемого вводимого значения);

- `<select_expr>` — оператор SELECT, в котором употребляются SQL-выражения и возвращающий множество значений.

То есть в условии поиска можно задавать, чтобы вводимое значение было:

- равно, меньше, больше некоторой величины или значения, выбираемого оператором SELECT из некоторой таблицы;
- находилось между заданными величинами и т. д.

Например, утверждение

```
check (percent_change between 25 and 30)
```

задает ограничение на величину изменения процента между 25 и 30. Для более полной иллюстрации контроля ввода данных, принятого в Interbase, покажем это на примере двух таблиц: Budjet (табл. 1.17) и Dept (табл. 1.16). Последняя содержит сведения о департаментах (код, имя, место расположения и код руководителя), а первая — сведения о бюджете каждого департамента.

Таблица 1.16. *Таблица Dept*

Dept_No	Dept_Name	Dept_Location	Head_Dept
11	Sales	Moscow	11
12	Sales	S.Petersburg	12
13	Finace	N.Novgorod	13
14	Engeneering	Pscov	14

Таблица 1.17. *Таблица Budjet*

Dept_No	Dept_Name	Dept_Location	Head_Dept	Budjet
11	Sales	Moscow	11	11000
12	Sales	S.Petersburg	12	12000
13	Finace	N.Novgorod	13	13000
14	Engeneering	Pscov	14	14000

Вот как может выглядеть описание таблицы Budjet с использованием контроля ввода данных (листинг 1.2).

Листинг 1.2

```
CREATE TABLE BUDJET
 (
  DEPT_NO INTEGER NOT NULL,
  DEPT_NAME  VARCHAR(50) ,
  DEPT_LOCATION  VARCHAR(50),
  HEAD_DEPT  COMPUTED BY (DEPT_NO),
  BUDGET FLOAT,
 CHECK(DEPT_NO >= 11),
/* предыдущую строку можно записывать в виде:
CHECK( DEPT_NO > (SELECT DEPT_NO FROM DEPT WHERE DEPT_NO=11)),
*/
 CHECK( DEPT_NAME = (SELECT DEPT_NAME FROM DEPT WHERE DEPT_NO=11)),
 CHECK( DEPT_NAME IN ('SALES','FINANCE','ENGENEERING')),
/*предыдущую строку можно записывать в виде:
CHECK( DEPT_NAME IN (SELECT DEPT_NAME FROM DEPT)),
*/
CHECK( DEPT_LOCATION IN (SELECT DEPT_LOCATION FROM DEPT))
);
```

В листинге 1.3 дано описание таблицы Dept.

Листинг 1.3

```
CREATE TABLE DEPT
 (
   DEPT_NO INTEGER NOT NULL,
   DEPT_NAME VARCHAR(50),
   DEPT_LOCATION VARCHAR(50),
   HEAD_DEPT INTEGER
);
```

Создание базы данных типа Interbase

Базу данных Interbase можно создавать как с помощью утилиты isql, запускаемой в командной строке, так и с помощью утилиты IBConsole.

Создание базы данных с помощью утилиты IBConsole

1. Запустить утилиту IBConsole через **Пуск | Программы | Interbase**.
2. В появившемся диалоговом окне зарегистрировать новый сервер, выполнив

   ```
   Server > Register
   ```

   В открывшемся диалоговом окне выбрать **Local Server**, ввести регистрационные данные (при установке Interbase принято, что User = SYSDBA, а Password = masterkey. Впоследствии эти величины можно изменить) и нажать **OK**. В итоге получим то, что показано на рис. 1.1.

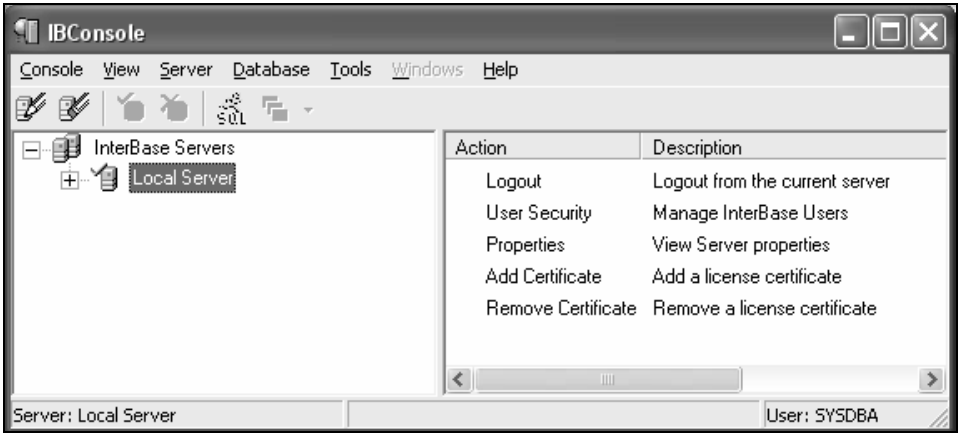


Рис. 1.1. Создание локального сервера для организации базы данных

Значок плюса в списке **Local Server** дает развертку, показанную на рис. 1.2.

3. Установим курсор мыши на позицию **Databases** в левом окне диалогового окна, показанного на рис. 1.2, и нажмем правую кнопку мыши (откроем контекстное меню элемента дерева с корнем **Local Server**). Получим картину, показанную на рис. 1.3 (такого же эффекта можно достичь, если выполнить команду **Database** из меню команд IBConsole).

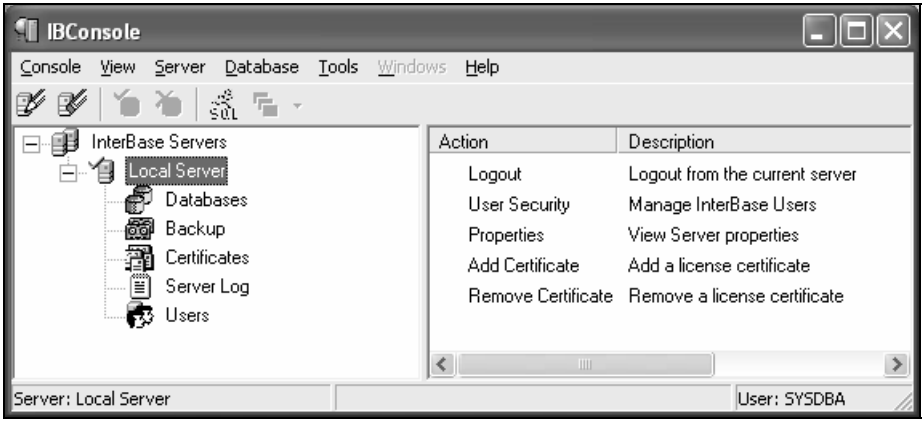


Рис. 1.2. Развертка структуры **Local Server**

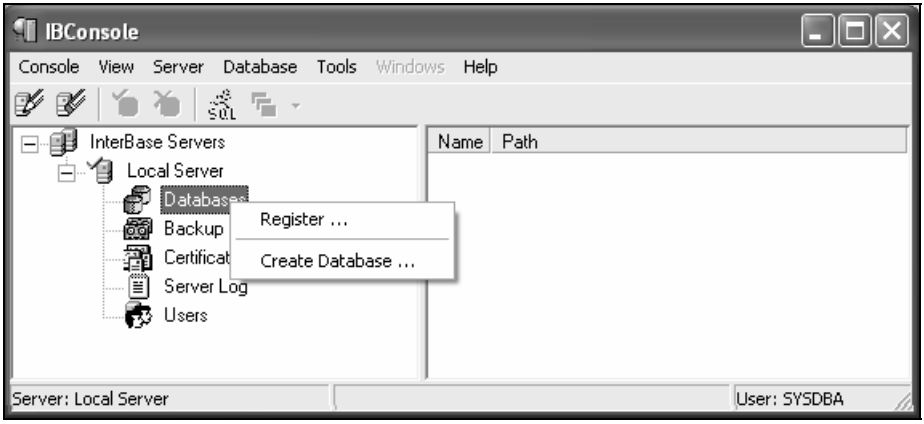


Рис. 1.3. Открытие контекстного меню элемента Databases

4. И теперь выполним опцию **Create Database**. Откроется диалоговое окно, показанное на рис. 1.4.
5. В этом диалоговом окне задаем информацию о файле базы данных, указывая его наименование с полным путем к нему и выбирая количество памяти под этот файл (количество страниц и размер страницы), а также символьный набор из выпадающего меню для отображения элементов БД, задаем алиас (имя, под которым будет фигурировать БД) и нажимаем **OK**. Получим картину, показанную на рис. 1.5.
6. Можно создать еще одну базу данных предложенным способом. Если при этом ее зарегистрировать, то получим картину, показанную на рис. 1.6.

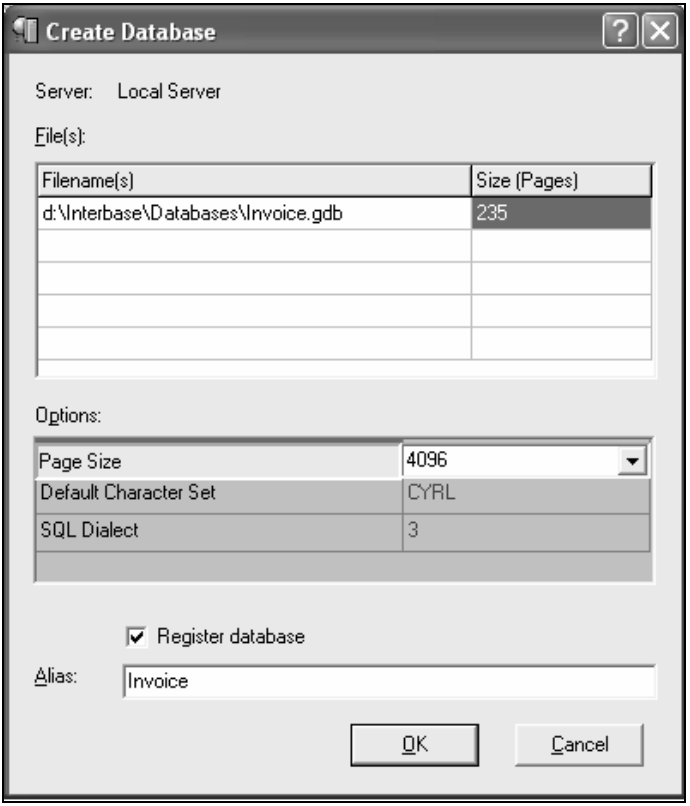


Рис. 1.4. Диалоговое окно для создания базы данных типа Interbase

Создание базы данных с помощью утилиты isql

Чтобы создать базу данных с помощью утилиты isql, надо перейти в командный режим (открыть окно MS-DOS), после чего можно поступить двояко: либо в командной строке набрать полностью команду на языке SQL, либо предварительно записать операторы создания базы данных в некоторый файл с помощью текстового редактора Блокнот, а затем в команде isql задать имя этого файла. В любом случае в окне DOS надо выполнить команду

```
c:\isql,
```

а далее набирать либо первый вариант данных, либо ввести

```
input "имя файла с путем к нему".
```

Кавычки ограничивают имя файла с путем к нему. В конце следует нажать <Enter>.

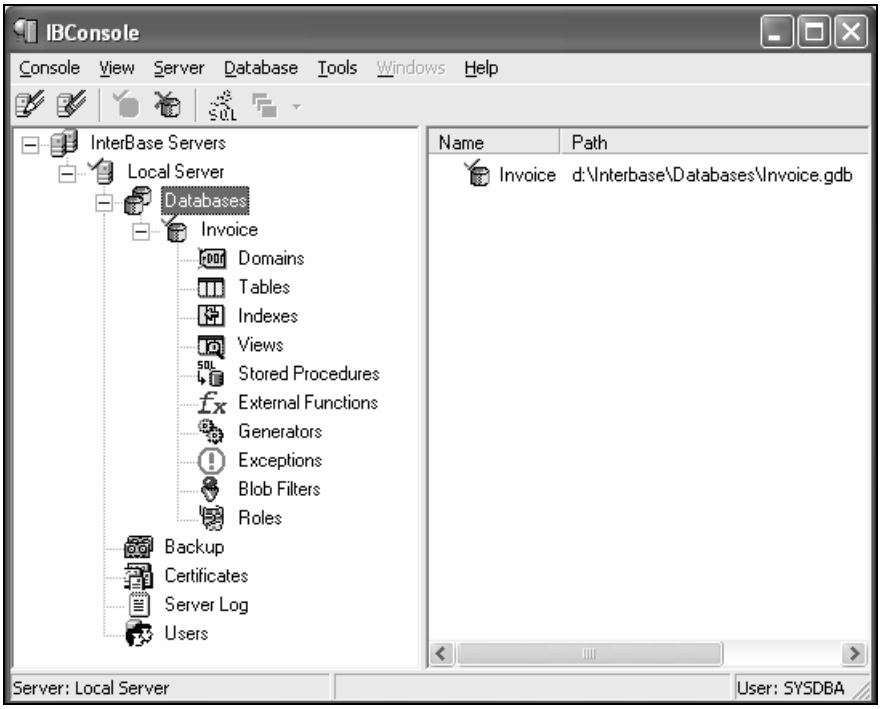


Рис. 1.5. Созданная и зарегистрированная БД

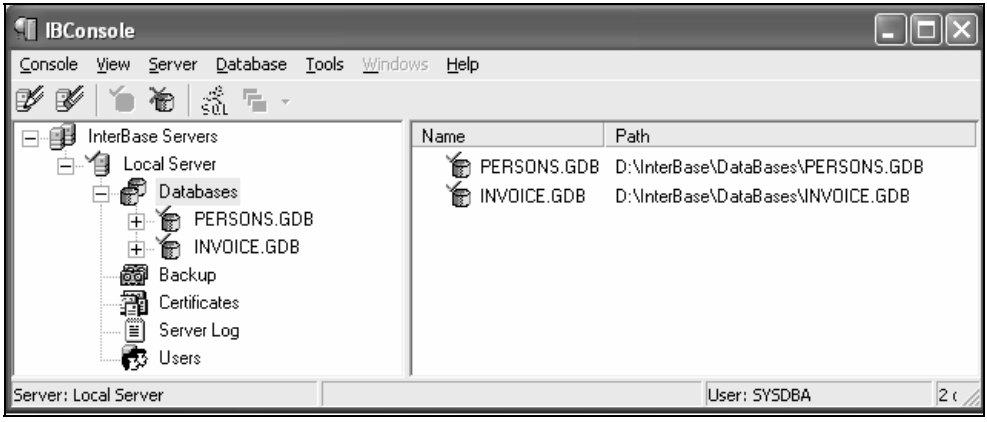


Рис. 1.6. На локальном сервере сформированы две БД

Пример создания базы данных

Пример показан на рис. 1.7. База данных создана с использованием утилиты IBConsole и утилиты isql. Обе используют интерактивный SQL.

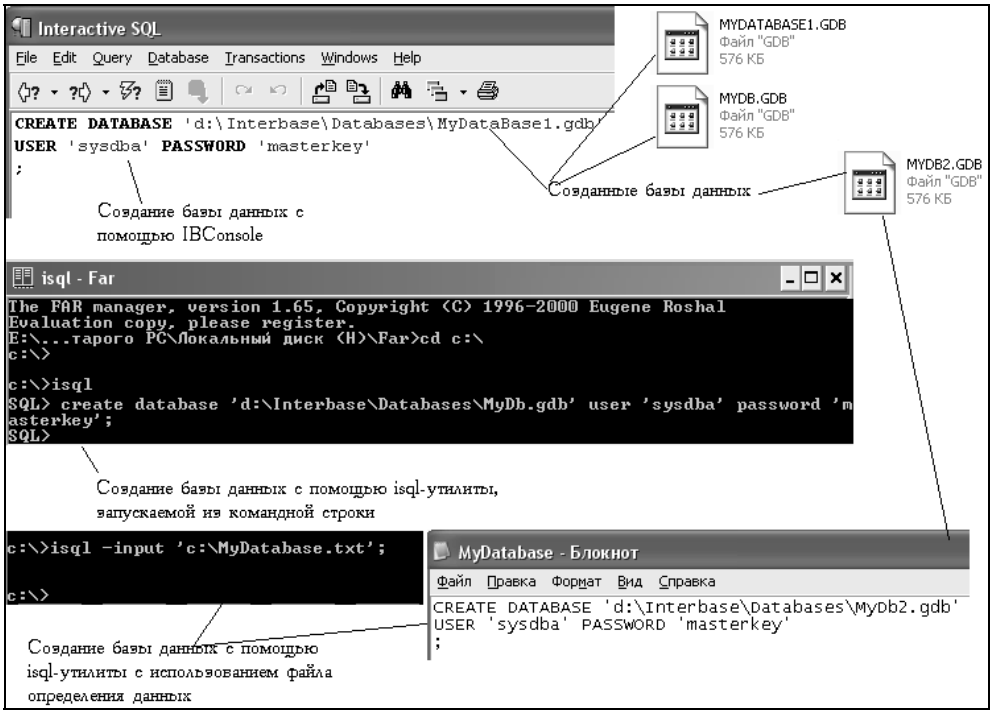


Рис. 1.7. Создание базы данных

Создание таблиц баз данных с помощью IBConsole

Таблица базы данных создается с помощью утилиты IBConsole следующим образом:

1. Создать описание таблицы с помощью оператора Create Table с применением редактора текстового файла, например, Блокнот (NotePad). Программа MS Word здесь не годится: у нее нетекстовый формат строк.
2. Запустить утилиту IBConsole путем исполнения команды **Пуск | Программы | Interbase | IBConsole**.

3. В диалоговом окне утилиты выбирается сервер, к которому надо подключиться (на его имени надо щелкнуть мышью).
4. Выполняется команда **Server/Login**. При этом откроется дерево с перечнем баз данных, расположенных на сервере, к которому мы подключились.
5. Надо выбрать базу данных, на которой требуется создать таблицу (щелкнуть мышью на ее имени в дереве) и нажать кнопку  для соединения с выбранной базой данных.
6. Нажать кнопку  для перехода в окно интерактивного SQL и после открытия этого окна вставить в него текст с командами создания таблицы из текстового файла.
7. Нажать кнопку  .

Хранимые процедуры

Это тоже элемент базы данных наряду с таблицами. Каковы его функции? Хранимая процедура — это специальная программа, написанная на так называемом языке процедур и триггеров системы InterBase и хранимая в качестве части данных базы данных. Поскольку вы создали хранимую процедуру, вы можете ее напрямую вызывать на выполнение из своего приложения. Хранимая процедура может получать входные параметры и возвращать определенные значения приложению.

Язык процедур и триггеров InterBase включает в себя операторы языка SQL и некоторые расширения, такие как IF THEN ELSE, WHILE DO, FOR SELECT DO, исключения и обработку ошибок.

Преимущества использования хранимых процедур:

- ❒ модульное оформление;
- ❒ приложения, обращающиеся к одной и той же базе данных, могут делить между собой хранимые процедуры (использовать одну и ту же хранимую процедуру), избегая дублирования кода (т. е. нет необходимости в повторном коде хранимой процедуры) и снижая таким образом размер самих приложений;
- ❒ рациональная поддержка. Когда процедура обновляется, ее изменения автоматически отражаются во всех приложениях, которые ее используют, и новая компиляция этих приложений не требуется. Приложения только однажды компилируются для каждого клиента;
- ❒ улучшение работы. И это — основное достоинство этого инструмента. Хранимые процедуры выполняются сервером, а не клиентом, что снижа-

ет загрузку сети и улучшает ее работу, особенно в условиях удаленного клиентского доступа. Например, в системе управления кадрами, когда общая база данных хранится на сервере и работники отдела кадров и руководство предприятия пользуются ее услугами, у некоторого клиента возникает потребность получить аналитические данные в некотором разрезе. При наличии в БД хранимой процедуры, получающей требуемую аналитику, нет нужды клиенту вытаскивать к себе все необходимые данные для расчета, а достаточно вызвать на сервере соответствующую хранимую процедуру, передав ей необходимые параметры, и эта процедура, на месте исполнив всю работу и не перегоняя большое количество данных по сети клиенту, выдаст ему только результат в виде небольшой таблицы. Этой же процедурой может воспользоваться и другой клиент и получить для себя такую же аналитику.

Работа с хранимыми процедурами

С помощью интерактивного SQL (isql) (это одна из разновидностей языка SQL в самой СУБД Interbase: наряду с ним существует DSQL (динамический SQL) и язык процедур и триггеров) можно создавать, модифицировать и удалять процедуры и исключения (о них — ниже). Существует два способа создавать, видоизменять и удалять хранимые процедуры с помощью интерактивного SQL:

- интерактивно;
- с помощью входного файла, содержащего операторы определения данных.

Обычно предпочтение отдают файлам, их легче модифицировать и обеспечивать для них соответствующую документацию. Для простых модификаций существующих процедур более подходит интерактивный режим. Пользователь, который создает процедуру, является ее собственником и может передавать привилегии исполнения процедуры другим пользователям, триггерам (о них — позже) и хранимым процедурам.

Использование файла определения данных

Для создания и изменения процедуры с использованием файла определения данных используют текстовый редактор, чтобы записать такой файл, затем сохраняют файл и исполняют с помощью интерактивного SQL (isql). Для этого либо применяют окно интерактивного SQL утилиты IBConsole, в которое вставляют данные из файла, либо используют команду isql в командной строке (режим DOS), задавая команду в виде

```
C:\ISQL -INPUT FILENAME DATABASE_NAME.
```

Здесь FILENAME — это имя файла определения данных, а database_name — имя используемой базы данных. Файл определения данных должен включать:

- операторы создания, модификации и удаления процедур и исключений;
- любые другие ISQL-операторы. Исключения должны быть созданы до ссылки на них в процедурах.

Вызов хранимых процедур

Приложения могут вызывать хранимые процедуры из SQL, DSQL, ISQL.

Существуют два типа хранимых процедур:

- SELECT-процедуры, которые приложение может использовать вместо таблицы или вьюера в операторе SELECT; результатом исполнения оператора SELECT над такой процедурой будет обычная таблица данных, какая получается, например, когда в SELECT стоит символ *. Например, если имеется SELECT-процедура с именем SP, то ее вызов на исполнение будет выглядеть так:

  ```
  select * from sp
  ```

- Executable-процедуры, которые приложение может вызывать оператором EXECUTE PROCEDURE. Executable-процедуры не обязательно должны возвращать значения вызывающим их программам. Синтаксис вызова таких процедур следующий:

  ```
  execute procedure proc_name [var [, var …]]
  [returning_values
  var [, var …]]
  ```

  Здесь:

  - `proc_name` — имя процедуры;
  - `[var [, var …]]` — необязательный список входных параметров;
  - `[returning_values var [, var …]]` — необязательный список выходных параметров, в которых процедурой возвращаются рассчитанные величины.

Оба типа процедур определяются оператором CREATE PROCEDURE и имеют одинаковый синтаксис. Различие состоит в том, как процедура написана, и как ее намерены использовать. SELECT-процедуры возвращают более одной строки базы данных, поэтому в вызывающих программах они появляются в качестве таблицы или вьюера.

Executable-процедуры — это подпрограммы, которые могут возвращать или не возвращать значения. Фактически одна и та же процедура может использоваться в качестве SELECT- или Executable-процедуры, хотя обычно процедура специально пишется для использования в операторе SELECT (SELECT-процедура) или в операторе EXECUTE PROCEDURE (Executable-процедура).

Привилегии для хранимых процедур

Чтобы использовать хранимые процедуры, пользователь должен быть либо создателем процедуры (создать ее у себя), или ей должна быть дана привилегия EXECUTE. Расширение к оператору GRANT назначает привилегию EXECUTE, а расширение к оператору REVOKE удаляет привилегию.

Сами хранимые процедуры иногда нуждаются в доступе к таблицам или вьюерам, для которых пользователь не имеет привилегий.

Создание процедур

Хранимая процедура создается оператором CREATE PROCEDURE в isql. Хранимая процедура состоит из заголовка и тела.

- Заголовок содержит:
  - имя хранимой процедуры, которое должно быть уникальным среди процедур, вьюеров и таблиц в базе данных;
  - список входных параметров (он может быть и пуст) с их типами, значения которых (параметров) процедура получает из вызывающей программы;
  - если процедура возвращает значения вызывающей программе, то оператор RETURNS сопровождается списком выходных параметров и их типами.
- Тело процедуры содержит:
  - список локальных переменных и их типов;
  - блок операторов на языке InterBase-процедур и триггеров, заключенный в скобки BEGIN и END. Блок сам может включать в себя другие блоки.

ПРИМЕЧАНИЕ

Так как каждый оператор в теле хранимой процедуры должен оканчиваться точкой с запятой (таково требование isql), вы должны определить различные символы для окончания оператора CREATE PROCEDURE в isql. Для этого надо использовать оператор SET TERM до оператора CREATE PROCEDURE, чтобы задать другой символ-окончание, чем точка с запятой. После оператора CREATE PROCEDURE можно включить другой оператор SET TERM для перехода обратно к точке с запятой.

Синтаксис оператора CREATE PROCEDURE:

```
CREATE PROCEDURE name
[(param datatype [, param datatype …])]
```

```
[RETURNS (param datatype [, param datatype …])]
AS
<procedure_body>;
<procedure_body> = [<variable_declaration_list>]
<block>
<variable_declaration_list> =
DECLARE VARIABLE var datatype;
[DECLARE VARIABLE var datatype;…]
<block> =
BEGIN
<compound_statement>
[<compound_statement> …]
END
<compound_statement> = {<block> | statement;}
```

Описание элементов приводится в табл. 1.18.

Таблица 1.18. *Описание аргументов создания процедуры*

Аргумент	Описание
Name	Имя процедуры. Должо быть уникальным среди имен процедур, таблиц и вьюеров в базе данных
param datatype	Входные параметры для передачи данных из вызывающей программы в процедуру. Param — имя входного параметра. Должно быть уникальным среди параметров процедуры. Datatype — тип данного (параметра), поддерживаемый в InterBase
RETURNS param datatype	Список выходных параметров, через которые процедура возвращает значения в вызывающую программу с их типами. Param — имя параметра, datatype — тип данного (параметра), поддерживаемый в InterBase. Процедура возвращает значения выходных параметров, когда встретит оператор SUSPEND в теле процедуры
AS	Ключевое слово, которое отделяет заголовок процедуры от ее тела
DECLARE VARIABLE var datatype	Объявляет локальные переменные (они используются только в теле процедуры). Каждое объявление переменной должно предваряться оператором DECLARE VARIABLE и заканчиваться точкой с запятой. Var — имя локальной переменной, datatype — ее тип
statement	Любой оператор на языке InterBase-процедур и триггеров

Модификация хранимых процедур

Для изменения хранимой процедуры используют оператор ALTER PROCEDURE. С его помощью изменяют определение существующей хранимой процедуры, и при этом ее зависимости от данных, на которые она ссылается, сохраняются. Изменения, совершенные относительно процедуры, становятся известными всем клиентским приложениям, ее использующим. Однако модифицировать процедуру может только пользователь с именем SYSDBA и собственник процедуры.

Для модификации процедуры следует выполнить следующие шаги:

1. Скопировать исходный файл определения данных, используемый для создания процедуры. Или использовать isql-extract оператор (речь идет о работе с IBConsole) для извлечения процедуры из базы данных в файл (чтобы увидеть процедуру в БД, надо щелкнуть мышью на позиции дерева БД, в которой хранится процедура). В результате в правом окне IBConsole появится перечень процедур в данной БД. Если открыть контекстное меню любой процедуры, то среди его команд будет и команда extract, извлекающая описание процедуры в окно. Это описание можно отредактировать и записать в некоторый файл.
2. Отредактировать файл, заменив оператор CREATE на ALTER, и изменить, при необходимости, определение процедуры.

Синтаксис этого оператора схож с синтаксисом оператора CREATE PROCEDURE:

```
ALTER PROCEDURE name
[(var datatype [, var datatype …])]
[RETURNS (var datatype [, var datatype …])]
AS
тело процедуры;
```

Имя процедуры (`name`) должно совпадать с именем существующей процедуры. Аргументы оператора ALTER PROCEDURE те же, что и аргументы для оператора CREATE PROCEDURE.

Удаление процедур

Оператор DROP PROCEDURE удаляет существующую хранимую процедуру из базы данных. Он может использоваться интерактивно с помощью isql или в файле определения данных. При удалении процедур имеются следующие ограничения:

- только пользователь с именем SYSDBA или собственник процедуры может ее удалить;

- нельзя удалить процедуру, которую используют другие процедуры, триггеры или вьюеры. Сначала надо разорвать связи, а потом уже удалять процедуру;
- нельзя удалить процедуру, зависящую рекурсивно или циклически от другой процедуры. Сначала надо модифицировать процедуру, чтобы удалить эту зависимость, а затем удалить саму процедуру;
- нельзя удалить процедуру, которая в данный момент используется активной транзакцией. Сначала надо завершить транзакцию, а затем удалять процедуру.

Синтаксис удаления процедуры таков:

```
DROP PROCEDURE name;
```

Здесь `name` — это имя существующей процедуры. Оператор

```
DROP PROCEDURE ACCOUNTS;
```

удаляет процедуру ACCOUNTS.

Модификация и удаление процедур, которые в данный момент используются

При модификации и удалении процедур, которые в данный момент используются, надо иметь в виду, что изменения в процедурах не видны в клиентских приложениях до тех пор, пока приложения не отсоединятся и снова не соединятся с базой данных.

Использование хранимых процедур

Хранимые процедуры могут использоваться приложениями различными способами. SELECT-процедуры используются вместо таблиц или вьюеров в операторе SELECT, Executable-процедуры используются в операторе EXECUTE PROCEDURE. Оба типа процедур определяются, как мы видели, через оператор CREATE PROCEDURE и имеют один и тот же синтаксис. Разница же между ними — в их написании и в намерении их использования. SELECT-процедуры всегда возвращают одну и более строк данных и поэтому по отношению к вызывающим их программам они являются как бы таблицами или вьюерами, в то время как Executable-процедуры — это простые подпрограммы, которые вызываются программами и могут возвращать или не возвращать данные.

В процессе разработки приложения следует создавать и тестировать хранимую процедуру на isql.

Использование Executable-процедур в isql

Executable-процедура вызывается с помощью оператора EXECUTE PROCEDURE.

Для исполнения хранимой процедуры на isql используется такой синтаксис:

```
EXECUTE  PROCEDURE name [() [param [, param …]] ()];
```

Имя процедуры (`name`) должно быть, естественно, задано, а каждый `param` задает значение входного параметра (константа). Должны быть определены все значения входных параметров. В isql нет возможности сформировать выходные параметры в операторе EXECUTE PROCEDURE, даже если процедура возвращает значения.

Использование SELECT-процедур в isql

SELECT-процедуры используют вместо таблиц или вьюеров в операторе SELECT. Они могут возвращать одну и более строк данных. Польза от применения таких процедур состоит в том, что:

- они могут принимать входные параметры, которые воздействуют на выход;
- они могут содержать логику, применение которой не доступно в обычных запросах или вьюерах;
- они могут возвращать строки из многих таблиц при использовании элемента UNION.

Синтаксис оператора SELECT при вызове процедуры таков:

```
SELECT <col_list> from name ([param [, param …]])
WHERE <условие поиска>
ORDER BY <order_list>;
```

Здесь `name` — имя вызываемой хранимой процедуры, а `param` — это константа, передаваемая соответствующему входному параметру. Все входные параметры должны получить свои значения:

- `col_list` — это список выходных параметров, разделенных запятыми, или символом *, если надо выбрать все строки;
- `WHERE` задает условие поиска для выбора подмножества строк;
- `ORDER BY` указывает, как станут упорядочиваться строки при их возврате.

Следующий код задает процедуру GET_EMP_PROJ, возвращающую EMP_PROJ — количество проектов, приходящихся на одного конкретного

работника, если передается его табельный номер EMP_NO в качестве входного параметра. Допустим, что мы определили таблицу EMPLOYEE_PROJECT как в табл. 1.19.

Таблица 1.19. *Описание EMPLOYEE_PROJECT*

EMP_NO	PROJ_ID	LAST_NAME
44	dgpii	Smith
47	vbase	Jenner
24	hwrii	Stevens

Процедура описана в листинге 1.4.

Листинг 1.4

```
SET TERM !! ;
CREATE PROCEDURE GET_EMP_PROJ (EMP_NO SMALLINT)
RETURNS (EMP_PROJ  VARCHAR(15) )AS
BEGIN
FOR SELECT PROJ_ID
FROM EMPLOYEE_PROJECT
WHERE EMP_NO = :EMP_NO
INTO :EMP_PROJ
DO
SUSPEND;
END !!
```

Следующий оператор исполняет процедуру GET_EMP_PROJ в isql, передавая значение 24 в качестве значения параметра EMP_NO:

```
SELECT * FROM GET_EMP_PROJ(24);
```

Результатом будет строка

```
Hwrii.
```

Создание хранимой процедуры в БД Persons и ее исполнение показаны на рис. 1.8—1.10.

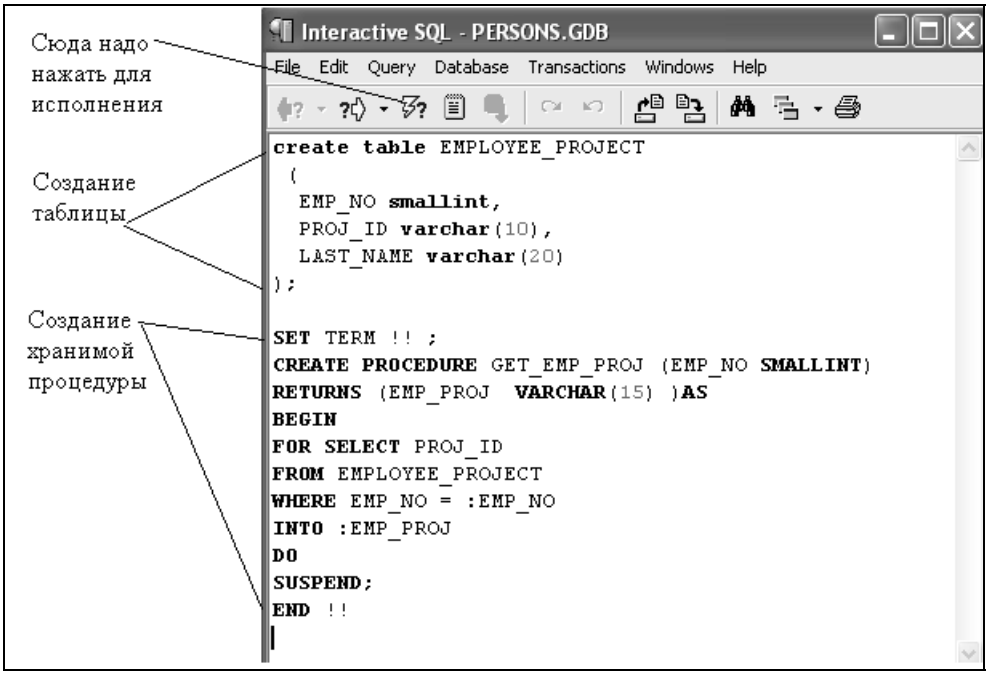


Рис. 1.8. Операторы создания таблицы и хранимой процедуры в окне интерактивного SQL перед их исполнением

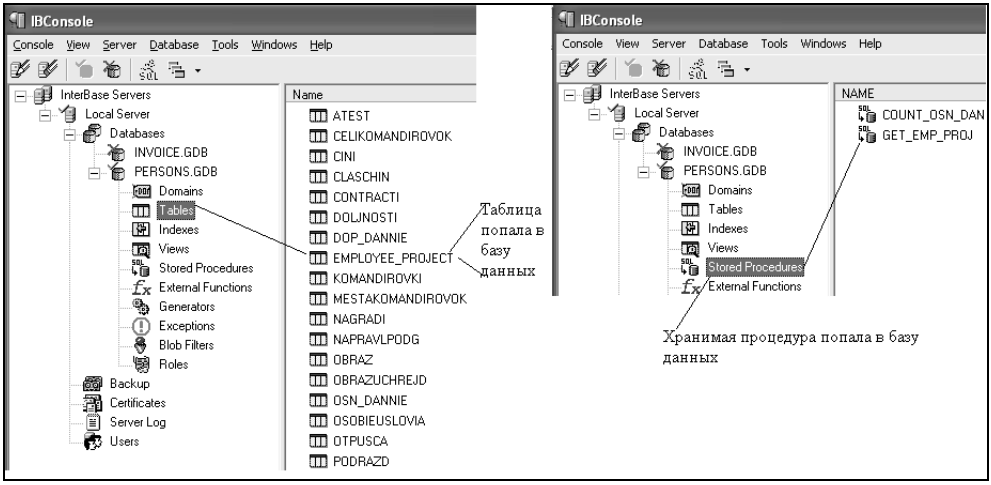


Рис. 1.9. Проверка попадания описаний создания таблицы и хранимой процедуры в БД

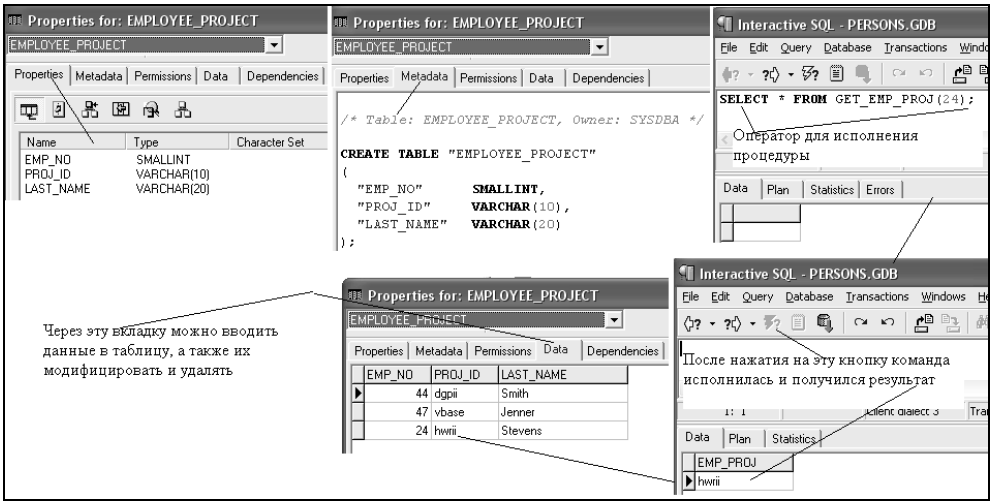


Рис. 1.10. Окна **Свойства таблицы** (видна ее структура и предоставлена возможность наполнять и модифицировать таблицу) и **Интерактивный SQL** для задания и исполнения команд SQL

Исключения

Исключение — это сообщение об ошибке, возникающее в хранимой процедуре или триггере. Исключения создаются оператором CREATE EXCEPTION, модифицируются оператором ALTER EXCEPTION и удаляются оператором DROP EXCEPTION. Хранимая процедура выдает исключение оператором

```
EXCEPTION name
```

При своем возникновении исключение возвращает вызывающей программе сообщение об ошибке и завершает выполнение процедуры, его вызывавшей, если только исключение не управляется оператором WHEN. Как и процедуры, исключения создаются и хранятся в базе данных, откуда они могут использоваться любой процедурой, которой они потребуются. Исключения должны быть созданы, естественно, до того, как они возникнут.

Создание исключений

Чтобы создать исключение, используют оператор

```
CREATE EXCEPTION name '<message>';
```

Например, следующий оператор создает исключение с именем REASSIGN_SALES:

```
CREATE EXCEPTION REASSIGN_SALES 'Reassign the sales records
before deleting this employee.';
```

Модификация исключений

Для изменения сообщения, возвращаемого исключением, используется оператор

```
ALTER EXCEPTION name '<message>';
```

Изменять исключение может только его создатель. В рамках предыдущего примера исключения изменим исключение:

```
ALTER EXCEPTION REASSIGN_SALES 'Can't delete employee — Reassign Sales';
```

Удаление исключений

Для удаления исключения используют оператор

```
DROP EXCEPTION name;
```

Например, оператор

```
DROP EXCEPTION REASSIGN_SALES;
```

удаляет исключение с именем REASSIGN_SALES.

При удалении исключений действуют следующие ограничения:

- удалить исключение может только пользователь, его создавший;
- исключения, которые находятся в использовании, не удаляются.

Оператор SHOW PROCEDURES выводит список зависимостей, процедур, исключений и таблиц, используемых хранимой процедурой. Оператор

```
SHOW PROCEDURE name
```

выводит заголовок и тело процедуры с именем `name`. Оператор

```
SHOW TRIGGERS table
```

выводит все триггеры, определенные для таблицы. Оператор

```
SHOW TRIGGER name
```

выводит заголовок и тело триггера с именем `name` (триггеры рассмотрены в следующем разделе).

Чтобы вызвать исключение в хранимой процедуре, надо выполнить оператор

```
EXCEPTION name;
```

Здесь `name` — это имя исключения, уже существующего (созданного) в базе данных. Точнее, это будет так: вы создаете хранимую процедуру и в ней обрабатываете некоторую подозрительную ситуацию, например, хотите из таб-

лицы удалить некоторую строку. Перед удалением строки надо проверить, не занята ли в данный момент таблица каким-либо клиентом. Если занята, вы выдаете команду на возникновение исключения

```
EXCEPTION name;
```

с выдачей соответствующего сообщения, которое, заранее созданное, хранится в базе данных под своим именем `name`. Когда исключение возникло, выполняются следующие действия:

- процедура, в которой возникло исключение, завершается и все выполненные ею действия аннулируются;
- вызвавшему процедуру приложению выдается сообщение об ошибке. В isql сообщение об ошибке выводится на экран. Но исключение может обрабатываться и оператором WHEN. Тогда поведение системы будет им и определяться.

Следующий пример показывает процедуру возникновения исключения и выдачу сообщения с именем REASSIGN_SALES:

```
IF (any_sales > 0) THEN
EXCEPTION REASSIGN_SALES;
```

Триггеры

О триггерах

Триггер — это отдельная подпрограмма, связанная с таблицей или вьюером, которая автоматически выполняет действие, когда модифицируется таблица или вьюер на уровне строки, а именно: когда строка вставляется, удаляется или модифицируется. Триггер никогда не вызывается напрямую, а только при модификации таблиц или вьюеров. Триггеры могут, как и хранимые процедуры, использовать аппарат исключений. В чем польза от применения триггеров?

- Принудительная обработка ограничений, что позволяет пользователю быть уверенным, что он вводит в строки только достоверные данные.
- Изменения в триггере автоматически отображаются у всех приложений, связанных с таблицей, что не требует их перекомпиляции, так как все происходит в базе данных, а не у каждого приложения в отдельности.
- Автоматическая регистрация изменений в таблицах. Приложение может регистрировать с помощью триггера изменения в таблице.
- Автоматическая регистрация изменений в базе данных с помощью обработчиков событий в триггерах.

Работа с триггерами

Триггеры можно создавать, модифицировать, удалять с помощью isql. Это можно осуществлять двумя способами:

- интерактивно;
- с помощью файла, содержащего соответствующие операторы (файл определения данных).

Предпочтительнее использовать файл, его легче модифицировать.

Использование файла определения данных

Чтобы создать или модифицировать триггер посредством файла, для работы с файлом надо использовать текстовый редактор и обработать файл с помощью isql. Для этого надо использовать команду

```
isql -input filename database_name.
```

Здесь `filename` — это имя обрабатываемого файла определения данных, а `database_name` — имя используемой базы данных. Или использовать окно интерактивного SQL в утилите IBConsole.

Файл определения данных может включать в себя:

- операторы создания, модификации или удаления триггеров, процедур и исключений;
- любые другие isql-операторы.

Создание триггеров

Триггер создается оператором CREATE TRIGGER, состоящим из заголовка и тела.

Заголовок триггера содержит:

- имя триггера, которое не должно повторяться в базе данных;
- имя таблицы базы данных, с которой связывается триггер;
- операторы, определяющие, когда триггер запускается.

Тело триггера содержит список локальных переменных с их типами и блок операторов на InterBase-языке процедур и триггеров, заключенный в операторные скобки BEGIN и END. Эти операторы выполняются, когда триггер запускается. Блок может включать в себя другие блоки операторов. Каждый оператор в теле триггера должен оканчиваться точкой с запятой. Но и тело триггера должно заканчиваться точкой с запятой. Поэтому этот момент както надо отрегулировать, иначе компилятор не поймет, где конец оператора, а где — тела. Для этого включают оператор SET TERM перед оператором CREATE TRIGGER, чтобы задать символ окончания триггера иной, чем

точка с запятой. Но после конца тела триггера выполняют другой SET TERM, чтобы изменить введенный ранее символ окончания снова на точку с запятой.

Синтаксис оператора CREATE TRIGGER:

```
CREATE TRIGGER name FOR {table | view}
[ACTIVE | INACTIVE]
{BEFORE | AFTER} {DELETE | INSERT | UPDATE}
[POSITION number]
AS <trigger_body>
<trigger_body> = [<variable_declaration_list>] <block>
<variable_declaration_list> =DECLARE VARIABLE variable datatype;
[DECLARE VARIABLE variable datatype; …]
<block> =
BEGIN
<compound_statement> [<compound_statement> …]
END
<compound_statement> = <block> | statement;
```

Описание аргументов:

- `name` — имя триггера. Должно быть уникальным в базе данных;
- `table` — имя таблицы или вьюера, с которыми связан триггер;
- `ACTIVE|INACTIVE` — указывается действие триггера по концу транзакции (напомним, что транзакция — это группа действий по выполнению некоторой операции, например, банковской — осуществление банковской проводки, когда с одного счета надо деньги снять, а на другой зачислить. Пока все это не будет выполнено, транзакция считается незавершенной):
  - `ACTIVE` (по умолчанию) — триггер действует;
  - `INACTIVE` — триггер не действует;
- `BEFORE|AFTER` — обязательный элемент. Определяет, когда запустится триггер;
  - `BEFORE` — перед соответствующей операцией по модификации строки БД;
  - `AFTER` — после модификации;
- `DELETE | INSERT | UPDATE` — операции модификации БД. Выбирается одна из них;
- `POSITION number` — задается порядок запуска триггеров после одной и той же операции. Величина `number` должна быть целым числом в диапазоне между 0 и 32 767, включая граничные значения;

- `DECLARE VARIABLE var <datatype>` — объявления локальных переменных триггера:
  - `var` — имя локальной переменной (уникальное в триггере);
  - `<datatype>` — тип локальной переменной.

Триггеры с более низкими номерами запускаются первыми. При величине 0 (по умолчанию) запускается первый триггер. Триггеры для таблицы не должны быть последовательными в отношении номеров позиций. Триггеры при одинаковом действии с таблицами и с одинаковыми номерами позиций будут запускаться в алфавитном (по их имени) порядке.

Следующий пример демонстрирует, как элемент POSITION определяет порядок запуска триггера. Пусть имеются четыре заголовка триггеров для таблицы ACCOUNTS:

```
CREATE TRIGGER A FOR ACCOUNTS BEFORE UPDATE POSITION 5 AS …
CREATE TRIGGER B FOR ACCOUNTS BEFORE UPDATE POSITION 0 AS …
CREATE TRIGGER C FOR ACCOUNTS AFTER UPDATE POSITION 5 AS …
CREATE TRIGGER D FOR ACCOUNTS AFTER UPDATE POSITION 3 AS …
```

Пусть имеет место такое обновление таблицы:

```
UPDATE ACCOUNTS SET C = 'canceled' WHERE C2 = 5;
```

Тогда произойдут следующие события: запустится триггер B, запустится триггер A, произойдет обновление таблицы, запустится триггер D, запустится триггер C.

InterBase-язык процедур и триггеров

Это полный язык программирования для работы с хранимыми процедурами и триггерами. Он включает в себя:

- SQL-операторы для манипулирования данными: INSERT, UPDATE, DELETE, SELECT;
- SQL-операторы и выражения;
- расширения SQL, включая операторы присваивания и управления, контекстные переменные, операторы событийного характера, исключения и операторы-обработчики ошибок. Хотя хранимые процедуры и триггеры используются для разных целей, язык их создания у них общий. Но имеются некоторые исключения:
  - контекстные переменные должны быть уникальны для триггеров;
  - входные и выходные параметры и операторы SUSPEND и EXIT, возвращающие значения, должны быть уникальны для хранимых процедур.

Язык хранимых процедур и триггеров не включает в себя многие операторы, доступные в DSQL. Следующие типы операторов не поддерживаются в хранимых процедурах и триггерах:

- операторы определения данных:
  - CREATE;
  - ALTER;
  - DROP;
  - DECLARE EXTERNAL FUNCTION;
  - DECLARE FILTER;
- операторы управления транзакциями:
  - SET TRANSACTION;
  - COMMIT;
  - ROLLBACK;
- операторы динамического SQL (DSQL):
  - PREPARE;
  - DESCRIBE;
  - EXECUTE;
- CONNECT/DISCONNECT и операторы посылки данных другой базе данных;
- GRANT/REVOKE;
- SET GENERATOR;
- EVENT INIT/WAIT;
- BEGIN/END DECLARE SECTION;
- BASED ON;
- WHENEVER;
- DECLARE CURSOR;
- OPEN;
- FETCH.

Следующий список показывает расширения языка для триггеров:

- BEGIN END — определяют блок операторов, который выполняется как один оператор. BEGIN — начинает блок, END — оканчивает его. После них точка с запятой не ставится;
- VARIABLE = expression. Оператор присвоения, который присваивает значение выражения переменной;

- /* comment_text */ — комментарий программиста, в котором может быть любое количество строк;
- EXCEPTION exception_name — запускается исключение с именем exception_name;
- EXECUTE PROCEDURE proc_name [var [, var]] [RETURNING_VALUES var [, var]] — исполняет хранимую процедуру с именем proc_name со списком входных параметров и возвращает значения выходных параметров;
- FOR select_statement DO compound_statement — повторяет исполнение оператора или блока, начинающегося после DO для каждой строки, извлеченной оператором select_statement, где select_statement — это обычный оператор SELECT, в котором требуется элемент INTO, который должен идти последним. compound_statement — это либо отдельный оператор, либо блок;
- IF (condition) THEN compound_statement [ELSE compound_statement] — оператор проверки условия: если оно TRUE, выполняется оператор или блок, следующий за элементом THEN, в противном случае выполняется оператор или блок, следующий за элементом ELSE, если тот присутствует в операторе. Condition — это булево выражение (TRUE, FALSE или UNKNOWN);
- NEW.column — контекстная переменная, которая показывает новое значение колонки таблицы в операциях INSERT или UPDATE;
- OLD.column — контекстная переменная, которая показывает значение колонки таблицы до операций UPDATE или DELETE. Например, следующий триггер запускается после обновления таблицы EMPLOYEE и сравнивает старую и новую зарплату работника. Если в зарплате произошли изменения, триггер вставляет данные в таблицу SALARY_HISTORY (создается история обновлений):

```
SET TERM !! ;
CREATE TRIGGER SAVE_SALARY_CHANGE  FOR EMPLOYEE
AFTER UPDATE
AS
BEGIN
IF (old.salary <> new.salary) THEN
INSERT INTO SALARY_HISTORY (EMP_NO, CHANGE_DATE,
UPDATER_ID, OLD_SALARY, PERCENT_CHANGE)
VALUES (old.emp_no, 'now', USER, old.salary,
(new.salary - old.salary) * 100 / old.salary);
END !!
SET TERM ; !!;
```

- POST_EVENT event_name — объявляет событие с именем event_name;
- WHILE (условие) DO compound_statement — пока условие = TRUE, выполняется оператор compound_statement;
- WHEN {error [, error]|ANY} DO compound_statement — оператор обработки ошибки:
  - если происходит одна из указанных в списке ошибок, выполняется оператор compound_statement;
  - error — это EXCEPTION exception_name;
  - ANY — обрабатывается любая ошибка.

Использование в isql оператора SET TERM

Как отмечалось уже, каждый оператор в теле триггера должен заканчиваться точкой с запятой. Поэтому надо иметь какой-то другой символ, чтобы отмечать момент окончания тела триггера (его надо отмечать в соответствии с синтаксисом). Для этой цели в isql включен оператор SET TERM, который должен быть выполнен перед оператором CREATE TRIGGER, чтобы определить символ окончания, который будет писаться после тела триггера. После же окончания триггера надо выполнить новый оператор SET TERM, чтобы вернуться к точке с запятой. Следующий пример иллюстрирует сказанное выше. Временным символом (терминатором) окончания тела триггера выбрана комбинация символов "!!":

```
SET TERM !! ;
CREATE TRIGGER SIMPLE FOR EMPLOYEE
AFTER UPDATE
AS
BEGIN
…
END !! // конец тела триггера
SET TERM  ; !!
```

После оператора SET TERM должен быть пробел. Кроме этого, каждый оператор SET TERM сам заканчивается текущим терминатором. Поэтому в первом SET TERM терминатором служит ";" , а в последнем — "!!".

> ***Примечание***
>
> В версии Interbase 7 такого оператора уже нет: система сама распознает терминаторы.

Вьюеры

Пользователи баз данных обычно нуждаются в доступе к подмножеству данных базы данных. Кроме того, требования к данным у одного пользователя или у группы пользователей часто совпадают. Вьюеры обеспечивают возможность создания пользовательского набора из таблиц. Они выводят из таблиц только группы данных, интересующие пользователя. Как только вьюер определен, его можно выводить и выполнять над ним операции, как над обыкновенной таблицей. Вьюер может происходить от одной и более таблиц или от другого вьюера. То есть вьюеры выглядят как обычные таблицы, хотя они физически не хранятся в базе данных. База данных хранит только их определения (шаблоны) и использует их для фильтрации данных при запросе вьюера. Важно понять, что создание вьюера не создает копию данных в другой таблице. Когда вы меняете данные через вьюер, на самом деле вы их меняете в соответствующей таблице. И наоборот: когда в базовой таблице происходит изменение, то оно автоматически отображается во вьюере, производном от этой таблицы. Вьюер — это как бы окно или фрейм, через которые можно видеть нужные данные в таблице или в группе таблиц. Определения данных, шаблон вьюера — это окно, фрейм.

Вьюер может быть создан из подмножества колонок одной таблицы. Пусть, например, таблица JOB в базе данных employee.gdb имеет 8 колонок: JOB_CODE, JOB_GRADE, JOB_COUNTRY, JOB_TITLE, MIN_SALARY, MAX_SALARY, JOB_REQUIREMENT и LANGUAGE_REQ. Следующий вьюер выводит список уровней зарплаты (подмножество колонок) для всех работ (все строки) в таблице JOB:

```
CREATE VIEW JOB_SALARY_RANGES AS
SELECT JOB_CODE, MIN_SALARY, MAX_SALARY FROM JOB;
```

Вьюер может быть создан из подмножества строк одной таблицы. Следующий вьюер выводит все колонки таблицы JOB, но только подмножество строк из этих колонок, отвечающих условию MAX_SALARY, меньше, чем 15 000:

```
CREATE VIEW LOW_PAY AS SELECT * FROM JOB
WHERE MAX_SALARY < 15000;
```

Вьюер может быть создан из подмножества комбинации строк и колонок одной таблицы. Следующий вьюер выводит только колонки JOB_CODE и JOB_TITLE , в которых MAX_SALARY меньше 15 000:

```
CREATE VIEW ENTRY_LEVEL_JOBS AS
SELECT JOB_CODE, JOB_TITLE FROM JOB
WHERE MAX_SALARY < 15000;
```

Вьюер может быть создан из подмножества строк и столбцов многих таблиц (так называемые объединения). Следующий пример показывает вьюер, созданный из таблиц JOB и EMPLOYEE. Последняя содержит 11 колонок: EMP_NO, FIRST_NAME, LAST_NAME, PHONE_EXT, HIRE_DATE, DEPT_NO, JOB_CODE, JOB_GRADE, JOB_COUNTRY, SALARY, FULL_NAME. Вьюер выводит две колонки из таблицы JOB и две из EMPLOYEE и возвращает только те строки, в которых SALARY меньше, чем 15 000:

```
CREATE VIEW ENTRY_LEVEL_WORKERS AS
SELECT JOB_CODE, JOB_TITLE, FIRST_NAME, LAST_NAME
FROM JOB, EMPLOYEE
WHERE JOB.JOB_CODE = EMPLOYEE.JOB_CODE AND SALARY < 15000;
```

Польза от использования вьюеров

Основные выгоды состоят в следующем:

- упрощенный доступ к данным. Вьюеры позволяют инкапсулировать подмножества данных из одной или более таблиц, чтобы потом использовать эти данные в запросах, не создавая каждый раз запросы заново;
- заказной доступ к данным. Вьюеры обеспечивают возможность создания множества запросов заранее по заказу потребителей с разными потребностями;
- независимость от данных. Вьюеры защищают пользователей от эффекта изменения структуры таблиц, на которых построены вьюеры. Например, администратор базы данных решает разбить одну таблицу на две, и вьюер должен быть создан как объединение двух новых таблиц, о чем пользователь может и не догадываться. Администратором будет переделан вьюер, а пользователям станет поступать через вьюер прежняя информация;
- безопасность данных. Она осуществляется при использовании вьюеров за счет ограничения доступа к неподходящим пользователю данным. Например, вы можете просматривать только информацию о работе, но связанную с ней информацию о зарплате не сможете, так как она не представлена во вьюере. То есть здесь действует правило: каждому пользователю — свой набор вьюеров, что намного проще осуществить, чем создавать специальные средства и правила доступа различных категорий пользователей к информации базы данных в самом приложении.

Создание вьюеров

Вьюер создается с помощью оператора CREATE VIEW, который создает виртуальную таблицу на основе одной или более таблиц базы данных. При

этом во вьюерах можно употреблять те же операции (SELECT, PROJECT, JOIN и UNION), что и для таблиц. Пользователь, который создает вьюер, является его собственником и имеет на него все привилегии, включая возможность передавать привилегии другим пользователям, триггерам и хранимым процедурам. Причем делаться это может без доступа к базовым таблицам, на основе которых создан вьюер. Синтаксис оператора CREATE VIEW таков:

```
CREATE VIEW name [(view_col [, view_col …])]
AS <select> [WITH CHECK OPTION];
```

> ***Примечание***
>
> Нельзя определять вьюер, базирующийся на результатах работы хранимой процедуры.

Задание имен колонок вьюера: `view_col` — это имя одной или более колонок вьюера. Вьюер может включать колонки, базирующиеся на выражениях. При задании колонок надо помнить, что их имена соответствуют по порядку и количеству колонкам оператора SELECT, поэтому необходимо указывать колонки вьюера для каждой выбираемой колонки или не указывать вовсе. Имена колонок должны быть уникальны среди всех имен колонок вьюера. Если имена колонок не указаны, вьюер берет по умолчанию имена колонок соответствующей таблицы, на основе которой он строится. Если определение вьюера включает выражение, имена колонок задавать обязательно. Язык isql не поддерживает определения вьюеров, содержащие элементы UNION. Вы должны написать приложение для создания этого типа вьюера.

Использование оператора SELECT

Оператор SELECT задает критерий выборки для строк, которые должны включаться во вьюер. В SELECT указывается список колонок, которые должны быть включены во вьюер из базовой таблицы. Когда вместо списка используется символ звездочки, то все колонки базовой таблицы будут включены во вьюер. Колонки выводятся в том порядке, в котором они расположены в базовой таблице. Следующий пример показывает создание вьюера, содержащего все колонки из таблицы EMPLOYEE:

```
CREATE VIEW MY_VIEW AS SELECT * FROM EMPLOYEE;
```

В элементе FROM указывается исходная таблица (таблица-источник). В приведенном ранее примере — это таблица EMPLOYEE. При необходимости в элементе WHERE указывается условие выбора строк.

В следующем примере задано, что выберутся только строки, отражающие работников, работающих в США:

```
CREATE VIEW USA_EMPLOYEES
  AS SELECT * FROM EMPLOYEE
    WHERE JOB_COUNTRY = 'USA';
```

В операторе SELECT может задаваться режим WITH CHECK OPTION, предваряющий операторы INSERT или UPDATE во вьюере. Как используется этот режим? WITH CHECK OPTION определяет правила модификации данных таблиц через вьюер. Этот режим может быть включен лишь в том случае, если вьюер имеет свойство обновлять таблицы (updatable). Вьюеры, созданные с режимом WITH CHECK OPTION, дают возможность InterBase проверить перед окончанием завершения операции, вставилась или обновилась строка с помощью вьюера. Через вьюер можно вставлять данные только в колонки, названные в этом вьюере, что вполне естественно. InterBase засылает NULL-значения для неизвестных колонок. WITH CHECK OPTION защищает вас от вставки или обновления значений, которые не удовлетворяют условию, указанному в элементе WHERE оператора SELECT.

Примеры

Допустим, что требуется создать вьюер, который позволяет иметь доступ к информации о всех департаментах с бюджетом в диапазоне $10 000—$500 000. Вот как определяется вьюер с именем SUB_DEPT:

```
CREATE VIEW SUB_DEPT (DEPT_NAME, DEPT_NO, SUB_DEPT_NO, LOW_BUDGET) AS
SELECT DEPARTMENT, DEPT_NO, HEAD_DEPT, BUDGET
FROM DEPARTMENT WHERE BUDGET BETWEEN 10000 AND 500000
WITH CHECK OPTION;
```

Вьюер SUB_DEPT основан на единственной таблице DEPARTMENT. Если вы — создатель вьюера или имеете привилегии INSERT, вы можете вставить новые данные в колонки DEPARTMENT, DEPT_NO, HEAD_DEPT и BUDGET таблицы DEPARTMENT. WITH CHECK OPTION страхует, чтобы все значения, введенные через вьюер и удовлетворяющие условию, указанному в элементе WHERE, попали туда, куда им положено, а не удовлетворяющие — чтобы были отвергнуты.

Следующий оператор вставляет новую строку через вьюер SUB_DEPT в департамент Publications:

```
INSERT INTO SUB_DEPT (DEPT_NAME, DEPT_NO, SUB_DEPT_NO, LOW_BUDGET)
VALUES ('Publications', '7735', '670', 250000);
```

InterBase вставит NULL во все остальные колонки таблицы DEPARTMENT, которые напрямую не указаны во вьюере.

> ***Примечание***
>
> При создании вьюера в операторе SELECT нельзя употреблять выражение ORDER BY.

Использование выражений при определении колонок

Выражение — это комбинация любых SQL-операторов, участвующих в сравнении или в вычислении и возвращающая определенное значение. Примером выражений служит сцепление символов строки, выполнение действий над числовыми данными, выполнение сравнений с использованием операций отношения (<, >, <= и т. д.) или булевых операций (AND, OR, NOT). Выражение должно возвращать единственное значение и не может быть массивом или возвращать массив. Любая колонка, используемая в выражении, должна существовать. Вот пример вьюера, в котором колонка задана выражением:

```
CREATE VIEW 10%_RAISE (EMPLOYEE, NEW_SALARY) AS
SELECT EMP_NO, SALARY *1.1 FROM EMPLOYEE;
```

Здесь из таблицы EMPLOYEE формируется вьюер, в котором станут отображаться те работники, у которых произошел рост зарплаты на 10% (значение колонки SALARY с коэффициентом — выражение).

Типы вьюеров: read-only (только для чтения) и updatable (обновляющие)

Когда вы обновляете вьюер, изменения происходят и в соответствующей таблице БД, на основе которой создан вьюер, но только при определенных условиях. Если вьюер соответствует этим условиям, он имеет свойство обновлять, т. е. является обновляющим. Если не соответствует — остается только способным предоставлять информацию, но не изменять ее. Тогда он служит только для чтения информации. Чтобы изменить определение вьюера, его надо сначала удалить, а затем вновь воссоздать с новыми качествами.

Вьюер может иметь свойства обновления, если выполняются следующие условия:

- он является подмножеством единственной таблицы или другого обновляющего вьюера;
- все колонки базовой таблицы, исключенные из определения вьюера, допускают NULL-значения;
- оператор SELECT во вьюере не содержит подзапросов, утверждения DISTINCT, элемента HAVING, функций агрегирования, объединенных таблиц, функций, определенных пользователем, и хранимых процедур.

Если же вьюер не отвечает этим условиям, он служит только для чтения (имеет свойств read only). Однако такие вьюеры все-таки можно сделать обновляющими, если воспользоваться триггерами и уникальными индексами (этот вопрос мы не рассматриваем).

Привилегии вьюера

Создатель вьюера должен обладать следующими привилегиями:

- создавать read-only-вьюеры. Создатель нуждается в SELECT-привилегиях (в привилегиях выборки) для любой таблицы, из которой создается вьюер;
- создавать updatable-вьюеры. Создатель нуждается во всех привилегиях для любой таблицы, из которой создается вьюер.

В следующем примере создается updatable-вьюер:

```
CREATE VIEW EMP_MNGRS (FIRST, LAST, SALARY) AS
SELECT FIRST_NAME, LAST_NAME, SALARY
FROM EMPLOYEE
WHERE JOB_CODE = 'Mngr';
```

Следующий оператор использует уже внутренний запрос (подзапрос), чтобы создать вьюер, поэтому такой вьюер будет иметь свойство read-only:

```
CREATE VIEW ALL_MNGRS AS
SELECT FIRST_NAME, LAST_NAME, JOB_COUNTRY FROM EMPLOYEE
WHERE JOB_COUNTRY IN
(SELECT JOB_COUNTRY FROM JOB
WHERE JOB_TITLE = 'manager');
```

Нижеследующий оператор создает вьюер из двух таблиц, поэтому созданный вьюер будет иметь свойство read-only:

```
CREATE VIEW PHONE_LIST AS
SELECT EMP_NO, FIRST_NAME, LAST_NAME, PHONE_EXT, LOCATION, PHONE_NO
FROM EMPLOYEE, DEPARTMENT
WHERE EMPLOYEE.DEPT_NO = DEPARTMENT.DEPT_NO;
```

Вставка данных через вьюер

Строки могут вставляться и обновляться через вьюер, если соблюдены следующие условия:

- вьюер имеет свойство updatable;
- хранимая процедура пользователя имеет привилегию INSERT для вьюера;
- вьюер создавался с использованием режима WITH CHECK OPTION.

Можно имитировать обновление read-only-вьюера, если написать триггеры, которые выполняют соответствующие записи в базовую таблицу.

Удаление вьюеров

Удаление вьюера из базы данных может производить только его создатель. Удаление выполняется с помощью оператора DROP VIEW. Никакого воздействия при этом на соответствующие таблицы БД не происходит. Если вьюер используется другим вьюером или хранимой процедурой, удаление не выполнится. Синтаксис оператора удаления таков:

```
DROP VIEW name;
```

Здесь `name` — имя вьюера.

Как отмечалось выше, модифицировать напрямую вьюер нельзя. Сначала его надо удалить, а затем вновь создать оператором CREATE VIEW с новыми возможностями.

Привилегии

Обзор привилегий доступа при использовании SQL

Безопасность доступа с помощью SQL к БД управляется на уровне привилегий доступа к таблицам или вьюерам — списка операций, которые пользователю позволено выполнять над таблицей или вьюером. Привилегии доступа задаются оператором GRANT для определенных пользователей, для ролевых имен, определяющих классы пользователей, которым разрешен доступ к объекту, а также для объектов, таких как хранимые процедуры или триггеры. Оператор GRANT может также предоставить возможность пользователям или хранимым процедурам выполнять хранимые процедуры с помощью EXECUTE-привилегии (привилегии на исполнение) и может передавать ролевые имена пользователям. Оператор REVOKE удаляет привилегии, назначенные оператором GRANT.

GRANT может использоваться в следующих вариантах:

```
Grant SELECT, INSERT, UPDATE, DELETE, REFERENCES
```

Он задает привилегии для работы с таблицей для пользователей, триггеров, хранимых процедур, вьюеров (может применяться утверждение WITH GRANT OPTION). Оператор

```
Grant SELECT, INSERT, UPDATE, DELETE
```

задает привилегии для работы с вьюером для пользователей, триггеров, хранимых процедур, вьюеров (может применяться утверждение WITH GRANT OPTION). Оператор

```
Grant SELECT, INSERT, UPDATE, DELETE, REFERENCES
```

задает привилегии для работы с таблицей для ролей — классов пользователей, которым разрешен доступ к объекту. Оператор

```
Grant SELECT, INSERT, UPDATE, DELETE
```

задает привилегии для работы с вьюером для ролей. Оператор

```
GRANT A ROLE TO USERS
```

передает роль пользователям (может применяться утверждение WITH ADMIN OPTION). Оператор

```
GRANT EXECUTE
```

дает разрешение на выполнение хранимой процедуры пользователями, триггерами, хранимыми процедурами, вьюерами (может применяться утверждение WITH GRANT OPTION).

Доступ и безопасность по умолчанию

Все таблицы и хранимые процедуры защищены от неавторизованного доступа еще при их создании. Первоначально только создатель таблицы, ее собственник, имеет доступ к таблице. И только ее собственник может использовать оператор GRANT, чтобы назначить привилегии другим пользователям или процедурам. Только создатель процедуры может выполнить или вызвать процедуру и только он может назначить EXECUTE-привилегии другим пользователям или процедурам. Interbase поддерживает также SYSDBA-пользователя (пользователя с таким именем, которое принято по умолчанию для пользователя БД Interbase), который имеет доступ ко всем объектам базы данных.

Доступные привилегии

Вот список привилегий доступа, которые могут передаваться и отменяться:

- ALL — привилегии по выборке, вставке, обновлению и удалению данных, а также по ссылке на первичный ключ из ключа другой таблицы;
- SELECT — привилегия по чтению (выборке) данных;
- INSERT — привилегия по записи данных;
- UPDATE — привилегия по изменению существующих данных;
- DELETE — привилегия по удалению данных;
- EXECUTE — привилегия по выполнению или вызову хранимой процедуры;
- REFERENCES — привилегия по возможности ссылки первичного ключа с помощью ключа другой таблицы;
- ROLE — все привилегии назначаются роли (группе пользователей, относящихся к одному классу).

Программирование в среде Borland C++Builder

Настройка Interbase-драйвера

Для работы с Interbase в среде Borland C++Builder следует оптимизировать InterBase SQL Links-драйвер. Оптимизация проводится с помощью BDE-администратора. После его запуска надо войти во вкладку, раскрыть узел **Drivers**, затем раскрыть узел **Native** дерева **Configuration**, щелкнуть на позиции **INTRBASE**, чтобы вывести установки драйвера. Они появятся в правом окне BDE (рис. 1.11).

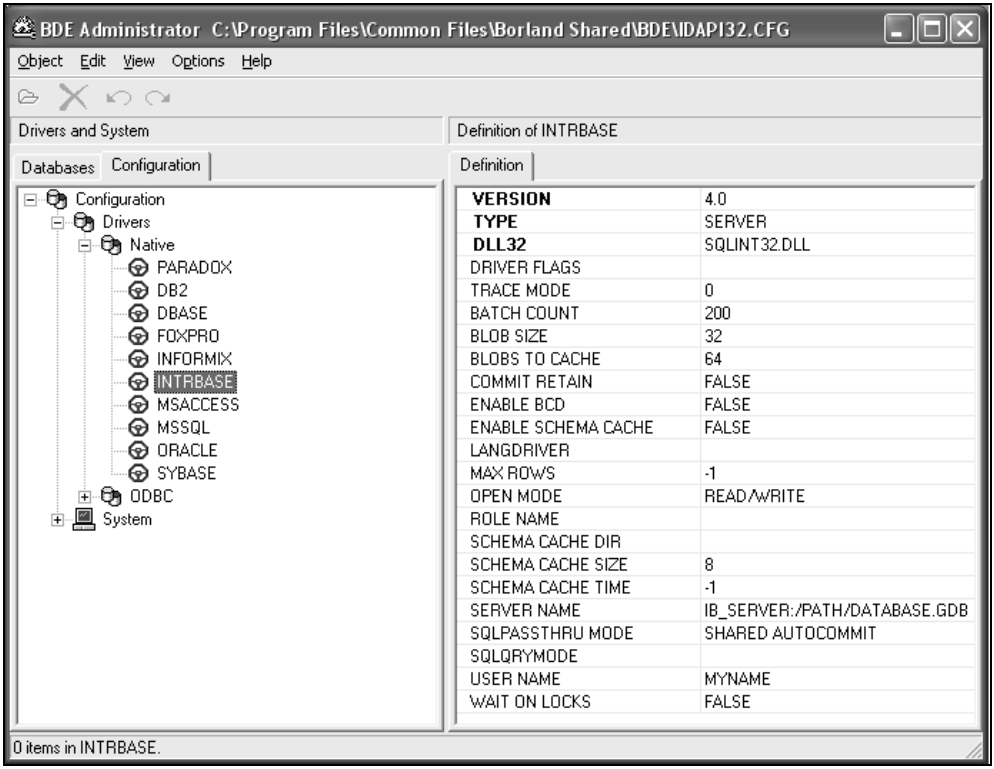


Рис. 1.11. Установки InterBase-драйвера

Для оптимизации InterBase-драйвера необходимо изменить следующие опции: **DRIVER FLAGS**, **SQLPASSTHRU MODE**, **SQLQUERY MODE**.

Рассмотрим их детально:

- Установка **DRIVER FLAGS**. В зависимости от потребности вашей базы данных можно установить опцию **DRIVER FLAGS** в значения либо 512, либо 4608. Рекомендуемое значение — 4608. Если установить **DRIVER FLAGS** в значение 512, это сделает режим исполнения транзакций, принятый по умолчанию, таким, что даже при изменении отдельной записи BDE станет делать пересмотр всех записей в наборе данных. Если установить **DRIVER FLAGS** в значение 4608, то режим исполнения транзакций, принятый по умолчанию, станет таким, что улучшится производительность базы данных при обновлении больших наборов данных.
- Установка **SQLPASSTHRU MODE**. Этот режим определяет, могут ли делить между собой одно и то же соединение с базой данных BDE и операторы SQL-транзитной пересылки:
  - по умолчанию **SQLPASSTHRU MODE** установлен в **SHARED AUTOCOMMIT**;
  - чтобы снизить издержки по автоматическому контролю транзакций, этот режим устанавливают в **SHARED NOAUTOCOMMIT**;
  - однако если вы хотите передать контроль за транзакциями вашему серверу, этот режим следует установить в **NOTSHARED**. В зависимости от количества данных это может повысить производительность InterBase в несколько раз.

  Рекомендуемое значение для этого режима — **SHARED NOAUTOCOMMIT**.
- Установка **SQLQUERY MODE**. Установка **SQLQUERY MODE** в значение **SERVER** позволит InterBase вместо BDE интерпретировать и исполнять SQL-операторы.

Работа с компонентом TQuery

При работе с таблицами базы данных в Borland C++Builder, как известно, используют компонент TTable. Но при работе с удаленными базами данных вместо компонента TTable следует использовать компонент TQuery. И в InterBase, естественно, TTable лучше не использовать. Дело в том, что, хотя TTable очень подходит для работы с данными и методами, он не может использоваться в приложениях "клиент-сервер": он предназначен для работы на небольших таблицах локальной базы данных. TTable собирает информацию о данных в таблице базы данных и пытается работать с набором данных в кэш-памяти. Он обновляет данные, когда вы исполняете метод Post или TDatabase->Rollback-метод. Это вызывает огромную перегрузку сети для большинства баз данных, используемых в режиме "клиент-сервер" и содер-

жащих большие наборы данных. В архитектуре клиент-сервер намного выгоднее использовать компонент TQuery.

Для улучшения производительности InterBase следует установить следующие свойства и методы TQuery:

- свойство CachedUpdates должно быть установлено в FALSE, чтобы позволить серверу управлять обновлениями, удалениями и конфликтами;
- свойство RequestLive должно быть установлено в FALSE, чтобы избавить VCL от хранения копии строк с клиентской стороны. Это увеличит производительность за счет уменьшения количества данных, перемещаемых по сети.

Кроме этого, для улучшения производительности необходимо:

- использовать метод Locate только для локальных наборов данных;
- избегать использования свойства подсчета количества записей RecordCount в таблице;
- давайте возможность серверу, а не вашему TQuery фильтровать информацию перед отсылкой ее вам по сети;
- метод Commit также вызывает работу со всеми данными, когда режим BDE **DRIVER FLAGS** не установлен в 4096, или когда вы используете явно управление транзакциями.

Использование генераторов

Использование InterBase-триггера для модификации значения первичного ключа таблицы может явиться причиной появления сообщения об ошибке удаления ключа или записи, порождаемой BDE. Эту ситуацию можно обойти, если к триггеру добавить генератор.

Генератор — это механизм, который создает уникальный последовательный номер, который автоматически вставляется в колонку базы данных с типом read-write, когда происходят SQL-операции манипуляции с данными, такие как INSERT или UPDATE. Генераторы обычно используются для формирования уникальных значений, которые вставляются в колонку, используемую в качестве первичного ключа. Например, программист, создающий приложение по обработке счетов-фактур, хочет быть уверенным, что каждый номер счета-фактуры, введенный в базу данных, будет уникальным. Тогда он создает генератор, чтобы номера счетов-фактур формировались автоматически, вместо того, чтобы писать специальный участок программы для этой цели. В базе данных может создаваться любое число генераторов. Главное, чтобы они имели уникальные имена. Генератор, когда он определен, является глобальным в базе данных.

Но вернемся к вопросу о возникновении ошибки. Например, когда ваш клиент посылает запись на сервер, первичный ключ имеет значение NULL. Используя триггер, InterBase вставляет значение в первичный ключ и посылает запись. Когда BDE пытается проверить существование только что вставленной записи, он ищет запись с первичным ключом, имеющим значение NULL, которую он не может найти, и поэтому формирует сообщение об ошибке. Чтобы обойти эту ситуацию, надо выполнить следующие действия:

- Создать такой триггер: оператор if проверяет, вставлялся ли первичный ключ NULL. Если да, то сформировать значение, создаваемое генератором. Если нет, то ничего не делать. Вот код:

```
Create Trigger COUNTRY_INSERT for COUNTRY
active before Insert position 0
as
if (new->Pkey == NULL)
new->Pkey = gen_id(COUNTRY_GEN,1);
```

- Необходимо также создать хранимую процедуру, возвращающую значение, сформированное генератором:

```
Create Procedure COUNTRY_Pkey_Gen returns (avalue INTEGER)
as
avalue = gen_id(COUNTRY_GEN,10);
```

- Следует добавить компонент TStoredProc в ваше приложение и связать его с хранимой процедурой COUNTRY_Pkey_Gen.
- Надо добавить TQuery в ваше приложение и в обработчик события BeforePost вставить такой код:

```
If(TQuery->state == dsinsert)
{
StoredProc1->ExecProc;
TQuery->FieldByName("Pkey")-> AsInteger =
StoredProc1->ParamByName("avalue")->AsInteger;
}
```

Это решение позволяет клиенту извлекать сгенерированное значение с сервера с помощью компонента TStoredProc и хранимой процедуры. Это гарантирует, что C++ Builder-клиент будет знать величину первичного ключа при отсылке записи.

Глава 2

Задача управления кадрами

Краткая сущность задачи

Задача управления кадрами на предприятии, в учреждении и в любой другой функционирующей единице, в которой трудится персонал, является одной из первоочередных задач управления. Знаменитый лозунг тридцатых годов двадцатого века "Кадры решают все" не устарел и поныне. Предлагаемый проект имеет целью помочь начинающему программисту или программисту, осваивающему работу в среде Borland C++Builder, в приобретении навыков работы с базами данных, в частности, с системой управления базами данных (СУБД) Interbase. Использование этой СУБД поможет в дальнейшем легче осваивать другие СУБД, так как в них с Interbase есть много общего. Кроме этого, предлагаемая задача — один из примеров производственного применения среды Borland C++Builder.

Основные подзадачи этого комплекса следующие:

- создание сетевого варианта, чтобы базой данных могли пользоваться работники с разных рабочих мест;
- ведение картотеки служащих;
- составление различных запросов к картотеке;
- сохранение типовых запросов в базе данных;
- формирование и печать типовых отчетов — штатного расписания, реестра служащих, личного дела служащего и других;
- создание собственных отчетов по базе данных;
- ведение картотеки распоряжений по кадровому составу с возможностью хранения в базе данных;
- поддержка возможности пополнения комплекса новыми программными модулями.

Вся работа по проектированию задачи состоит из двух основных этапов: разработки структуры базы данных и разработки программного обеспечения задачи. Из всех задач мы рассмотрим наиболее интересную: создание картотеки персонала, так как картотека — это база для решения всех остальных задач.

Учетная карточка. Разработка структуры базы данных

Технология формирования

В основе всего комплекса лежит так называемая "Учетная карточка работника" (в дальнейшем — Карточка), которая заполняется и ведется работником кадровой службы на основе распоряжений и приказов по предприятию и документов, предоставляемых работником.

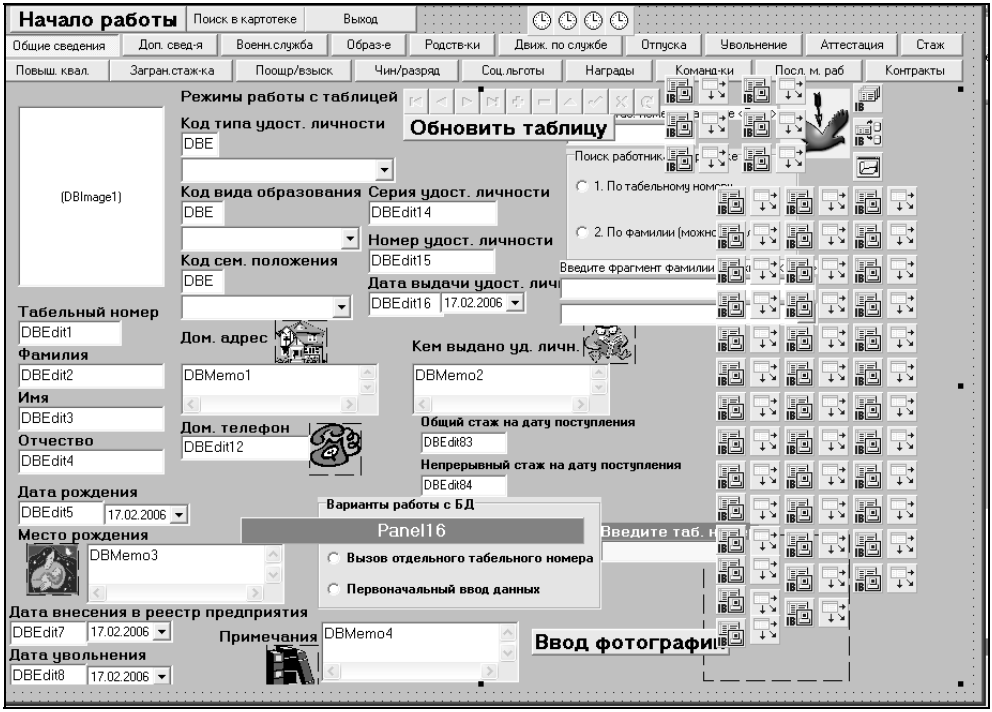


Рис. 2.1. Вид Карточки в режиме дизайна

Доступ к работе с картотекой должен осуществляться по специальному паролю, который открывает строку меню задачи "Кадры" для вызова этой подзадачи. Работа автоматизирована: учетная карточка заполняется на основе данных самой базы данных, в том числе и справочников. При работе с определенным разделом вызывается соответствующая таблица базы данных. Оператор (работник отдела кадров) может ее модифицировать. Технология получения Карточки по конкретному работнику такова: оператор вызывает на выполнение подзадачу "Картотека работников предприятия" и задает либо табельный номер, либо фамилию (можно не полностью) работника для поиска его карточки. В последнем случае ему выдается перечень фамилий с табельными номерами (подсказка), из которого он выбирает нужную строку и щелкает на ней мышью. Оператору выдается Карточка (ее раздел **Общие сведения** в режиме дизайна показан на рис. 2.1, а в режиме исполнения — на рис. 2.2), которую он может модифицировать, войдя в соответствующий раздел. Вкладки разделов помещены в верхней части Карточки.

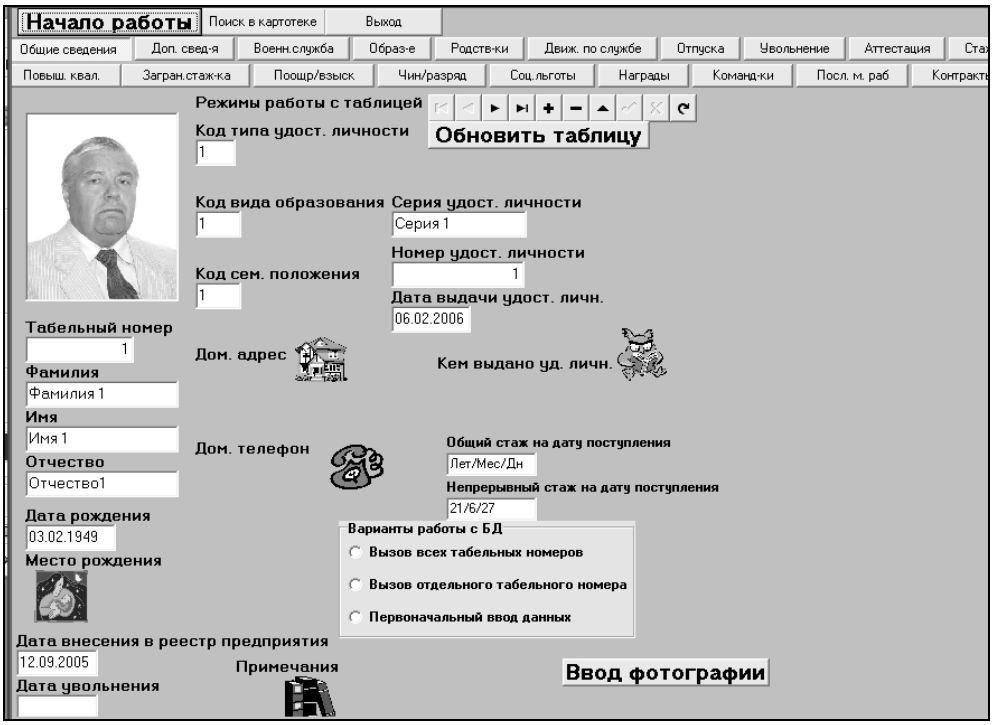


Рис. 2.2. Вид Карточки в режиме исполнения

Оператор входит в соответствующий раздел, щелкая на его вкладке левой кнопкой мыши, вызывая тем самым соответствующую таблицу, данные которой отображаются в полях страницы-раздела. Оператор может модифицировать таблицу, внося данные в соответствующие поля. Если есть необходимость ввести новую Карточку, оператор, пользуясь режимами работы с таблицей, добавляет к картотеке новую строку в разделе **Общие сведения** и начинает заполнять соответствующие поля в каждой вкладке, формируя тем самым новую Карточку. Заполнение Карточки начинается с вкладки **Общие сведения** — это отражение главной таблицы БД. С этой таблицей связаны все остальные таблицы, на основе которых формируется Карточка. Связь выполнена таким образом, что, двигаясь по записям главной таблицы, мы синхронно движемся по остальным таблицам. То есть, если на странице **Общие сведения** находятся данные по табельному номеру X, то во всех остальных вкладках Карточки отражены данные по этому же табельному номеру X. Такой подход позволяет поддерживать таблицы базы данных в актуальном состоянии. Поддержка таблиц базы данных осуществляется не сама по себе, а только через Карточку, что позволяет избежать ошибок рассогласования таблиц. Кроме того, предусмотрена целостность данных в базе данных с применением аппарата первичных и внешних ключей.

Структура Карточки

Карточка построена на основе применения компонента TPageControl, который позволяет создавать отдельные страницы для каждого раздела Карточки. Для создания Карточки поместим в форму компонент TPageControl и с помощью его контекстного меню (правая кнопка мыши) создадим необходимое количество страниц. Так как каждая страница отдельный объект со своим набором характеристик, часть из которых отражена в Инспекторе объекта, воспользуемся этой возможностью и дадим свои имена каждой странице, присвоив свойству **Caption** требуемое имя (рис. 2.3).

Описание реквизитов каждой страницы приводится далее. Отметим, что часть реквизитов (даты, тип удостоверения личности, сведения об образовании, семейное положение и др.) заполняются не непосредственным вводом в поле, а путем выборки из соответствующего списка. Это позволяет повысить достоверность ввода. Списки данных формируются на основе справочников, ведение которых организовано на соответствующих страницах (вкладках).

Приведем перечень справочников:

- профессии;
- учреждения, в которых получают образование;

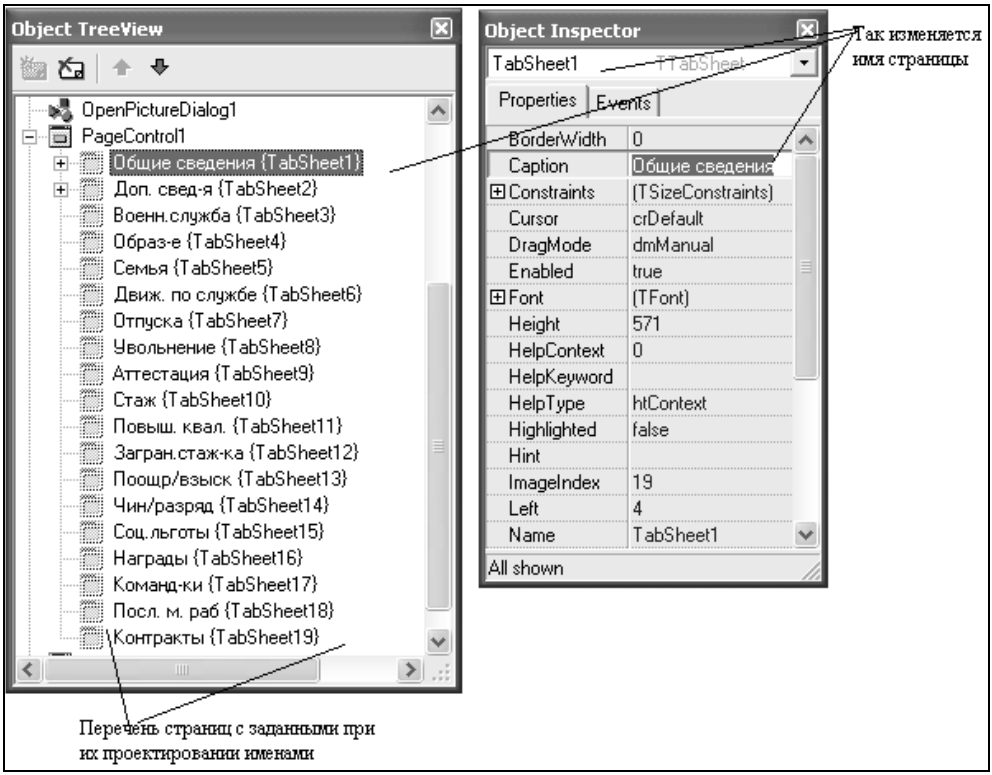


Рис. 2.3. Переименование страниц Карточки

- степени родства;
- структурные подразделения предприятия;
- праздничные и выходные дни;
- виды отпусков;
- виды результатов аттестаций;
- виды обучений;
- виды направлений подготовки;
- виды средств обучения;
- страны;
- виды поощрений и взысканий;
- виды разрядов и классных чинов;
- виды социальных льгот;

- виды причин увольнения;
- виды наград;
- виды результатов командировок;
- перечень мест командировок;
- перечень целей командировок;
- перечень должностей;
- перечень особых условий работы;
- виды процентных надбавок к зарплате;
- типы удостоверений личности;
- виды образования;
- типы обучения;
- типы образования;
- перечень специальностей.

Описание разделов учетной карточки

Для удобства пользования все реквизиты группируются в специальные разделы, наименования которых отражены во вкладках Карточки:

- **Общие сведения**;
- **Дополнительные данные**;
- **Военная служба**;
- **Образование**;
- **Родственники и иждивенцы**;
- **Прохождение службы на предприятии**;
- **Отпуска**;
- **Увольнение**;
- **Аттестация**;
- **Стаж работы**;
- **Повышение квалификации**;
- **Стажировка за границей**;
- **Взыскания/поощрения**;
- **Классные чины/квалификационные разряды**;
- **Социальные льготы**;
- **Награды**;

- **Командировки**;
- **Последнее место работы**;
- **Контракты**.

Раздел карточки *Общие сведения*

В этом разделе заполняются общие сведения о работнике. Поля, использующие данные из справочника, заполняются путем выбора значения из выпадающего списка. Если в списке нет соответствующего значения, необходимо заполнить справочник, на основе которого создан список. Ведение справочников для данного раздела организовано на вкладке **Дополнительные сведения** из-за отсутствия необходимого места в разделе **Общие сведения**. Для остальных разделов ведение справочников, используемых в этих разделах, организовано в тех же разделах. В раздел **Общие сведения** данные вносятся из таблицы, которая создается в базе данных под именем Osn_dannie и содержит поля:

- табельный номер работника;
- ФИО;
- дата рождения;
- место рождения;
- дата внесения в реестр;
- дата увольнения;
- вид образования;
- семейное положение;
- адрес места жительства;
- домашний телефон;
- тип удостоверения личности;
- серия удостоверения личности;
- номер удостоверения личности;
- дата выдачи удостоверения личности;
- кем выдано удостоверение личности;
- общий трудовой стаж на дату поступления на предприятие;
- непрерывный трудовой стаж на дату поступления на предприятие;
- фотография;
- примечания.

Раздел карточки *Дополнительные данные*

Раздел содержит дополнительные данные на служащего, например, адрес его электронной почты и т. п. Данные вносятся из таблицы, которая создается в базе данных под именем Dop_dannie и содержит поля:

- ❒ табельный номер работника;
- ❒ адрес электронной почты;
- ❒ сведения об оплате алиментов.

Раздел карточки *Военная служба*

Раздел содержит данные о воинской обязанности работника. Данные вносятся из таблицы, которая создается в базе данных под именем Voenn и содержит поля:

- ❒ табельный номер работника;
- ❒ номер военного билета;
- ❒ дата выдачи;
- ❒ кем выдан;
- ❒ звание;
- ❒ группа учета;
- ❒ категория учета;
- ❒ код военно-учетной специальности;
- ❒ перечень пройденных учебных сборов;
- ❒ перечень участия в войнах, конфликтах и других боевых действиях;
- ❒ перечень правительственных наград;
- ❒ перечень полученных ранений и контузий;
- ❒ наличие мобпредписания.

Раздел карточки *Образование*

Раздел содержит таблицу с образовательными учреждениями, в которых обучался работник, и таблицу профессий. Образовательные данные вносятся из таблицы, которая создается в базе данных под именем Obraz и содержит поля:

- ❒ табельный номер работника;
- ❒ код образовательного учреждения (в БД есть таблица ObrazUchrejd);
- ❒ дата поступления;

- дата окончания;
- полученная специальность;
- тип образования (довузовское, вуз, послевузовское);
- профильное/непрофильное образование (необходимо для выделения профильного образования при наличии у работника нескольких высших образований);
- код профессии (выбирается из таблицы профессий);
- дата получения профессии;
- номер подтверждающего документа;
- дата подтверждающего документа;
- кем выдан документ.

В таблицу профессий данные вносятся из таблицы, которая создается в базе данных под именем Profess и содержит поля:

- код профессии (для данного предприятия в соответствии с множеством применяемых на нем профессий составляется подмножество из Общероссийского классификатора профессий рабочих, должностей служащих и тарифных разрядов (ОКПДТР), являющегося составной частью Единой системы классификации и кодирования информации (ЕСКК) Российской Федерации);
- название профессии.

Список образовательных учреждений находится в БД в виде таблицы с именем ObrazUchrejd. Структура таблицы такова:

- код учреждения;
- наименование учреждения;
- адрес места расположения учреждения.

Раздел карточки *Родственники и иждивенцы*

Раздел содержит данные о членах семьи. Данные вносятся из таблицы, которая создается в базе данных под именем Rodstvo и содержит поля:

- табельный номер работника;
- фамилия, имя, отчество родственника;
- год рождения;
- степень родства;
- адрес проживания.

Раздел карточки *Прохождение службы на предприятии*

В раздел заносятся данные о прохождении службы в данном учреждении (предприятии). Данные вносятся из таблицы, которая создается в базе данных под именем Slujba и содержит поля:

- ❒ табельный номер работника;
- ❒ код структурного подразделения;
- ❒ номер телефона;
- ❒ дата поступления в подразделение;
- ❒ дата и номер распоряжения о поступлении;
- ❒ должность;
- ❒ номер контракта;
- ❒ дата убытия из подразделения;
- ❒ дата и номер распоряжения об убытии;
- ❒ причина убытия.

Раздел карточки *Отпуска*

В раздел заносятся данные об отпусках. При заполнении полей начала отпуска и количества дней автоматически рассчитывается дата окончания отпуска с учетом праздничных дней из справочника "Праздничные дни". Данные вносятся из таблицы, которая создается в базе данных под именем Otpuska и содержит поля:

- ❒ табельный номер работника;
- ❒ вид отпуска;
- ❒ количество дней;
- ❒ дата начала отпуска;
- ❒ дата окончания отпуска;
- ❒ заработная плата, начисленная за отпуск.

Раздел карточки *Увольнение*

В раздел заносятся данные об увольнении. После заполнения даты увольнения работник получает статус "Уволенный" с этой даты (у него становится непустым поле **Дата увольнения**). Можно восстановить служащего в картотеке, удалив дату об увольнении. Данные вносятся из таблицы, которая создается в базе данных под именем Uvoln и содержит поля:

- ❒ табельный номер работника;
- ❒ дата увольнения;

- ❒ причина;
- ❒ номер и дата приказа об увольнении.

Раздел карточки *Аттестация*

В раздел заносятся данные о проведенных и планируемых аттестациях. Данные вносятся из таблицы, которая создается в базе данных под именем Atest и содержит поля:

- ❒ табельный номер работника;
- ❒ дата аттестации (план);
- ❒ дата аттестации (факт);
- ❒ код результата аттестации.

Раздел карточки *Стаж*

В раздел заносятся данные для расчета стажа служащего. Данные автоматически рассчитываются при входе в данную вкладку. Раздел содержит поля:

- ❒ табельный номер работника;
- ❒ общий стаж (лет/месяцев/дней);
- ❒ непрерывный стаж (лет/месяцев/дней);
- ❒ стаж работы на данном предприятии (лет/месяцев/дней);
- ❒ возраст работника.

Раздел карточки *Повышение квалификации/Профессиональная переподготовка*

В раздел вносятся данные из таблицы, которая создается в базе данных под именем PovKval и содержит поля:

- ❒ табельный номер работника;
- ❒ тип обучения (повышение квалификации или переподготовка);
- ❒ дата начала обучения;
- ❒ дата окончания обучения;
- ❒ вид обучения;
- ❒ название учебного заведения;
- ❒ направление подготовки (для отчетных форм Госкомстата 1-ГС, 1-МС);
- ❒ средства обучения;
- ❒ полученная профессия.

Раздел карточки *Стажировка за границей*

В этот раздел заносятся данные о стажировке работника в иностранных учебных заведениях. Данные вносятся из таблицы, которая создается в базе данных под именем Zagran и содержит поля:

- ❒ табельный номер работника;
- ❒ дата начала стажировки;
- ❒ дата окончания стажировки;
- ❒ название страны;
- ❒ название учебного заведения;
- ❒ направление стажировки;
- ❒ средства обучения.

Раздел карточки *Взыскания/Поощрения*

В этот раздел заносятся данные из таблицы, которая создается в базе данных под именем VziscPooshr и содержит поля:

- ❒ табельный номер работника;
- ❒ код действия по отношению к работнику;
- ❒ дата и номер директивного документа.

Раздел карточки *Классные чины/Квалификационные разряды*

Раздел служит для ведения классных чинов или квалификационных разрядов работника. Данные вносятся из таблицы, которая создается в базе данных под именем ClasChin и содержит поля:

- ❒ табельный номер работника;
- ❒ название классного чина или квалификационного разряда;
- ❒ период действия (количество лет);
- ❒ дата присвоения;
- ❒ кем издан директивный документ о присвоении;
- ❒ дата и номер директивного документа.

Раздел карточки *Социальные льготы*

В раздел заносятся социальные льготы, которые предоставлены работнику. Данные вносятся из таблицы, которая создается в базе данных под именем SotsLgoti и содержит поля:

- ❒ табельный номер работника;

- ❒ наименование социальной льготы;
- ❒ сумма (или процент) выплаты.

Раздел карточки *Награды*

В раздел заносится список наград. Данные вносятся из таблицы, которая создается в базе данных под именем Nagradi и содержит поля:

- ❒ табельный номер работника;
- ❒ наименование награды;
- ❒ дата присвоения награды;
- ❒ кем присвоена;
- ❒ дата и номер директивного документа.

Раздел карточки *Командировки*

В раздел заносится список служебных командировок. Данные вносятся из таблицы, которая создается в базе данных под именем Komandirovki и содержит поля:

- ❒ табельный номер работника;
- ❒ дата начала командировки;
- ❒ дата окончания командировки;
- ❒ количество дней командировки;
- ❒ цель командировки;
- ❒ наименование места командировки;
- ❒ результат командировки;
- ❒ номер директивного документа;
- ❒ дата директивного документа.

Раздел карточки *Последнее место работы*

В раздел заносятся данные о последнем месте работы. Данные вносятся из таблицы, которая создается в базе данных под именем PoslMestoRab и содержит поля:

- ❒ табельный номер работника;
- ❒ наименование последнего места работы;
- ❒ должность;
- ❒ заработная плата;

- дата увольнения;
- причина увольнения.

Раздел карточки *Контракты*

В раздел заносятся данные о всех контрактах, заключенных с работником на данном предприятии. Сведения вносятся из таблицы, которая создается в базе данных под именем Contracti и содержит поля:

- табельный номер;
- номер контракта;
- дата начала действия контракта;
- дата окончания действия контракта;
- оклад (тарифная ставка);
- код процентной надбавки;
- код вида отпуска по контракту;
- код особых условий.

Создание таблиц

Прежде чем формировать таблицы, необходимо создать на сервере саму базу данных, зарегистрировать ее и соединиться с ней, т. е. получить к ней доступ, иначе просто некуда будет писать создаваемые таблицы. Об этом говорилось в *главе 1*. Команды создания БД на SQL таковы:

```
CREATE DATABASE "d:\Interbase\Databases\Persons.gdb"
USER "sysdba" PASSWORD "masterkey";
```

Эти команды надо вставить в окно интерактивного SQL утилиты IBConsole и выполнить операторы, нажав кнопку с пиктограммой  . Для этого запустим утилиту, зарегистрируем сервер (Server/Login), ответив на запрос пароля БД вводом в поле пароля значения masterkey. При этом сервер запустится, и мы перейдем в окно интерактивного SQL, нажав кнопку  , и введем в окно команды, указанные выше.

Таблицы базы данных создаются тоже с помощью утилиты IBConsole и тоже в режиме использования интерактивного SQL.

Перед созданием таблиц надо подключиться к созданной базе данных (чтобы в ней создавать таблицы). Для этого следует щелкнуть на строке с именем нужной БД (выделить) и щелкнуть кнопку  на панели IBConsole. При этом получим результат, показанный на рис. 2.4.

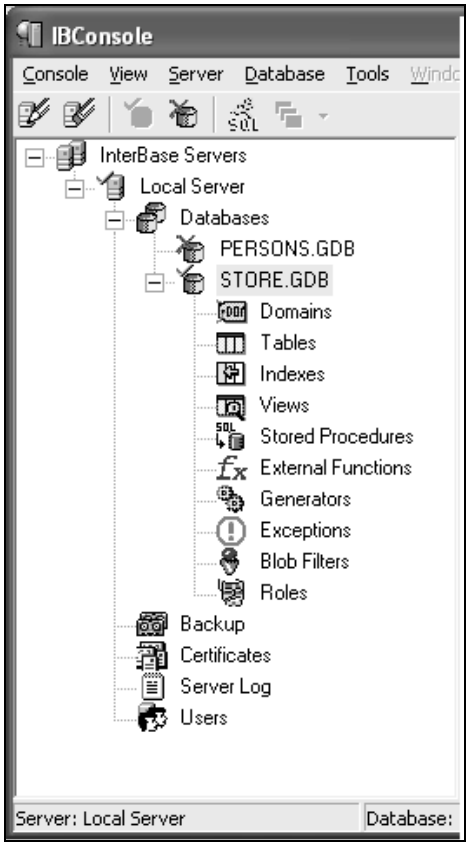


Рис. 2.4. Подключение к базе данных Store

После подключения высвечиваются входы в составные части подключенной базы данных: в домены, таблицы, индексы, вьюеры, хранимые процедуры и т. д. Если щелкнуть пиктограмму Tables, то получим список только системных таблиц базы данных, к которой мы подключены на сервере: своих таблиц мы еще не сформировали (рис. 2.5).

Файл SQL-операторов создания таблиц

Напишем на SQL программы создания таблиц для рассмотренных уже справочников и входных-выходных документов и поместим их в файл с помощью редактора Блокнот, откуда их удобно вставлять в окно IBConsole. Программы создания таблиц приведены в листинге 2.1.

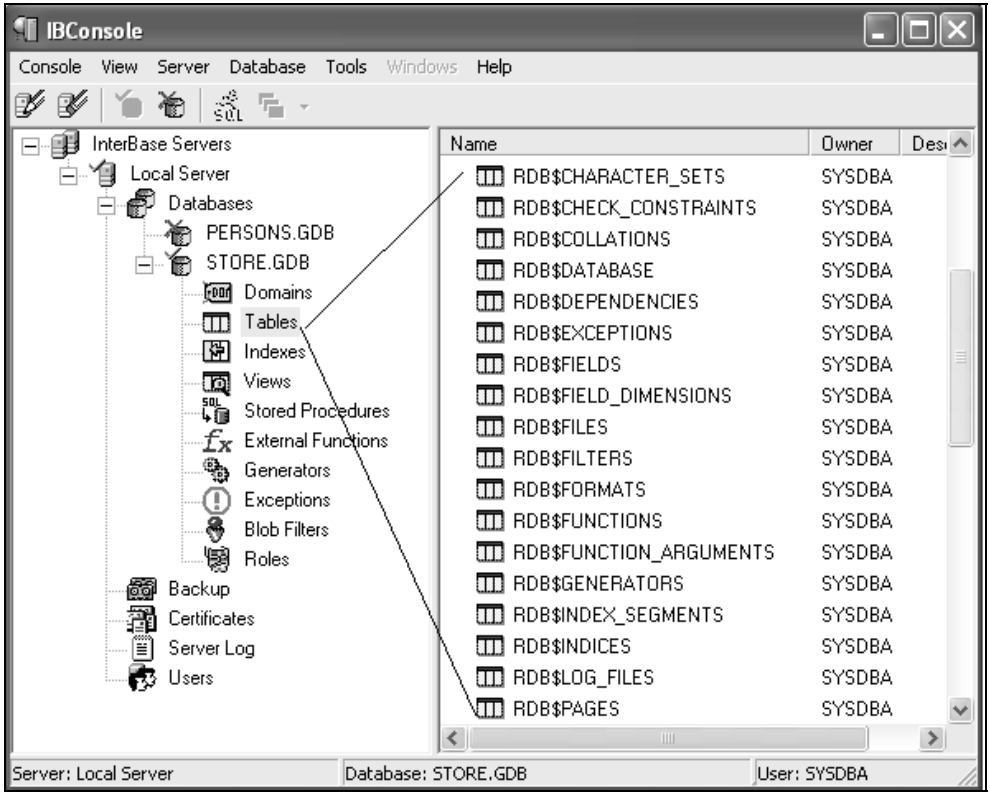


Рис. 2.5. Список системных таблиц базы данных Persons

Листинг 2.1

```
Create Table osn_dannie /*создание таблицы для основных данных карточки
*/
 (
       tn float not null,
       familia varchar(25) not null,
       imea varchar(15) not null,
       otchestvo varchar(15) not null,
       data_rojd date not null,
       mesto_rojd varchar(100) not null,
       data_v_reestre date not null,
       data_uvoln date default null,
```

```
        vid_obraz char(15) not null,
        sem_poloj char(15) not null,
        dom_adres varchar(100) not null,
        dom_telefon varchar(15),
        tip_ud_lichn varchar(30) not null,
        seria_ud_lichn varchar(15) not null,
        nom_ud_lichn float not null,
        data_vidachi_ud_lichn date not null,
        kem_vid_ud_lichn varchar(50) not null,
        foto blob,
                obshstajnadatupost varchar(10),
                neprstajnadatupost varchar(10),
        primechania varchar(200),
         primary key (tn)
   );


/*Изменение полей таблицы Osn_dannie*/
Alter Table Osn_Dannie
drop Vid_Obraz, /*удаление двух полей и вставка вместо них двух других*/
drop Sem_Poloj,
add CodVidaObraz integer,
add CodSemPoloj integer;
/*Изменение полей таблицы Osn_dannie*/

Create Table dop_dannie

/*при удалении первичного ключа
таблицы osn_dannie автоматически будет удалена строка с таким же ключом
этой таблицы*/

(
        tn float not null,
        adr_e_mail varchar(25),
        alimenti varchar(100),
```

```
       primary key (tn),
        foreign key (tn) references osn_dannie (tn)
       on delete cascade
);

/*В таблицах, в которых отсутствуют PRIMARY KEY (TN),
FOREIGN KEY (TN)..., может быть более одной строки для данного табельного
номера*/
Create Table Voenn
 (
   Tn float Not Null,
   NBileta VarChar(15) Not Null,
   DataVidachi Date Not Null,
   KemVidan  VarChar(50) Not Null,
   Zvanie VarChar(30),
   GrUch  VarChar(10),
   KatUch VarChar(10),
   CodVUS VarChar(10),
   UchSbori VarChar(300),
   UchVVoinah VarChar(300),
   Nagradi VarChar(500),
   Ranenia VarChar(500),
   MobPredp char(1),
   Primary Key(Tn),
   Foreign Key(Tn) References Osn_Dannie(Tn)
      On Delete Cascade
);

Create Table Obraz
 (
   Tn float Not Null,
   CodObrZaved Integer Not Null,
   DataPostup Date Not Null,
   DataOconch Date,
   CodSpecialn integer Not Null,
   CodTipaObraz integer,
```

```
   Profilnoe Char(1),  /*да-нет*/
   CodProfes integer,
   DataPoluchProf Date,
   NomPodtvDoc VarChar(15),
   DataPodtvDoc Date,
   KemVidanDoc VarChar(75)
);

Create Table Specialnosti
 (
   Cod Integer Not Null,
  Naim VarChar(50)
  );

Create Table Professii
 (
   Cod Integer Not Null,
   NazvProf VarChar(75)
  );

Create Table ObrazUchrejd
 (
   Cod Integer Not Null,
   Nazv VarChar(75),
   AdresMestaRasp VarChar(75)
 );

Create Table Rodstvo
 (
  Tn float Not Null,
   Fio VarChar(75),
   GodRojd  Date,
   CodStepeniRodstva integer,
   Adres VarChar(75)
    );
```

```
Create Table Slujba
 (
  Tn float Not Null,
   CodPodr integer,
   Telefon   VarChar(15),
   DataPostup Date,
   NomPric VarChar(15),
   DataPric Date,
   CodDolj integer,
   NomContr VarChar(15),
   DataUbit Date,
   NomRasp VarChar(15),
   DataRasp Date,
   CodPrich integer
);

Create Table Podrazd
 (
  Cod Integer Not Null,
   Nazv VarChar(75),
   FIOShefa VarChar(50),
   Tel VarChar(15)
    );

Create Table Otpusca
 (
  Tn float Not Null,
  CodVidaOtp integer,
   KolDnei integer,
   DataNach Date,
   DataOconch Date,
   NomPricaza VarChar(15),
   DataPricaza Date,
   ZarplataZaOtp float
  );
```

```
Create Table VidiOtpuscov
 (
  CodVidaOtp integer,
  NainVidaOtp VarChar(30),
  Norma integer
);

Create Table PrazdDni
 (
  NomMes integer,
  NomeraPrazdDnei VarChar(30),
  MaxDeni integer
);

Create Table Uvoln
 (
  Tn float Not Null,
   DataUvoln Date,
   NomPricaza VarChar(15),
   DataPricaza Date,
   CodPrichini integer,
  Primary Key(Tn),
   Foreign Key(Tn) References Osn_Dannie(Tn)
      On Delete Cascade
  );

Create Table VidiPrichinUvoln
 (
  CodPrichini integer,
  NaimPrichini VarChar(75)
);

Create Table Atest
 (
  Tn float Not Null,
```

```
   DataAtPlan Date,
   DataAtFact Date,
   CodRezult Integer,
   Primary Key(Tn)
  );

Create Table ResultAtest
 (
  Cod integer,
  NaimResult VarChar(150)
);

Create Table PovKval
 (
  Tn float Not Null,
  CodTipaObuch integer,
   DataNachOb Date,
   DataOkonchOb Date,
   CodVidaObuch integer,
   CodUchZav integer,
   CodNapravlPodg integer,
   CodSredstvaObuch integer,
   CodPoluchProfes integer
  );

 Create Table TipiObuch
 (
  Cod integer,
  Naim VarChar(50)
);


Create Table VidiObuch
 (
  Cod integer,
```

```
  NaimVidaObuch VarChar(75)
);

Create Table NapravlPodg
 (
  Cod integer,
  NaimNapr VarChar(75)
);

Create Table SredsvaObuch
 (
  Cod integer,
  NaimSredstva VarChar(75)
);

Create Table Zagran
 (
  Tn float Not Null,
  DataNachSt Date,
   DataOkonchSt Date,
   CodStrani integer,
   CodUchZav integer,
   CodNapravlSt integer,
   CodSredstvaObuch integer
  );

Create Table Strani
 (
  Cod integer,
  NaimStrani VarChar(50)
);

Create Table VziscPooshr
 (
  Tn float Not Null,
```

```
  NomDocVziscPoosh VarChar(15),
  DataDocVziscPoosh Date,
  CodVziscPoosh integer
  );

Create Table SpisVzPooshr
 (
  CodVziscPoosh  integer,
  NaimVziscPoosh  VarChar(50)
);

Create Table ClasChin
 (
  Tn float Not Null,
  CodChina integer,
  PeriodDeistvia integer,
  DataPrisvoenia Date,
  NomDirDoc VarChar(15),
  DataDirDoc Date
  );

Create Table Cini
 (
  CodChina integer,
  NaimChina VarChar(50)
);

Create Table  VidiSotsLgot
 (
  CodVida integer,
  NaimVida VarChar(50)
);

Create Table  SotsLgoti
 (
  Tn float Not Null,
```

```
  CodSotsLgoti integer,
 ProcVipl float,
 SummaVipl float
);

Create Table Nagradi
 (
  Tn float Not Null,
  CodNagr integer,
  DataPrisv Date,
  KemPrisvoena VarChar(45),
  NomDirDoc VarChar(15),
  DataDirDoc Date
 );

Create Table VidiNagrad
 (
  CodNagr  integer,
  NaimNagr  VarChar(50)
);



Create Table Komandirovki
 (
  Tn float Not Null,
  DataNachKom Date,
   KolDnei integer,
   Celi VarChar(75),
   CodMesta integer,
   CodResultata integer,
   NomDirDoc VarChar(15),
   DataDirDoc Date
  );

Create Table MestaKomandirovok
 (
```

```
  CodMesta  integer,
  NaimMesta  VarChar(75)
);

Create Table ResultKomandirovok
 (
  CodResult integer,
  NaimResult  VarChar(75)
);

Create Table CeliKomandirovok
 (
  CodCeli integer,
  NaimCeli VarChar(75)
);

Create Table PoslMestoRab
 (
  Tn float Not Null,
   NaimPredpr VarChar(75),
   CodBivshDoljn integer,
   Zarplata float,
   DataUvolnSPredMesta Date,
   CodPrichiniUvoln integer,
Primary Key(Tn),
   Foreign Key(Tn) References Osn_Dannie(Tn)
      On Delete Cascade
  );

Create Table ProcNadbavki
 (
  CodProcNadbavki integer,
  NaimProcNadb VarChar(75),
  ProcNadbavki float,
  SummaProcNadb float
);
```

```
Create Table OsobieUslovia
 (
  CodOsUsl  integer,
  NaimOsUsl VarChar(50)
);

Create Table Contracti
 (
  Tn float Not Null,
  NomContracta VarChar(15),
   DataNach Date,
   DataOconch Date,
   Oclad float,
   CodProcNadbavki integer,
   CodVidaOtp integer,
   CodOsobUsl integer
  );

Create Table Doljnosti
 (
  Cod integer,
  Naim VarChar(50)
);

Create Table TipiUdostLichnosti
 (
  Cod integer,
  Naim VarChar(35)
);

Create Table VidiOrazovania
 (
  Cod integer,
  Naim VarChar(50)
);
```

```
Create Table SemPoloj
  (
   Cod integer,
   Naim VarChar(50)
);

Create Table TipiObrazovania
  (
   Cod integer,
   Naim VarChar(25)
);
```

Шаги использования утилиты IBConsole для создания таблиц в базе данных показаны на рис. 2.6.

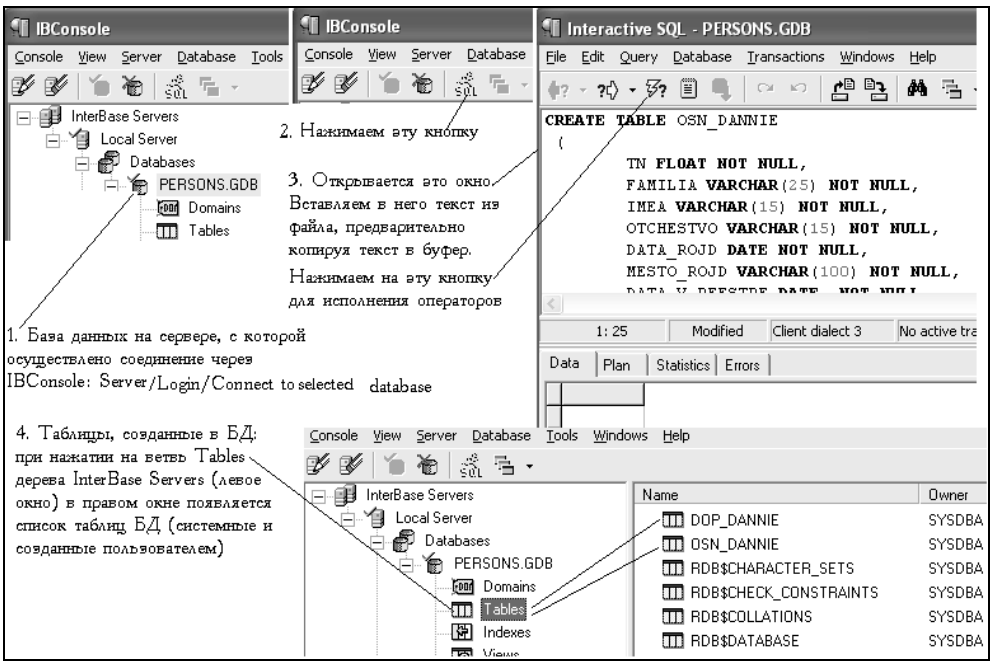


Рис. 2.6. Ход задания таблиц в базе данных

Формирование связей между таблицами Карточки

Все таблицы базы данных, сведения из которых поступают в Карточку, привязаны к главной таблице Osn_dannie по табельному номеру: значение этого поля у всех таблиц одинаково. Поэтому когда во вкладке **Общие сведения** выбирается Карточка с конкретным табельным номером, то при входе в любую другую вкладку на ней высвечиваются сведения по этому же табельному номеру. Таблицы, в которых не может быть строк с одинаковыми табельными номерами, связаны с главной таблицей по табельному номеру с помощью ключей primary key и foreign key. Это значит, что ни одна строка в таких таблицах не может быть удалена "без ведома" главной таблицы. Таблицы же, которые допускают наличие строк с повторяющимися табельными номерами (например, у одного работника может быть несколько социальных льгот), упомянутой связи не имеют.

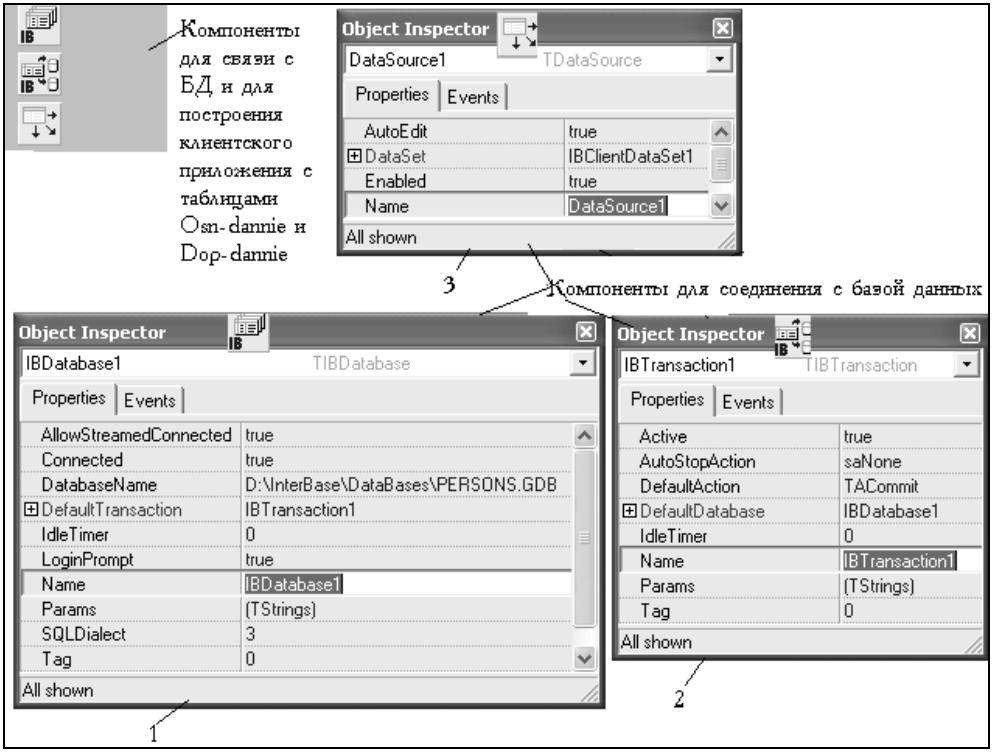


Рис. 2.7. Компоненты соединения клиента с БД

Для работы с каждой таблицей применяется компонент TIBClientDataSet, который позволяет строить клиентское приложение: сама база данных располагается на сервере.

Но в самом начале следует определить связь консольного приложения с базой данных, чтобы затем уже работать с таблицами. Связь с базой данных осуществляется с помощью Interbase-компонентов TIBDatabase, TIBTransaction, TDataSource. В TIBDatabase в свойстве DatabaseName определяется путь к базе данных и в свойстве DefaultTransaction выбирается компонент IBTRansaction1, через который станет осуществляться обмен данными клиента с БД. TDataSource участвует в работе с БД косвенно: он соединяет компонент TIBClientDataSet с TIBDatabase. Инспекторы объекта с определенными в них свойствами компонентов приведены на рис. 2.7.

Теперь рассмотрим, как применяется компонент TIBClientDataSet для связи с таблицей БД. Эта связь показана на рис. 2.8.

Пояснения приведены в рисунке.

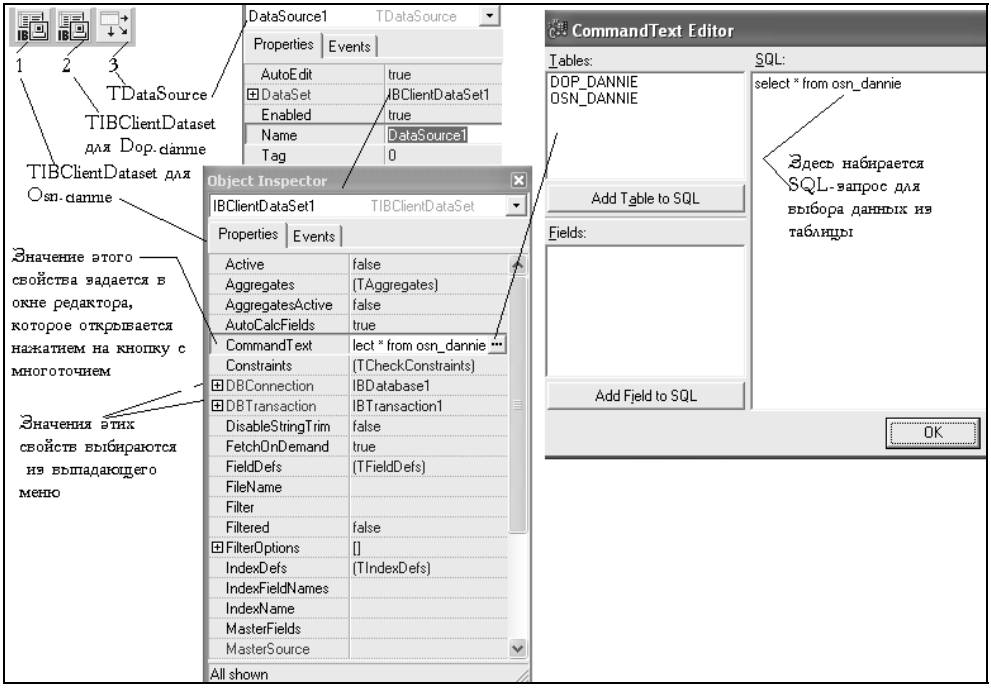


Рис. 2.8. Настройка TIBClientDataSet на таблицу БД

Ввод данных

Ввод изображения (фотографии)

Изображение вводится через кнопку **TButton** (Ввод фотографии) в ее обработчике с применением компонента TOpenPictureDialog, который позволяет выбрать необходимое изображение из соответствующей папки, где оно хранится. Затем изображение вставляется в компонент TImage, из которого затем копируется в буфер, а уже из буфера вставляется в компонент TDBImage, связанный с таблицей.

Ввод остальных полей данных

Ввод полей данных, не являющихся изображениями, выполняется с использованием элементов TDBText для отображения небольших текстов и настроенных на соответствующие поля соответствующих таблиц. Там, где это возможно, вместо обычного ввода данных в поле применяется выборка из элементов ComboBox и DateTimePicker (для задания дат). Объемные текстовые данные вводятся в элементы DBMemo.

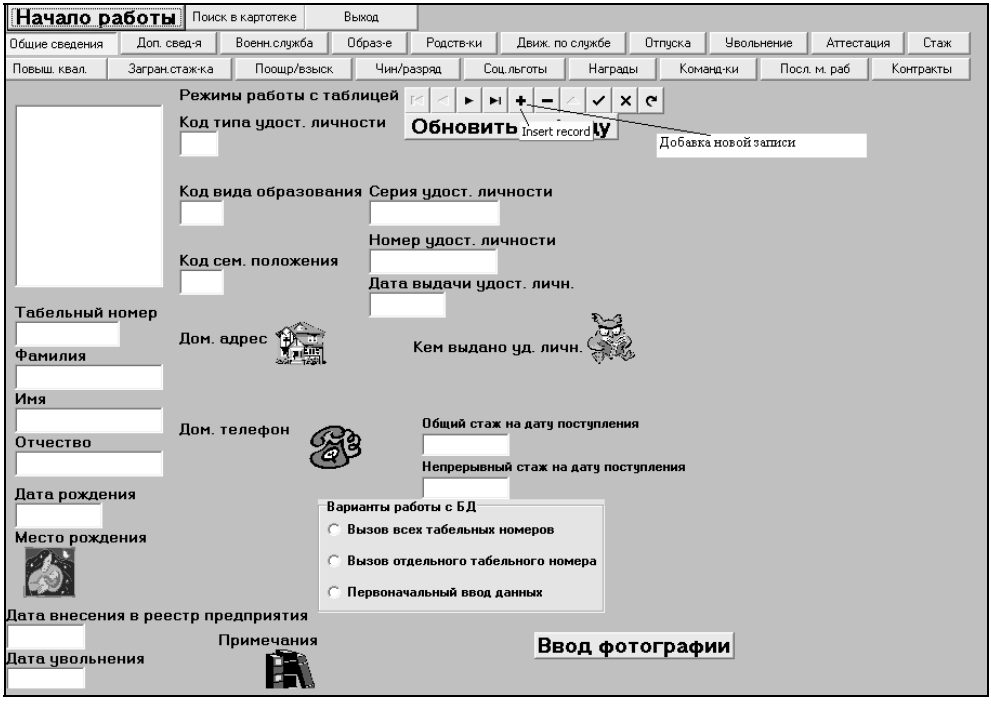


Рис. 2.9. Применение TDBNavigator

Ввод идет с применением компонента TDBNavigator, который обеспечивает добавление пустых записей для ввода новых данных, установку режима редактирования полей одной записи и т. п. (рис. 2.9).

Ввод осуществляется в заданном порядке, определенном в программе путем автоматической передачи фокуса ввода соответствующему полю ввода, начиная с поля, которое следует за табельным номером. Для ввода в поля, которые предусматривают значительный объем ввода, предусмотрено использование значков: надо щелкнуть на значке, в результате чего появится memo-поле, в которое и можно вводить текст. Если после окончания ввода снова щелкнуть на значке, memo-поле исчезнет. При вводе в поля, для которых предусмотрена выборка данных из справочника, при щелчке в их поле (при передаче полю фокуса ввода) под полем появляется элемент выпадающего меню (его реализует компонент TComboBox), на кнопке которого следует щелкнуть для того, чтобы раскрылся список. Из этого списка надо выбрать подходящее значение и щелкнуть на нем. При этом список закроется, сам элемент станет невидимым, а выбранное значение попадет в поле (рис. 2.10).

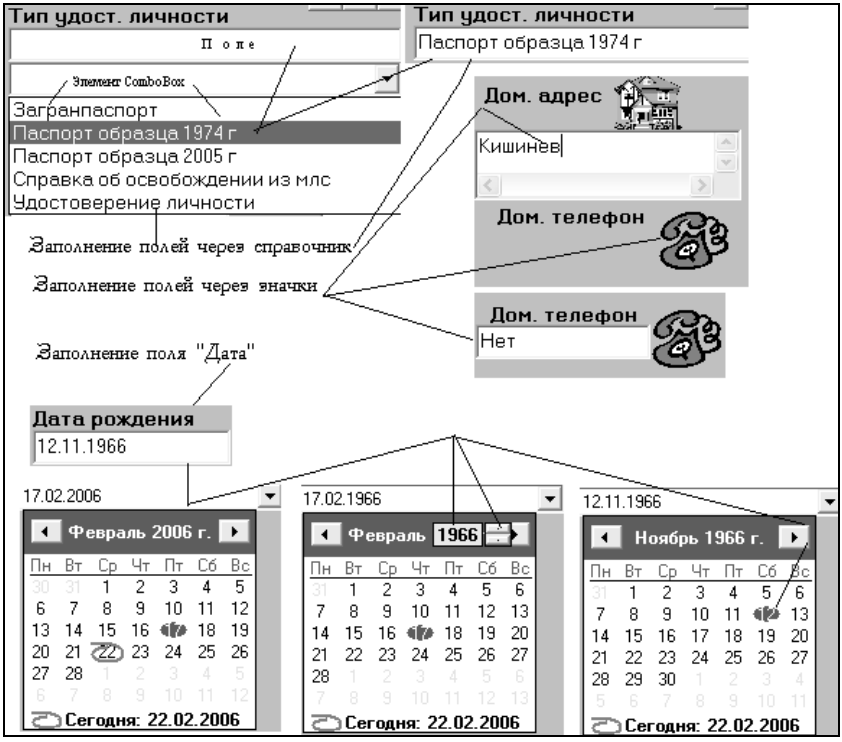


Рис. 2.10. Заполнение полей с помощью применения значков, справочников и календарей

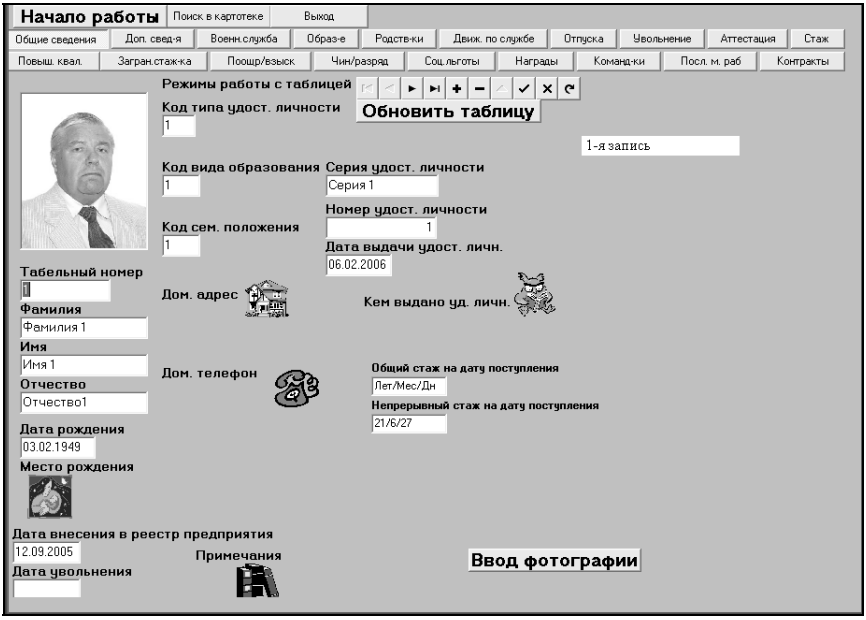


Рис. 2.11. Раздел **Общие сведения**. Запись 1

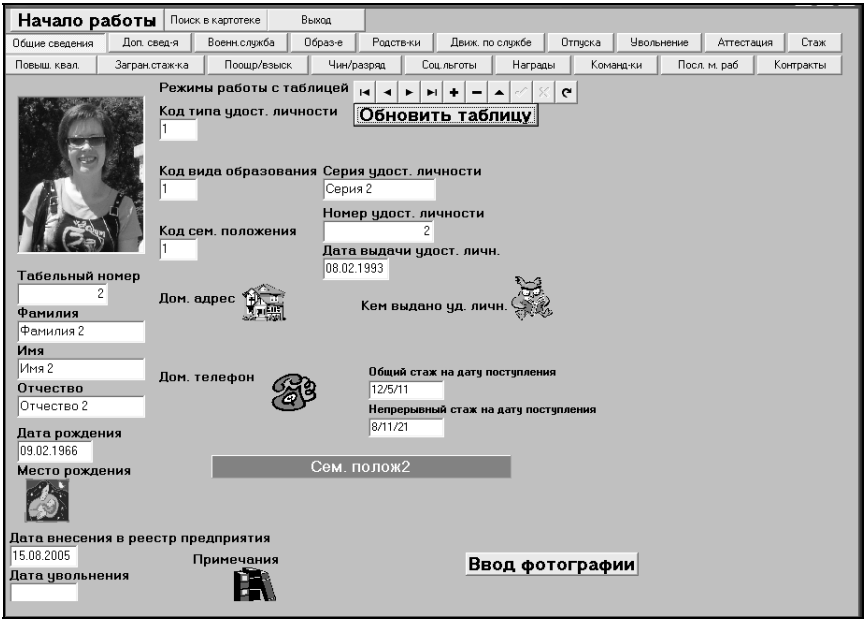


Рис. 2.12. Раздел **Общие сведения**. Запись 2

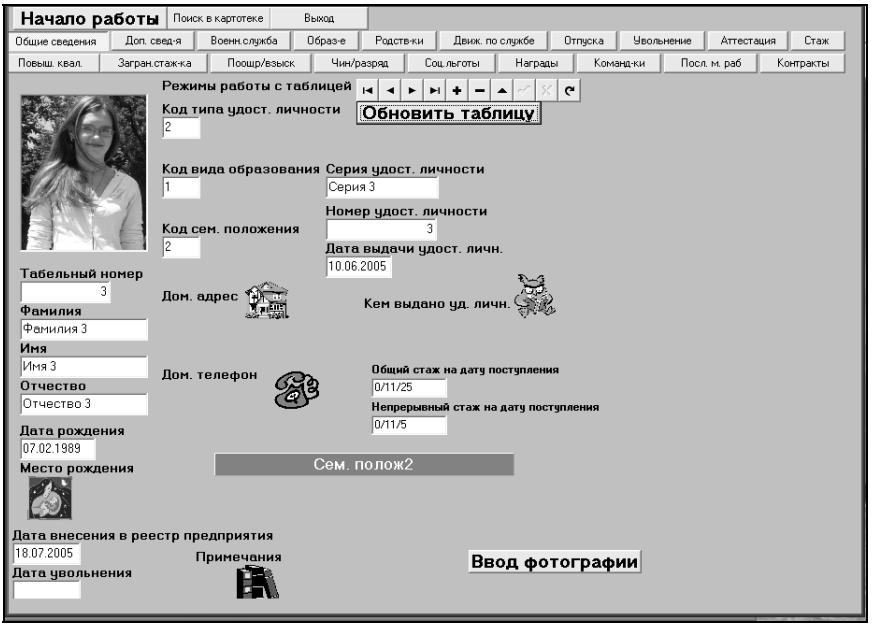


Рис. 2.13. Раздел **Общие сведения**. Запись 3

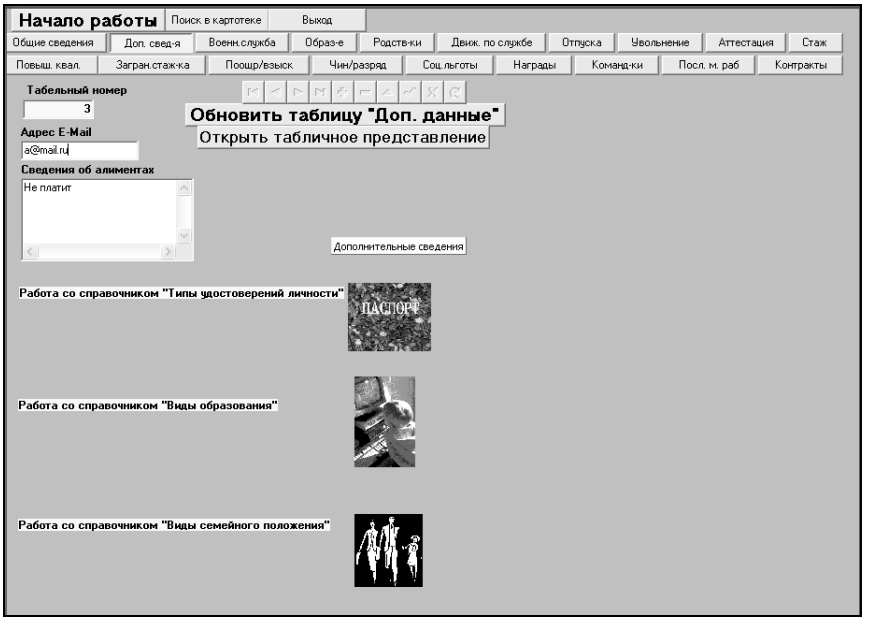


Рис. 2.14. Раздел **Дополнительные сведения**. Запись 3

На рис 2.11—2.13 приведены три записи, введенные в Карточку для раздела **Общие сведения**, и одна запись из раздела **Дополнительные сведения**, которые помогут в дальнейшем продемонстрировать режим поиска по картотеке.

Обновление базы данных

Когда при работе с картотекой имела место модификация данных, базу данных (т. е. таблицы, претерпевшие изменения) следует обновить, так как все произведенные изменения физически не попали в БД, а находятся на клиентском компьютере. Чтобы переслать измененные данные в БД, применяется метод ApplyUpdates() компонента TIBClientDataSet. Метод применяется во всех компонентах TIBClientDataSet, которые связаны с соответствующими таблицами. Например, в случае, когда рассматривались только таблицы Osn_dannie и Dop_dannie, выполнению подлежат операторы:

```
IBClientDataSet1->ApplyUpdates(-1); // обновление таблицы Osn_dannie
IBClientDataSet2->ApplyUpdates(-1); // обновление таблицы Dop_dannie
```

Организация поиска в картотеке

Поиск в картотеке организован по двум направлениям: если известен табельный номер работника, то — по табельному номеру, а если табельный номер не известен, то по фамилии или по ее фрагменту. При этом после набора фрагмента в выпадающем списке получаются фамилии с табельными номерами по всем записям, в фамилии которых входит введенный фрагмент. После этого остается выбрать из списка нужную фамилию и щелкнуть на ней мышью. Шаги оператора по поиску показаны на рис. 2.15.

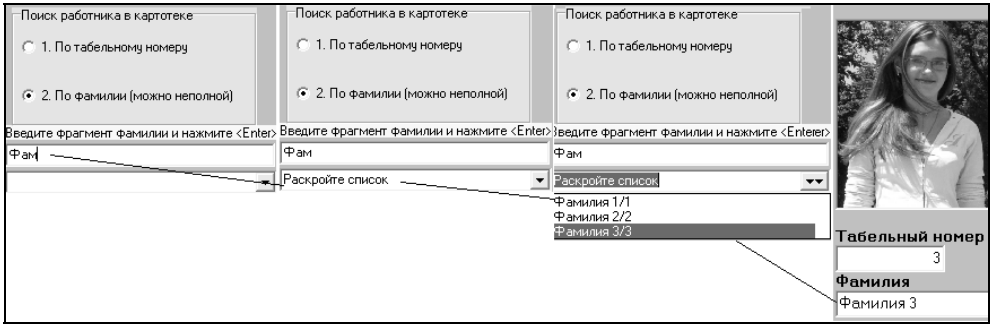


Рис. 2.15. Поиск по фрагменту фамилии

Об организации обмена данными между клиентом и сервером

Итак, данные для обработки мы получаем из базы данных через компоненты TIBClientDataSet с помощью запроса, формируемого в окне редактора запросов и попадающего после этого в свойство CommandText этих компонентов (см. рис. 2.7, 2.8). Как следует из рисунков, из базы данных выбираются все строки из таблиц. Эти строки должны пройти по сети между клиентом и сервером и попасть на сохранение в буфер клиента. Если таблицы по объему невелики, то проблем с обменом данными между сервером и клиентами не возникает. Проблемы появляются, когда таблицы достаточно объемны, и перегонять огромные потоки информации по сети становится накладно: может случиться ситуация, когда время получения данных от сервера и отсылки обновленных данных серверу столь велико, что сама задача потеряет смысл. С другой стороны, следует иметь в виду, что если не выбирать всю таблицу с сервера, а только необходимую запись по конкретному работнику и ее же отсылать назад (при модификации, естественно), то для каждой записи надо будет "бегать" по сети, на что уходит определенное время. Если предстоит модифицировать много записей, то совокупное время на движение "туда-сюда" по пути "клиент-сервер" тоже может стать неприемлемым: легче подождать, пока прочитаются все таблицы полностью, а их корректировка займет мало времени, так как она будет происходить в памяти. Обновление же базы данных опять займет большое время, но это может происходить уже без участия оператора. Какой же вывод? Напрашивается разработка обоих вариантов, чтобы дать пользователю возможность выбора в зависимости от перечисленных условий. Мы еще не отметили, что сервер может одновременно работать со многими клиентами. Пока что мы рассмотрели ситуацию, когда оператору-пользователю не было необходимости получать так называемые агрегированные данные: сводные таблицы, различного рода расчетные данные и т. п. Для иллюстрации возьмем крайний случай: пусть руководству потребовалась общая численность работников предприятия и в каком-нибудь разрезе. Допустим, что работников на предприятии — два миллиона (например, это какое-то министерство). Неужели надо будет пересылать данные по двум миллионам работников на клиентский компьютер, чтобы получить несколько итоговых строк? Сколько времени займет одна эта пересылка! Ясно, что такая организация работы нежелательна. Выход из положения состоит в применении аппарата хранимых процедур, которые организуются в базе данных как ее элементы и которые обеспечивают необходимые расчеты прямо на месте, в базе данных, получая от клиента только значения необходимых параметров и возвращая клиенту результаты расчета в виде малого объема итоговой информации.

Итак, лучше разработать два варианта работы клиентского приложения с базой данных: один, первый, тот, что и был ранее. Он обеспечивает выборку всех записей таблиц и работу с таблицами в памяти машины. Другой вариант — выборка только одной записи из всех таблиц по заданному табельному номеру. Поэтому в Карточке появится кнопка **Варианты работы**.

Пример построения хранимой процедуры

Допустим, требуется подсчитать количество записей таблицы Osn_Dannie. Сначала создадим в БД структуру **Хранимая процедура**, а затем воспользуемся компонентом TIBStoredProcedure вкладки Interbase. Процедура в БД создается с помощью оператора CREATE PROCEDURE (табл. 2.1). Он создает на языке SQL саму процедуру, ее входные и выходные параметры и ее действия. Структура оператора показана в листинге 2.2.

Листинг 2.2

```
CREATE PROCEDURE
CREATE PROCEDURE name
[(param <datatype> [, param <datatype> …])]
[RETURNS <datatype> [, param <datatype> …])]
AS <procedure_body> [terminator]
<procedure_body> =
[<variable_declaration_list>]
<block>
<variable_declaration_list> =
DECLARE VARIABLE var <datatype>;
[DECLARE VARIABLE var <datatype>; …]
<block> =
BEGIN
<compound_statement>
[<compound_statement> …]
END
<compound_statement> = {<block> | statement;}
<datatype> = SMALLINT| INTEGER| FLOAT| DOUBLE PRECISION
| {DECIMAL | NUMERIC} [(precision [, scale])]
| {DATE | TIME | TIMESTAMP)
```

```
| {CHAR | CHARACTER | CHARACTER VARYING | VARCHAR}
[(int)] [CHARACTER SET charname]
| {NCHAR | NATIONAL CHARACTER | NATIONAL CHAR} [VARYING] [(int)]
```

Таблица 2.1. *Описание аргументов оператора Create Procedure*

Аргумент	Описание
name	Имя процедуры. Должно быть уникальным среди имен элементов БД
param datatype	Входные параметры, через которые вызывающая программа передает процедуре определенные значения: param — имя входного параметра, datatype — InterBase-тип входного параметра
RETURNS param datatype	Выходные параметры, через которые процедура возвращает значения вызывающей программе: param — имя параметра, datatype – InterBase-тип параметра. Процедура возвращает значения выходных параметров, если в своем теле она содержит оператор возврата SUSPEND
AS	Это ключевое слово отделяет заголовок процедуры от ее тела
DECLARE VARIABLE var datatype	Это выражение объявляет локальные переменные процедуры. После объявления переменной должна стоять точка с запятой (;), var — имя локальной переменной, datatype — InterBase-тип переменной
statement	Любой одиночный оператор языка InterBase-процедур и триггеров. Должен заканчиваться точкой с запятой (;), исключая операторы-скобки BEGIN и END
terminator	Terminator — знак окончания тела процедуры. Определяется оператором SET TERM. Используется только в интерактивном SQL

Описание оператора CREATE PROCEDURE

Оператор определяет новую хранимую процедуру в базе данных. Хранимая процедура — это программа, написанная на InterBase-языке хранимых процедур и триггеров и хранимая как часть метаданных (данных, описывающих структуры БД). Хранимая процедура может получать входные данные через описанные в ней входные параметры и выдавать результаты вызвавшей ее программе через описанные в ней выходные параметры. InterBase-язык процедур и триггеров включает в себя все операторы SQL и некоторые усиливающие SQL расширения, включающие операторы IF THEN ELSE, WHILE DO, FOR SELECT DO, исключения и обработку ошибок. То есть этот язык максимально приближен к обычному алгоритмическому языку для составления программ.

Существуют два типа процедур:

- SELECT-процедуры, которые приложение может использовать в качестве таблиц или вьюеров в операторе SELECT. Такие процедуры должны возвращать одно или более значений или ошибку, которая произошла при их выполнении;
- Executable-процедуры, которые приложение может выполнять напрямую с помощью оператора EXECUTE PROCEDURE или соответствующего метода компонента TStoredProcedure. Такие процедуры необязательно должны возвращать какие-то значения вызывающей программе.

Так как каждый оператор в теле процедуры должен оканчиваться точкой с запятой, а само тело тоже должно иметь какой-то признак окончания, который, естественно, должен отличаться от точки с запятой, то для него следует определить свой символ окончания. Для этого используют оператор SET TERM перед оператором CREATE PROCEDURE, чтобы задать такой символ-терминатор. После окончания оператора CREATE PROCEDURE следует написать другой оператор SET TERM, чтобы изменить введенный сисмвол-терминатор процедуры на точку с запятой. InterBase-язык процедур и триггеров включает в себя операторы:

- SQL-операторы INSERT, UPDATE, DELETE и одиночный SELECT;
- SQL-операторы и выражения, включая генераторы и функции, определенные пользователем (UDF), связанные с базой данных.

Расширения SQL, включая операторы присвоения, управления, контекстные переменные (для триггеров), операторы, отсылающие события, исключения, а также операторы обработки ошибок. В табл. 2.2 приведены расширения SQL, принятые в InterBase-языке процедур и триггеров.

Таблица 2.2. *Расширения SQL*

Оператор	**Описание**
BEGIN ... END	Задает блок операторов, который исполняется как один оператор. BEGIN начинает блок, END определяет конец блока. Точку с запятой после него ставить нельзя
variable = expression	Оператор присвоения значения выражения expression переменной, входному или выходному параметру
/* comment_text */	Так задается комментарий, который может содержать любое количество текстовых строк
EXCEPTION exception_name	При исполнении этого оператора возникает исключение с именем name. Исключение — это ошибка, определенная пользователем, возвращающая некоторое, определенное пользователем, сообщение

Таблица 2.2 *(окончание)*

Оператор	Описание
EXECUTE PROCEDURE proc_name [var [, var ...]] [RETURNING_VALUES var [, var ...]]	Оператор исполняет хранимую процедуру proc_name
EXIT	Оператор передает управление на конец процедуры: на ее последний оператор END
FOR select_statement DO compound_statement	Этот оператор повторяет оператор или блок после слова DO для каждой строки, извлеченной оператором select_statement. select_statement — это обычный оператор SELECT за тем исключением, что он должен в самом конце содержать утверждение INTO, задающее в какие переменные должен пересылаться результат
compound_statement	Это либо одиночный оператор, либо блок операторов, заключенный в операторные скобки BEGIN END
IF (condition) THEN compound_statement [ELSE compound_statement]	Проверяет условие condition: если оно равно TRUE, выполняет оператор или блок, следующий за THEN — в противном случае выполняет оператор или блок, следующий за ELSE, если то присутствует
NEW.column	Это новая контекстная переменная, которая указывает новое значение колонки в операциях INSERT или UPDATE
OLD.column	Это старая контекстная переменная, которая указывает значение колонки перед операциями UPDATE или DELETE
POST_EVENT event_name | col	Посылает событие event_name или использует значение в col в качестве имени события
SUSPEND	Используется в процедуре типа SELECT. Приостанавливает исполнение процедуры до момента, когда от вызывающего приложения не поступит следующая команда FETCH (сделать выборку), и возвращает выходные значения вызывающему приложению
WHILE (condition) DO compound_statement	Пока условие condition равно TRUE, выполняется оператор compound_statement
WHEN {error [, error ...] | ANY} DO compound_statement	Оператор обработки ошибок: пока имеет место одна из заданных ошибок (error) или пока имеет место любая ошибка (ANY), выполняется оператор compound_statement. error — это EXCEPTION exception_name, SQLCODE errcode или GDSCODE number

Создадим теперь хранимую процедуру, которая подсчитывает количество записей в картотеке (мы помним, что мы ввели три записи в таблицу Osn_dannie). Процедуру можно определить через редактор Блокнот, а затем вставить в окно интерактивного SQL, а можно сразу записать в окно интерактивного SQL утилиты IBConsole. Вид этого окна и созданная хранимая процедура показаны на рис. 2.16.

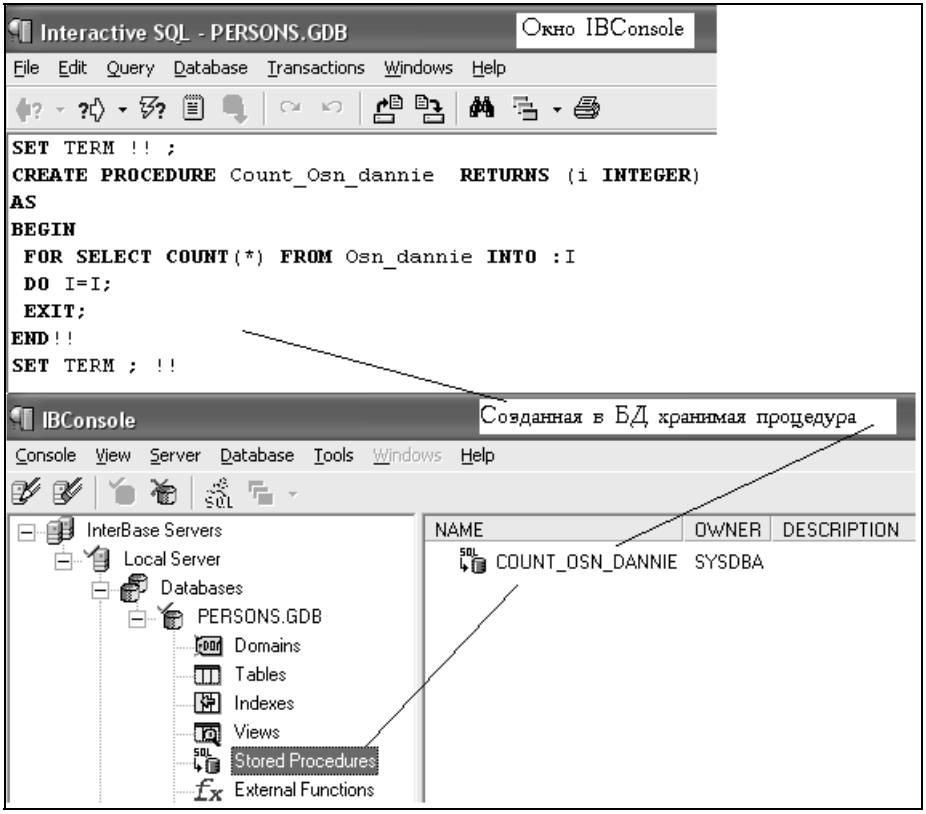


Рис. 2.16. Вид окна интерактивного SQL утилиты IBConsole и создаваемая хранимая процедура

Для проверки работы хранимой процедуры было создано вспомогательное приложение, вид формы и инспекторы объекта компонентов которого показаны на рис. 2.17.

Результат выполнения процедуры, которая подсчитала количество записей в таблице по основным данным карточки (в таблице было три записи), показан на рис. 2.18.

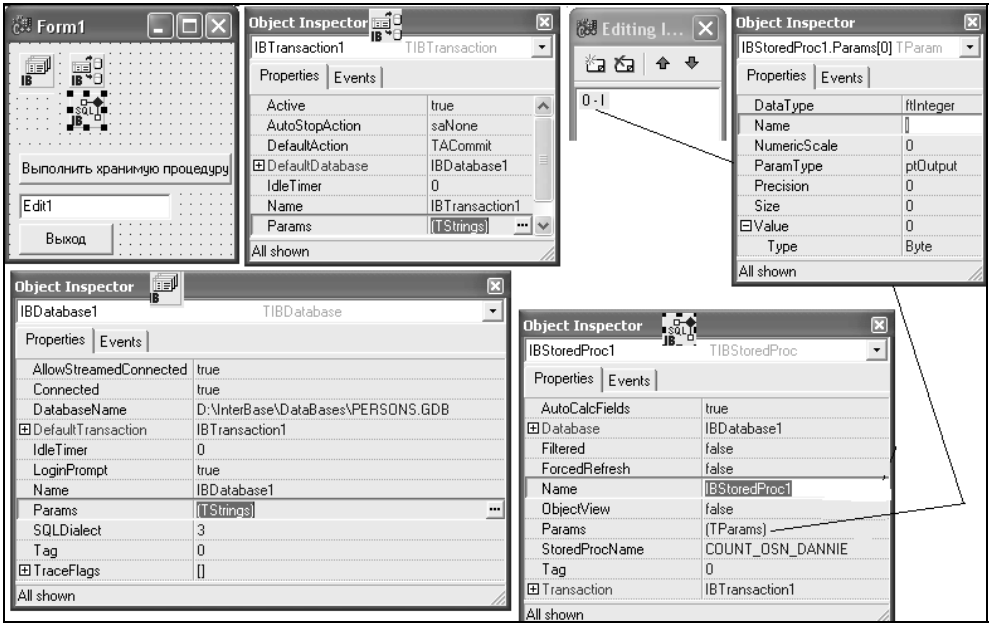


Рис. 2.17. Компоненты приложения для проверки работы хранимой процедуры

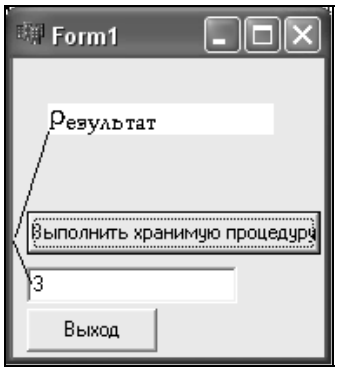


Рис. 2.18. Результат работы хранимой процедуры

Текст приложения по проверке хранимой процедуры приведен в листинге 2.3.

Листинг 2.3

Cpp-файл

```
//---------------------------------------------------------------------
```

```
#include <vcl.h>
#pragma hdrstop

#include "Unit1.h"
//---------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
 Application->Terminate();
}
//---------------------------------------------------------------------

void __fastcall TForm1::Button2Click(TObject *Sender)
{
  IBStoredProc1->Prepare();
  IBStoredProc1->ExecProc();
  Edit1->Text = IBStoredProc1->ParamByName("i")->AsInteger;
}
```

h-файл

```
//---------------------------------------------------------------------

#ifndef Unit1H
#define Unit1H
//---------------------------------------------------------------------
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
#include <DB.hpp>
#include <DBClient.hpp>
#include <DBGrids.hpp>
#include <DBLocal.hpp>
#include <DBLocalI.hpp>
#include <ExtCtrls.hpp>
```

```
#include <Grids.hpp>
#include <IBDatabase.hpp>
#include <Provider.hpp>
#include <IBCustomDataSet.hpp>
#include <IBStoredProc.hpp>
//---------------------------------------------------------------------
class TForm1 : public TForm
{
__published: // IDE-managed Components
        TIBDatabase *IBDatabase1;
        TIBTransaction *IBTransaction1;
        TEdit *Edit1;
        TButton *Button1;
        TIBStoredProc *IBStoredProc1;
        TButton *Button2;
        void __fastcall Button1Click(TObject *Sender);
        void __fastcall Edit1KeyDown(TObject *Sender, WORD &Key,
          TShiftState Shift);
        void __fastcall Button2Click(TObject *Sender);

private:      // User declarations
public:             // User declarations
        __fastcall TForm1(TComponent* Owner);
};
//---------------------------------------------------------------------
extern PACKAGE TForm1 *Form1;
//---------------------------------------------------------------------
#endif
```

Компоненты Interbase и таблицы БД

Приведем перечень компонентов Interbase, настроенных на соответствующие таблицы базы данных Persons. Их имена необходимы при создании приложений на основе разработанной картотеки учета персонала:

- IBClientDataSet1 — настроен на таблицу OSN_DANNIE;
- IBClientDataSet2 — настроен на таблицу DOP_DANNIE;
- IBClientDataSet3 — настроен на таблицу VOENN;

- IBClientDataSet4 — настроен на таблицу PROFESSII;
- IBClientDataSet5 — настроен на таблицу OBRAZUCHREJD;
- IBClientDataSet6 — настроен на таблицу OBRAZ;
- IBClientDataSet7 — настроен на таблицу RODSTVO;
- IBClientDataSet8 — настроен на таблицу SLUJBA;
- IBClientDataSet9 — настроен на таблицу PODRAZD;
- IBClientDataSet10 — настроен на таблицу OTPUSCA;
- IBClientDataSet11 — настроен на таблицу PRAZDDNI;
- IBClientDataSet12 — настроен на таблицу VIDIOTPUSCOV;
- IBClientDataSet13 — настроен на таблицу UVOLN;
- IBClientDataSet14 — настроен на таблицу ATEST;
- IBClientDataSet15 — настроен на таблицу RESULTATEST;
- IBClientDataSet16 — настроен на таблицу VIDIOBUCH;
- IBClientDataSet17 — настроен на таблицу NAPRAVLPODG;
- IBClientDataSet18 — настроен на таблицу SREDSVAOBUCH;
- IBClientDataSet19 — настроен на таблицу POVKVAL;
- IBClientDataSet20− настроен на таблицу STRANI;
- IBClientDataSet21 — настроен на таблицу ZAGRAN;
- IBClientDataSet22 — настроен на таблицу SPISVZPOOSHR;
- IBClientDataSet23 — настроен на таблицу VZISCPOOSHR;
- IBClientDataSet24 — настроен на таблицу CINI;
- IBClientDataSet25 — настроен на таблицу CLASCHIN;
- IBClientDataSet26 — настроен на таблицу VIDISOTSLGOT;
- IBClientDataSet27− настроен на таблицу SOTSLGOTI;
- IBClientDataSet28 — настроен на таблицу VIDIPRICHINUVOLN;
- IBClientDataSet29 — настроен на таблицу VIDINAGRAD;
- IBClientDataSet30 — настроен на таблицу NAGRADI;
- IBClientDataSet31 — настроен на таблицу RESULTKOMANDIROVOK;
- IBClientDataSet32 — настроен на таблицу MESTAKOMANDIROVOK;
- IBClientDataSet33 — настроен на таблицу KOMANDIROVKI;
- IBClientDataSet34 — настроен на таблицу CELIKOMANDIROVOK;
- IBClientDataSet35 — настроен на таблицу POSLMESTORAB;

- IBClientDataSet36 — настроен на таблицу DOLJNOSTI;
- IBClientDataSet37 — настроен на таблицу OSOBIEUSLOVIA;
- IBClientDataSet38 — настроен на таблицу PROCNADBAVKI;
- IBClientDataSet39 — настроен на таблицу CONTRACTI;
- IBClientDataSet40 — настроен на таблицу TIPIUDOSTLICHNOSTI;
- IBClientDataSet41 — настроен на таблицу VIDIORAZOVANIA;
- IBClientDataSet42— настроен на таблицу SEMPOLOJ;
- IBClientDataSet43— настроен на таблицу TIPIOBUCH;
- IBClientDataSet44— настроен на таблицу TIPIOBRAZOVANIA;
- IBClientDataSet45— настроен на таблицу SPECIALNOSTI.

Некоторые общие принципы разработки приложения по созданию картотеки персонала

Здесь мы рассмотрим, как формировать поле на основе справочных данных и на основе выбора даты, и как получать отклик в виде текста наименования в ответ на двойной щелчок на коде этого наименования.

Формирование поля на основе справочных данных

Возьмем пример, показанный на рис. 2.19.

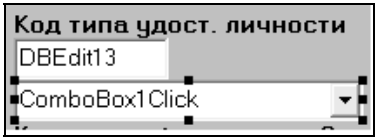


Рис. 2.19. Формирование поля DBEdit13 на основе справочных данных

Поле **Код типа удостоверения личности** надо сформировать на основе соответствующего справочника типов удостоверения личности. Справочные данные станем формировать в элементе TComboBox, чтобы затем выборкой из него отослать нужный код в требуемое поле. Строку станем формировать

в виде значения кода, за которым следует разделитель в виде наклонной черты, после которой пойдет наименование. Устанавливаем в форму TComboBox и в элементе DBEdit13 с помощью Инспектора объекта создаем обработчик события OnEnter, чтобы как только DBEdit13 получит фокус ввода (то ли на нем будет щелчок мыши, то ли фокус ввода ему передаст какой-либо другой элемент), управление передалось в обработчик события OnEnter. В обработчике же события надо записать команды работы со справочником типов удостоверений личности таким образом, чтобы из него извлеклись все строки и поместились в элемент TComboBox, который при помещении его в форму в режиме дизайна делается невидимым: его свойство Visible устанавливается в значение FALSE. Как только строки помещены, TComboBox делается видимым и ему передается фокус ввода, чтобы из него можно было тут же выбирать необходимую информацию. Текст обработчика приведен в листинге 2.4. Сам справочник "достается" из БД с помощью компонента IBClientDataSet40.

Листинг 2.4

```
    AnsiString s;
// Формирование списка компонента ComboBox1 для выборки кода
//удостоверения личности
 if(!IBClientDataSet40->Active)
    IBClientDataSet40->Open();
 IBClientDataSet40->First();
 ComboBox1->Clear();

 for(int i=0; i <IBClientDataSet40->RecordCount; i++)
  {
   s=IntToStr(IBClientDataSet40->Fields->Fields[0]->AsInteger) + "/" +
    IBClientDataSet40->Fields->Fields[1]->AsString;
   ComboBox1->Items->Add(s);
   IBClientDataSet40->Next();
  }
   ComboBox1->Show();
   ComboBox1->SetFocus();
```

Теперь следует организовать выборку необходимой строки из элемента ComboBox. Для этого следует создать обработчик его события OnClick: как только произойдет щелчок на компоненте (а его сделают тогда, когда найдут

выбранную строку и щелкнут на ней), строка должна попасть в поле DBEdit13. После пересылки элемента строки в нужное поле (а это будет выделенный из строки код) надо наименование тоже выделить и отобразить на элементе TPanel для удобства работы. И только после этого надо передать фокус ввода следующему для обработки полю. Текст обработчика приведен в листинге 2.5.

Листинг 2.5

```
  //выборка из ComboBox1 строки и засылка из нее кода типа удостоверения
личности AnsiString s1,s=ComboBox1->Items->Strings[ComboBox1->ItemIndex];
  int pos=s.AnsiPos("/");
  s1=s.SubString(1,pos-1);
  DBEdit13->Field->AsInteger=StrToInt(s1);
  Panel16->Show();
  Panel16->Caption=s.SubString(pos+1,150);
  ComboBox1->Hide();
  DBEdit9->SetFocus();
```

Формирование поля-даты

Рассмотрим пример, показанный на рис. 2.20.

Рис. 2.20. Формирование поля-даты

Устанавливаем в форму компонент TDateTimePicker. В Инспекторе объекта для компонента DBEdit5 задаем обработку события OnEnter: как только DBEdit5 получит фокус ввода, начнет работать обработчик события OnEnter. В обработчике запишем команды **Показать TDateTimePicker** (в режиме раз-

работки этот компонент устанавливается состояние невидимости, чтобы он не мешал при работе приложения и появлялся на экране только в необходимых случаях) и **Передать фокус ввода компоненту TDateTimePicker**:

```
DateTimePicker1->Show();
DateTimePicker1->SetFocus();
```

В самом же TDateTimePicker надо организовать обработку события OnCloseUp в режиме исполнения после выбора даты в выпадающем календаре, который появляется при нажатии кнопки справа в поле TDateTimePicker. В обработчике следует записать команды удаления с экрана TDateTimePicker, пересылки выбранной из календаря даты в поле DBEdit5 и передачи фокуса ввода следующему полю, которое предстоит обрабатывать после ввода даты:

```
DateTimePicker1->Hide();
DBEdit5->Field->AsDateTime = DateTimePicker1->Date;
DBEdit6->SetFocus();
```

Получение отклика в виде текста наименования в ответ на двойной щелчок на коде этого наименования

В компьютере как можно большее количество данных хранится в закодированном виде: это не только экономит память, но и создает удобства для обработки информации. Но удобств без неудобств практически не бывает. В нашем случае неудобство состоит в том, при работе с Карточкой в полях появляются коды соответствующих реквизитов вместо их названий. Чтобы получить название, в приложении используется другое событие компонента, в котором находится код: двойной щелчок на этом компоненте. Создается обработчик этого события, который читает из таблицы уже не весь справочник, как в случае, когда в компонент вводился код, а только строку, содержащую заданный в поле компонента код. Такую выборку позволяет сделать компонент TIBClientDataSet, у которого есть свойство CommandText: именно через это свойство и задается информация о выборке из таблицы БД. В нем задается оператор SQL с ограничителем where, в котором определяется выборка конкретной строки с заданным значением кода. Свойство CommandText формируется параметрически в специальной функции

```
AnsiString PoiskNaim(TIBClientDataSet *a, AnsiString TabNaim,int
Cod,AnsiString NazvPoleaCod)
```

В ней параметр `a` задает компонент типа TIBClientDataSet, через который будет обрабатываться та или иная таблица-справочник БД, параметр TabNaim задает имя конкретной таблицы-справочника, параметр `Cod` задает конкретное значение кода, строку с которым надо выбрать из таблицы, и,

наконец, параметр NazvPoleaCod задает имя поля кода в таблице. Пример обработки двойного щелчка компонента DBEdit13:

```
AnsiString s=DBEdit13->Field->AsString;
 s=s.Trim();
 int cod=StrToInt(s);
 Panel16->Caption=  Poisk-
Naim(IBClientDataSet40,"TipiUdostLichnosti",cod,"cod");
 Panel16->Show();
```

Работа с вкладками Карточки

Работа начинается с основной вкладки: **Общие сведения**, а в ней — с нажатия кнопки **Начало работы**. После ее нажатия появляется диалоговое окно для задания пароля доступа к БД. Мы создавали БД с паролем masterkey, его и набираем. После этого в нижнем поле вкладки появляется подсвеченное поле выбора режимов работы, показанное на рис. 2.21.

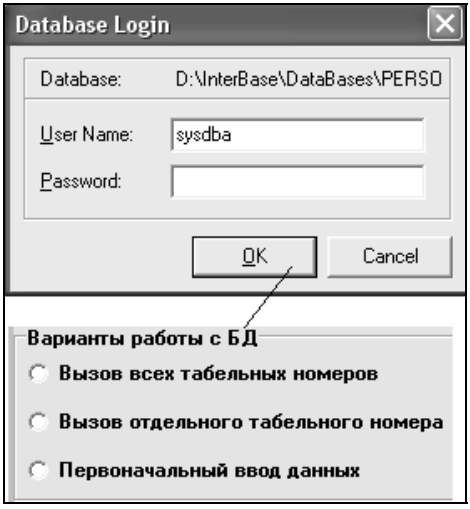
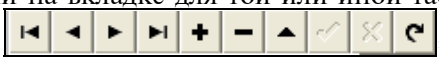


Рис. 2.21. Выбор режима работы с картотекой

Режимы первый и третий практически равносильны: они открывают возможность движения по всем строкам таблицы "Основные данные". Только третий режим используется тогда, когда таблица еще пуста и только заводится, а первый — когда в таблице уже есть строки и которую потом можно пополнять. Третий режим введен просто для "неуверенного" пользователя,

хотя заведение таблицы можно было бы осуществлять и в первом режиме. Второй же режим дает доступ только к работе с одной записью таблицы "Основные данные" и с соответствующими записями других таблиц. Чтобы перейти к записи с другим табельным номером (а именно этот реквизит запрашивается при начале работы во втором режиме), следует повторно выбрать второй режим работы.

После модификации какой-либо таблицы (справочника) на вкладке следует нажать на кнопку **Обновить** для соответствующей таблицы, чтобы изменения попали в БД, так как модификация происходит в памяти клиентского приложения, а БД, как известно, находится на сервере.

При работе с полями на вкладке для той или иной таблицы следует пользоваться навигатором  , который автоматически настраивается на ту таблицу, с которой происходит работа. Нажимая на кнопки навигатора, вы можете осуществлять движение по записям таблицы (первые четыре кнопки слева имеют смысл **В начало таблицы**, **Предыдущая запись**, **Последующая запись**, **В конец таблицы**), следующая кнопка с изображением плюса нужна, когда требуется добавить новую запись: она создает пустую запись, в поля которой и вводятся новые данные, кнопка с изображением минуса используется, когда надо удалить текущую запись, кнопка с треугольником используется, когда надо модифицировать запись: если она не нажата (имеет доступный, черный, цвет), то модификацию поля выполнять нельзя. Следующая кнопка отсылает модифицированные данные в БД (в нашем случае они на сервер не попадают: все равно надо нажимать кнопку **Обновить**), при нажатии на кнопку с крестом изменения, внесенные в поля записи, ликвидируются (отменяются). Кнопка с изогнутой стрелкой нужна при работе в сети, когда несколько пользователей одной и той же таблицы имеют к ней доступ и право модификации. Чтобы увидеть последнее состояние таблиц, и надо нажать эту кнопку — **Освежить**.

При переходе от одного табельного номера к другому (нажатием одной из первых слева четырех кнопок) следует затем нажать кнопку со знаком треугольника, чтобы перевести таблицу в состояние модификации, иначе ни одно поле нельзя будет подвергнуть никакому изменению.

На всех вкладках, кроме главной, предусмотрена возможность работы не только с одной видимой записью, а сразу со всеми записями таблицы: просматривать их, модифицировать и т. д. Это так называемое табличное представление информации. Инструмент представления информации БД в табличной форме (а им является компонент DBGrid) автоматически настраивается на активную в данный момент таблицу БД.

При создании вкладок для лучшего понимания начинающим программистом программных частей использовались повторы программных блоков, которые, в принципе, можно было бы превратить в функции и тем самым сократить размер текста программы и сделать его более изящным. Автор придерживается мнения, что жертвовать пониманием текста в угоду его со-

кращению или изяществу следует очень-очень аккуратно. Особенно когда имеешь дело с начинающим изучать этот предмет. Часть блоков все-таки пришлось превратить в функции, которые все расположены в начале текста приложения перед Конструктором. Кроме того, с описанной уже целью некоторые программные участки реализованы по-разному. Например, в главной вкладке для поиска применена функция Locate(), тогда как в остальных — выборка с помощью оператора SQL.

Действия пользователя на всех вкладках практически однотипны.

- На вкладке **Стаж** никаких действий не происходит: при ее появлении автоматически рассчитываются данные по возрасту работника и его видам стажа на данную дату.
- На вкладке **Отпуска** при задании начальной даты отпуска и количества дней отпуска автоматически высвечивается конечная дата отпуска, рассчитанная с использованием справочника выходных и праздничных дней, которую пользователь может ввести в соответствующее поле. А может ввести и свою дату. Справочник-календарь надо поддерживать, как и все остальные справочники, в актуальном состоянии.
- Вид вкладок (кроме основной) в режиме разработки и исполнения показан на рис. 2.22—2.57.

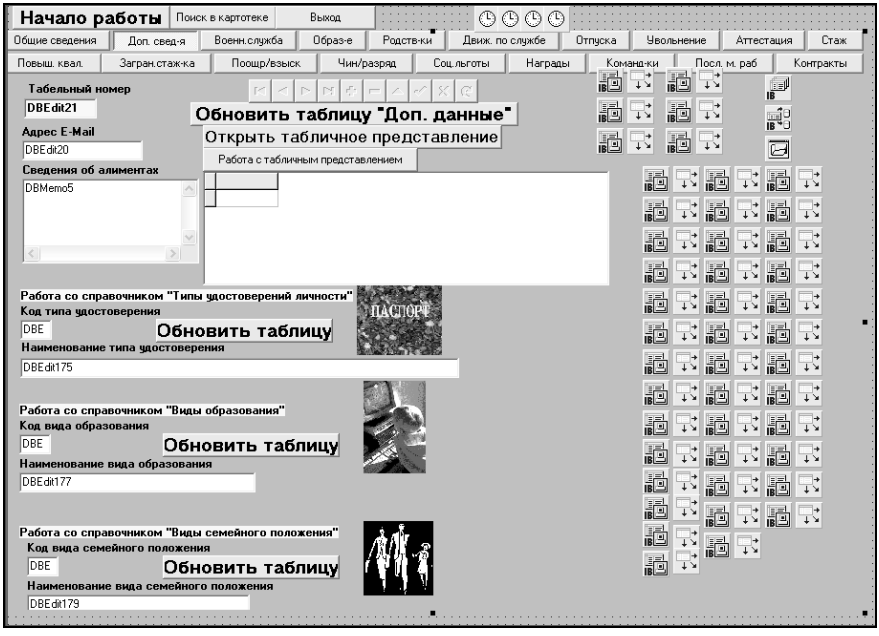


Рис. 2.22. Вкладка **Дополнительные сведения** в режиме разработки

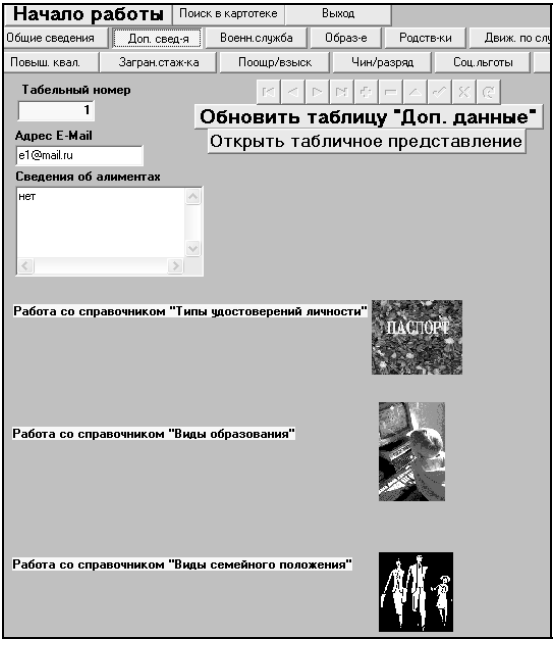


Рис. 2.23. Вкладка **Дополнительные сведения** в режиме исполнения

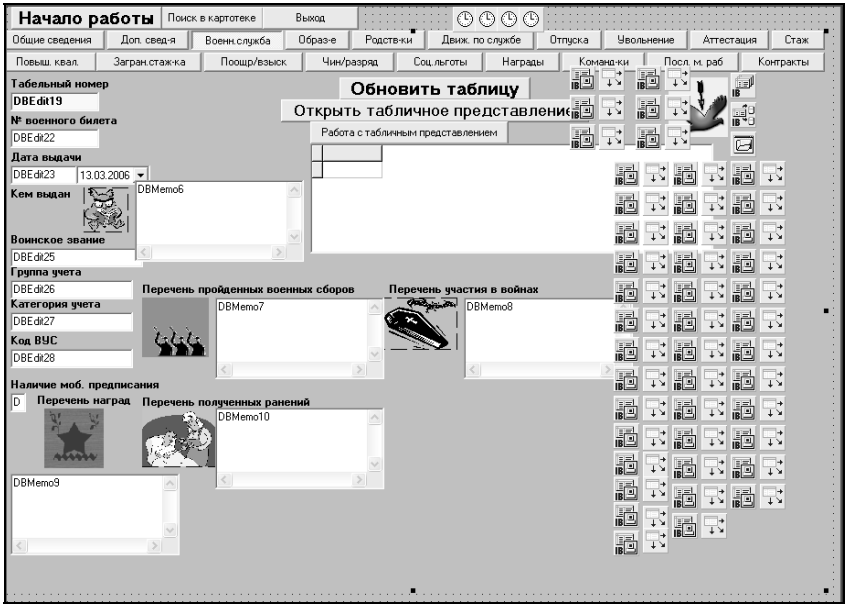


Рис. 2.24. Вкладка **Военная служба** в режиме разработки

Рис. 2.25. Вкладка **Военная служба** в режиме исполнения

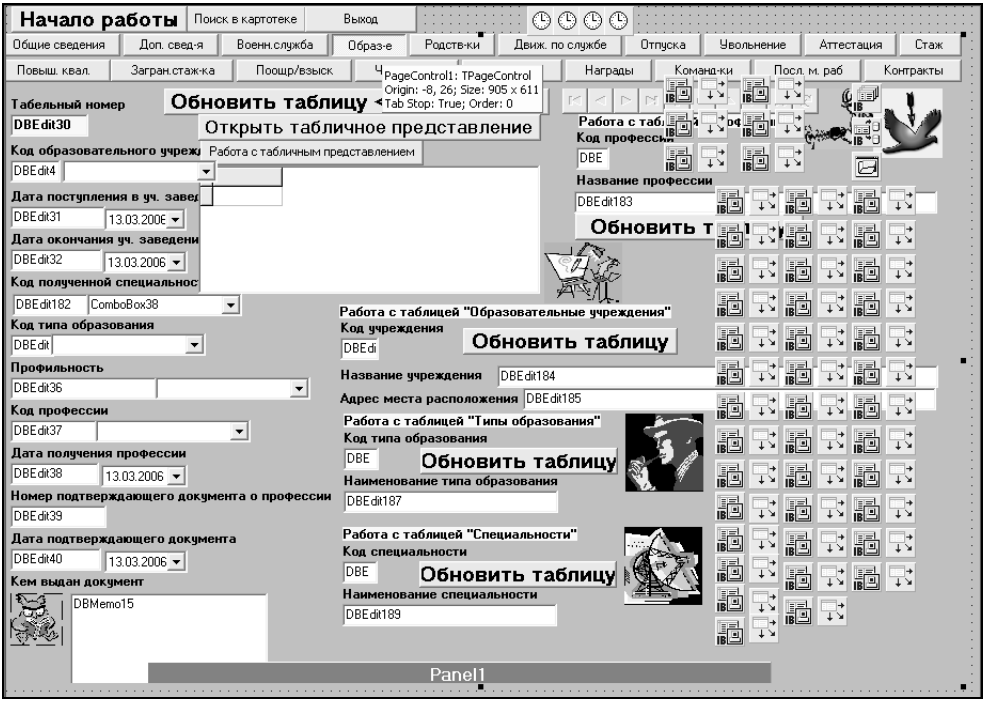


Рис. 2.26. Вкладка **Образование** в режиме разработки

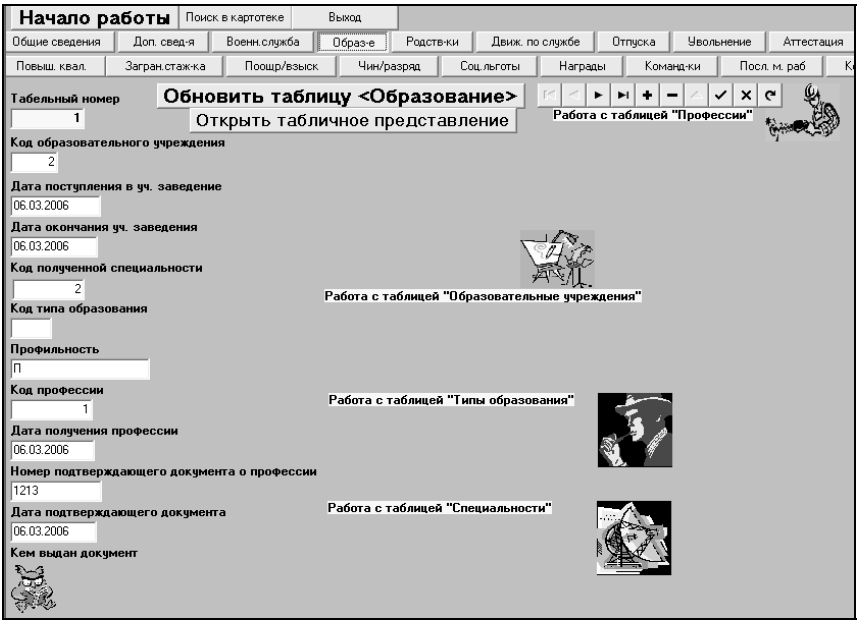


Рис. 2.27. Вкладка **Образование** в режиме исполнения

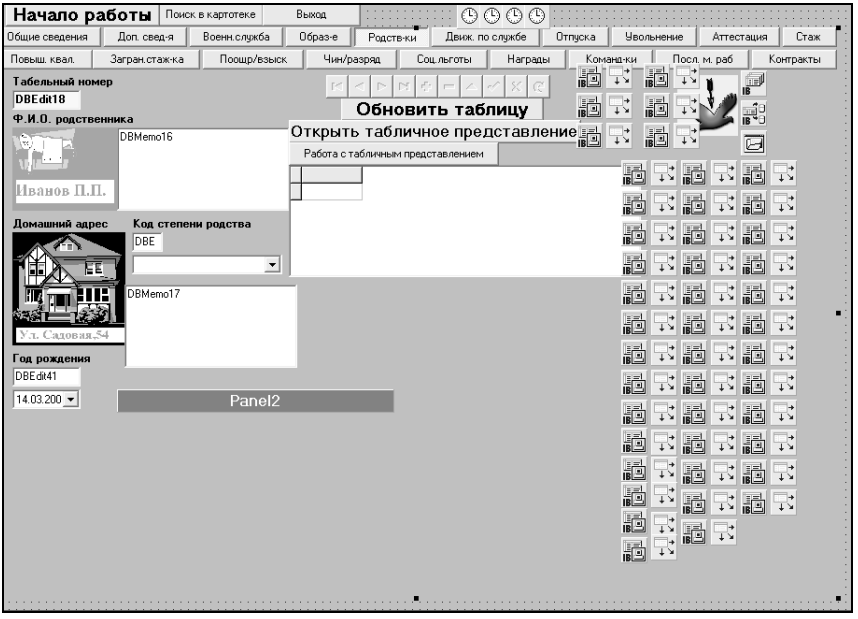


Рис. 2.28. Вкладка **Родственники и иждивенцы** в режиме разработки

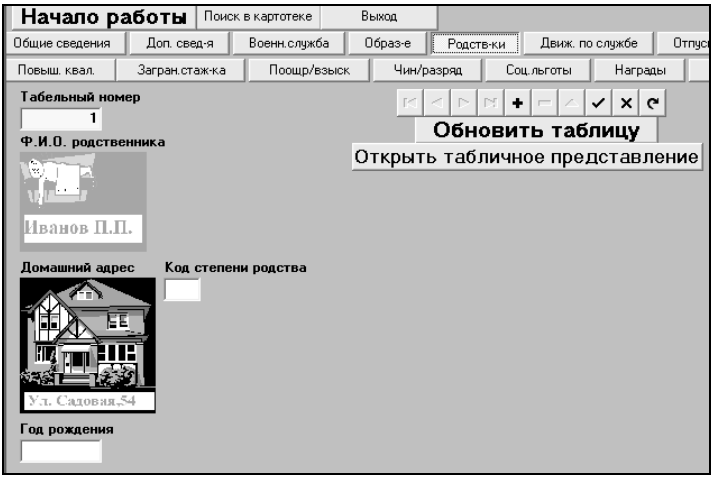


Рис. 2.29. Вкладка **Родственники и иждивенцы** в режиме исполнения

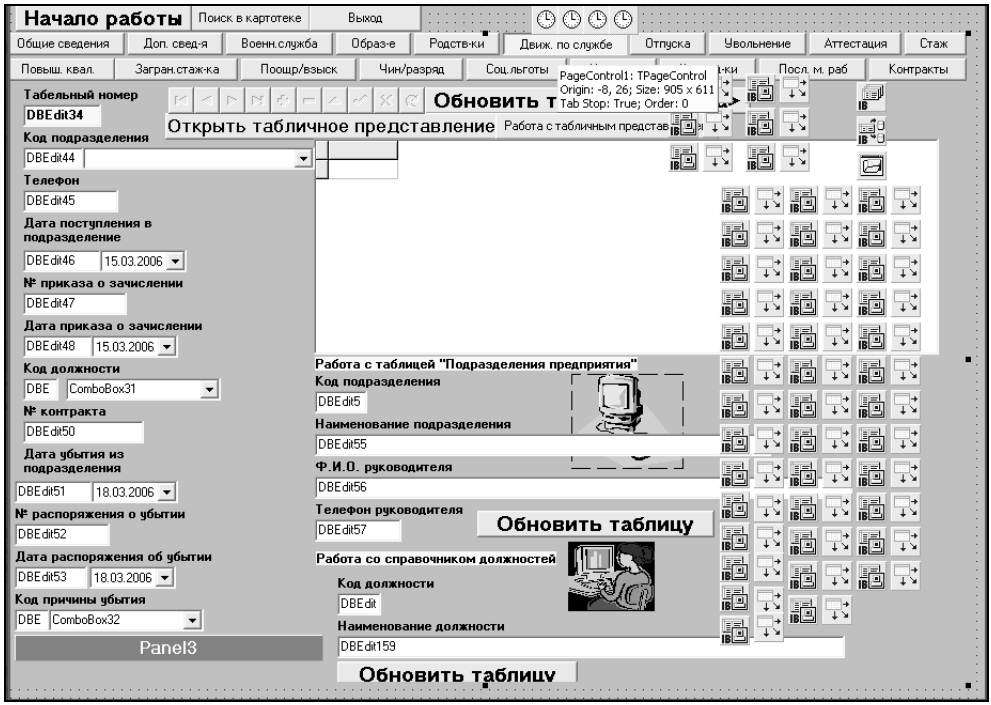


Рис. 2.30. Вкладка **Движение по службе** в режиме разработки

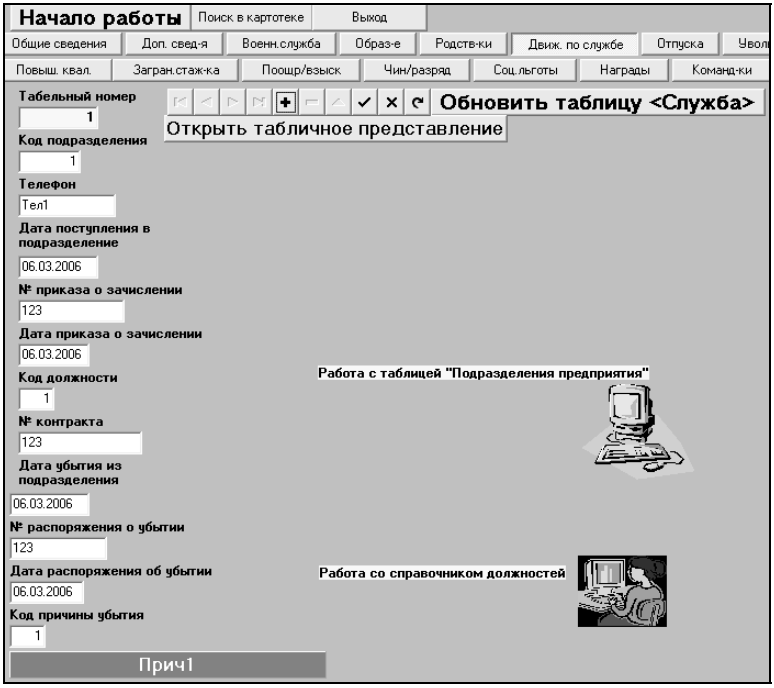


Рис. 2.31. Вкладка **Движение по службе** в режиме исполнения

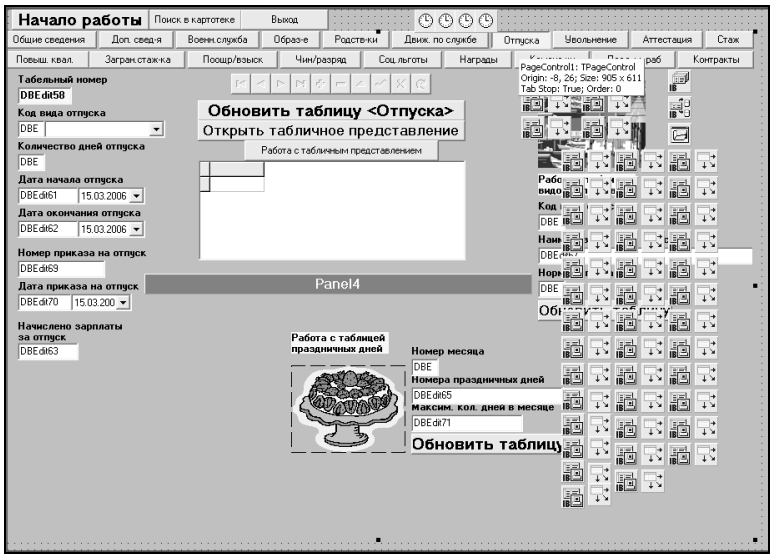


Рис. 2.32. Вкладка **Отпуска** в режиме разработки

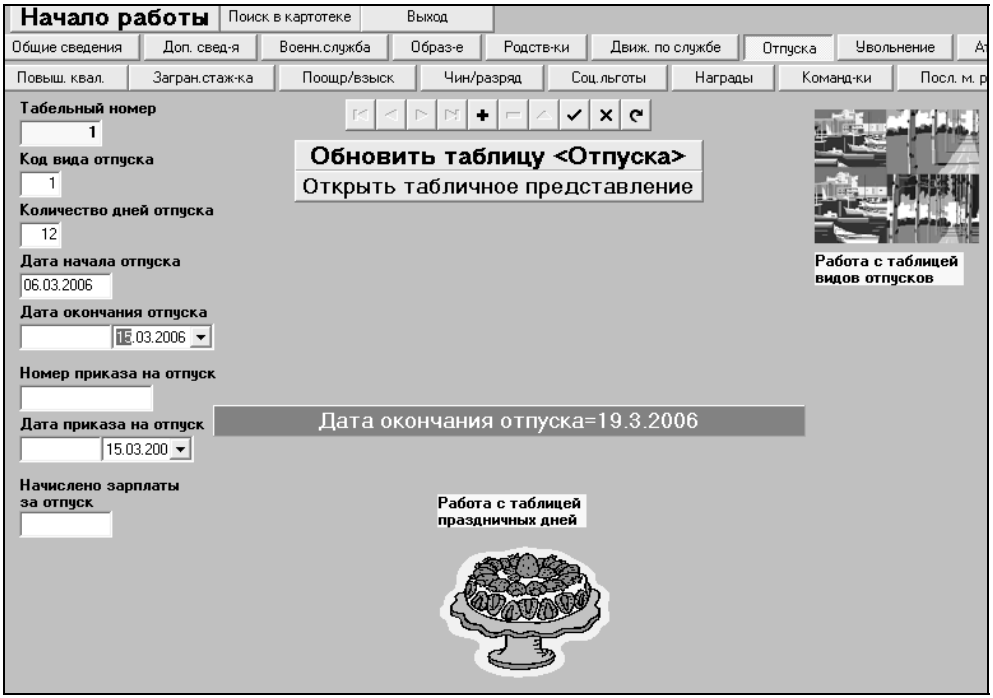


Рис. 2.33. Вкладка **Отпуска** в режиме исполнения

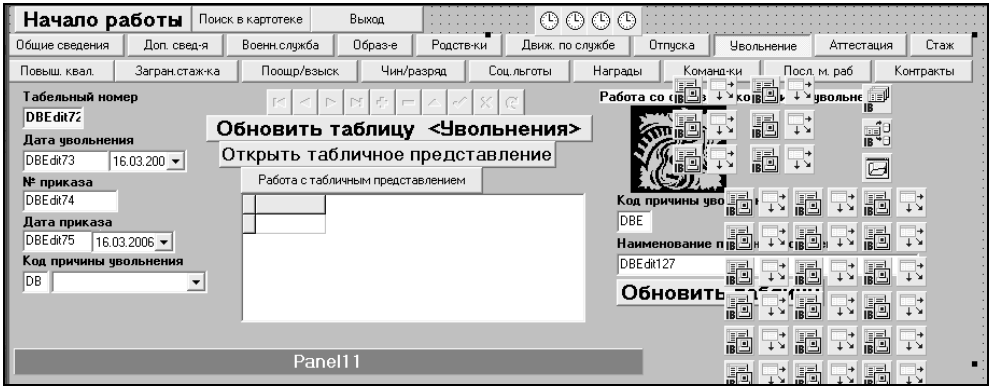


Рис. 2.34. Вкладка **Увольнение** в режиме разработки

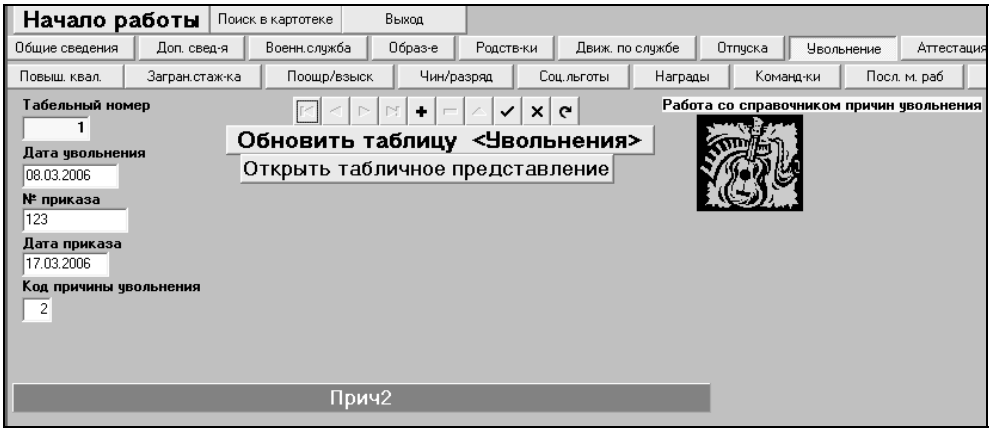


Рис. 2.35. Вкладка **Увольнение** в режиме исполнения

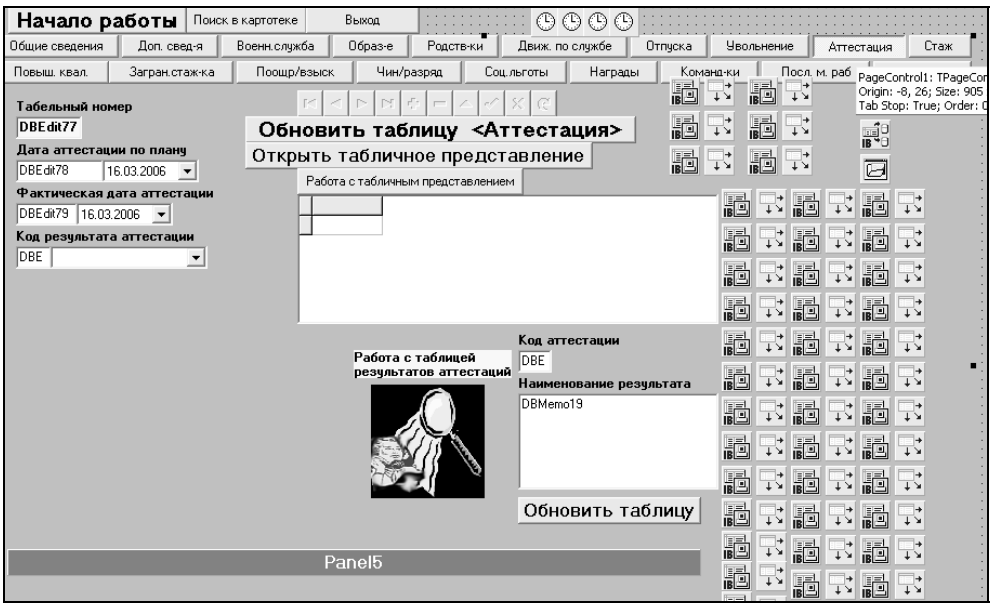


Рис. 2.36. Вкладка **Аттестация** в режиме разработки

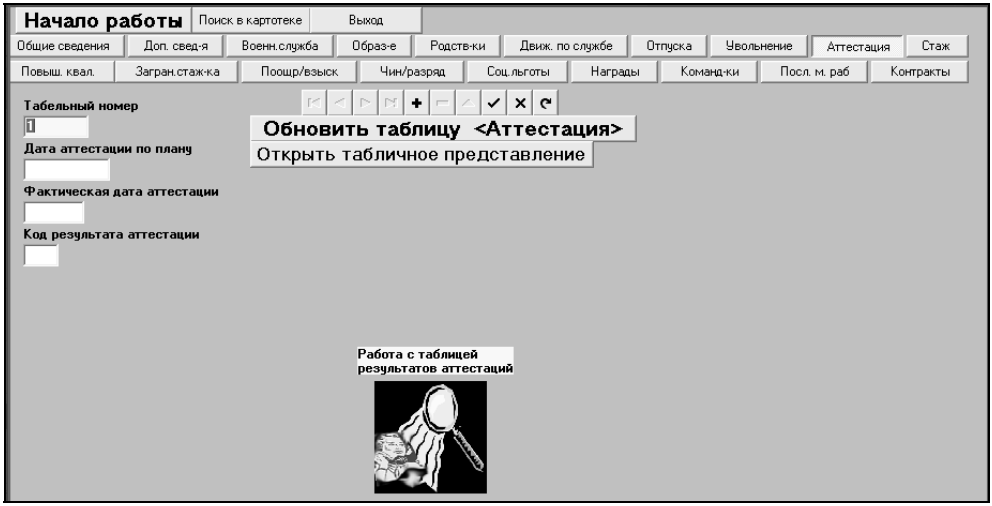


Рис. 2.37. Вкладка **Аттестация** в режиме исполнения

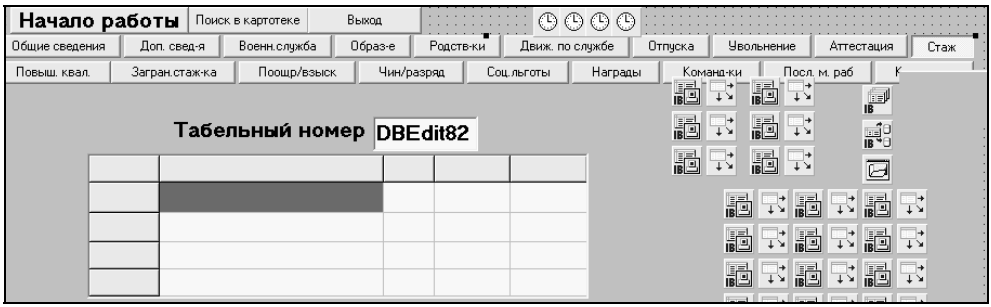


Рис. 2.38. Вкладка **Стаж** в режиме разработки

Рис. 2.39. Вкладка **Стаж** в режиме исполнения

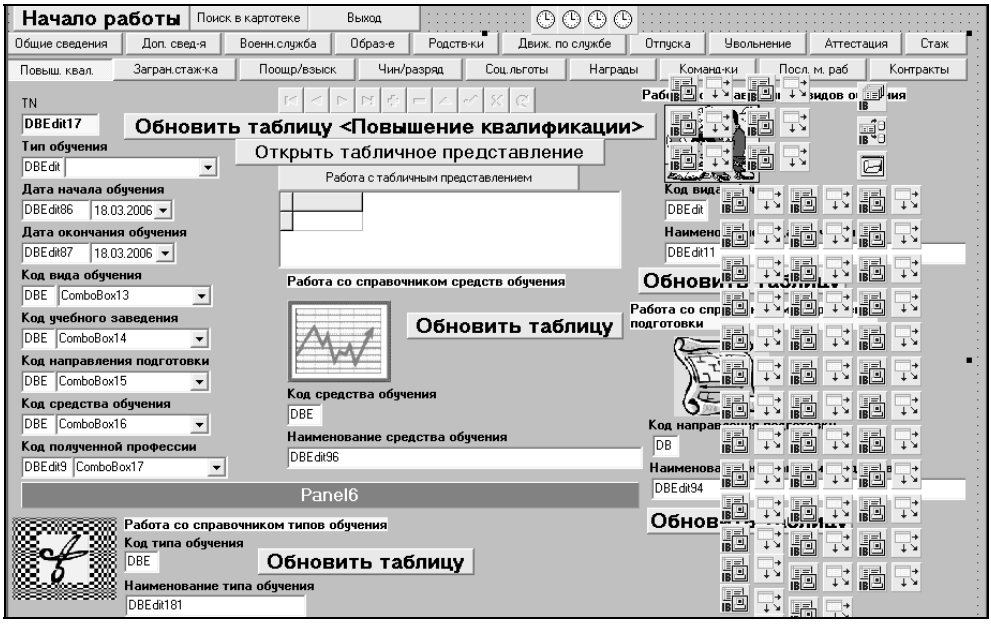


Рис. 2.40. Вкладка **Повышение квалификации** в режиме разработки

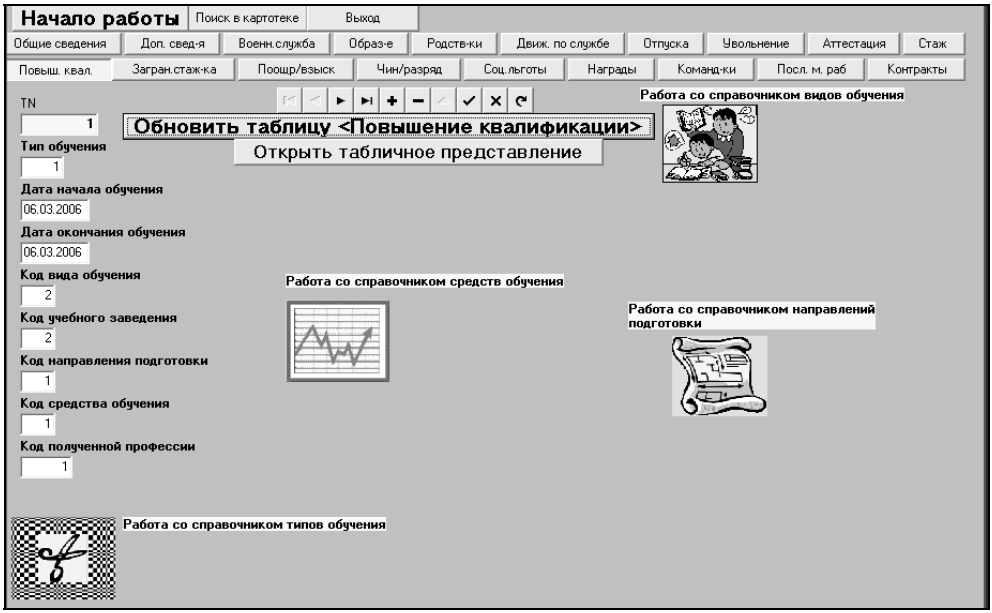


Рис. 2.41. Вкладка **Повышение квалификации** в режиме исполнения

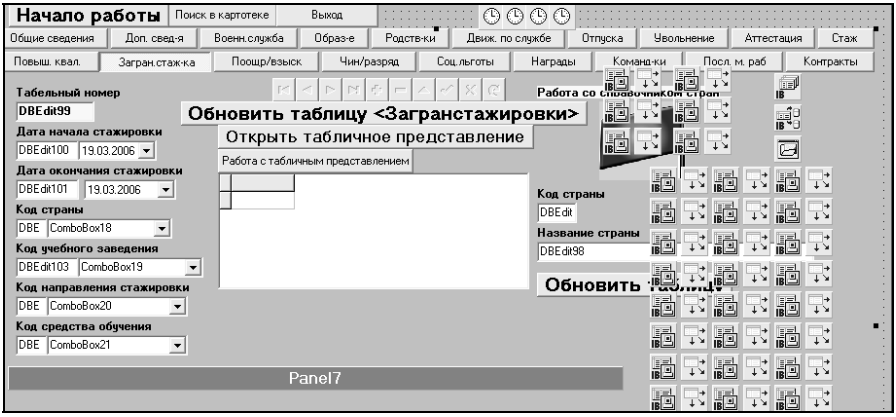


Рис. 2.42. Вкладка **Стажировка за границей** в режиме разработки

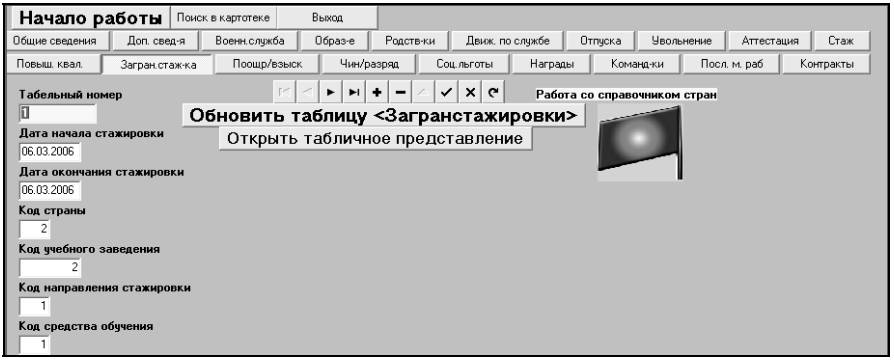


Рис. 2.43. Вкладка **Стажировка за границей** в режиме исполнения

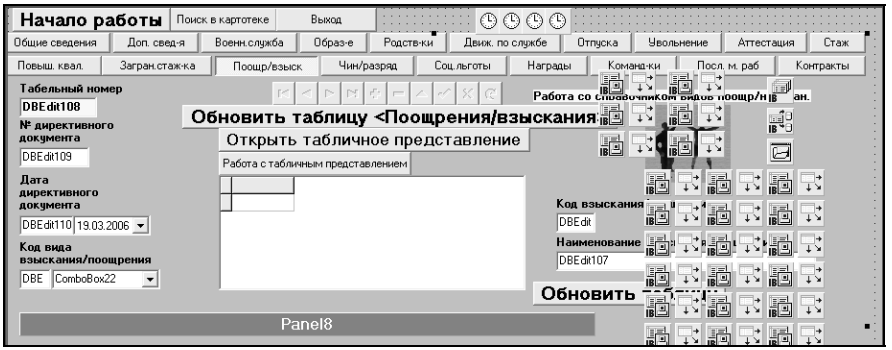


Рис. 2.44. Вкладка **Поощрения/взыскания** в режиме разработки

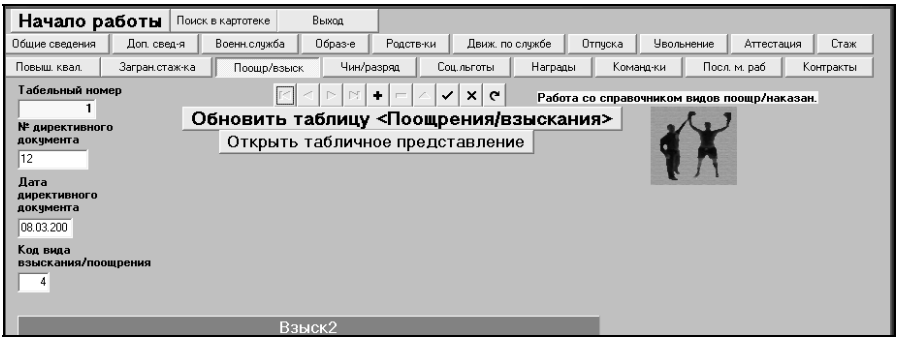


Рис. 2.45. Вкладка **Поощрения/взыскания** в режиме исполнения

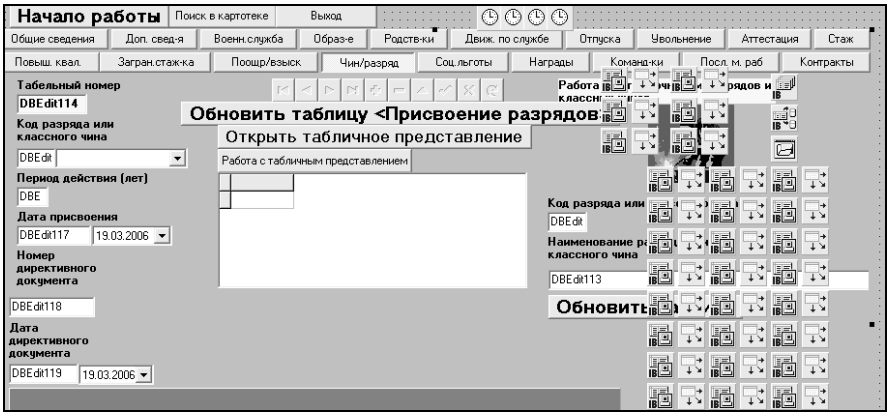


Рис. 2.46. Вкладка **Чины/разряды** в режиме разработки

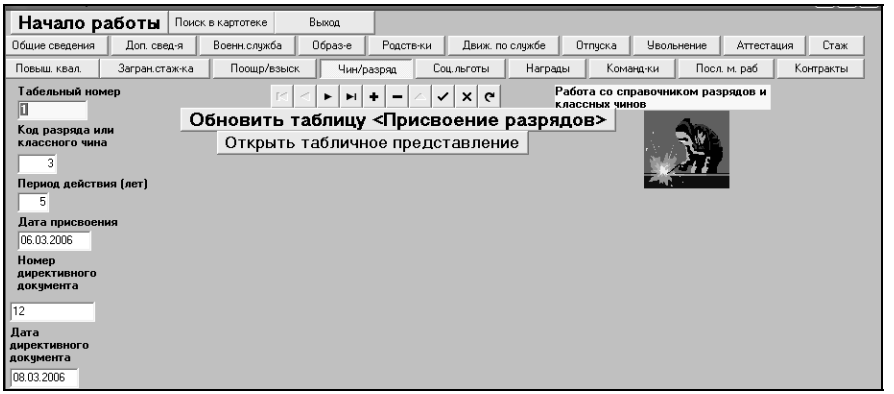


Рис. 2.47. Вкладка **Чины/разряды** в режиме исполнения

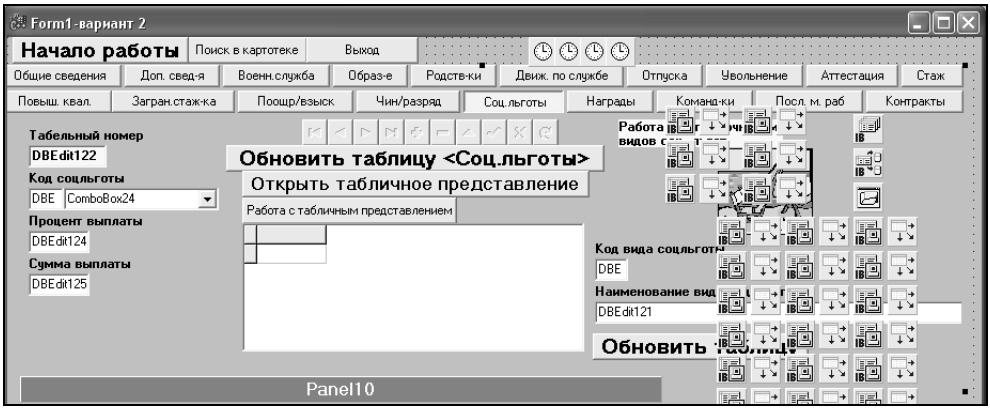


Рис. 2.48. Вкладка **Социальные льготы** в режиме разработки

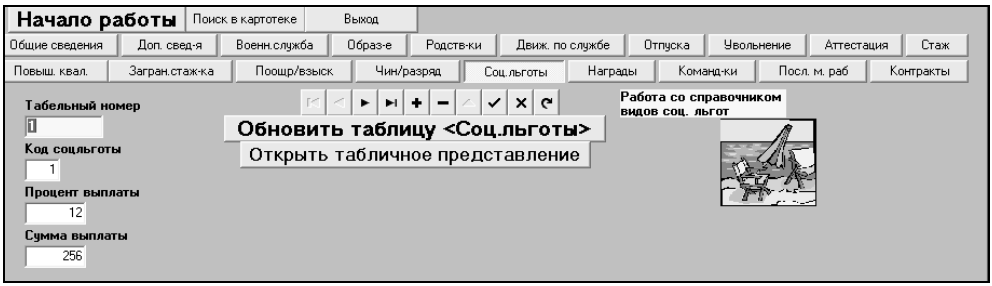


Рис. 2.49. Вкладка **Социальные льготы** в режиме исполнения

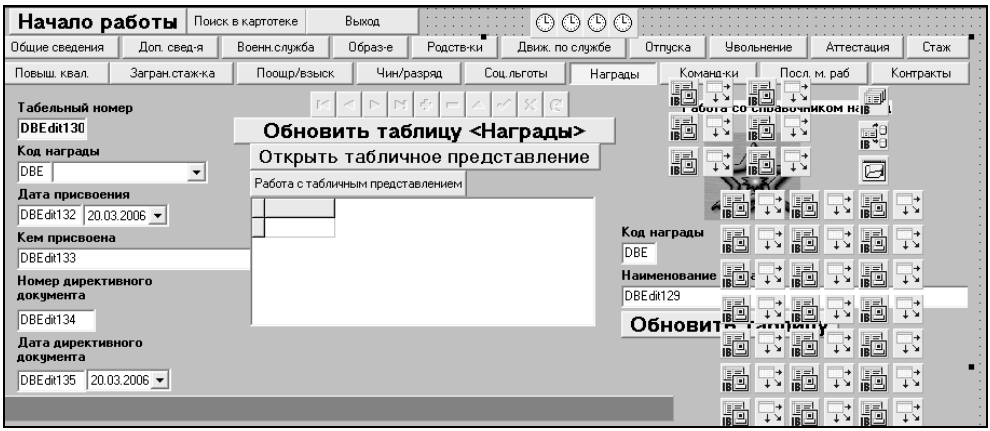


Рис. 2.50. Вкладка **Награды** в режиме разработки

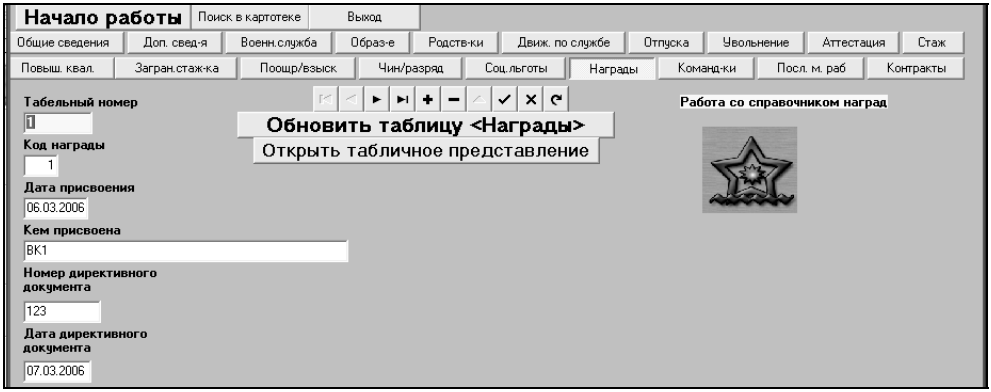


Рис. 2.51. Вкладка **Награды** в режиме исполнения

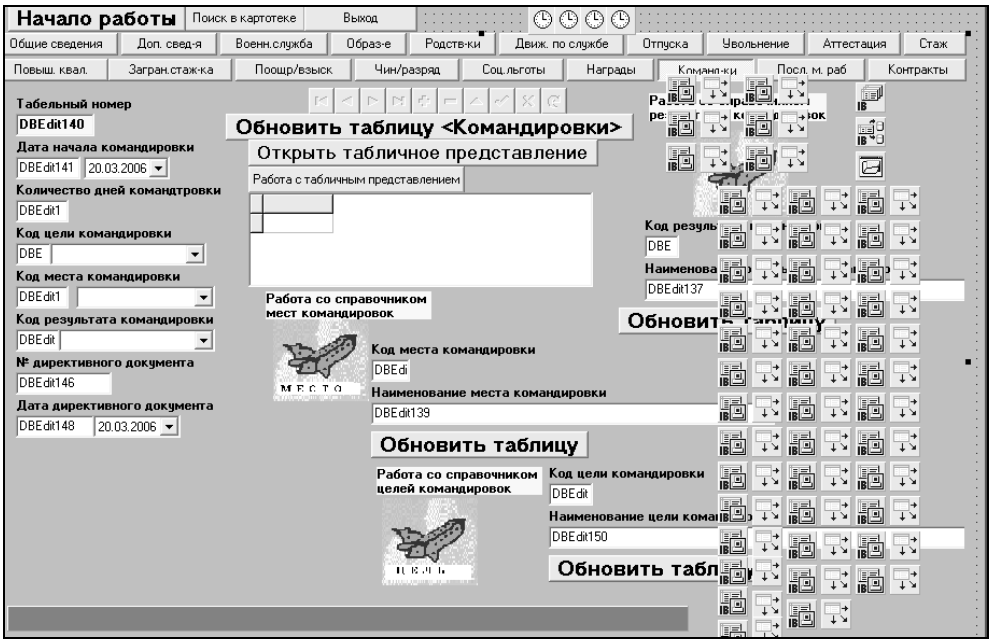


Рис. 2.52. Вкладка **Командировки** в режиме разработки

Рис. 2.53. Вкладка **Командировки** в режиме исполнения

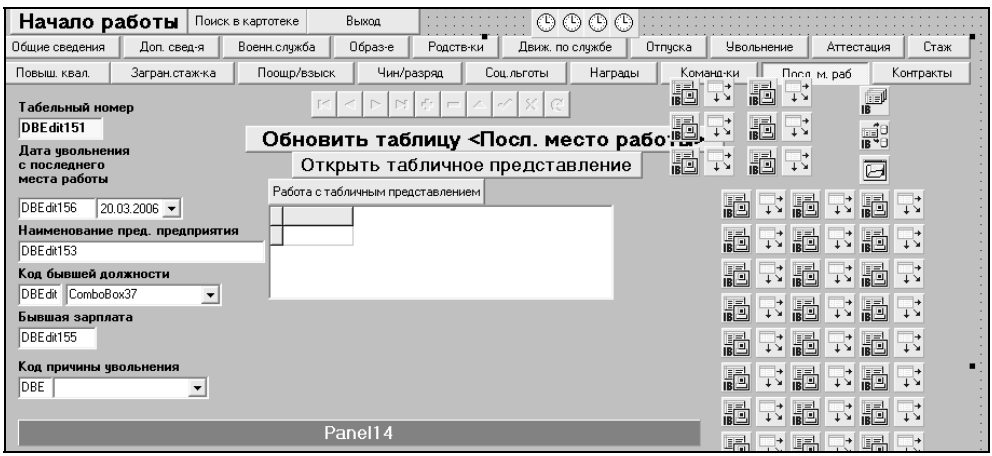


Рис. 2.54. Вкладка **Последнее место работы** в режиме разработки

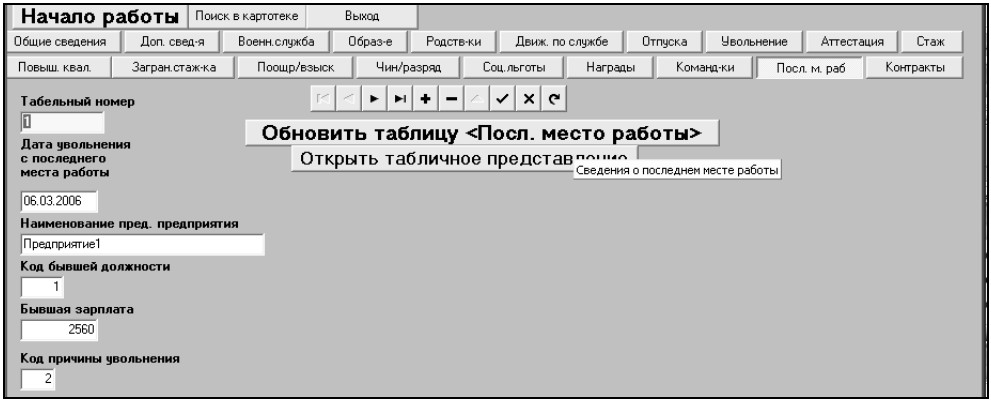


Рис. 2.55. Вкладка **Последнее место работы** в режиме исполнения

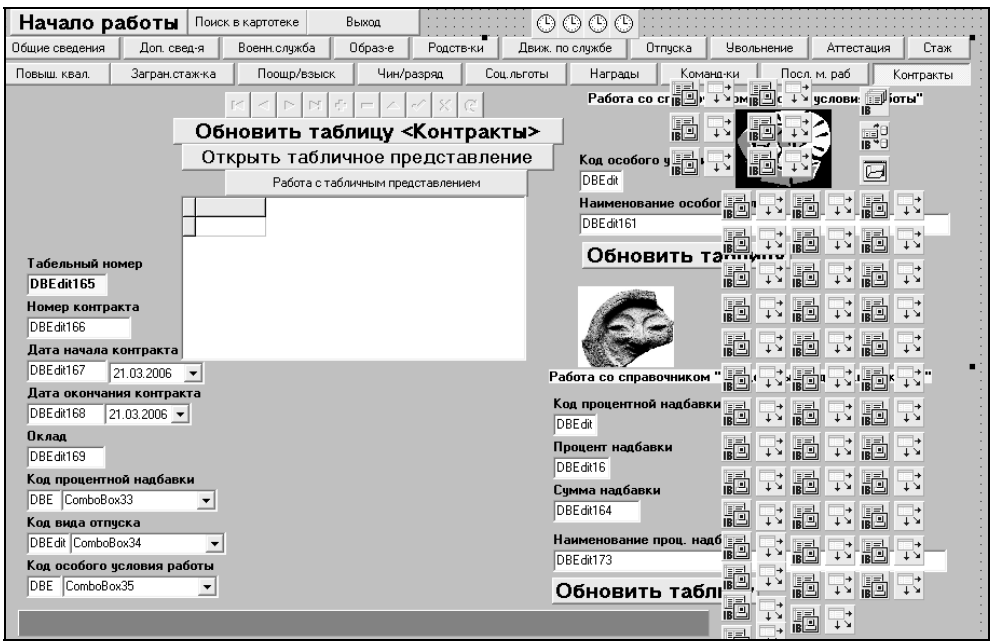


Рис. 2.56. Вкладка **Контракты** в режиме разработки

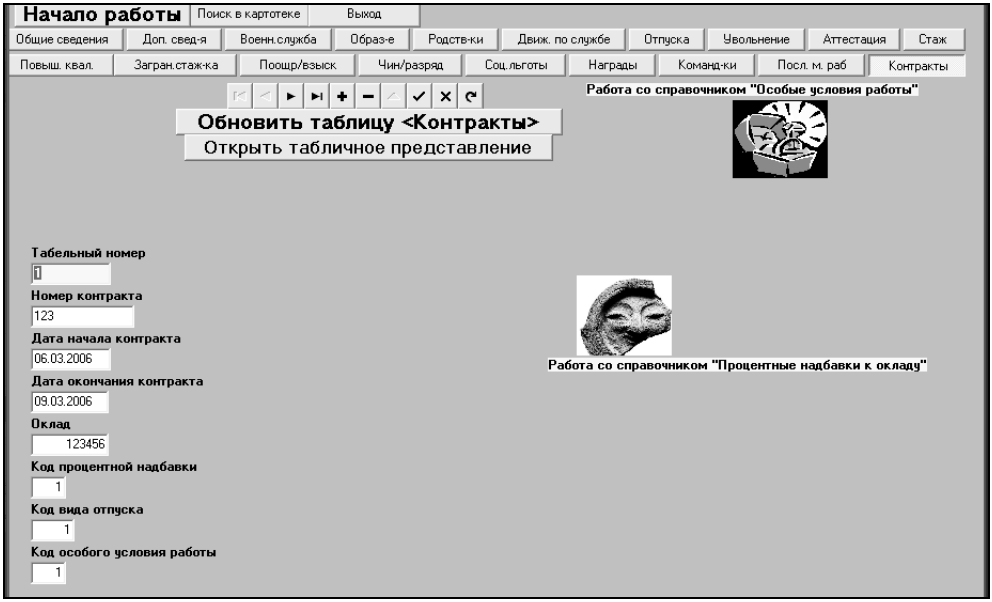


Рис. 2.57. Вкладка **Контракты** в режиме исполнения

Глава 3

Учет движения материалов на складах

Постановочная часть

Экономико-организационная сущность

Учет движения материалов на складе предприятия призван обеспечить контроль движения материальных ценностей и их сохранности по каждому складу и материально-ответственному лицу. Текущий учет движения материалов ведется в номенклатурном разрезе (в разрезе товаров) в натуральном и стоимостном выражении и осуществляется на основании приходно-расходных документов. К понятию материалов мы относим все предметы, находящиеся на складе (остатки материалов), поступающие на склад (приход материалов) и выдающиеся со склада (расход материалов). Поступление материалов может быть как от подразделений самого предприятия (готовые изделия или возврат неизрасходованных материалов), так и от внешних поставщиков (возврат ранее приобретенных у предприятия материалов или просто поставка своего материала). Выдача со склада может быть как какому-либо подразделению этого же предприятия, так и внешним клиентам предприятия. Таким образом, поставщиками и клиентами склада могут как собственные подразделения предприятия, так и предприятия не только своей страны, но и расположенные за ее пределами.

Чтобы выполнить цели, поставленные перед задачей учета движения материалов по складам, надо решить такие подзадачи, как "Учет поступления материалов в разрезе складов", "Учет расхода материалов со складов в разрезе получателей " и "Учет остатков материалов на складах".

Приходно-расходные документы поступают на склад из бухгалтерии предприятия, и на их основании осуществляется деятельность склада. По каж-

дому поступившему на склад материалу заводится так называемая "Карточка складского учета" (в дальнейшем — Карточка), в которую заносится вся информация о приходе, расходе и остатке материала с указанием реквизитов документов, на основании которых произошло изменение в Карточке по данному материалу. Таким образом, видно, что Карточка является не первичным, а вторичным документом, формируясь на основании первичных (приходно-расходных) документов. На основании Карточки бухгалтерия в дальнейшем строит различные аналитические таблицы.

Чтобы автоматизировать учет движения материалов, потребуется воспользоваться определенной справочной информацией. Так, например, надо зашифровать все материалы, хранящиеся на складе, указав их характеристики (цену, единицу измерения и т. д.), всех поставщиков и получателей материалов (мы специально не сказали "покупателей", так как среди тех, кто приобретает материал со склада, может быть и "свой"). И если на предприятии не введена внутренняя рыночная система, то тот "свой", кто получает материал со склада, будет просто получателем, а не покупателем. В дальнейшем эту ситуацию можно разрешить, если в коде покупателя будет отражена его принадлежность данному предприятию. Вот мы и подошли к тому, что надо иметь справочник поставщиков-покупателей, в которых эти экономические агенты, имеющие свои специфические характеристики (адрес, телефон), будут зашифрованы или — закодированы, что то же самое. Откуда становится известным, какие точно справочники потребуются для решения задачи, и какова будет их структура? Эти выводы делаются после проведения постановщиком задачи следующих шагов:

1. Анализируется вся выходная документация по данной задаче, т. е. документы и их структура, которые заказчик задачи желает получать: какие в них входят реквизиты и как они между собой взаимосвязаны.
2. Анализируется, какие для этого должны быть входные документы, т. е. документы и их структура, на основании которых станут формироваться выходные документы.
3. Сравниваются имеющиеся в автоматизируемом объекте входные документы с теми, которые выявляются в результате анализа выходной документации, и строятся общие входные документы.
4. Среди входных документов выделяются документы, которые станут иметь так называемую условно-постоянную информацию (мало меняющуюся). Это и будут справочники и нормативы (например, нормы отпуска материалов в подразделения предприятия на те или иные нужды).

Нормативно-справочная информация

Для решения задачи "Учет движения материалов на складах" используются следующие справочники:

- ❒ внутренний справочник материалов следующей структуры:
  - номенклатурный номер материала;
  - наименование материала;
  - шифр единицы измерения материала;
  - структурированный шифр материала (шифр, имеющий структуру в виде групп, подгрупп и т. д., к которым относится данный материал);
  - шифр балансового счета, на который будет отнесен этот материал (например, счет "Материалы", счет "Готовая продукция");
  - величина минимального запаса;
  - шифр денежной единицы;
  - шифр цены приобретения;
  - шифр цены отпуска;
  - шифр списка налогов на данный материал;
  - шифр страны-производителя;
- ❒ справочник единиц измерения материала, имеющий структуру:
  - шифр единицы измерения;
  - наименование единицы измерения;
  - шифр единицы измерения, в которую может переводиться данная единица (например, может потребоваться, чтобы килограммы были переведены в тонны);
  - коэффициент перевода в другую единицу измерения;
- ❒ справочник материалов, относящихся к группе агрегирования 1-го уровня:
  - код группы 1-го уровня;
  - наименование группы 1-го уровня;
- ❒ справочник материалов, относящихся к группе агрегирования 2-го уровня:
  - код группы 2-го уровня;
  - наименование группы 2-го уровня;
- ❒ справочник материалов, относящихся к группе агрегирования 3-го уровня:
  - код группы 3-го уровня;
  - наименование группы 3-го уровня;

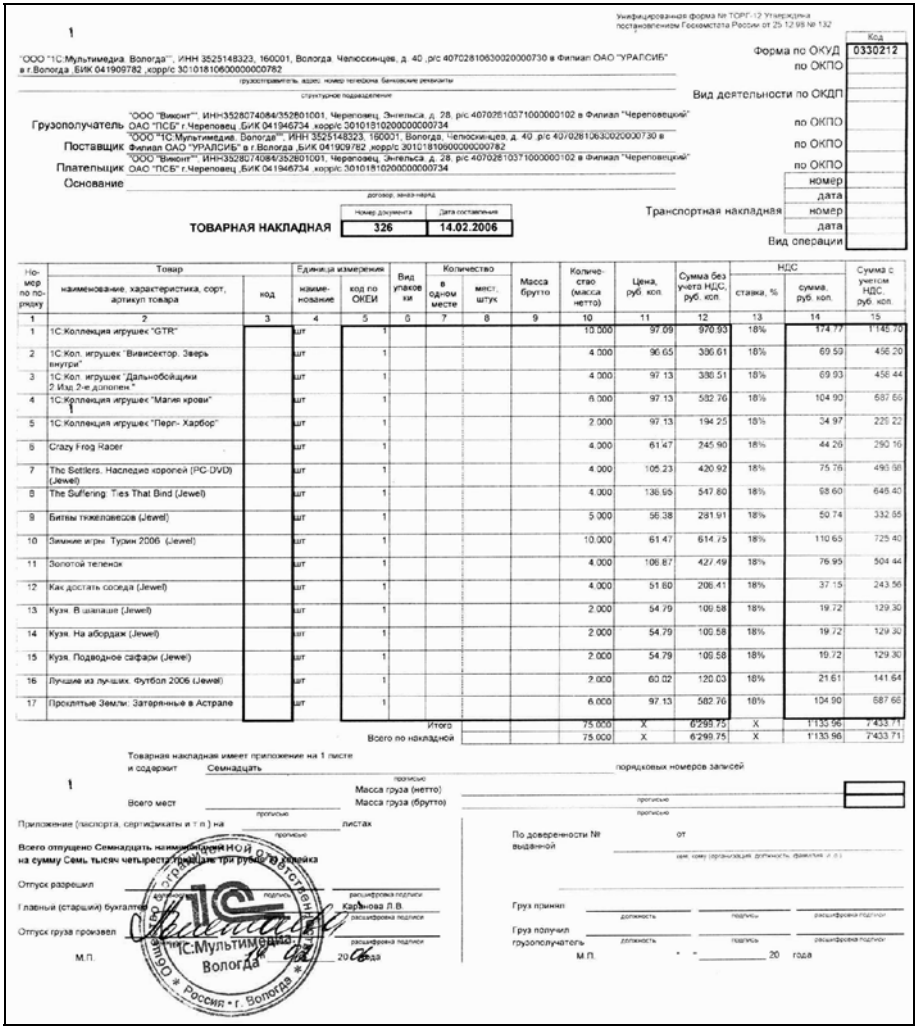
- ❒ справочник материалов, относящихся к группе агрегирования 4-го уровня:
  - код группы 4-го уровня;
  - наименование группы 4-го уровня;
- ❒ справочник материалов, относящихся к группе агрегирования 5-го уровня:
  - код группы 5-го уровня;
  - наименование группы 5-го уровня;
- ❒ справочник балансовых счетов, имеющий структуру:
  - код балансового счета;
  - наименование балансового счета;
- ❒ справочник денежных единиц, имеющий структуру:
  - код денежной единицы;
  - наименование денежной единицы;
  - знак денежной единицы;
- ❒ справочник цен приобретения материалов, имеющий структуру:
  - код денежной единицы;
  - код цены приобретения;
  - цена приобретения;
- ❒ справочник цен отпуска материалов, имеющий структуру:
  - код денежной единицы;
  - код цены отпуска;
  - цена отпуска;
- ❒ справочник списков видов налогов:
  - код списка;
  - код вида налога;
  - наименование вида налога;
  - процентная ставка налога;
  - ограничительная сумма налога;
- ❒ справочник стран, имеющий структуру:
  - код страны;
  - наименование страны;

- ❒ справочник поставщиков-покупателей со структурой (используется и для кодирования данного предприятия, в этот справочник вносятся и реквизиты данного предприятия):
  - код поставщика-покупателя;
  - полное наименование поставщика-покупателя;
  - юридический адрес;
  - фактический адрес;
  - телефоны;
  - факс;
  - e-mail;
  - банк;
  - расчетный счет;
  - корреспондентский счет:
  - директор;
  - главный бухгалтер;
  - контакт с представителем;
  - прочие данные;
  - код валюты взаиморасчетов;
- ❒ справочник новых курсов валют со структурой:
  - код валюты;
  - курс валюты;
  - дата изменения курса;
- ❒ справочник видов документов:
  - код документа;
  - наименование документа.

Входные документы

На отпуск товара со склада внутренним получателям и на прием от них материалов оформляется документ под названием "Накладная на внутреннее перемещение".

Когда товар поступает на склад от внешнего поставщика, он сопровождается документами "Товарная накладная" и "Счет-фактура", составленными поставщиком (рис. 3.1 и 3.2).



Унифицированная форма № ТОРГ-12 Утверждена постановлением Госкомстата России от 25.12.98 № 132

Код

Форма по ОКУД 0330212

по ОКПО

Вид деятельности по ОКДП

Грузополучатель "ООО "Вымпел"", ИНН 3528074084/352801001, Череповец, Энгельса, д. 28, р/с 40702810371000000102 в Филиал "Череповецкий" ОАО "ПСБ" г.Череповец ,БИК 041946734 ,корр/с 30101810200000000734 — по ОКПО

Поставщик "ООО "1С:Мультимедиа. Вологда"", ИНН 3525148323, 160001, Вологда, Челюскинцев, д. 40 ,р/с 40702810630020000730 в Филиал ОАО "УРАЛСИБ" в г.Вологда ,БИК 041909782 ,корр/с 30101810900000000782 — по ОКПО

Плательщик "ООО "Вымпел"", ИНН 3528074084/352801001, Череповец, Энгельса, д. 28, р/с 40702810371000000102 в Филиал "Череповецкий" ОАО "ПСБ" г.Череповец ,БИК 041946734 ,корр/с 30101810200000000734 — по ОКПО

Основание — договор, заказ-наряд — номер, дата

Транспортная накладная — номер, дата

Вид операции

Номер документа 326 | Дата составления 14.02.2006

ТОВАРНАЯ НАКЛАДНАЯ

Номер по порядку	Товар: наименование, характеристика, сорт, артикул товара	Товар: код	Единица измерения: наименование	Единица измерения: код по ОКЕИ	Вид упаковки	Количество: в одном месте	Количество: мест, штук	Масса брутто	Количество (масса нетто)	Цена, руб. коп.	Сумма без учета НДС, руб. коп.	НДС: ставка, %	НДС: сумма, руб. коп.	Сумма с учетом НДС, руб. коп.
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1С:Коллекция игрушек "GTR"		шт	1					10.000	97.09	970.93	18%	174.77	1'145.70
2	1С:Кол. игрушек "Вивисектор. Зверь внутри"		шт	1					4.000	96.65	386.61	18%	69.59	456.20
3	1С:Кол. игрушек "Дальнобойщики 2.Изд.2-е дополнен."		шт	1					4.000	97.13	388.51	18%	69.93	458.44
4	1С:Коллекция игрушек "Магия крови"		шт	1					8.000	97.13	582.76	18%	104.90	687.66
5	1С:Коллекция игрушек "Перл- Харбор"		шт	1					2.000	97.13	194.25	18%	34.97	229.22
6	Crazy Frog Racer		шт	1					4.000	61.47	245.90	18%	44.26	290.16
7	The Settlers. Наследие королей (PC-DVD) (Jewel)		шт	1					4.000	105.23	420.92	18%	75.76	496.68
8	The Suffering: Ties That Bind (Jewel)		шт	1					4.000	138.95	547.80	18%	98.60	646.40
9	Битвы тяжеловесов (Jewel)		шт	1					5.000	56.38	281.91	18%	50.74	332.65
10	Зимние игры. Турин 2006 (Jewel)		шт	1					10.000	61.47	614.75	18%	110.65	725.40
11	Золотой теленок		шт	1					4.000	106.87	427.49	18%	76.95	504.44
12	Как достать соседа (Jewel)		шт	1					4.000	51.60	206.41	18%	37.15	243.56
13	Кузя. В шапке (Jewel)		шт	1					2.000	54.79	109.58	18%	19.72	129.30
14	Кузя. На абордаж (Jewel)		шт	1					2.000	54.79	109.58	18%	19.72	129.30
15	Кузя. Подводное сафари (Jewel)		шт	1					2.000	54.79	109.58	18%	19.72	129.30
16	Лучшие из лучших. Футбол 2006 (Jewel)		шт	1					2.000	60.02	120.03	18%	21.61	141.64
17	Проклятые Земли: Заторяйные в Астрале		шт	1					6.000	97.13	582.76	18%	104.90	687.66
					Итого				75.000	X	6'299.75	X	1'133.96	7'433.71
					Всего по накладной				75.000	X	6'299.75	X	1'133.96	7'433.71

Товарная накладная имеет приложение на 1 листе и содержит Семнадцать порядковых номеров записей

прописью

Всего мест — прописью — Масса груза (нетто) — Масса груза (брутто) — прописью — прописью

Приложение (паспорта, сертификаты и т.п.) на — прописью — листах

Всего отпущено Семнадцать наименований товаров на сумму Семь тысяч четыреста тридцать три рубля 71 копейка

По доверенности № — от — выданной — кем, кому (организация, должность, фамилия, и. о.)

Отпуск разрешил — подпись — расшифровка подписи

Главный (старший) бухгалтер — подпись — Карпова Л.В. — расшифровка подписи

Отпуск груза произвел — подпись — расшифровка подписи

М.П. — "__" ____ 20__ года

Груз принял — должность — подпись — расшифровка подписи

Груз получил грузополучатель — должность — подпись — расшифровка подписи

М.П. — "__" ____ 20__ года

Общество с ограниченной ответственностью "1С:Мультимедиа" Вологда * Россия * г. Вологда *

Рис. 3.1. Товарная накладная

Сравнительный состав реквизитов обоих документов показан на рис. 3.3.

Эти два документа с грузом приходят вместе, хотя имеют разные пункты назначения внутри предприятия. В обоих документах находятся сведения об одних и тех же товарах. Поэтому, чтобы не повторять ввод по одним и тем же товарам (а их может быть в документах много) и учитывая небольшое расхождение по реквизитам в обоих документах, имеет смысл для этих документов сделать общий макет ввода данных и хранить сведения о поступивших товарах в одном экземпляре.

Приложение №1
к Правилам ведения журналов учета полученных и выставленных счетов-фактур, книг покупок и книг продаж при расчетах по налогу на добавленную стоимость, утвержденным постановлением Правительства Российской Федерации от 2 декабря 2000 г. N 914 (в редакции постановлений Правительства Российской Федерации от 15 марта 2001 г. N 189, от 27 июля 2002 г. N 575, от 16 февраля 2004 г. N 84)

СЧЕТ-ФАКТУРА № 326 от 14.02.06

Продавец: "ООО "1С:Мультимедиа. Вологда""
Адрес: 160001, Вологда, Челюскинцев, д. 40
ИНН/КПП продавца: 3525148323
Грузоотправитель и его адрес: Он же
Грузополучатель и его адрес: "ООО "Виконт"", Адрес: Череповец, Энгельса, д. 28
К платежно-расчетному документу от . .
Покупатель: "ООО "Виконт""
Адрес: Череповец, Энгельса, д. 28
ИНН/КПП покупателя: 3528074084/352801001

Валюта: руб.

Наименование товара (описание выполненных работ, оказанных услуг)	Единица измерения	Количество	Цена (тариф) за единицу измерения	Стоимость товаров (работ, услуг), всего без налога	В том числе акциз	Налоговая ставка	Сумма налога	Стоимость товаров (работ, услуг), всего с учетом налога	Страна происхождения	Номер грузовой таможенной декларации
1	2	3	4	5	6	7	8	9	10	11
1С:Коллекция игрушек "GTR"	шт	10	97.09	970.93	--	18%	174.77	1'145.70	Россия	
1С:Кол. игрушек "Вивисектор. Зверь внутри"	шт	4	96.65	386.61	--	18%	69.59	456.20	Россия	
1С:Кол. игрушек "Дальнобойщики 2.Изд.2-е дополнен."	шт	4	97.13	388.51	--	18%	69.93	458.44	Россия	
1С:Коллекция игрушек "Магия крови"	шт	6	97.13	582.76	--	18%	104.90	687.66	Россия	
1С:Коллекция игрушек "Перл-Харбор"	шт	2	97.13	194.25	--	18%	34.97	229.22	Россия	
Crazy Frog Racer	шт	4	61.48	245.90	--	18%	44.26	290.16	Россия	
The Settlers. Наследие королей (PC-DVD) (Jewel)	шт	4	105.23	420.92	--	18%	75.76	496.68	Россия	
The Suffering: Ties That Bind (Jewel)	шт	4	136.95	547.80	--	18%	98.60	646.40	Россия	
Битвы тяжеловесов (Jewel)	шт	5	56.38	281.91	--	18%	50.74	332.65	Россия	
Зимние игры. Турин 2006 (Jewel)	шт	10	61.48	614.75	--	18%	110.65	725.40	Россия	
Золотой теленок	шт	4	106.87	427.49	--	18%	76.95	504.44	Россия	
Как достать соседа (Jewel)	шт	4	51.60	206.41	--	18%	37.15	243.56	Россия	
Кузя. В шалаше (Jewel)	шт	2	54.79	109.58	--	18%	19.72	129.30	Россия	
Кузя. На абордаж (Jewel)	шт	2	54.79	109.58	--	18%	19.72	129.30	Россия	
Кузя. Подводное сафари (Jewel)	шт	2	54.79	109.58	--	18%	19.72	129.30	Россия	
Лучшие из лучших. Футбол 2006 (Jewel)	шт	2	60.02	120.03	--	18%	21.61	141.64	Россия	
Проклятые Земли: Затерянные в Астрале	шт	6	97.13	582.76	--	18%	104.90	687.66	Россия	
Всего к оплате							1'133.96	7'433.71		

Руководитель организации _____ /Каранов А.Л./ (подпись) (Ф.и.о)
Главный бухгалтер _____ /Каранова Л.В./ (подпись) (Ф.и.о)
Индивидуальный предприниматель _____ _____ (подпись) (Ф.и.о)
(реквизиты свидетельства о государственной регистрации индивидуального предпринимателя)

ПРИМЕЧАНИЕ. Первый экземпляр - покупателю, второй экземпляр - продавцу

Рис. 3.2. Счет-фактура

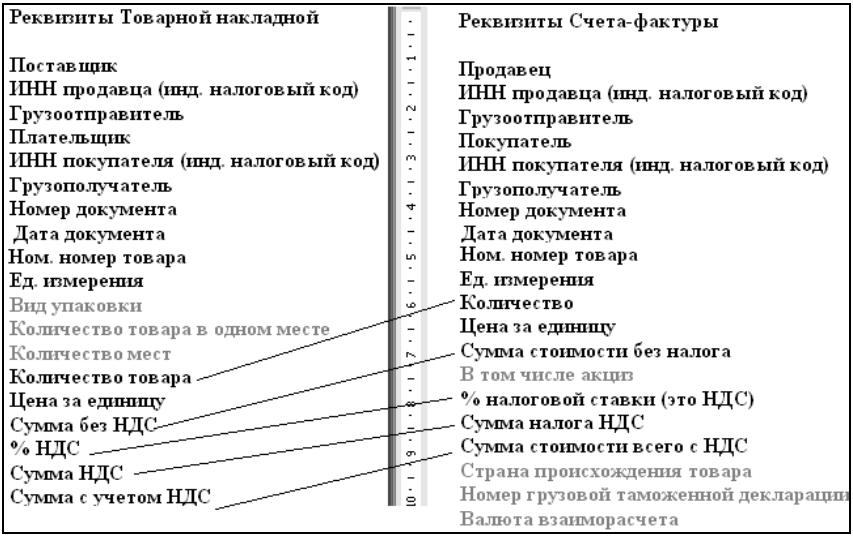


Рис. 3.3. Сравнительный состав реквизитов товарной накладной и счета-фактуры

И информационные нужды бухгалтерии можно удовлетворить — по каждому товару станут храниться данные, складу не необходимые, но полезные в целом для предприятия. Поступление на склад обозначается бухгалтерской операцией "Приход".

На склад могут поступать материалы из собственных подразделений предприятия (например, готовая продукция или остатки неиспользованных материалов). Такие материалы сопровождаются документом "Накладная на внутреннее перемещение" (приход). Но склад не только принимает товар и материалы, но также и отпускает их. Такая операция называется "Расход". Отпуск товаров и материалов со склада осуществляется также на основе определенных документов. Для склада они также являются входными документами. При внутреннем перемещении сырья и материалов это такие документы, как "Накладная на внутреннее перемещение" (на расход), "Требование на отпуск материала со склада", "Лимитно-заборная карта". Данные по этим документам вводятся в компьютер, и на их основе модифицируются данные по материалам, хранящимся на складе. Когда же со склада надо выдать товар стороннему покупателю, и в этой ситуации предприятие выступает в роли поставщика, оно готовит для покупателя сопроводительную документацию в виде товарной накладной и счета-фактуры. В этом плане при разработке задачи можно поступить двояко: довести дело до полной автоматизации, когда эти документы будут выписываться на компьютере в рамках решения задачи, а можно остановиться на неполной автоматизации, когда товарная накладная и счет-фактура поступят на склад уже составленными, а на складе они станут обрабатываться точно так же, как это делалось с аналогичными документами от сторонних поставщиков. Это намного упрощает проблему. В учебных целях мы так и поступим: нашей целью является показать подходы к решению проблемы складского учета с точки зрения применения баз данных и среды Borland C++Builder.

На складе ведется так называемая "Карточка складского учета", данные которой служат источником для ввода остатков материалов при внедрении компьютерной обработки информации и для работы с остатками при текущей работе склада. Кроме того, на складе периодически проводится инвентаризация: проверка фактического наличия материалов на данный период. Результаты проверки отражаются в специальной инвентаризационной описи, которая может служить входным документом для корректировки остатков материалов: данные фактического наличия материала и его учетные данные могут не совпадать по тем или иным причинам.

При проектировании машинной обработки документов следует стремиться к их унификации, чтобы, по возможности, снизить количество обрабатываемых форм документов и не усложнять тем самым технологический процесс. На основании всего сказанного можно следующим образом определить

структуру входных документов склада (каждый входной документ позже превратится в таблицу базы данных):

- ❒ накладная на приход-расход материала при его внутреннем перемещении, требование на отпуск материала со склада и лимитно-заборная карта:
  - код склада;
  - номер документа;
  - дата документа;
  - код вида документа (накладная, требование);
  - тип документа (приходный/расходный);
  - код поставщика/получателя;
  - номенклатурный номер товара (материала);
  - количество;
- ❒ остатки материалов:
  - код склада;
  - номер документа;
  - дата документа;
  - код вида документа (инвентаризационная опись, учетная карточка);
  - номенклатурный номер товара (материала);
  - количество;
- ❒ товарная накладная — счет-фактура:
  - код поставщика;
  - ИНН (индивидуальный налоговый номер) поставщика;
  - код грузоотправителя;
  - код плательщика;
  - ИНН плательщика;
  - код грузополучателя;
  - номер товарной накладной;
  - дата товарной накладной;
  - номер счета-фактуры;
  - дата счета-фактуры;
  - страна происхождения товара;
  - номер таможенной декларации;
  - тип документа (приходный, расходный);

- номенклатурный номер товара;
- расчетные характеристики товара;
- количество.

При вводе этого документа могут быть два случая: сведения о полученном товаре уже существуют в справочнике материалов, и наоборот: товар получен впервые, и сведения о нем следует занести в справочник материалов. Рассмотрим оба этих случая.

Допустим, сведения о полученном товаре уже существуют в справочнике материалов. Это значит, что по введенному с документа номенклатурному номеру в справочнике материалов находится соответствующая запись, из которой можно извлечь характеристики товара и сравнить их с теми, которые приведены в сопроводительном документе. Это такие характеристики, как наименование товара, единица измерения, денежная единица расчета, цена приобретения, страна-производитель, перечень налогов, которыми облагается данный товар и минимальный запас товара на складе (используется при выписке товара сторонним организациям). По каждому несоответствию бухгалтерией предприятия принимается решение о внесении/невнесении корректировок в справочник материалов. Кроме этого, в момент ввода документов следует рассчитать суммы в денежном выражении с учетом налогов, чтобы дать возможность сравнить расчетные суммы с теми, которые указаны во вводимом документе. Вопрос об устранении причин несовпадения сумм решается бухгалтерией. Все эти характеристики товара, полученные на основании расчетов с использованием справочников, запоминаются в поле документа "Расчетные характеристики товара": если документ приходный, то они станут использоваться при дальнейшей обработке документа без повторения расчетов. Если же документ расходный, то они используются для печати товарной накладной и счета-фактуры.

Если товар получен впервые, и сведения о нем следует занести в справочник материалов, оператор, осуществляющий ввод документа, должен перейти в режим корректировки справочников и занести в соответствующие справочники необходимые данные, находящиеся во вводимом документе, и возвратиться на продолжение ввода документа.

Выходные документы

Для бухгалтерии составляется так называемый "Приходно-расходный ордер" — документ, обеспечивающий учет поступивших на склад материалов. А для самого склада формируется учетный документ "Карточка складского учета", которая заводится на каждый материал, хранящийся на складе, и отражающая состояния этого материала: приход, расход, остатки, от кого и когда поступил, кому и когда отпущен, и на основании каких документов с мате-

риалом производились соответствующие операции. Структура документов такова:

- ❒ приходно-расходный ордер:
  - код склада;
  - номер документа;
  - дата документа;
  - код операции;
  - код получателя;
  - код поставщика;
  - наименование поставщика;
  - номер товарной накладной;
  - номенклатурный номер материала;
  - наименование материала;
  - наименование единицы измерения;
  - количество товара по документу;
  - количество товара, принятое на склад;
  - наименование денежной единицы расчета;
  - налоги, рассчитанные для данного товара;
  - цена покупки (продажи) товара;
  - сумма по всем налогам;
  - сумма по товару без учета налогов;
  - сумма по товару с учетом налогов;
- ❒ карточка складского учета материала:
  - код склада;
  - номенклатурный номер материала;
  - характеристики материала;
  - дата документа-основания для перемещения материала;
  - номер документа-основания для перемещения материала;
  - тип операции (приход-расход);
  - код поставщика (получателя);
  - остаток материала на начало перемещения;
  - остаток материала после перемещения;
  - количество поступившего (отпущенного) материала.

При расчете приходно-расходного ордера задается диапазон дат поступления материалов на склад. Входом для документа служат накладная на внутреннее перемещение и товарная накладная. Обе с типом документа "Приход". Таким же порядком можно рассчитать и расходную часть ордера, если изменить выборку на документы с типом "Расход".

Для расчета Карточки складского учета (КСУ) входом служат остатки материалов на складе, сама КСУ за предыдущий период и приходные документы на склад. Карточка рассчитывается после движения данного товара (материала) на складе: после поступления товара на склад и/или после его отпуска со склада. Расчет КСУ происходит в диапазоне дат. Если в этом диапазоне дат новых сведений в остатке по данному номенклатурному номеру нет, то остаток не включаются в КСУ. Сама КСУ будет находиться в таблице-картотеке.

Создание базы данных и таблиц

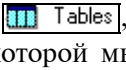
Механизм создания базы данных и таблиц

Базу данных для обработки информации по складам создадим с помощью утилиты IBConsole в окне интерактивного SQL. Для этого запустим утилиту, зарегистрируем сервер (Server/Login), ответив на запрос пароля БД вводом значения `masterkey`. При этом сервер запустится, мы перейдем в окно интерактивного SQL, нажав кнопку  , и введем в окно команды:

```
CREATE DATABASE "d:\Interbase\Databases\Store.gdb"
USER "sysdba"
PASSWORD "masterkey"
;
```

База данных с именем Store создастся. Если теперь посмотреть на базы данных нашего сервера, то увидим то, что показано на рис. 3.4.

Перед созданием таблиц надо подключиться к созданной базе данных (чтобы в ней создавать таблицы). Для этого следует щелкнуть на строке с именем нужной БД (выделить) и нажать на кнопку  на панели IBConsole. При этом получим результат, показанный на рис. 3.5.

При этом высвечиваются входы в составные части подключенной базы данных: в домены, таблицы, индексы, вьюеры, хранимые процедуры и т. д. Если нажать, например, на значок  , то получим список только системных таблиц базы данных, к которой мы подключены на сервере: своих таблиц мы еще не сформировали (рис. 3.6).

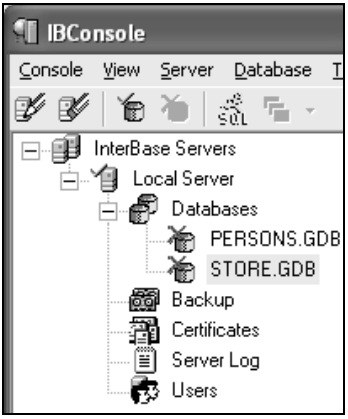


Рис. 3.4. Базы данных сервера

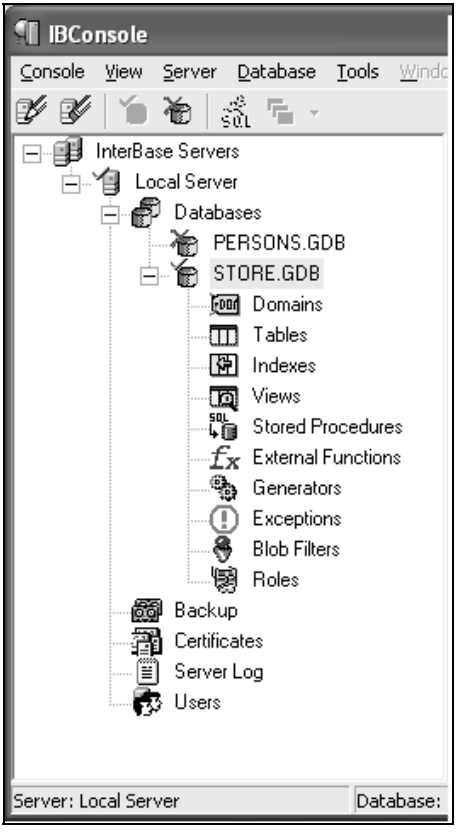


Рис. 3.5. Подключение к базе данных Store

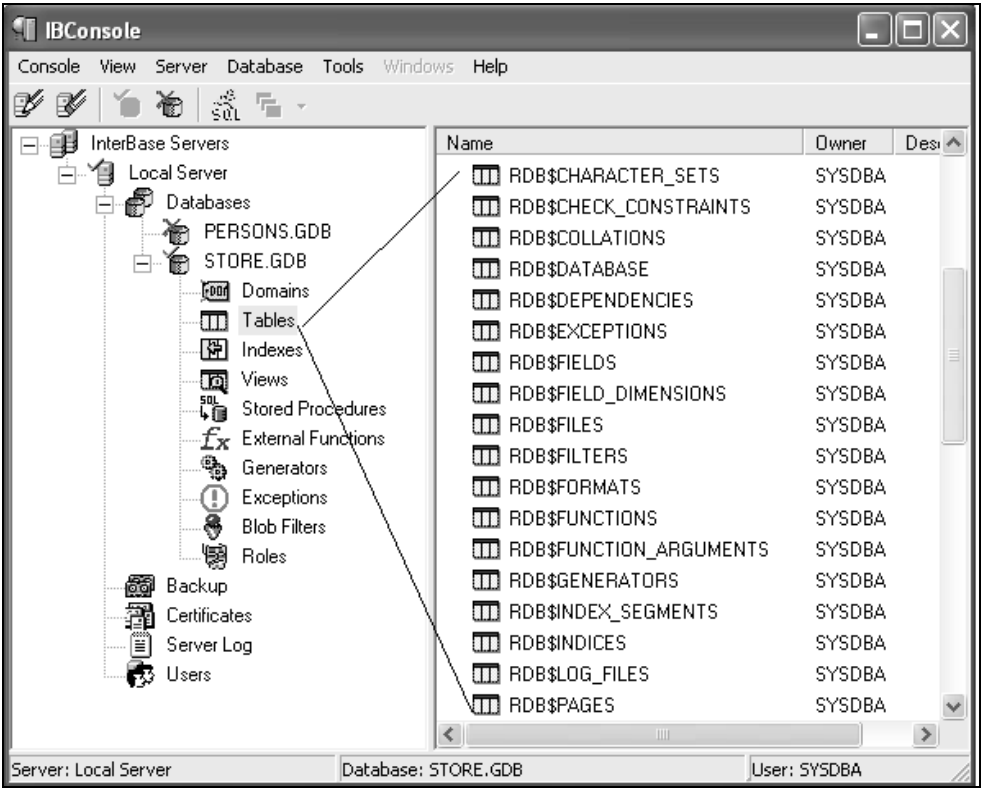


Рис. 3.6. Список системных таблиц базы данных Store

Напишем на SQL программы создания таблиц для рассмотренных уже справочников и входных-выходных документов и поместим их в файл с помощью редактора "Блокнот", откуда их удобно вставлять в окно IBConsole. Программы создания таблиц приведены в листинге 3.1.

Листинг 3.1

```
/* справочник материалов*/
Create table Materiali
 (
  NN integer not null,
   NaimMater VARCHAR(50) not null,
   EdIzm integer not null,
   StructuraCoda VARCHAR(20),
```

```
   BalSch integer not null,
   MinZapas float,
   DenEdRasch integer,
   CodCeniPocup integer,
   CodCeniProd integer,
   CodSpiscaNalogov integer,
   CodStraniProizv integer,
   PRIMARY KEY (NN)
);

/*справочник единиц измерения материала*/
Create table EdIzm
 (
  Cod integer not null,
  NaimEdIzm VARCHAR(10),
  CodVDr integer,
  KoefVDr float,
  PRIMARY KEY (Cod)
 );


/*справочник материалов, относящихся к первому уровню группировки*/
Create table Ur1
 (
  Cod VARCHAR(4) not null,
  NaimUr1 VARCHAR(30),
  PRIMARY KEY (Cod)
 );

Create table Ur2
 (
  Cod VARCHAR(4) not null,
  NaimUr2 VARCHAR(30),
  PRIMARY KEY (Cod)
 );
```

```
Create table Ur3
 (
  Cod VARCHAR(4) not null,
  NaimUr3 VARCHAR(50),
  PRIMARY KEY (Cod)
 );

Create table Ur4
 (
  Cod VARCHAR(4) not null,
  NaimUr4 VARCHAR(70),
  PRIMARY KEY (Cod)
 );

Create table Ur5
 (
  Cod VARCHAR(4) not null,
  NaimUr5 VARCHAR(90),
  PRIMARY KEY (Cod)
 );

/*Справочник балансовых счетов*/
create table BalSch
 (
  CodSch integer NOT NULL,
  NaimSch VARCHAR(60),
  PRIMARY KEY (CodSch)
);

/*Справочник денежных единиц*/
create table DenEd
 (
   Cod integer NOT NULL,
  Naim VARCHAR(25),
  Znak BLOB,
  PRIMARY KEY (Cod)
 );
```

```
/*Справочник цен приобретения материалов*/
CREATE TABLE CeniPocup
(
 CodDenEd integer NOT NULL,
 CodCeniPocup integer  NOT NULL,
 CenaPocup FLOAT NOT NULL,
 PRIMARY KEY (CodCeniPocup)
);

/*Справочник цен отпуска материалов*/
CREATE TABLE CeniProdaj
(
 CodDenEd integer  NOT NULL,
 CodCeniProd integer  NOT NULL,
 CenaProd FLOAT NOT NULL,
 PRIMARY KEY (CodCeniProd)
);

/*Справочник списков видов налогов*/
CREATE TABLE SpisciVidovNalogov
(
 CodSpisca integer NOT NULL,
 NomerVSpisce integer NOT NULL,
 NaimVSpisce VARCHAR(30),
 ProcStavcaVSpisce FLOAT NOT NULL,
 OgranSumma FLOAT,
 PRIMARY KEY (CodSpisca, NomerVSpisce)
);

/*Справочник стран*/
CREATE TABLE Strani
 (
  Cod integer NOT NULL,
  Naim VARCHAR(50),
 PRIMARY KEY (Cod)
 )
```

```
/*Справочник поставщиков-покупателей*/
CREATE TABLE PostPocup
 (
  Cod  INTEGER NOT NULL,
  NAIM VARCHAR(150),
 IurAdres VARCHAR(150),
 FactAdres VARCHAR(150),
 Telefoni  VARCHAR(50),
  Facs VARCHAR(15),
  Email VARCHAR(25),
  Banc VARCHAR(150),
  RaschSch VARCHAR(50),
 CorrSch VARCHAR(50),
 Director VARCHAR(25),
 GlBuh VARCHAR(25),
 Contact VARCHAR(30),
 CodValDliaRasch integer,
 Prochee VARCHAR(50),
PRIMARY KEY (Cod)
 );

/*Справочник курсов валют */
create table CursValut
 (
   CodValuti integer  NOT NULL,
   NaimValuti VARCHAR(15),
   CursV FLOAT NOT NULL,
   DataIzm Date NOT NULL,
   PRIMARY KEY (CodValuti )
 );

/*Справочник видов документов: накладная, счет-фактура,...*/
create table VidiDocum
 (
   Cod integer  NOT NULL,
   Naim VARCHAR(35),
```

```
   PRIMARY KEY (Cod)
 );

/*Справочник типов документов: приход, расход, остатки*/
create table TipiDoc
 (
   Cod integer  NOT NULL,
   Naim VARCHAR(15),
   PRIMARY KEY (Cod)
 );


/*Накладная на внутреннее перемещение*/
create table VxDocVnPer
 (
   CodSclada integer NOT NULL,
   CodVidaDoc  integer NOT NULL,
   NomDoc  VARCHAR(15) NOT NULL,
   DataDoc Date NOT NULL,
   TipDoc char(1) NOT NULL,
   CodPost  integer NOT NULL,
   NN float  NOT NULL,
   Characteristics VARCHAR(450), /*до 15-ти характеристик товара*/
   KolPoDocum  float,
   KolPriniato float,
   PRIMARY KEY (CodSclada,NomDoc,DataDoc, TipDoc, CodPost,NN),
       FOREIGN KEY (NN) REFERENCES materiali (NN)
       ON DELETE CASCADE
 );

/*Остатки*/
create table Ostatci
 (
   CodSclada integer NOT NULL,
   CodVidaDoc  integer NOT NULL,
   NomDoc  VARCHAR(15) NOT NULL,
```

```
   DataDoc Date NOT NULL,
   TipDoc char(1) NOT NULL,
   NN float NOT NULL,
   Characteristics VARCHAR(450), /*до 15-ти характеристик товара*/
   KolPoDocum float,
   KolPriniato float,
   PRIMARY KEY (CodSclada,NomDoc,DataDoc, CodVidaDoc,NN),
     FOREIGN KEY (NN) REFERENCES materiali (NN)
       ON DELETE CASCADE
 );

/*Товарная накладная счет-фактура*/
create table TovNakl
 (
   CodSclada integer NOT NULL,
   CodVidaDoc integer NOT NULL,
   CodPost integer  NOT NULL,
   InnPost float,
   CodGruzOtpr integer,
   CodPlat integer,
   InnPlat float,
   CodGruzPoluch integer,
   NomDoc  VARCHAR(15) NOT NULL, /*накладной*/
   DataDoc date NOT NULL,
   NomDocFact  VARCHAR(15) NOT NULL,
   DataDocFact date NOT NULL,
   Strana integer,
   NomTamojDecl VARCHAR(20),
  TipDoc char(1)  NOT NULL,
   NN float NOT NULL,
   Characteristics VARCHAR(450), /*до 15-ти характеристик товара*/
   KolPoDocum float,
   KolPriniato float,
   PRIMARY KEY (CodSclada,NomDoc,NN),
    FOREIGN KEY (NN) REFERENCES materiali (NN)
       ON DELETE CASCADE
 );
```

```
/*Приходный ордер*/
create table PrixOrder
 (
   CodSclada VARCHAR(2) ,
    NomOrdera VARCHAR(15),
    DataOrdera VARCHAR(10),
    TipDoc  VARCHAR(50),
    CodGruzPoluch VARCHAR(10),
    CodPost VARCHAR(10),
    NaimPost VARCHAR(75),
    NomNakl  VARCHAR(15),
    NN float,
    DenEd  VARCHAR(15),
    NaimMater VARCHAR(75),
    NaimEdIzm  VARCHAR(15),
    Strana VARCHAR(15),
    KolPoDocum  VARCHAR(10),
    KolPriniato VARCHAR(10),
    cena VARCHAR(10),
    nalogi VARCHAR(50),
    summaPoNal VARCHAR(50),
    summaPoTovBezNal VARCHAR(50),
    summaPoTovSNal VARCHAR(50)
   /*Таблица без первичного ключа: есть повторяющиеся строки */
 );

/*Карточка складского учета материала*/
create table CartSclUch
 (
   CodSclada integer NOT NULL,
   NN float NOT NULL,
   Characteristics VARCHAR(450),
    NomDoc VARCHAR(15) NOT NULL,
    DataDoc date NOT NULL,
   TipOper char(1) NOT NULL,
   CodPost integer  NOT NULL,
```

```
    OstNaNachalo  float,
    OstNaConec  float,
    Kol float

);
```

Откроем теперь окно интерактивного SQL (кнопка  на панели кнопок IBConsole) и вставим в него информацию листинга 3.1. Результат показан на рис. 3.7.

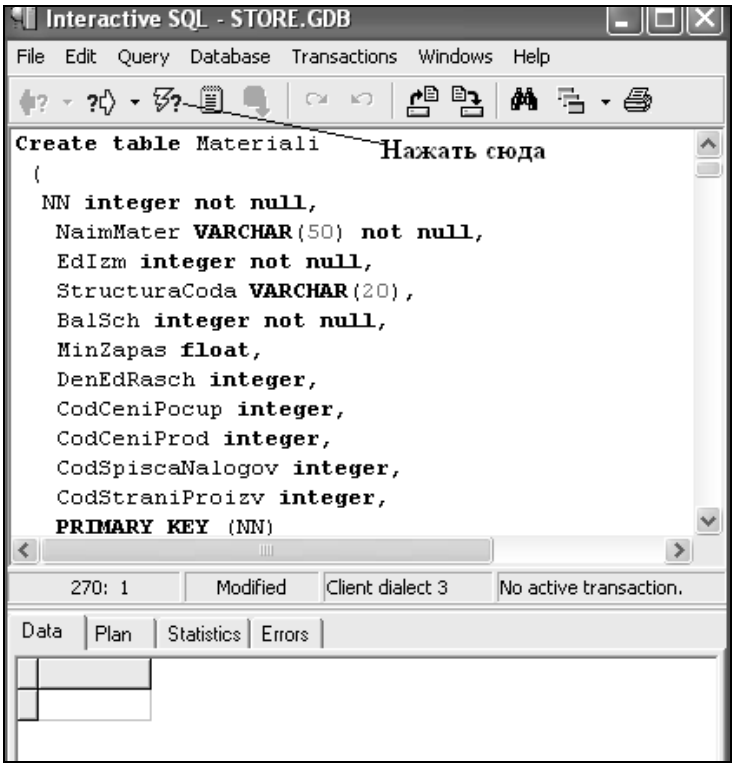


Рис. 3.7. Подготовка к созданию таблиц в базе данных

Нажмем теперь на кнопку  . Если команды создания таблиц составлены правильно, то в базе данных создадутся соответствующие таблицы, что видно из рис. 3.8.

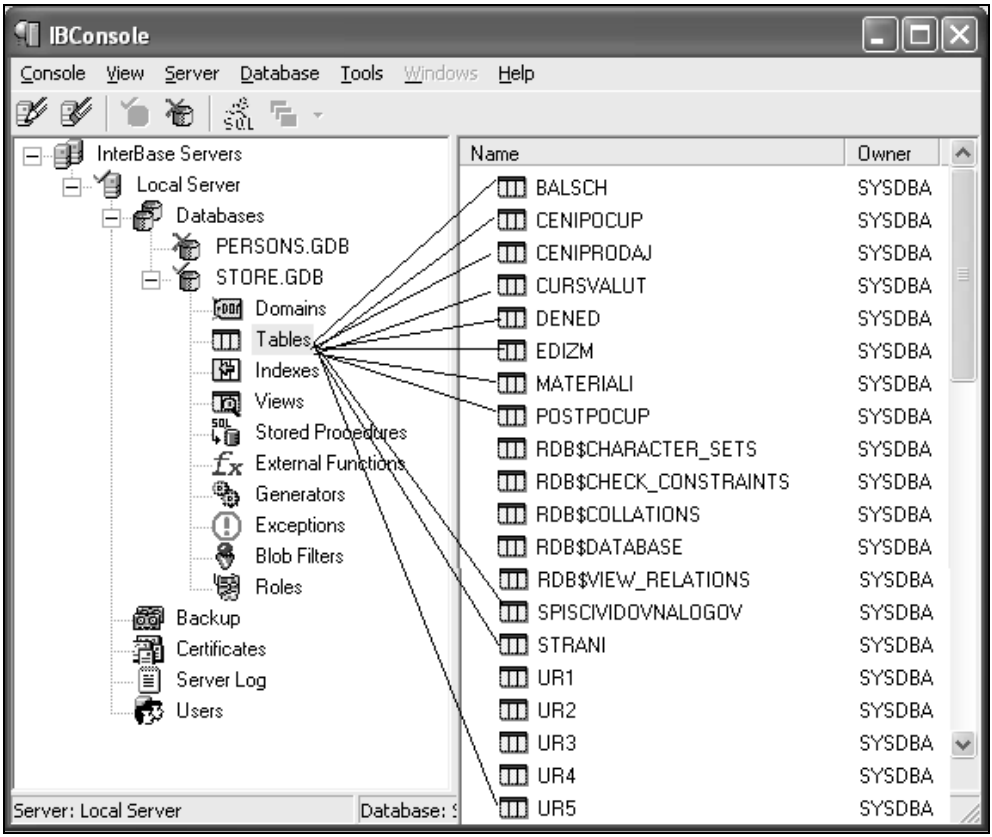


Рис. 3.8. Таблицы, созданные в БД Store

Общие принципы создания модуля ведения справочников

Создадим модуль ведения этих таблиц. Модуль будем строить на основе Interbase-компонента TIBClientDataSet среды Borland C++Builder. Этот компонент, как мы видели раньше, позволяет строить клиентское приложение и через него работать с сервером, на котором находятся наши таблицы. Общий подход для всех справочников будет одинаков: помещаем в форму экземпляры Interbase-компонентов IBDatabase1, IBTransaction1, которые обеспечат соединение с базой данных через свои свойства. Сначала определяется свойство Database Name из IBDatabase1: надо нажать на кнопку с многоточием в его поле и выбрать путь к нужной базе данных. Затем в этом же компоненте можно определить значение свойства Default Transaction: надо

щелкнуть в поле этого свойства мышью, и в нем появится кнопка выпадающего списка подключаемых компонентов для проведения транзакций. В списке будет имя установленного в форме компонента IBTransaction1. Его и надо выбрать. После этого следует определить значение свойства Default Database в компоненте IBTransaction1 таким же способом, как и для Default Transaction в предыдущем компоненте. Потом перевести свойство Connected из IBDatabase1 в значение true, чтобы подключиться к базе данных. Но как только вы попытаетесь это сделать, появится диалоговое окно с запросом на ввод пароля для доступа к БД. Мы создавали БД с именем пользователя sysdba и паролем masterkey. Их и надо ввести. После этого значение свойства **Connected** станет true, и доступ к базе данных будет разрешен. Но чтобы общаться с ней (а общение идет через компонент IBTransaction1), надо сделать активным компонент IBTransaction1, т. е. перебросить его свойство **Active** в true. Вид Инспекторов объекта для компонентов IBDatabase1 и IBTransaction1 показан на рис. 3.9.

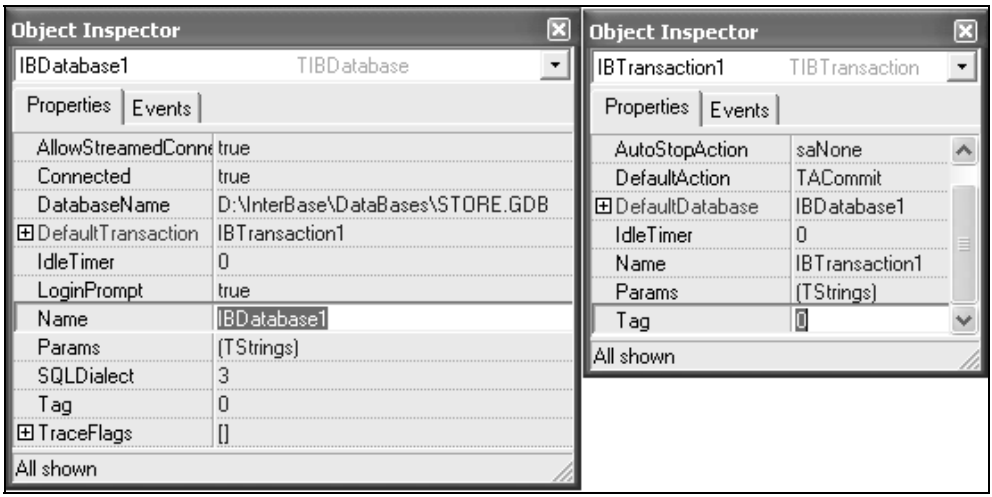


Рис. 3.9. Инспекторы объектов IBDatabase1 и IBTransaction1

Теперь можно взяться за настройку компонента TIBClientDataSet. Каждый такой компонент будет определять работу с одной таблице БД. А каждая таблица у нас представляет отдельный справочник или документ. Определим свойства DBConnection, DBTransaction конкретного экземпляра компонента, отвечающие за возможность подключения его к БД, и свойство CommandText, которое задает команду выборки данных из БД. Для работы с

таблицами мы станем выбирать все их строки, так как сами справочники будут относительно невелики. Поэтому для выборки применим оператор `select` вида

```
select * from <имя таблицы >
```

Для работы со всеми справочниками-таблицами, совокупность которых можно назвать картотекой справочников, воспользуемся компонентом TPageControl, как и в случае задачи управления персоналом, описанной в предыдущей главе. Каждая страница этого компонента будет отведена под работу с отдельным справочником. Надо будет ввести компонент TDBNavigator для управления движением по записям таблиц, а также кнопки для открытия таблиц и их обновления после их модификации.

Итак, для каждой таблицы вводим свой компонент TIBClientDataSet. После его настройки открываем правой кнопкой мыши его контекстное меню, выбираем опцию **Fields Editor**, в открывшемся окне опять нажимаем правую кнопку мыши и выбираем опцию **Add all fields**. В результате в окне увидим все поля даного компонента с заданным SQL-запросом из свойства CommandText. Опять открываем контекстное меню окна и выполняем опцию **Select all**, после чего устанавливаем курсор мыши на выделенную область всех полей и перетягиваем ее мышью в заданное место страницы. Все поля перекочуют на станицу с добавкой к каждому полю метки с название поля. Одновременно с полями автоматически на страницу попадает компонент TDataSource, настроенный на конкретный экземпляр TIBClientDataSet, из которого извлечены поля в страницу. Изменим у меток свойства Caption на необходимые нам названия. Через эти полученные поля типа DBEdit мы и станем общаться с таблицами БД — справочниками и документами. Вид таких компонентов для работы со справочником материалов показан на рис. 3.10.

При вводе изображения для денежного знака в таблице валют следует добавить в качестве промежуточного средства ввода компонент TImage и установить его свойство Stretch в значение true, а значение свойства Stretch в DBImage1, связанном с таблицей валют, также установить в true.

Следует также обеспечить подстройку свойства DataSource компонента DBNavigator1 при переходе от одной таблицы к другой, чтобы им можно было управлять текущей таблицей. Для этого надо воспользоваться событием OnShow вкладки компонента PageControl1: в обработчик этого события станем помещать команду присвоения

```
DBNavigator1-> DataSource= DataSource
```

для компонента, связанного с данной таблицей.

Например, для страницы "Денежные единицы" надо написать

```
DBNavigator1->DataSource=DataSource2;
```

Почему именно DataSource2? Да потому, что эту страницу мы формировали второй по порядку. А могли бы и пятой-десятой. Тогда бы и номер компонента-источника был соответствующий. В этом же обработчике можно поместить команды перекраски страницы.

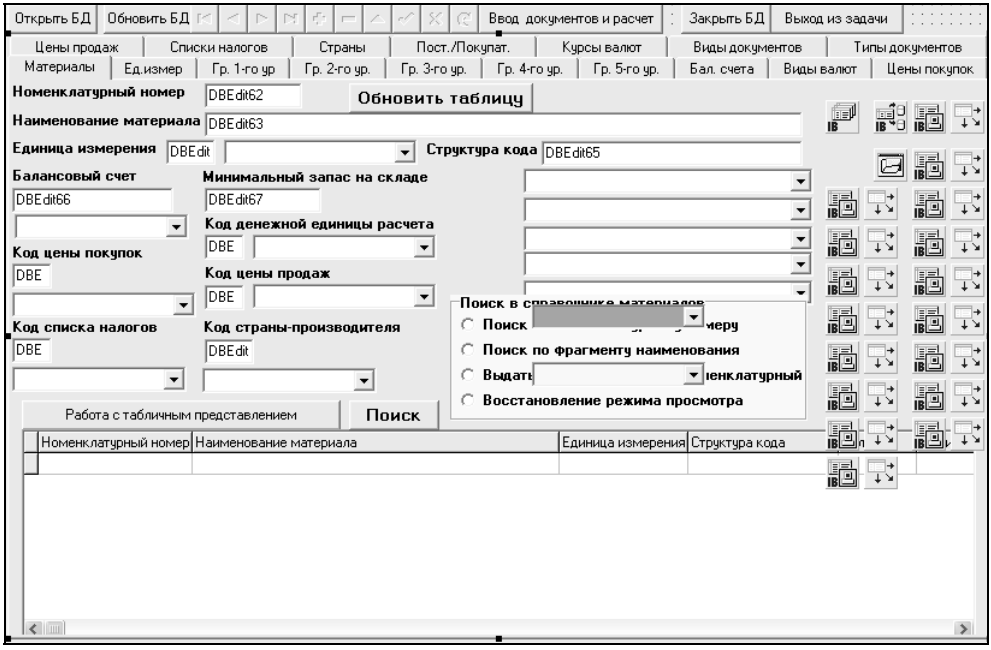


Рис. 3.10. Поля работы с таблицей материалов

Напомним, что для ввода в таблицу новой строки следует предварительно нажать на кнопку [+] навигатора, для удаления существующей строки — на кнопку [−] навигатора, чтобы модифицировать таблицу, следует нажать на кнопку [▲], после модификации таблицы следует нажать на кнопку [✓] навигатора, а уже после нее следует нажимать на кнопку обновления БД, которая обеспечивает обновление всех модифицированных таблиц картотеки.

Для работы со справочниками, требующими данных из других справочников, применяется механизм формирования этих данных в элементах типа ComboBox, откуда затем данные выбираются. Даты вносятся с помощью выборки из выпадающего календаря (компонент TDateTimePicker).

Для обслуживания результатов работы по вводу данных и расчету документов дополнительно предусмотрены выводы результатов в табличном представлении, для чего использовались компоненты TDBDrid. Некоторые из этих компонентов наделены свойствами причаливания (Docable). В частности, после щелчка на таком элементе он приобретает свойства перемещаемого и изменяемого свои размеры окна. Это сделано для таких справочников, как справочник материалов, справочник поставщиков-покупателей и др. Дело в том, что при проектировании таких справочников из-за большого количества имеющихся у них полей занята почти вся площадь рабочего стола. Чтобы видеть всю картину в момент отладки, желательно наблюдать и поля ввода, и, в целом, всю таблицу через элемент TDBDrid. А для него остается небольшое пространство где-нибудь в неудобном месте рабочего стола (см. рис. 3.10). Поэтому следует создать возможность перетаскивания мышью элемента TDBDrid в более удобное для пользователя место рабочего стола, одновременно давая возможность изменять размеры элемента.

Описание создания и функционирования элементов складского учета

Создание справочников

Вид одного из справочников — справочника материалов — в режиме проектирования был показан на рис. 3.10. Инспектор объекта экземпляра компонента TIBClientDataSet для работы со справочником и правила формирования его запроса на выборку данных из справочника показаны на рис. 3.11.

Посмотрим, как формируются элементы ввода (модификации) полей некоторого условного справочника. Проведем изучение на примере нескольких типичных полей. Все справочники задачи проектируются по одинаковому правилу, так что их детально рассматривать не станем, а только укажем на некоторые различия.

Когда мы разместили на рабочем столе поля справочника перетаскиванием их из окна (правила на рис. 3.12, 3.13), сначала следует изменить их названия на более удобные для использования. Как это делать, показано на рис. 3.14. Затем можно изменить некоторые свойства полей ввода-вывода, представленных в форме компонентами DBEdit.

Чтобы начать работать с полями ввода, сначала следует активизировать таблицу, которую представляют эти поля (в нашем случае для примера мы взяли таблицу Materiali, на которую настроен экземпляр IBClientDataSet1 ком-

понента TIBClientDataSet). То есть свойство Active из IBClientDataSet1 должно получить значение true, или должен быть выполнен оператор

```
IBClientDataSet1->Open()
```

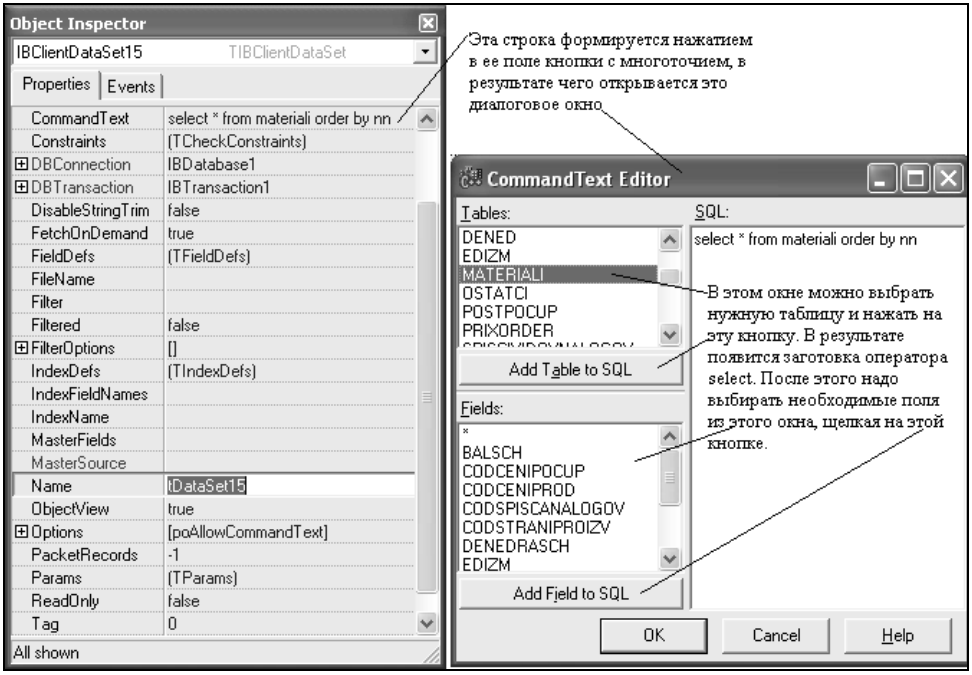


Рис. 3.11. Инспектор объекта для IBClientDataSet15 и правила формирования его свойства CommandText

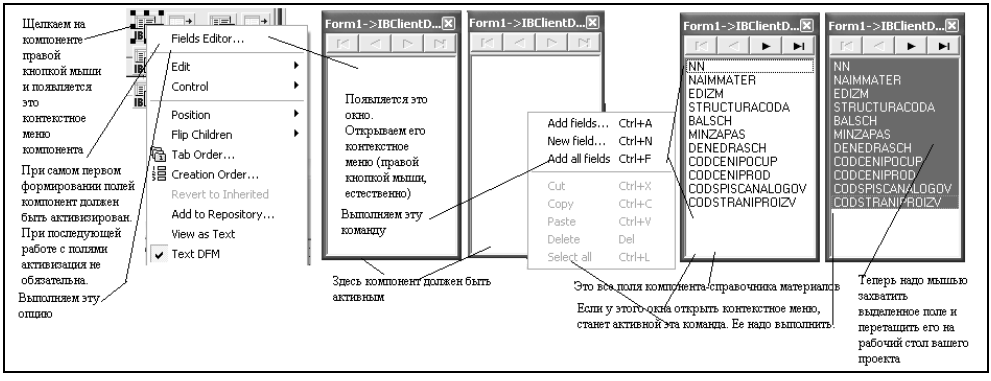


Рис. 3.12. Создание в форме полей ввода-вывода справочника. Часть 1

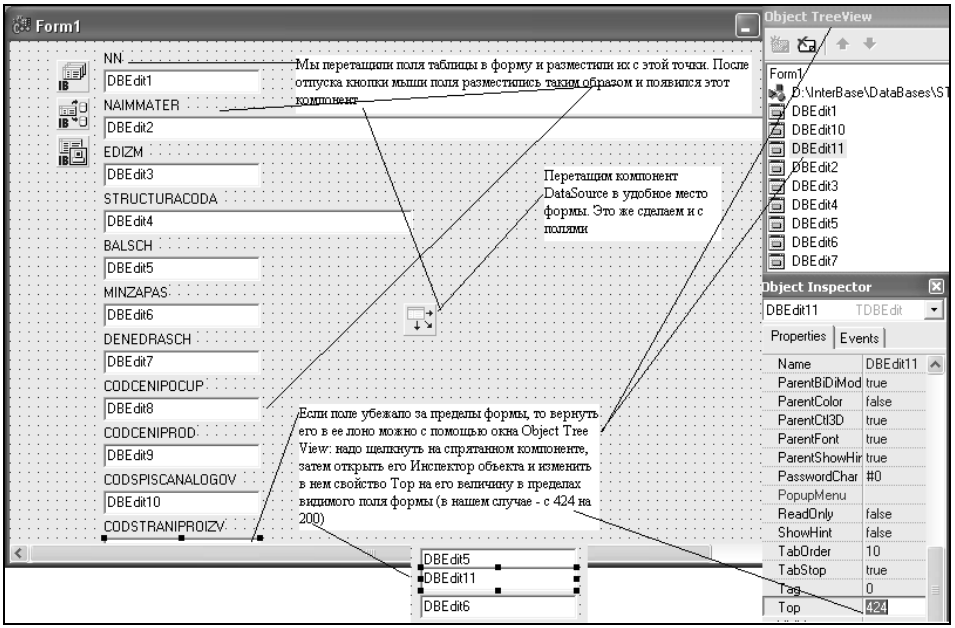


Рис. 3.13. Создание в форме полей ввода-вывода справочника. Часть 2

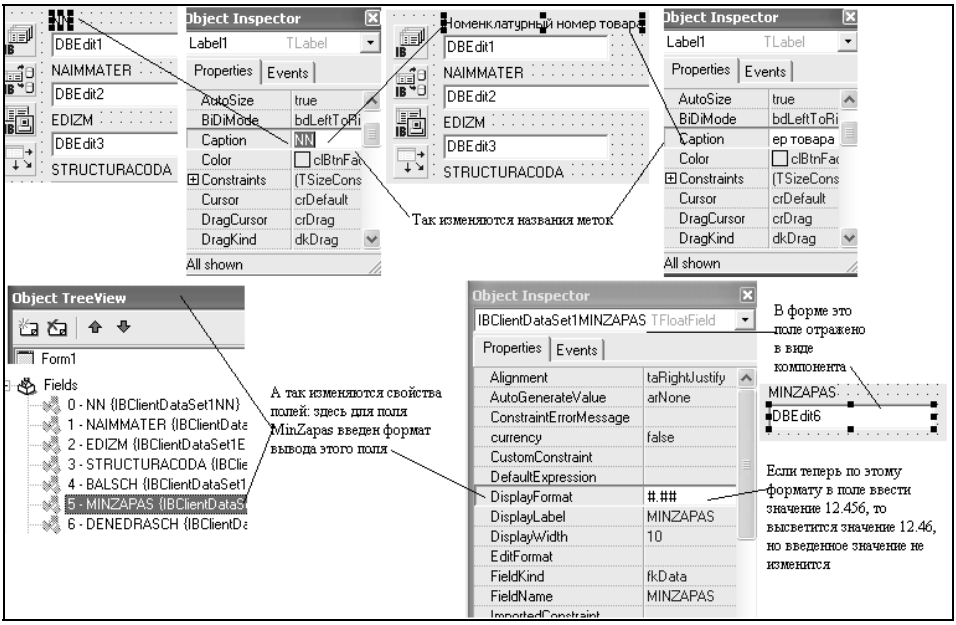


Рис. 3.14. Изменение некоторых свойств полей ввода-вывода

После активизации соответствующего поля (то ли щелчком мыши на нем, то ли выполнением команды SetFocus() в обработчике какого-то события) следует вводить в него данные.

Здесь возникают следующие вопросы.

- Если данные ввода не являются данными, которые можно почерпнуть из некоторого справочника, и их требуется просто вводить в поле с помощью клавиш клавиатуры, то как обеспечить окончание ввода данных в поле, чтобы потом автоматически перейти к вводу данных в следующее поле?
- Если данные можно почерпнуть из некоторого справочника, то как обеспечить выборку из этого справочника?

В первом случае поступают так: так как ввод идет путем нажатия на клавиши клавиатуры, то естественно было бы, чтобы ввод закончился тоже нажатием какой-то клавиши клавиатуры, например, клавиши <Enter>. Чтобы этого достичь, надо обработать событие OnKeyDown компонента, в поле которого осуществляется ввод. На рис. 3.15 это показано для условного компонента DBEdit1, который назван как номенклатурный номер товара.

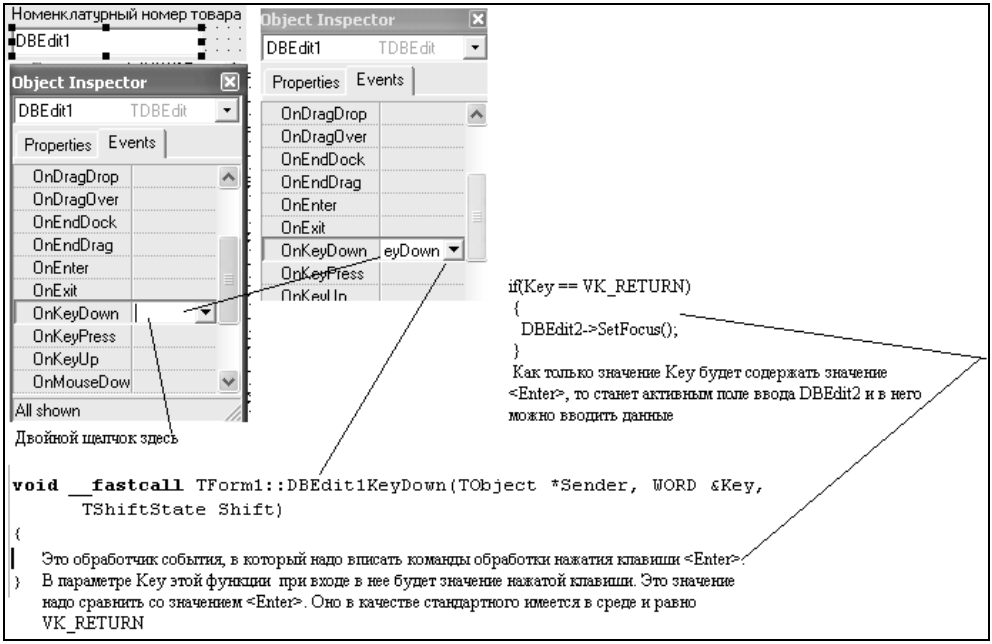


Рис. 3.15. Окончание ввода нажатием <Enter>

Во втором случае следует добиться такого положения, чтобы при входе в поле ввода активизировался компонент, в котором станут формироваться необходимые данные для выборки в поле ввода. Если, например, в поле надо вводить дату, то применяют компонент TDateTimePicker, который предоставляет возможность выбора даты.

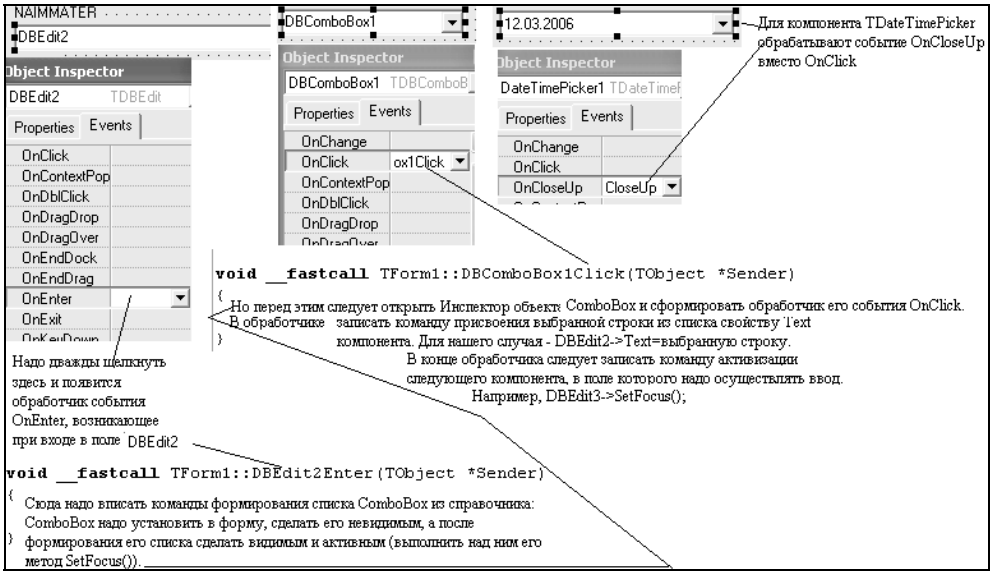


Рис. 3.16. Организация взаимодействия поля ввода и вспомогательного компонента, поставляющего информацию для поля ввода

Если в поле надо вводить некоторую строку из какого-то справочника (например, из справочника стран-производителей товара), то применяют компонент ComboBox, в котором формируют такой справочник, а затем, пользуясь его возможностями, выбирают из него необходимую строку. Организация взаимодействия поля ввода и вспомогательного компонента, обеспечивающего поставку данного в поле ввода, показана на рис. 3.16.

Работа со справочниками задачи в режиме проектирования и исполнения

Вид справочников показан на рис. 3.17—3.50.

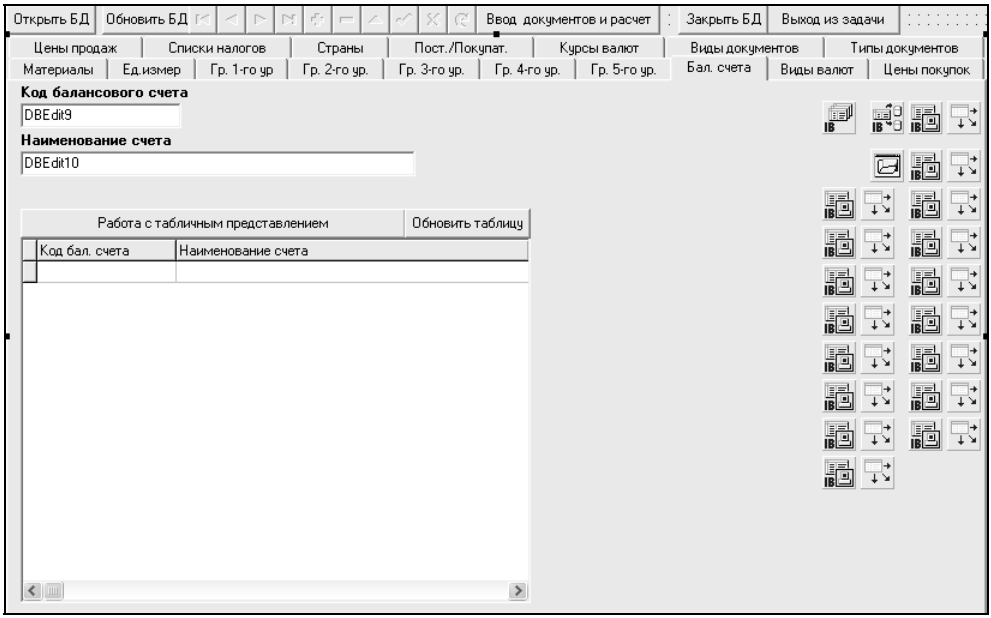


Рис. 3.17. Справочник балансовых счетов в режиме проектирования

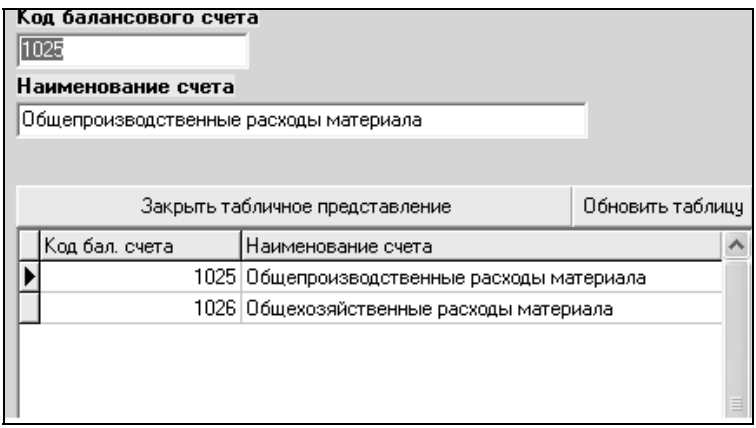


Рис. 3.18. Справочник балансовых счетов в режиме исполнения

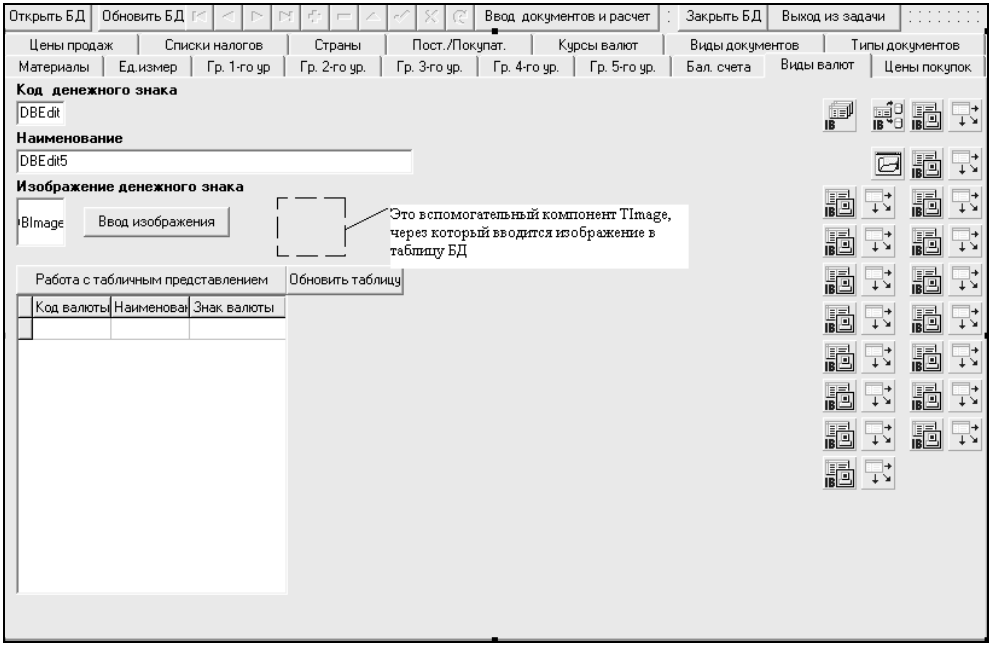


Рис. 3.19. Справочник видов валют в режиме проектирования

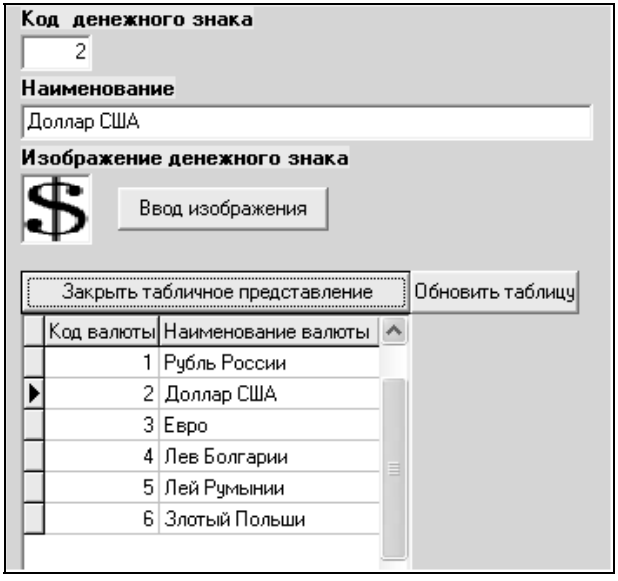


Рис. 3.20. Справочник видов валют в режиме исполнения

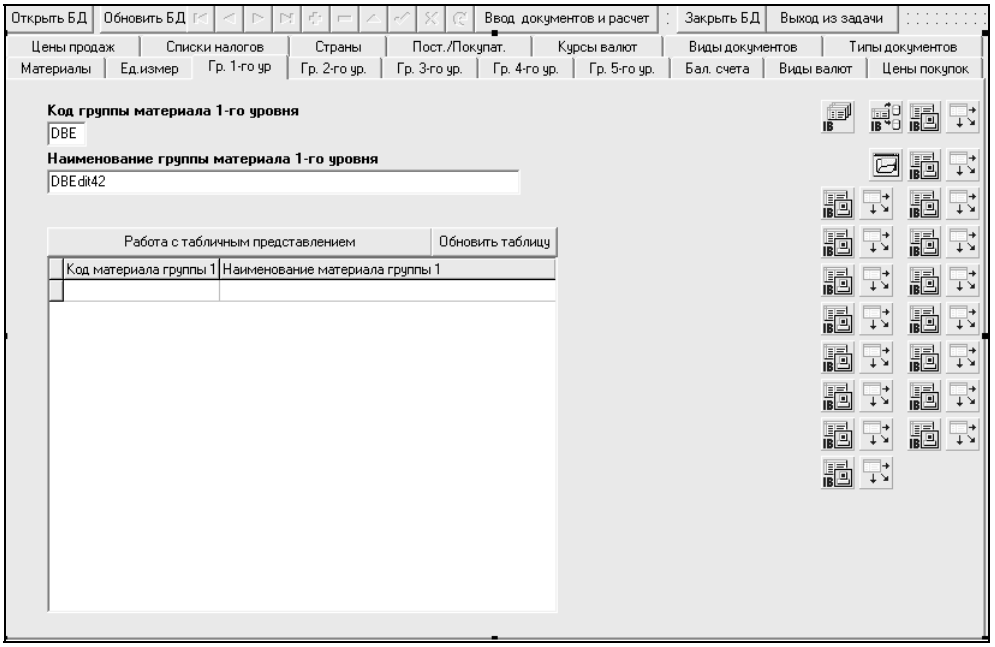


Рис. 3.21. Справочник групп товаров 1-го уровня в режиме проектирования

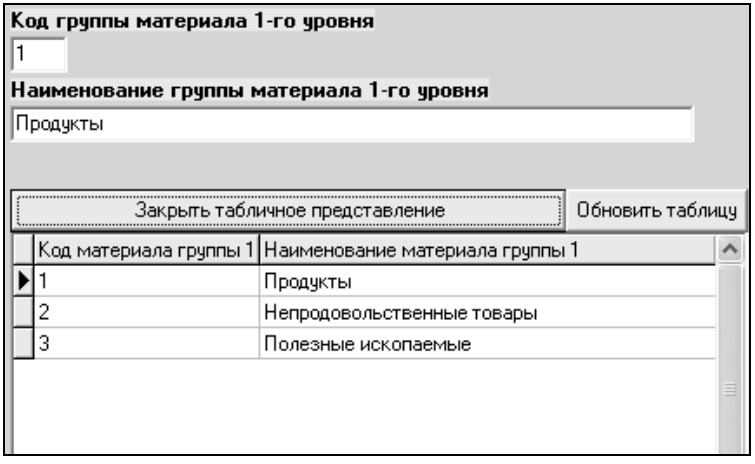


Рис. 3.22. Справочник групп товаров 1-го уровня в режиме исполнения

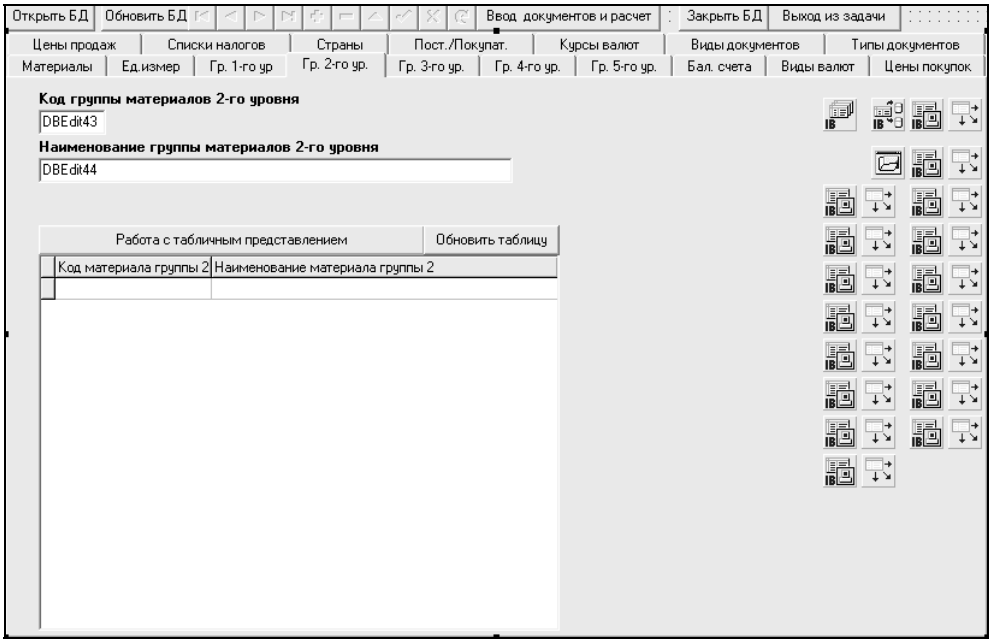


Рис. 3.23. Справочник групп товаров 2-го уровня в режиме проектирования

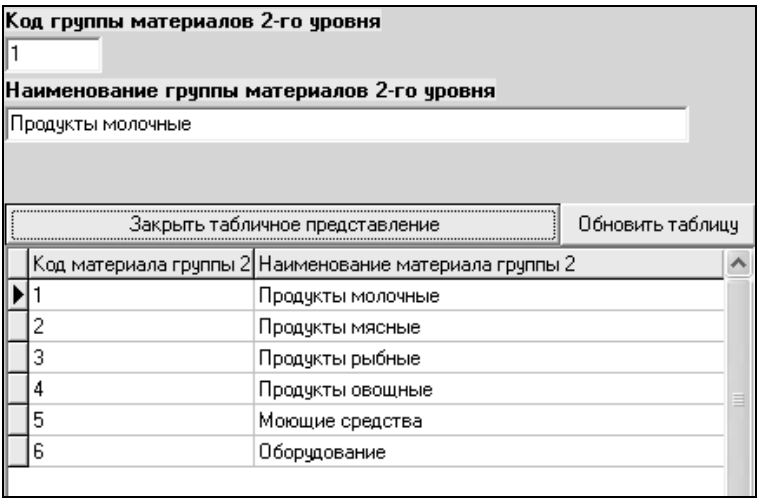


Рис. 3.24. Справочник групп товаров 2-го уровня в режиме исполнения

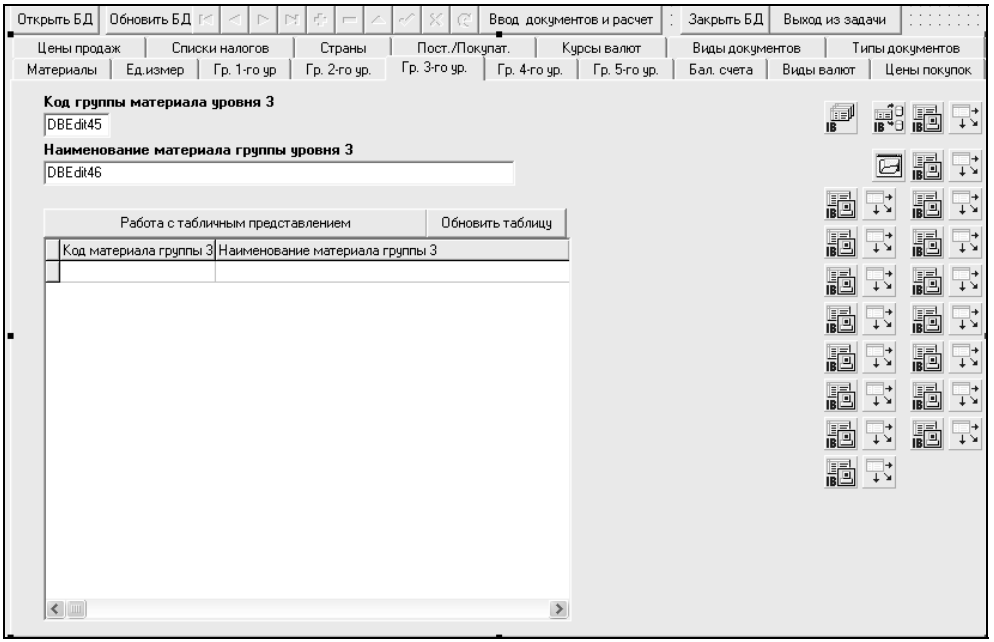


Рис. 3.25. Справочник групп товаров 3-го уровня в режиме проектирования

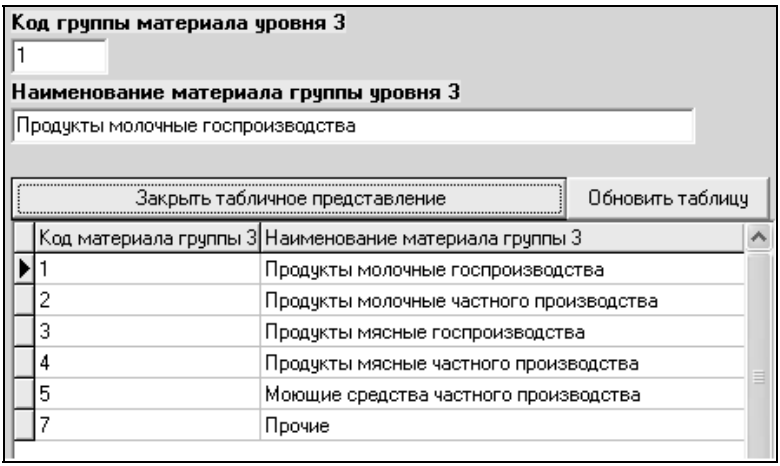


Рис. 3.26. Справочник групп товаров 3-го уровня в режиме исполнения

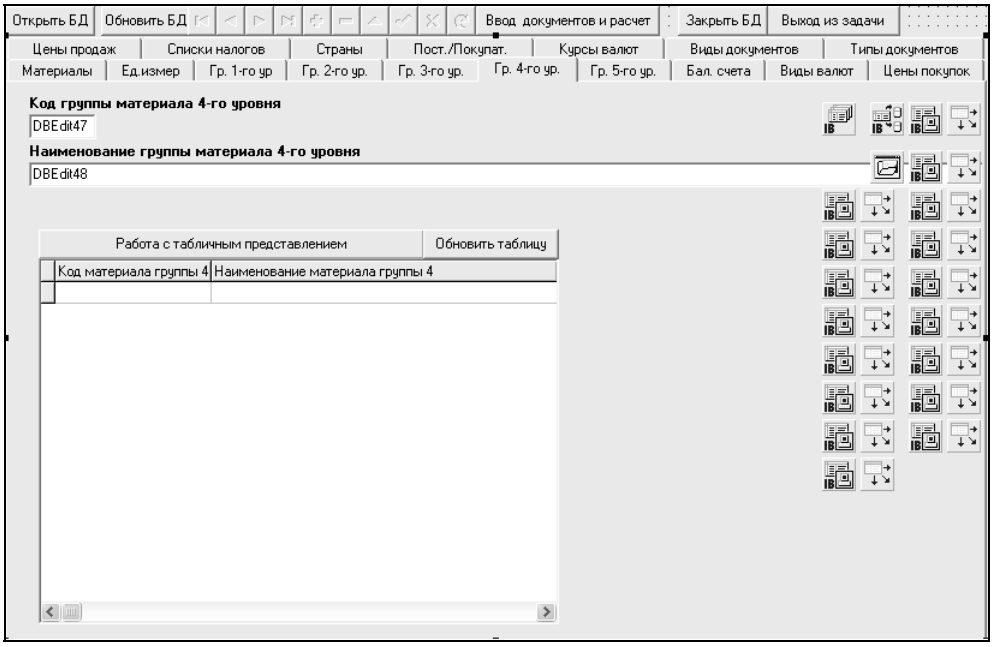


Рис. 3.27. Справочник групп товаров 4-го уровня в режиме проектирования

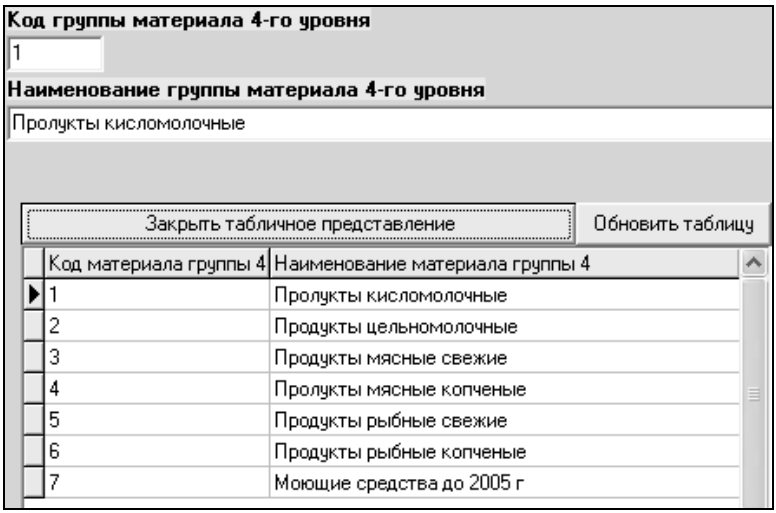


Рис. 3.28. Справочник групп товаров 4-го уровня в режиме исполнения

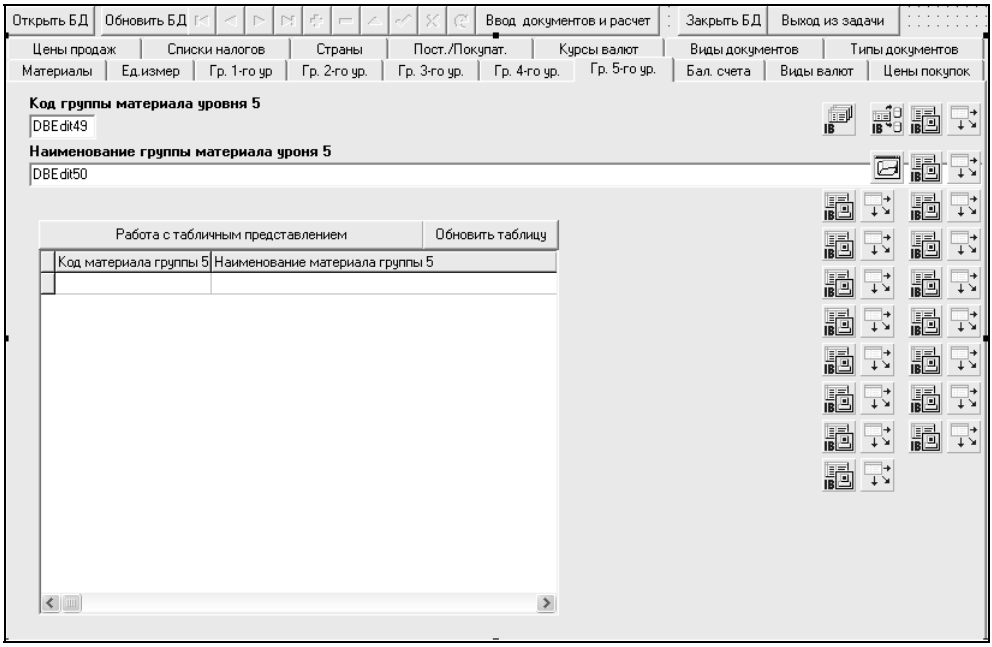


Рис. 3.29. Справочник групп товаров 5-го уровня в режиме проектирования

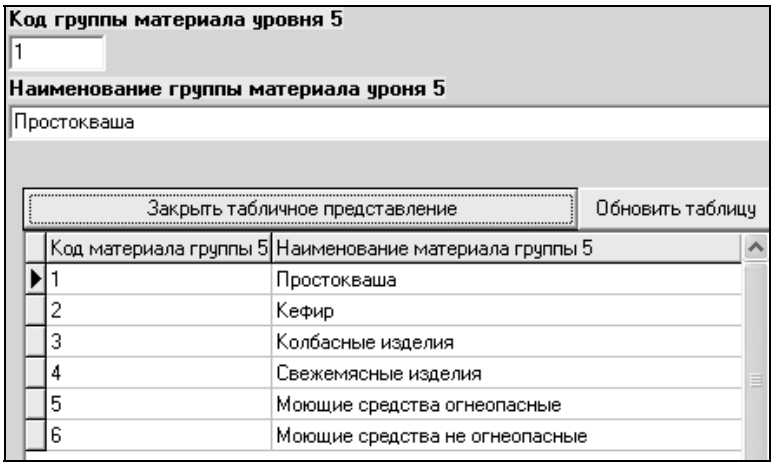


Рис. 3.30. Справочник групп товаров 5-го уровня в режиме исполнения

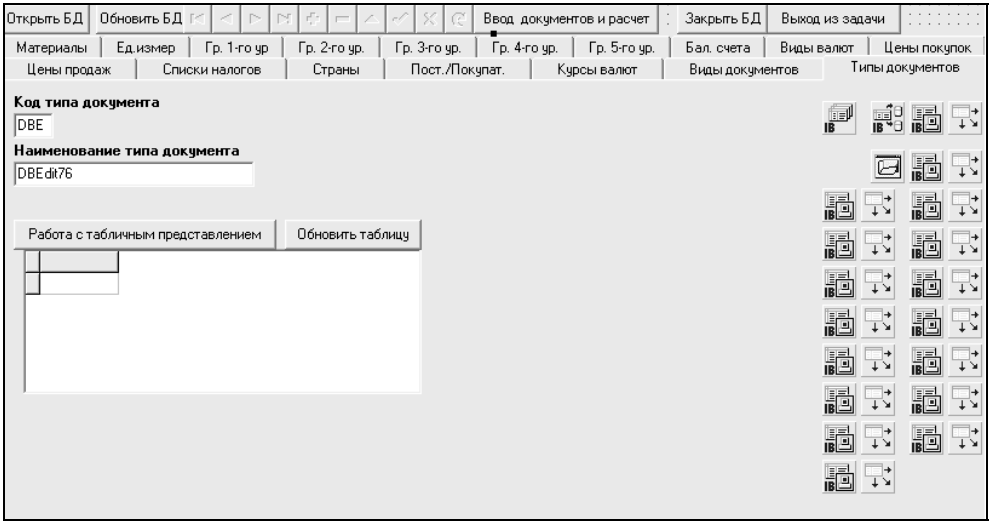


Рис. 3.31. Справочник типов документов в режиме проектирования

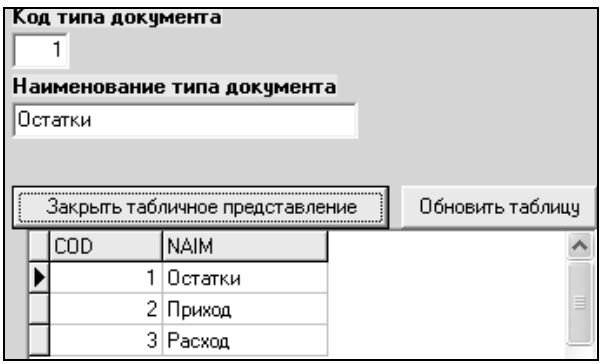


Рис. 3.32. Справочник типов документов в режиме исполнения

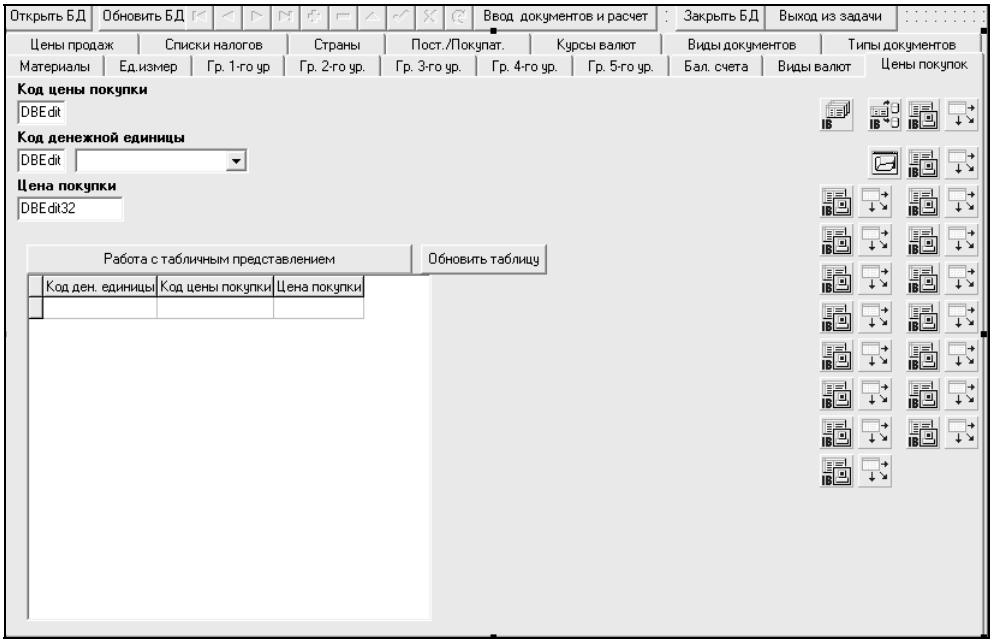


Рис. 3.33. Справочник цен покупок в режиме проектирования

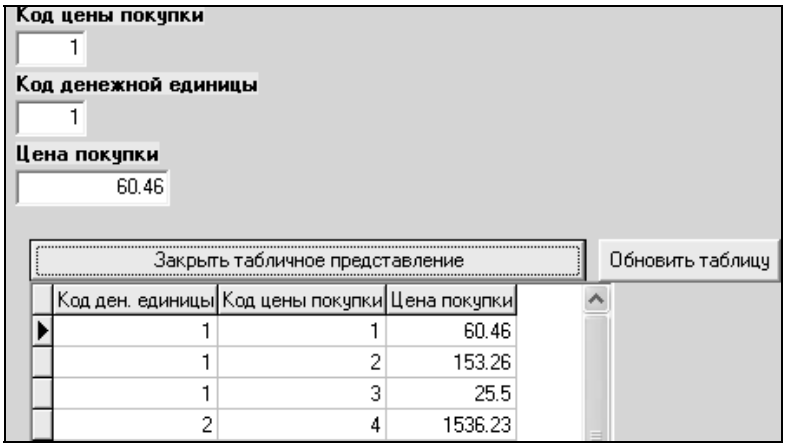


Рис. 3.34. Справочник цен покупок в режиме исполнения

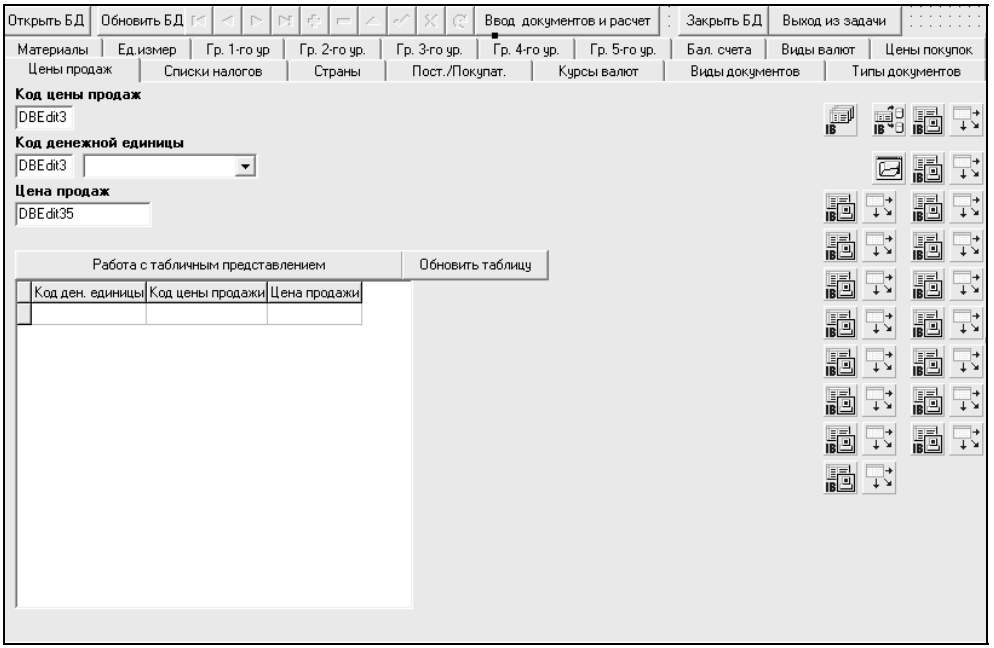


Рис. 3.35. Справочник цен продаж в режиме проектирования

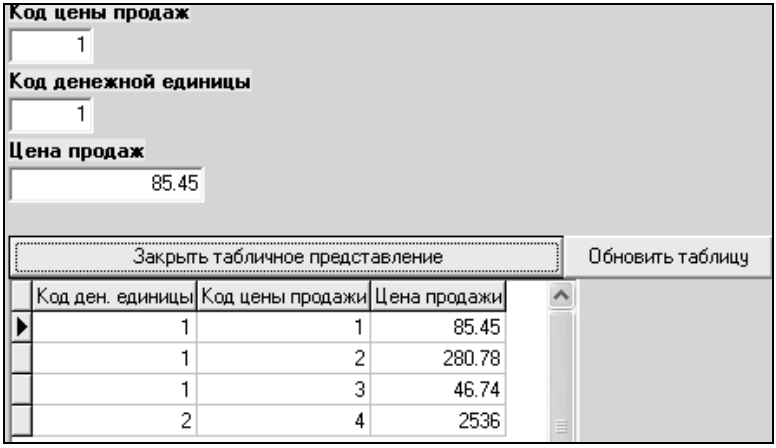


Рис. 3.36. Справочник цен продаж в режиме исполнения

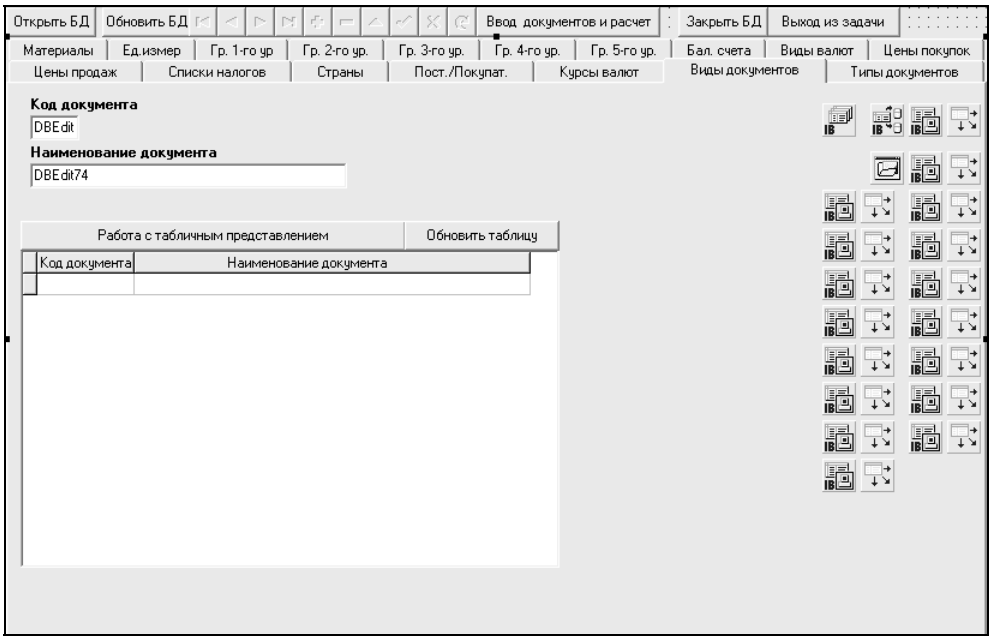


Рис. 3.37. Справочник видов документов в режиме проектирования

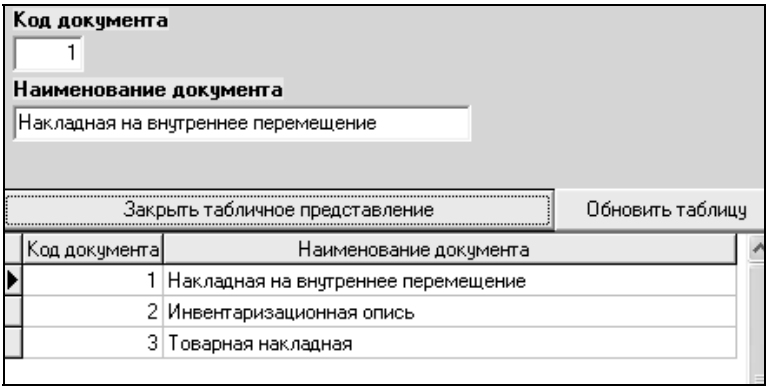


Рис. 3.38. Справочник видов документов в режиме исполнения

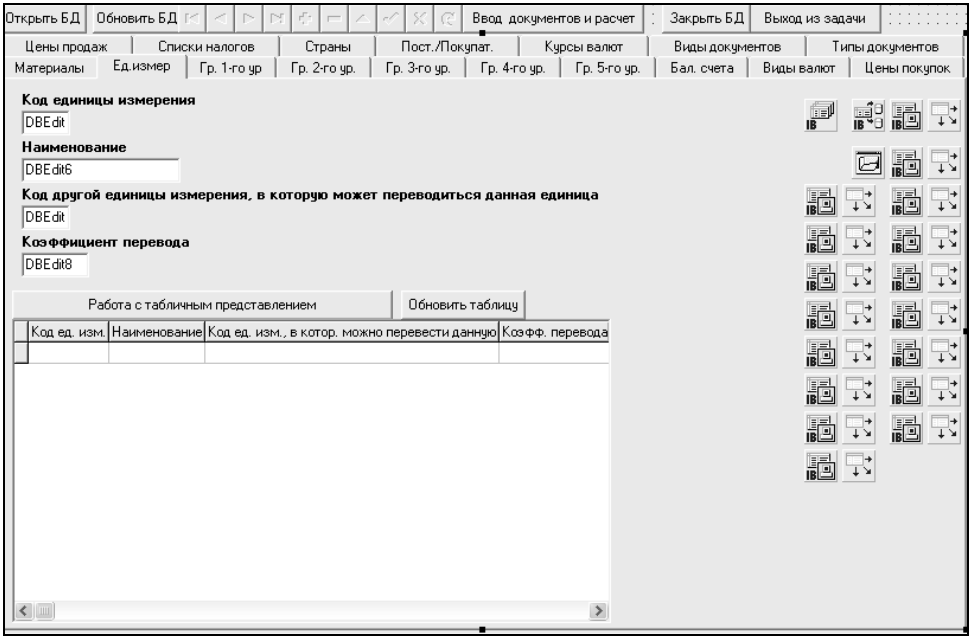


Рис. 3.39. Справочник единиц измерения в режиме проектирования

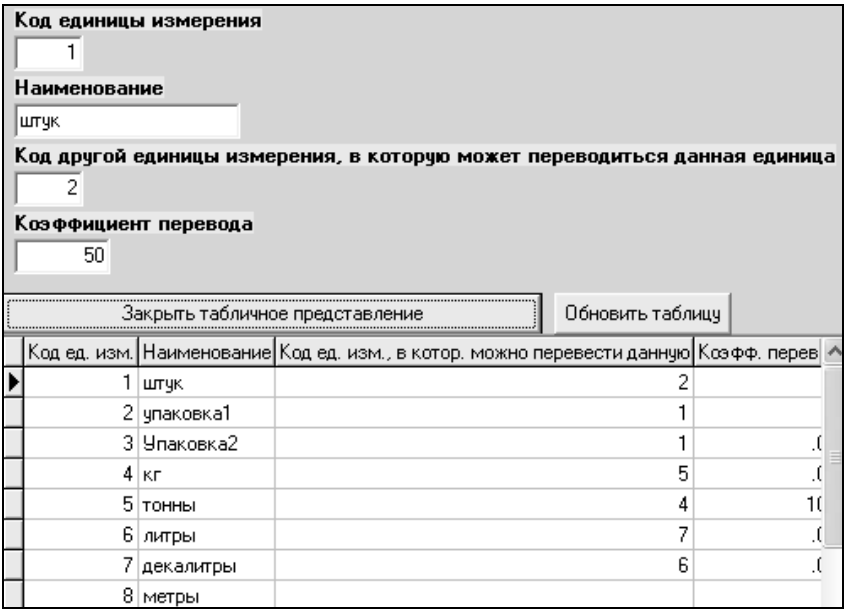


Рис. 3.40. Справочник единиц измерения в режиме исполнения

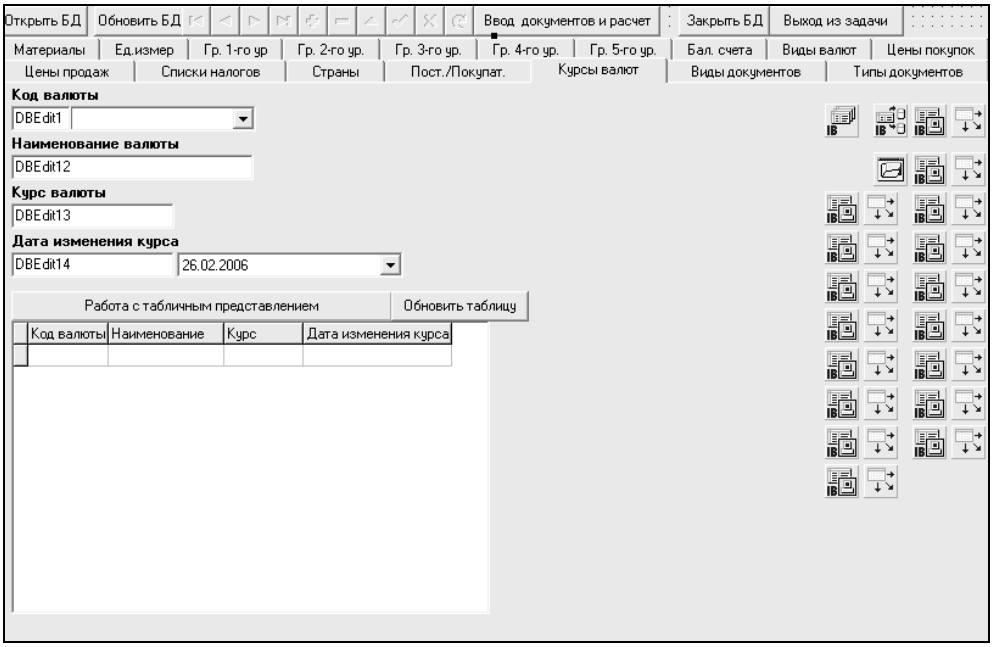


Рис. 3.41. Справочник курсов валют в режиме проектирования

Рис. 3.42. Справочник курсов валют в режиме исполнения

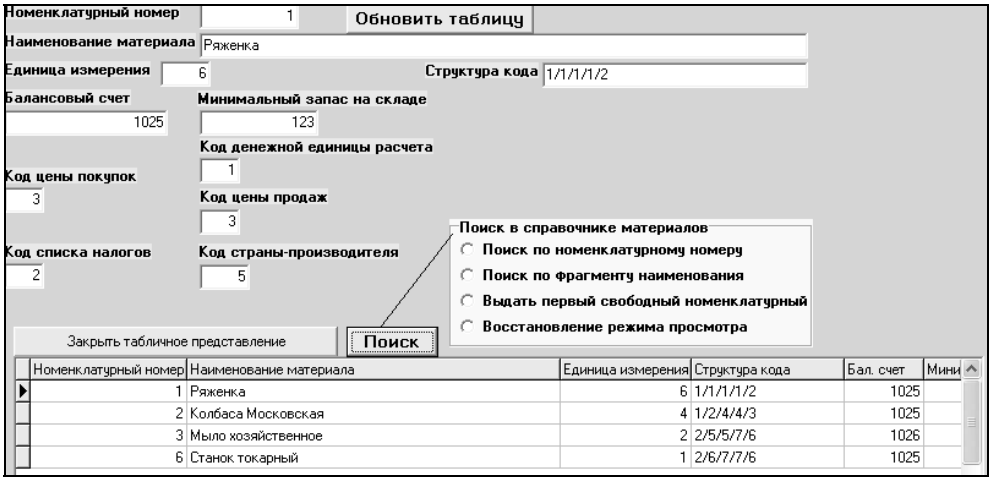


Рис. 3.43. Справочник материалов в режиме исполнения

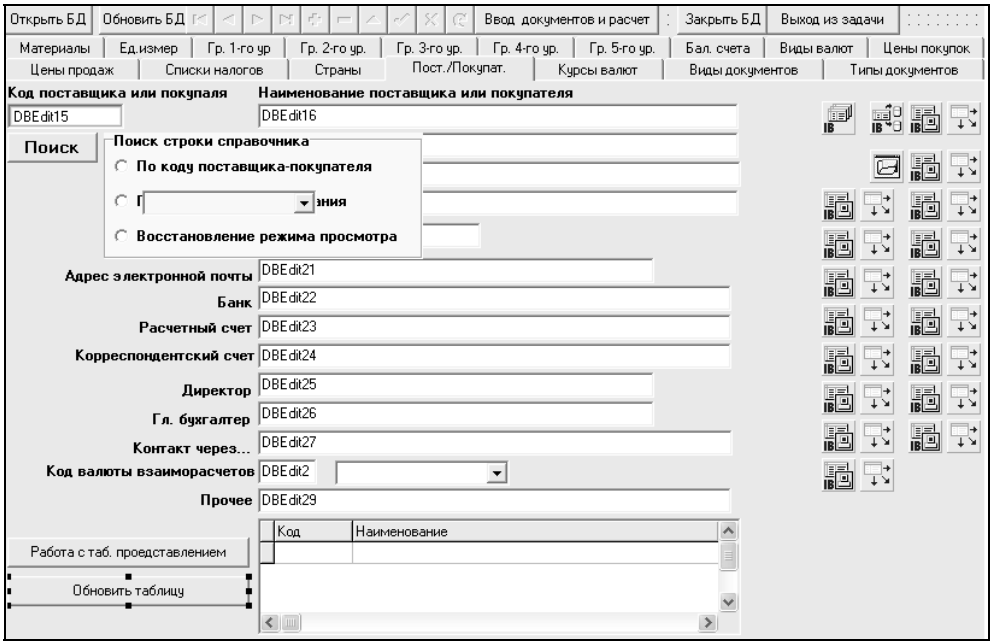


Рис. 3.44. Справочник поставщиков-покупателей в режиме проектирования

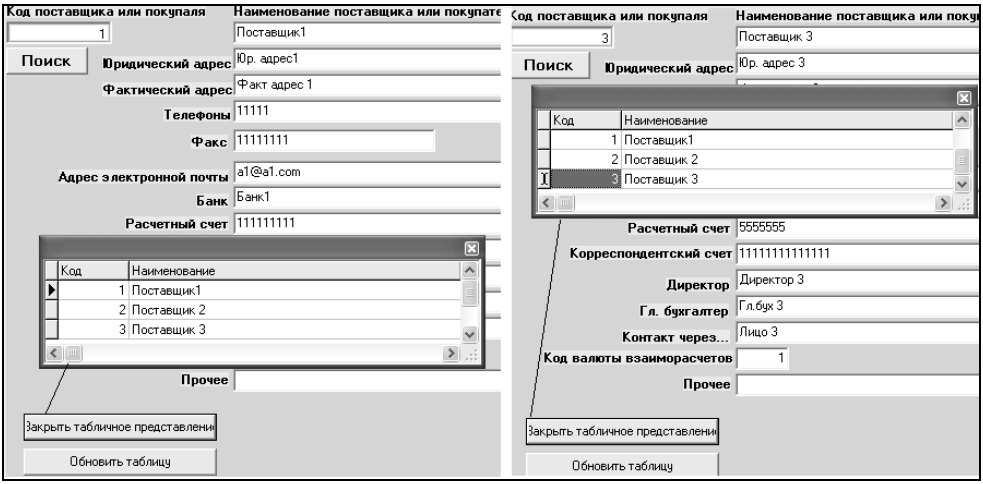


Рис. 3.45. Справочник поставщиков-покупателей в режиме исполнения: просмотр таблицы в перемещаемом окне

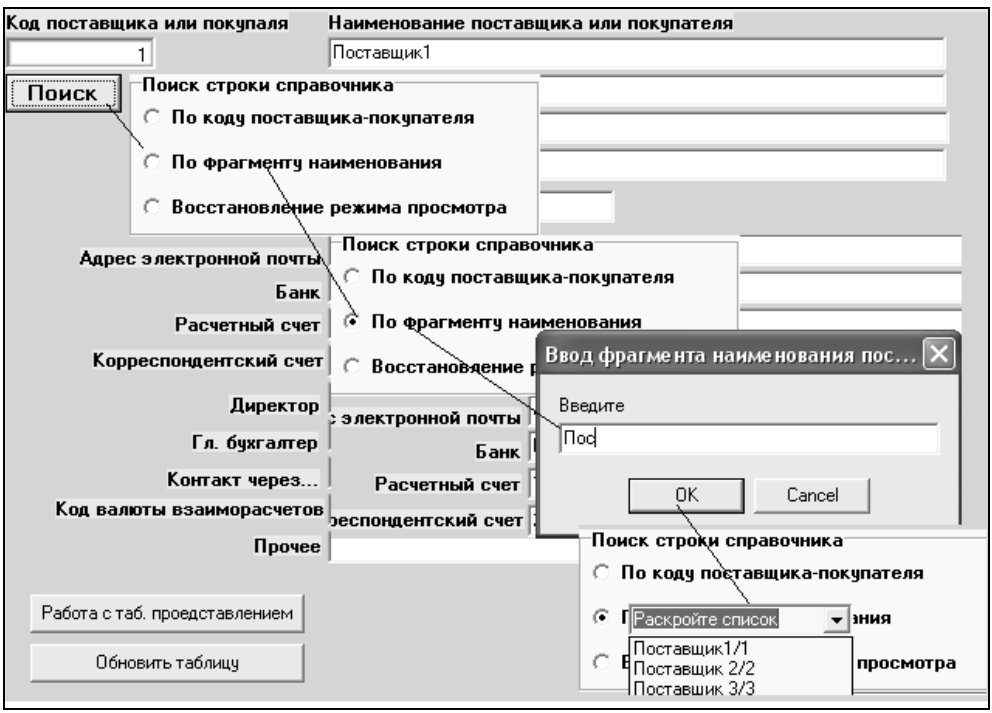


Рис. 3.46. Справочник поставщиков-покупателей в режиме исполнения: организация поиска

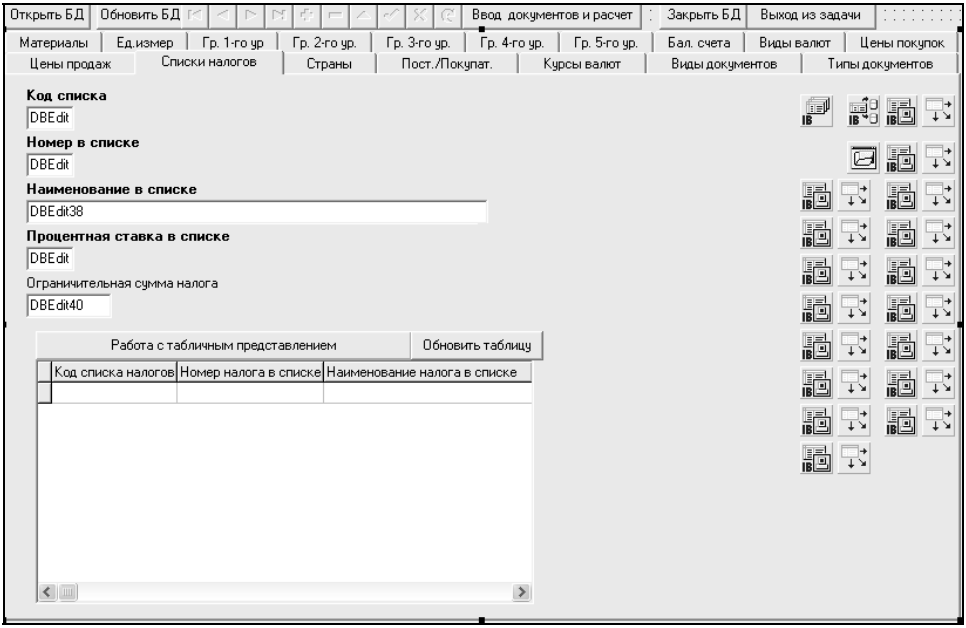


Рис. 3.47. Справочник списков налогов в режиме проектирования

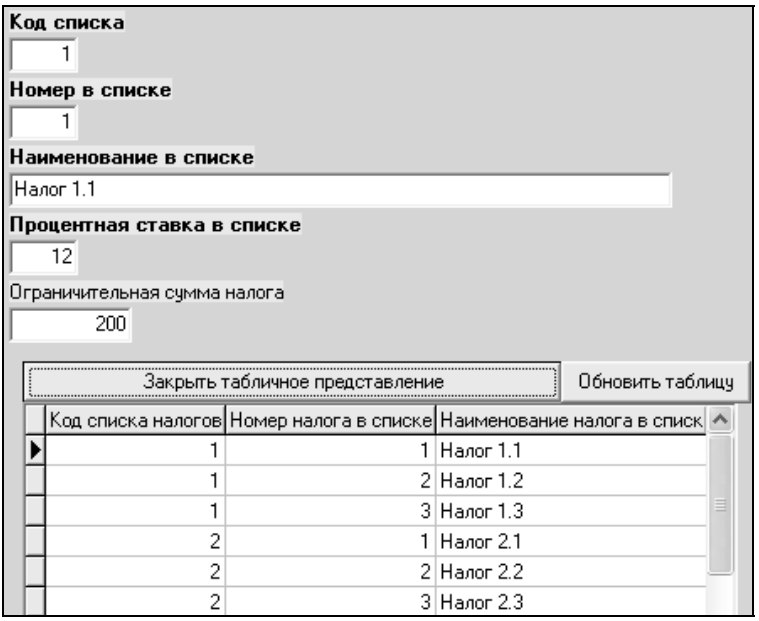


Рис. 3.48. Справочник списков налогов в режиме исполнения

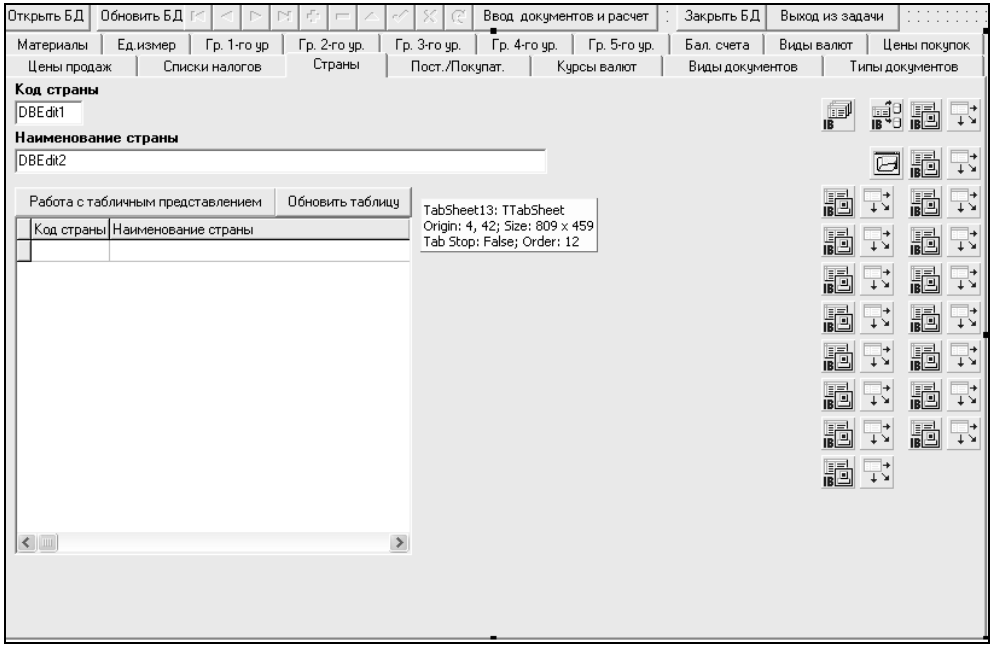


Рис. 3.49. Справочник стран в режиме проектирования

Рис. 3.50. Справочник стран в режиме исполнения

Работа с остатками

Остатки материалов на складах вводятся после очередной инвентаризации склада и в дальнейшем, естественно, влияют на количество соответствующего материала на складе. После ввода остатка по данному материалу "жизнь" его на складе начинается заново: Карточка складского учета по этому материалу должна быть пересчитана. Ход ввода остатков показан на рис. 3.51.

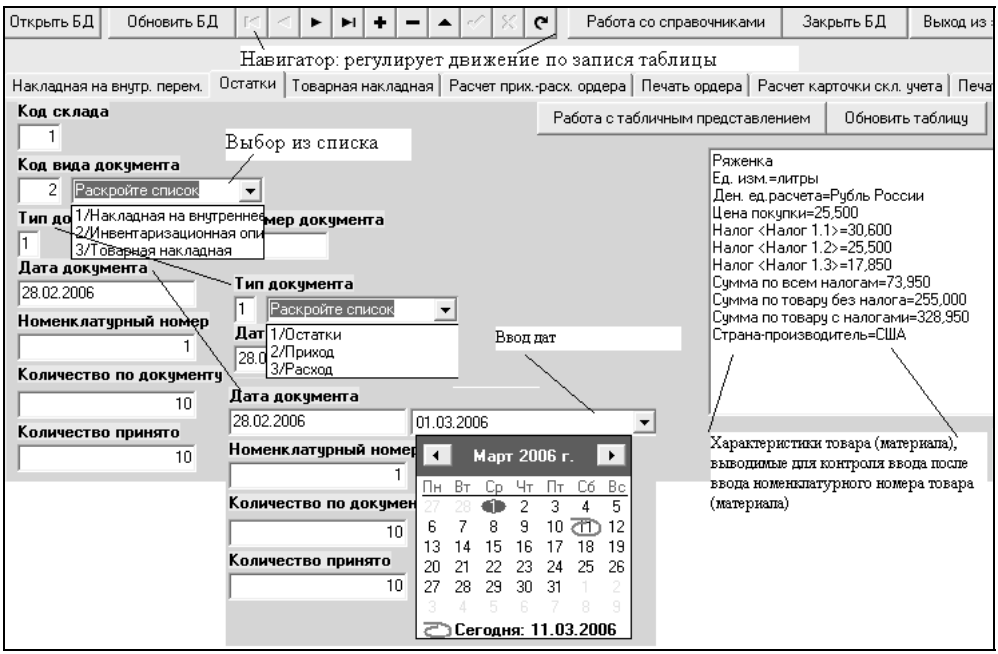


Рис. 3.51. Ввод остатков

При вводе используются справочники вида документов, типов документов и материалов. При активизации поля **Номенклатурный номер** автоматически формируется раскрывающийся список материалов, которому и передается фокус ввода. При наборе первого символа наименования список устанавливается на строку, которая начинается с данного символа, при наборе второго символа — на строку, у которой два первых символа совпадают с введенными, и т. д. Если строка с таким наименованием имеется в справочнике материалов, то щелчок на ней обеспечит ввод в поле номенклатурного номера значения номенклатурного номера, связанного с этой строкой. Если же такой строки нет, требуется переключиться на ведение справочника материалов и ввести отсутствующее наименование в справочник и затем про-

должить ввод остатков. После завершения ввода информации по данному материалу в поле формы высвечивается информация по характеристикам этого материала: наименование, единица измерения и т. п. Эти характеристики служат для контроля ввода, который осуществляет в момент ввода оператор ввода. После перехода к следующему материалу информация по характеристикам исчезает. Ее можно и загодя убрать, щелкнув в ее поле. Для движения по записям таблицы остатков, как и по записям других таблиц, надо пользоваться навигатором. Например, чтобы добавить новую строку в остатки, надо предварительно нажать на кнопку [+] навигатора, а чтобы перейти в режим модификации какого-либо поля — на кнопку [▲]. Страница по работе с остатками материалов в режиме модификации полей и просмотра записей показана на рис. 3.52.

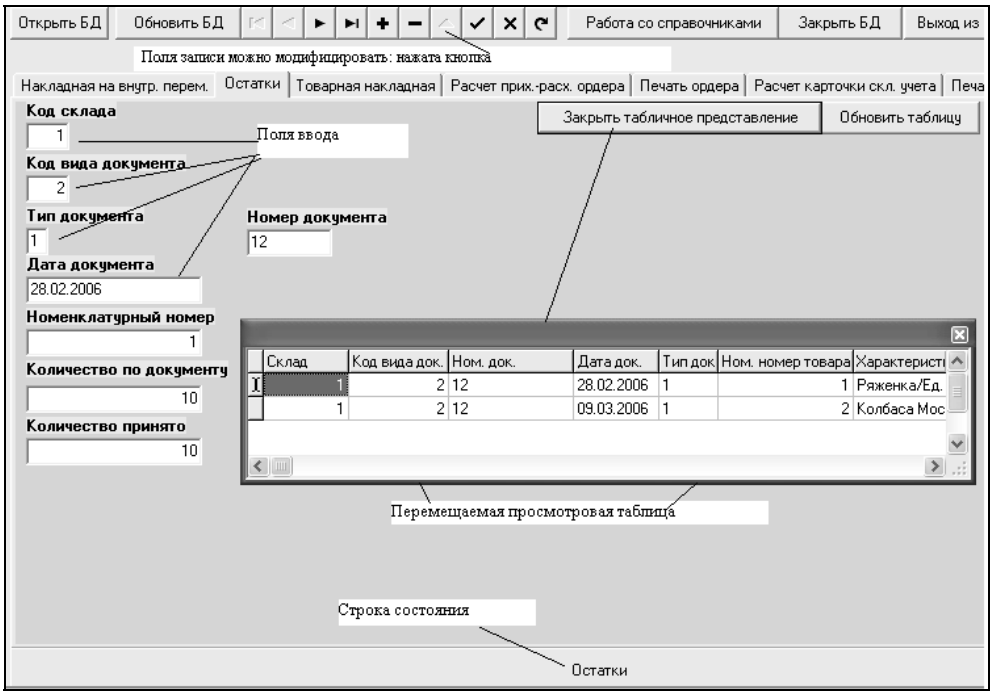


Рис. 3.52. Модификация полей записи и просмотр записей по остаткам

Работа с накладной на внутреннее перемещение

Работа с накладной на внутреннее перемещение материала отличается по технологии от работы с остатками только тем, что в момент ввода открывается диалоговое окно для запроса ввода количества принятого материала

(рис. 3.54). Дело в том, что количество материала, указанное в сопроводительном документе, может по тем или иным причинам не совпадать с количеством принятого материала. Вид накладной на внутреннее перемещение в режиме проектирования показан на рис 3.53.

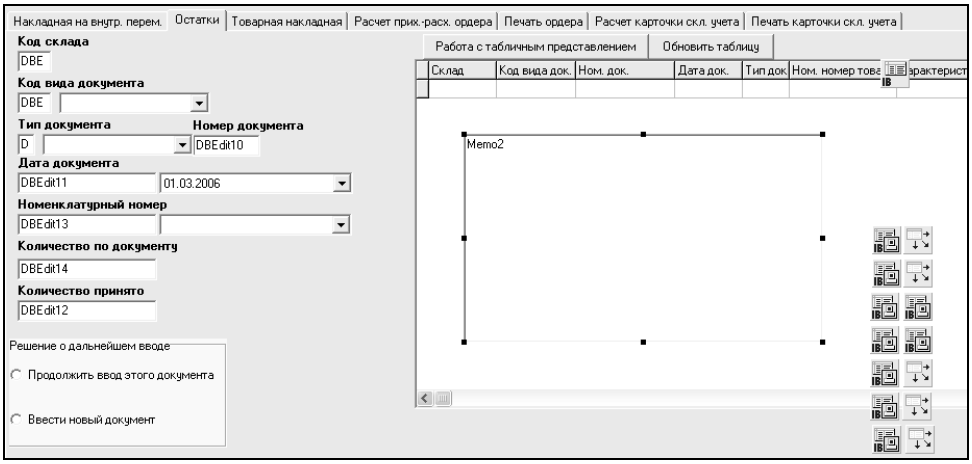


Рис. 3.53. Вид накладной на внутреннее перемещение в режиме проектирования

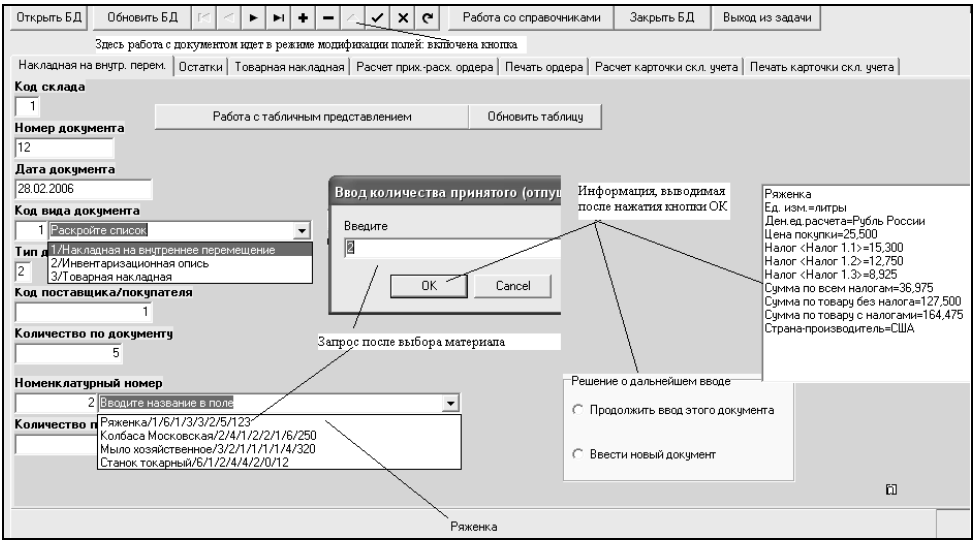


Рис. 3.54. Ввод и модификация накладной на внутреннее перемещение товара (режим исполнения)

Работа с товарной накладной

Вид товарной накладной в режиме проектирования показан на рис. 3.55.

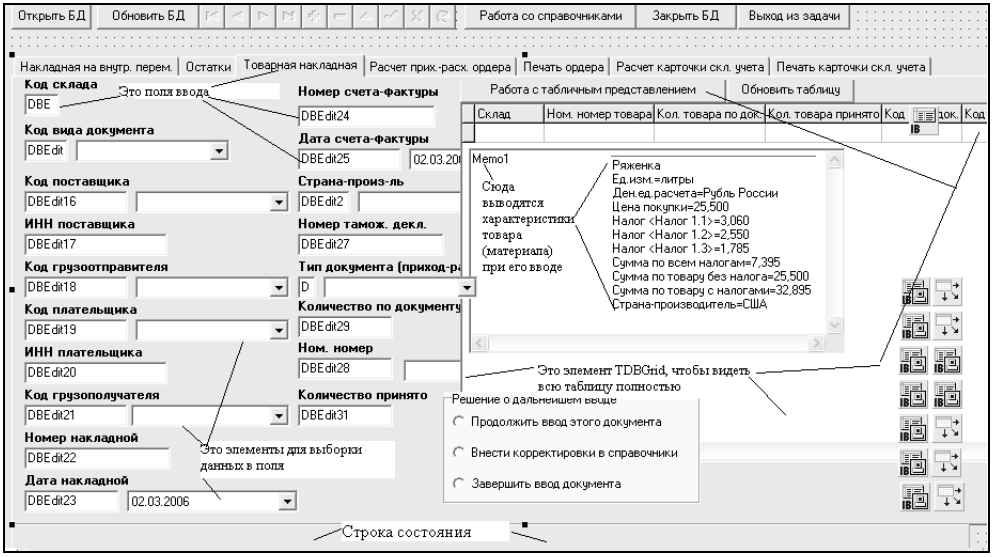


Рис. 3.55. Вид товарной накладной в режиме проектирования

Технологически работа с этим документом не отличается от работы с предыдущими двумя документами. Вид товарной накладной в режиме исполнения приведен на рис. 3.56 и 3.57.

Работа с приходно-расходным ордером

Приходно-расходный ордер рассчитывается в задаваемый интервал дат. Поэтому на вход расчета поступают накладные на внутреннее перемещение и товарные накладные за интервал дат. Задание интервала дат показано на рис. 3.58.

После выбора первой даты календарь исчезает, после повторного нажатия на кнопку **Выберите** — появляется и снова исчезает после выбора конечной даты. Значение дат высвечивается в строке состояния.

Когда даты выбраны, следует нажать на кнопку **Расчет**. Произойдет запрос на выборку типа документа (рис. 3.58), после чего автоматически осуществится расчет, и результат сохранится только в памяти, поэтому, чтобы увидеть ордер, его надо распечатать, перейдя на страницу **Печать ордера**.

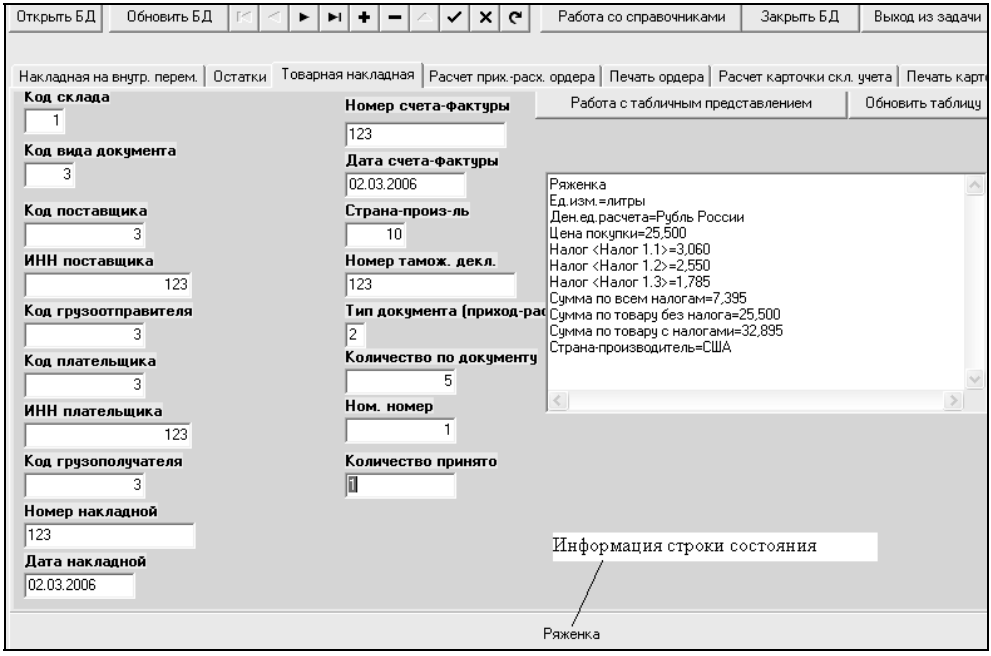


Рис. 3.56. Вид товарной накладной в режиме модификации полей записи

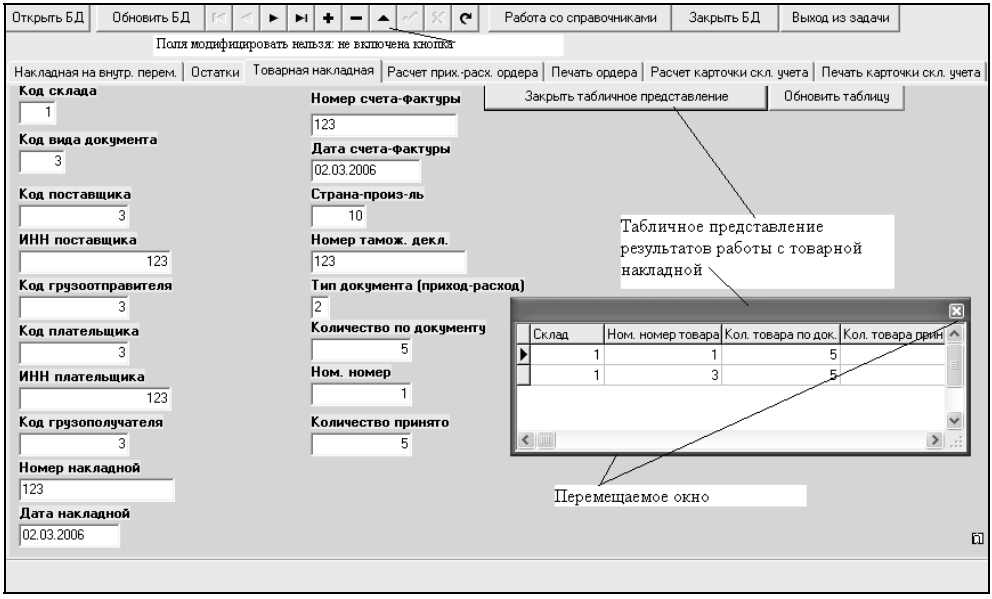


Рис. 3.57. Вид товарной накладной в режиме просмотра записей

Рис. 3.58. Задание интервала дат
для расчета приходно-расходного ордера

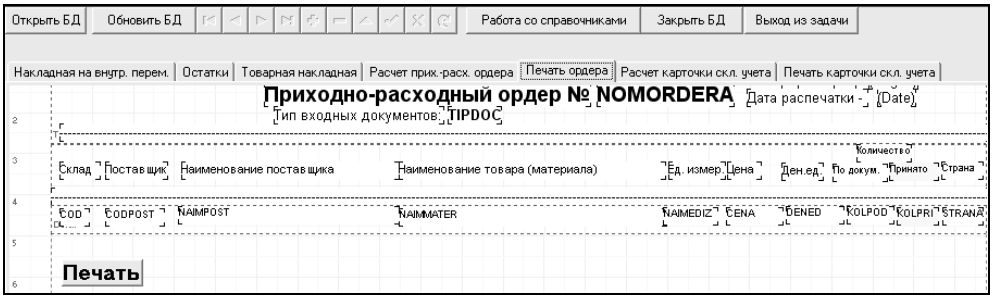


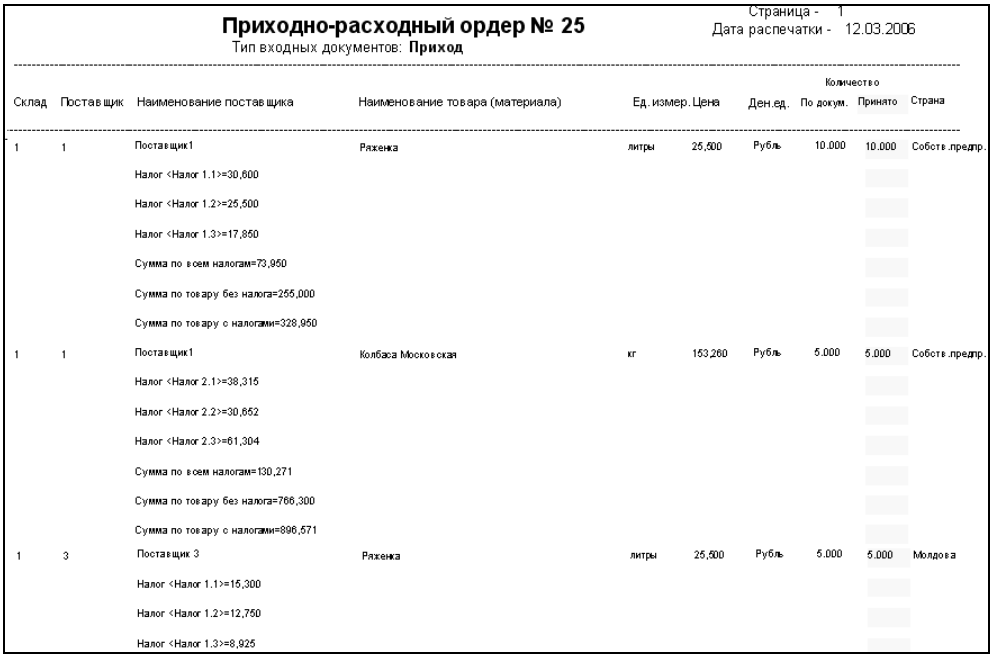
Рис. 3.59. Страница проектирования печати
приходно-расходного ордера

Вид этой страницы в режиме проектирования показан на рис. 3.59, а результат печати — на рис. 3.60. Для организации печати применены компоненты из палитры компонентов QReport.

Работа с карточкой складского учета

Расчет карточки складского учета похож на расчет приходно-расходного ордера, только вместо запроса типа обрабатываемых документов запрашивается номенклатурный номер, по которому станет рассчитываться карточка (рис. 3.61).

В отличие от приходно-расходного ордера, карточка сохраняется в базе данных. Вид рассчитанной карточки в табличном представлении показан на рис. 3.62.



Приходно-расходный ордер № 25

Тип входных документов: **Приход**

Страница - 1

Дата распечатки - 12.03.2006

Склад	Поставщик	Наименование поставщика	Наименование товара (материала)	Ед. измер.	Цена	Ден.ед.	Количество По докум.	Принято	Страна
1	1	Поставщик1	Ряженка	литры	25,500	Рубль	10.000	10.000	Собств.предпр.
		Налог <Налог 1.1>=30,600							
		Налог <Налог 1.2>=25,500							
		Налог <Налог 1.3>=17,850							
		Сумма по всем налогам=73,950							
		Сумма по товару без налога=255,000							
		Сумма по товару с налогами=328,950							
1	1	Поставщик1	Колбаса Московская	кг	153,260	Рубль	5.000	5.000	Собств.предпр.
		Налог <Налог 2.1>=38,315							
		Налог <Налог 2.2>=30,652							
		Налог <Налог 2.3>=61,304							
		Сумма по всем налогам=130,271							
		Сумма по товару без налога=766,300							
		Сумма по товару с налогами=896,571							
1	3	Поставщик 3	Ряженка	литры	25,500	Рубль	5.000	5.000	Молдова
		Налог <Налог 1.1>=15,300							
		Налог <Налог 1.2>=12,750							
		Налог <Налог 1.3>=8,925							

Рис. 3.60. Вид приходно-расходного ордера после печати

Рис. 3.61. Подготовка к расчету карточки складского учета

Чтобы отпечатать карточку, следует перейти на страницу **Печать карточки складского учета**. Вид этой страницы в режиме проектирования показан на рис. 3.63.

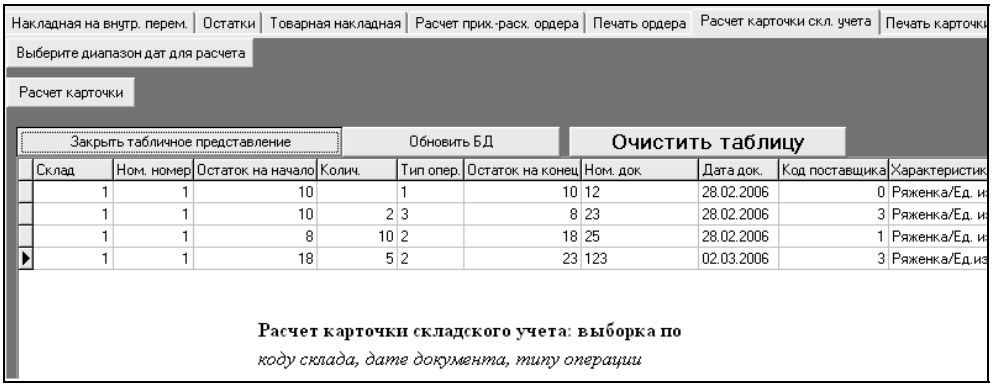


Рис. 3.62. Вид рассчитанной карточки в табличном представлении

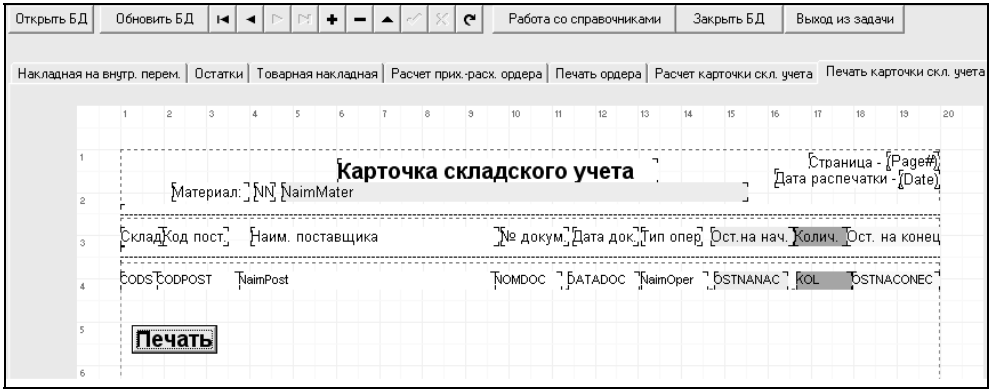


Рис. 3.63. Проектирование печати карточки складского учета

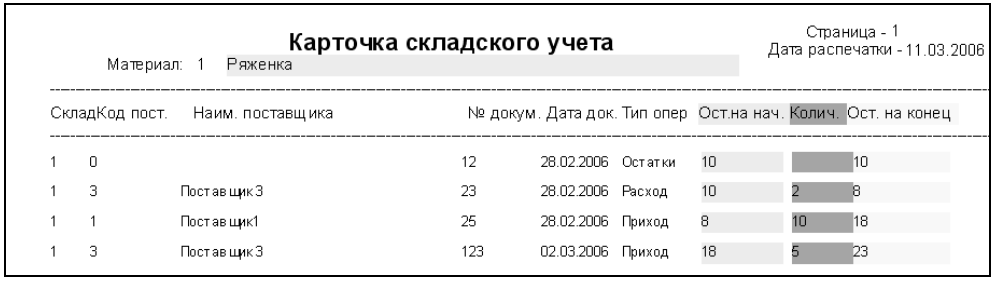


Рис. 3.64. Отпечатанная карточка складского учета

Вид отпечатанной карточки приведен на рис. 3.64.

Отметим, что при проектировании вывода карточки пришлось вводить поля наименований, чтобы вывести их при распечатке. Этих полей нет в выводимой таблице. Однако их можно получить из дополнительных таблиц так, как это показано на рис. 3.65.

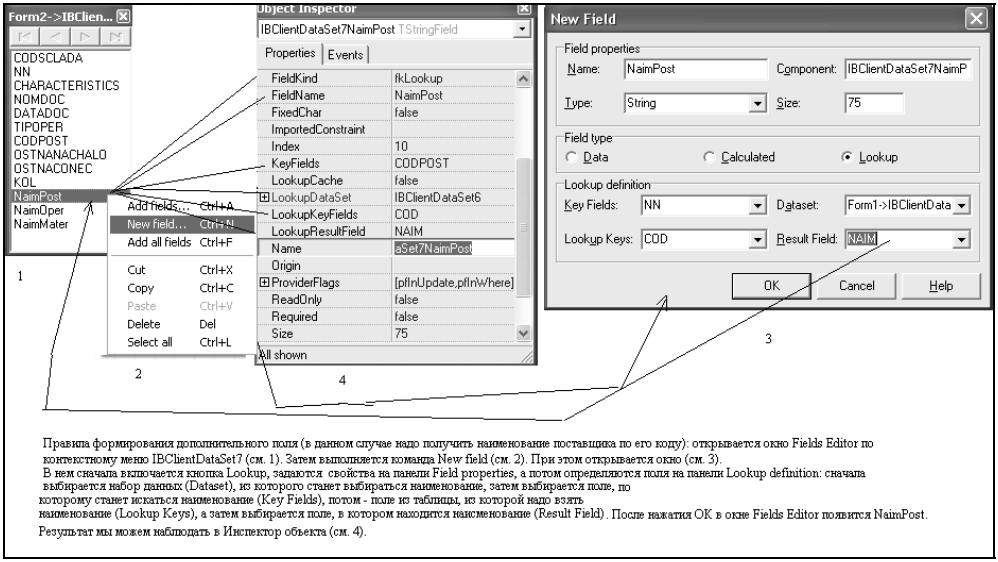


Рис. 3.65. Формирование дополнительных полей с наименованиями из других таблиц

Компоненты Interbase, через которые осуществляется связь с таблицами БД

Приведем перечень компонентов Interbase, настроенных на соответствующие таблицы базы данных Invoice. Их имена необходимы при создании приложений на основе разработанной картотеки учета материалов:

Для Form1 (ведение справочников):

- IBClientDataSet1 — настроен на таблицу STRANI;
- IBClientDataSet2 — настроен на таблицу DENED;
- IBClientDataSet3 — настроен на таблицу EDIZM;
- IBClientDataSet4 — настроен на таблицу BALSCH;
- IBClientDataSet5 — настроен на таблицу CURSVALUT;
- IBClientDataSet6 — настроен на таблицу POSTPOCUP;

- IBClientDataSet7 — настроен на таблицу CENIPOCUP;
- IBClientDataSet8 — настроен на таблицу CENIPRODAJ;
- IBClientDataSet9 — настроен на таблицу SPISCIVIDOVNALOGOV;
- IBClientDataSet10 — настроен на таблицу UR1;
- IBClientDataSet11 — настроен на таблицу UR2;
- IBClientDataSet12 — настроен на таблицу UR3;
- IBClientDataSet13 — настроен на таблицу UR4;
- IBClientDataSet14 — настроен на таблицу UR5;
- IBClientDataSet15 — настроен на таблицу MATERIALI;
- IBClientDataSet16 — настроен на таблицу VIDIDOCUM;
- IBClientDataSet17 — настроен на таблицу TIPIDOC.

Для Form2 (обработка информации на складе):

- IBClientDataSet1 — настроен на таблицу VXDOCVNPER;
- IBClientDataSet2 — настроен на таблицу OSTATCI;
- IBClientDataSet3 — настроен на таблицу POSTPOCUP;
- IBClientDataSet4 — настроен на таблицу VIDIDOCUM;
- IBClientDataSet5 — настроен на таблицу TOVNAKL;
- IBClientDataSet6 — настроен на таблицу PRIXORDER;
- IBClientDataSet7 — настроен на таблицу CARTSCLUCH.

Глава 4

Нахождение оптимального пути между двумя пунктами

Постановка задачи

Предположим, что вы являетесь владельцем крупного хлебозавода, и вам требуется дважды в день развозить хлеб в N точек. От вашего завода к каждой точке-получателю хлеба ведет множество путей — маршрутов. Каждый из маршрутов характеризуется множеством параметров с точки зрения экономичности доставки продукции: один маршрут короче, но на нем расходуется много горючего (доставка продукции осуществляется автотранспортом), другой длиннее, но на нем меньше светофоров и поток машин меньше, т. е. по нему можно быстрее доставить продукцию потребителю, но это обойдется дороже, и т. д. Налицо обыкновенная задача оптимального программирования. Чтобы не усложнять проблему, рассмотрим только одну характеристику маршрута — его длину. Задача состоит в том, чтобы из всех маршрутов, ведущих от хлебозавода к данной точке-потребителю продукции, выбрать тот, который будет короче остальных. При решении этой проблемы воспользуемся методом, предложенным корифеями оптимального программирования Р. Беллманом и Дж. Данцигом.

Предположим, что наилучший путь из точки A в точку B проходит через точки C и D, как это показано на рис. 4.1 жирной линией.

Допустим теперь, что нам надо пройти только через точки C и D. Каким будет оптимальный путь из C в D? Ясно, что он должен совпадать с частью оптимального пути из A в B. Действительно, если бы это было не так, то путь из A в B был бы не оптимальным. Точнее сказать, что любая часть оптимального пути тоже является оптимальной. Это и есть так называемый *принцип оптимальности Р. Беллмана*. Если внимательно посмотреть на выводы из этого принципа, то можно сказать, что, найдя оптимальный путь между двумя промежуточными точками, можно быть уверенным, что если одна из таких точек встретится на нашем пути, то следующей точкой на опти-

мальном пути будет точка, оптимальная к встреченной. Но как встретить такую точку, пока не ясно. Ясно только, что начальная точка нашего движения всегда находится на оптимальном пути. Для отыскания алгоритма выбора оптимального пути рассмотрим схему путей, состоящую из семи точек.

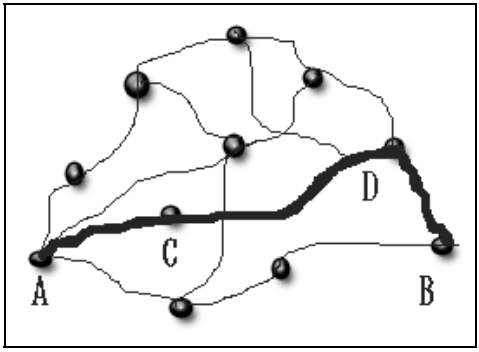


Рис. 4.1. Наилучший путь из точки *A* в точку *B*

Представим эту схему в виде некоторого графа — совокупности пронумерованных кружков-точек, соединенных между собой линиями, на которых обозначена их длина — расстояние между двумя рядом расположенными точками. Этот граф показан на рис. 4.2.

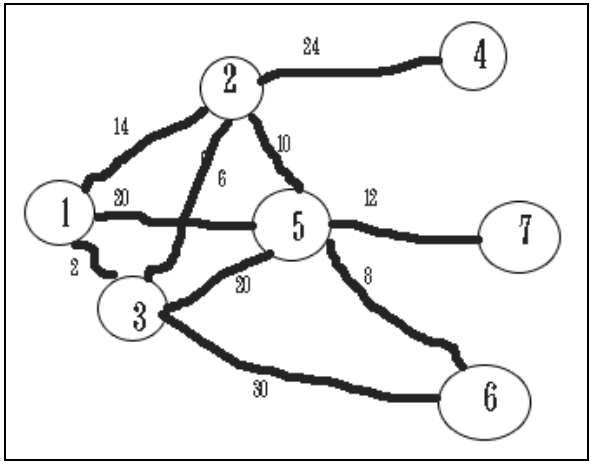


Рис. 4.2. Пример схемы путей, связывающих семь точек

Для нахождения оптимального пути поступим следующим образом.

Вычислим все одношаговые (между соседними точками) пути, исходящие из начальной точки (помним, что начальная точка всегда лежит на оптимальном пути). Получим такую схему:

Путь	1	1-2	1-3	1-5
Длина	0	14	2	20

Из таблицы выбираем минимальную длину и получаем путь 1-3. То есть отрезок 1-3 находится на оптимальном пути. Рассматриваем все пути, исходящие из точки 3, и выбираем среди них тот, у которого длина минимальная. Получим схему:

Путь	1-3-2	1-3-5	1-3-6
Длина	8	22	32

Таким образом, видим, что вместе с точкой 3 на оптимальном пути находится точка 2.

Теперь работаем с точкой 2: рассматриваем все пути, исходящие из точки 2, и выбираем среди них тот, у которого длина минимальная. Получим схему:

Путь	1-3-2-4	1-3-2-5
Длина	32	18

Точка 5 оказалась на оптимальном пути. Поступаем с ней точно так же, как и с предыдущими "оптимальными" точками: рассматриваем пути, исходящие из этой точки, и ищем минимальный. Получим схему:

Путь	1-3-2-5-6	1-3-2-5-7
Длина	26	30

В итоге получаем оптимальный (минимальный, самый короткий) путь по точкам: 1-3-2-5-6.

Мы решили задачу нахождения кратчайшего пути из точки 1. Попали, в конце концов, в точку 6. А если нам не надо в точку 6, а надо — в точку 7? Очевидно, что при таком ограничении кратчайший путь изменится. Как его

искать? Ясно, что при выборе точки (точнее, при выборе минимального отрезка пути) надо быть еще уверенным, что он лежит на пути к заданной точке. Напрашивается такой алгоритм: из исходной точки 1 выбираем одношаговые пути, а среди них — те, что ведут в заданную точку. А уже из последних выбираем минимальный отрезок. Затем берем его конец, принимаем его за начало нашего движения, и алгоритм повторяется уже с новым началом. То есть основная трудность состоит в том, чтобы на каждом отрезке определять, что он лежит на пути к нужной точке.

Введем дополнительное требование к информации: наряду с векторами исходящих из данного узла (точки) узлов надо иметь и вектора входящих в данный узел узлов. Теперь в рамках новых требований рассмотренная задача векторно станет выглядеть следующим образом:

- по входящим узлам:

```
{1: 0}, {2: 1,3}, {3: 1}, {4: 2}, {5: 2,1,3}, {6: 5,3}, {7: 5}
```

  Здесь номера перед двоеточием — это номера узлов, куда входят пути от других узлов, номера которых указаны после двоеточия;

- по исходящим узлам:

```
{1: 2,3,5}, {2: 4,5}, {3: 5,6}, {4: }, {5: 6,7}, {6: }, {7: }
```

  Здесь номера перед двоеточием — это номера узлов, откуда исходят пути в другие узлы, номера которых указаны после двоеточия.

  Теперь уже можно перейти к описанию алгоритма. Пусть нам надо найти кратчайший путь из точки 1 в точку 6. Сначала движемся "назад" и определяем по векторным записям множество узлов, находящихся на пути к узлу 6. Смотрим на записи по входящим узлам справа налево и получаем последовательно:

  1. В узел 6 входят узлы 5,3. Раскроем каждый из этих узлов. В узел 5 входят узлы 2,1,3. В узел 3 — узел 1. В итоге получим множество узлов M={6,5,3,2,1,3,1}. А теперь начнем работать с векторами исходящих узлов.
  2. Так как нам требуется найти путь от узла 1, то возьмем его исходящий вектор, но в нем только те элементы, которые входят в M{}. Для нашего примера все узлы входят в M{}. Вот на этом множестве исходящих узлов из узла 1, попавших в M{}, и рассчитываем минимальное расстояние.
  3. Для узла 2 расстояние будет равно 14, для узла 3 — 2, для узла 5 — 20. То есть выбираем наименьшее значение и получим, что надо выбрать узел 3. Он-то и будет лежать на кратчайшем пути к точке 6.
  4. Теперь смотрим все узлы, исходящие из узла 3 (узлы 2,5,6). Проверяем их на вхождение в M{} и отбираем те, что удовлетворяют требованию вхождения. В нашем примере они все входят в M{}. Находим среди

них узел с минимальным отрезком. Для узла 2 расстояние до узла 3 будет равно 6, для узла 5 — 20, для узла 6 — 30. Следовательно, выбираем наименьшее значение и получим, что надо выбрать узел 2. Он и будет лежать на кратчайшем пути к точке 6.

5. Работаем дальше уже с узлом 2. Из него выходят узлы 4,5. Узел 4 не принадлежит множеству M{}, он не лежит на пути к точке 6. Его не рассматриваем. Остается единственный узел 5. Из него исходят пути в узлы 6,7. Узел 7 не входит во множество M{}, следовательно его не рассматриваем, а узел 6 совпадает с требуемой конечной точкой. Все. Останов. Нашли наикратчайший путь из точки 1 в точку 6:1-3-2-5-6.

Создание базы данных и формирование таблиц

Создадим отдельную базу данных для этой задачи с помощью утилиты IBConsole: запускаем утилиту, выполняя цепочку команд: **Пуск | Программы | Interbase | IBConsole**, отмечаем строку **Local Server**, выполняем команду **Server/Login**, запуская сервер. Получим вид окна IBConsole, показанного на рис. 4.3.

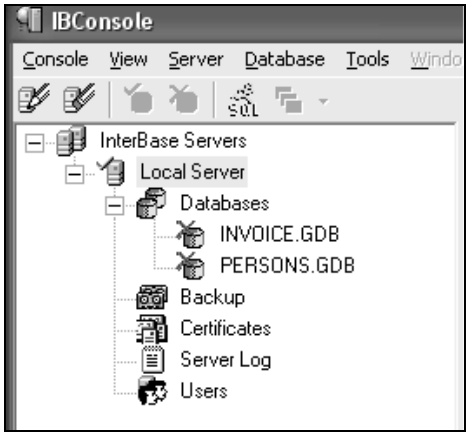


Рис. 4.3. Подключение к серверу для создания на нем базы данных

Из рисунка видно, что на сервере уже существуют две базы данных: для картотеки персонала и данных по складскому учету. В показанном на рис. 4.3 окне выполняем команды **Database/Create Database**. Открывается диалоговое окно, показанное на рис. 4.4.

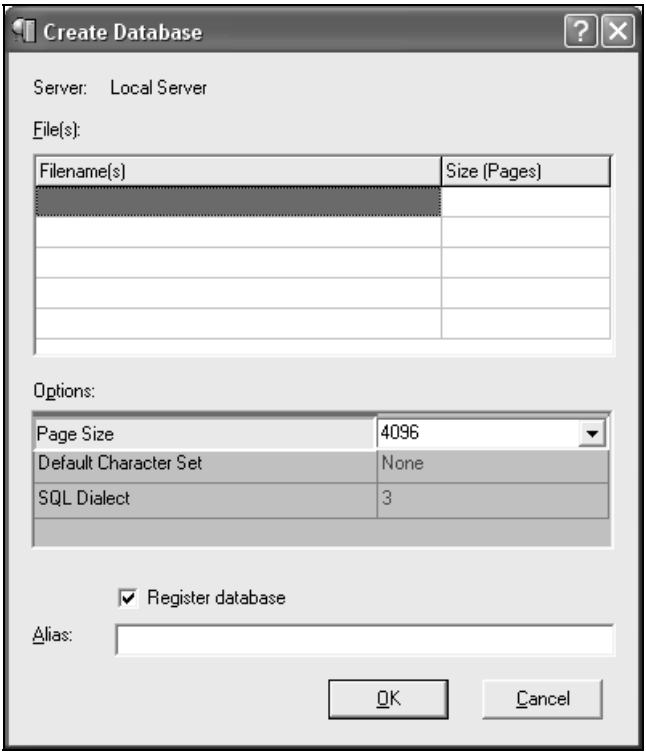


Рис. 4.4. Диалоговое окно для создания базы данных

Заполняем поля этого окна и получаем вид окна, показанный на рис. 4.5.

После нажатия на **OK** база данных создана, о чем свидетельствует содержимое соответствующего окна IBConsole (рис. 4.6).

Чтобы хранить всю схему (граф), создадим одну таблицу такой структуры, как табл. 4.1. Здесь NodNum — номер текущего узла, NodNumIn — номер ближайшего узла, путь от которого входит в данный узел (входящий узел).

В рамках нашего контрольного примера, рассмотренного ранее, таблица станет выглядеть так — табл. 4.1.

Таблица 4.1. *Представление схемы путей*

NodNum	NodNumIn	LengthIn
1	0	0
1	0	0
1	0	0
2	1	14

Таблица 4.1 *(окончание)*

NodNum	NodNumIn	LengthIn
2	3	6
3	1	2
3	0	0
3	0	0
4	2	24
5	2	10
5	1	20
5	3	20
6	5	8
6	3	30
7	5	12

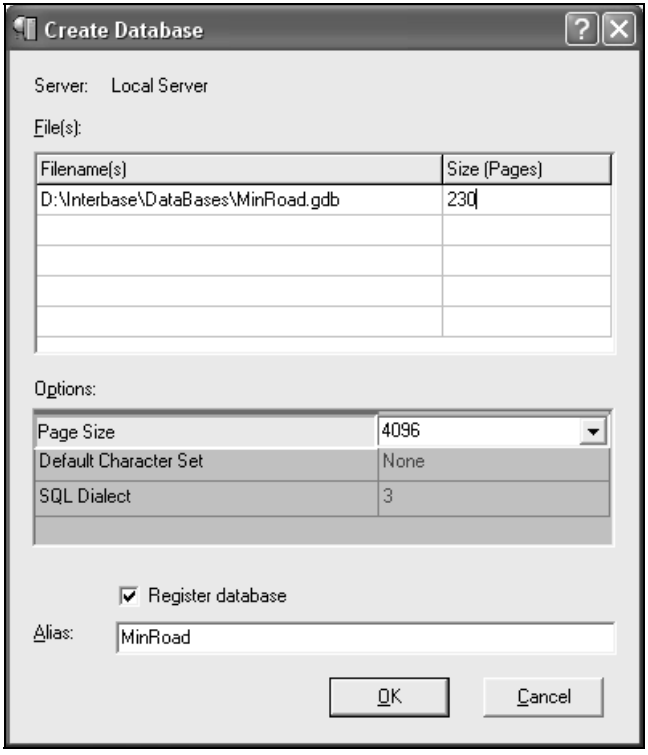


Рис. 4.5. Заполнение диалогового окна IBConsole для создания бааы данных

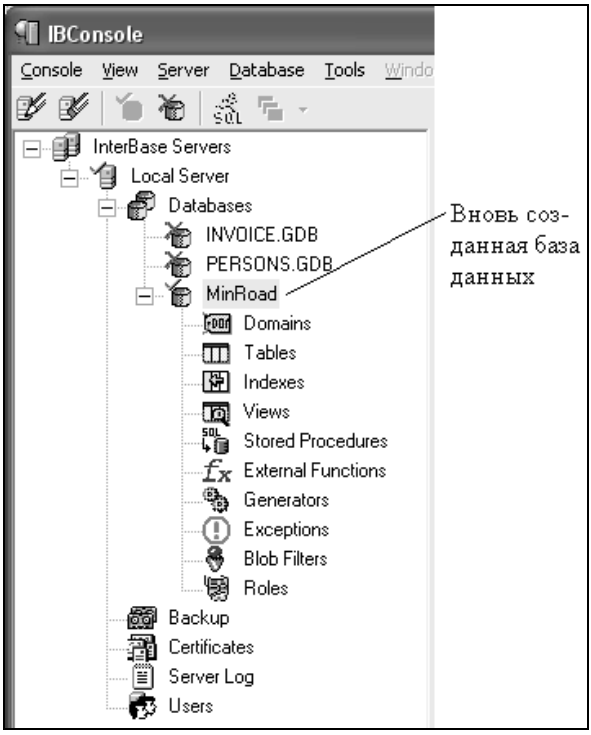


Рис. 4.6. Информация о вновь созданной базе данных в окне IBConsole

Описание таблицы приведено в листинге 4.1.

Листинг 4.1

```
Create Table InOutNods
 (
   NodNumber integer Not Null,
   NodNumIn  integer Not Null,
   LengthIn   float Not Null
  );
```

Кроме таблицы, создадим к ней индекс:

```
CREATE INDEX ind
ON INOUTNODS (NodNumber);
```

Он поможет нам в сортировке таблицы по первой колонке.

Программирование

Ввод данных в таблицу

Ввод данных в таблицу происходит по той же схеме, что и ввод данных в таблицы задач, рассмотренных в *главах 3, 4.*

Рис. 4.7. Вид формы в режиме проектирования

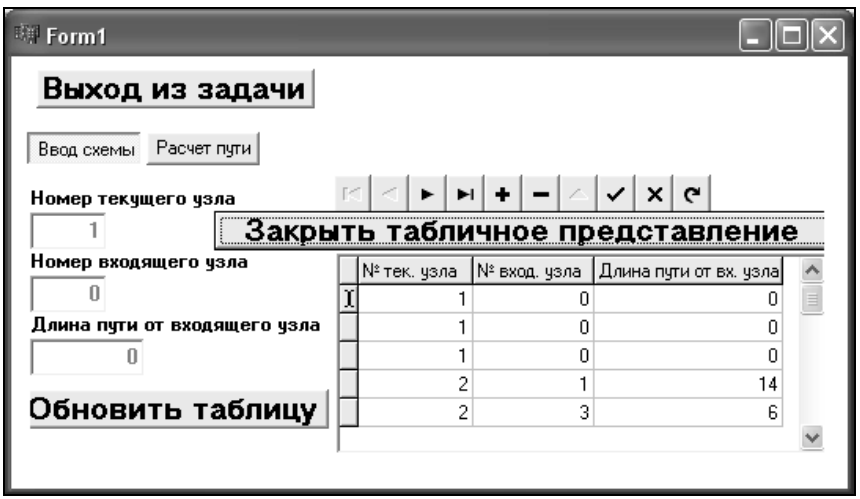


Рис. 4.8. Введенный контрольный пример

Вид формы в режиме проектирования показан на рис. 4.7.

Вид формы в режиме исполнения вместе с введенным контрольным примером показан на рис. 4.8.

Листинг программы ввода схемы путей представлен в листинге 4.2.

Листинг 4.2

cpp-файл

```
//---------------------------------------------------------------------------

#include <vcl.h>
#pragma hdrstop

#include "Unit1.h"

//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"

#define MaxInd 1000 //максимальное количество узлов сети
#define EndOfArr 999999  //признак конца массива

TForm1 *Form1;

TIBClientDataSet *ds1,*ds2;
int IndMas;
 struct st
   {
     int nn;
     int iz;
     float len;
     AnsiString sign;
    }M[MaxInd];
//для функции FloatToStr():
enum TStringFloatFormat {sffGeneral, sffExponent, sffFixed, sffNumber,
sffCurrency };
```

```
//функция развертывания узла назад:
//записывает в массив tr все узлы, входящие в nod с длинами их путей

 TLocateOptions Opts;
void unfolding(TIBClientDataSet *ds,struct st M[],int nod);
//---------------------------------------------------------------------------
void unfolding(TIBClientDataSet *ds,struct st M[],int nod)
 {
  //подгонка массива
  IndMas=0;
  while(M[IndMas].nn != EndOfArr)
   IndMas++;

  int  nodnumin;
   ds->Close();
   ds->CommandText="select * from inoutnods where nodnumber=
   "+ IntToStr(nod);
   ds->Open();

   if(ds->FieldByName("nodnumin")->AsInteger == 0)
    return; //в данный узел не входит ни один другой узел
   while (!ds->Eof)
    {
     nodnumin= ds->FieldByName("nodnumin")->AsInteger;
     if(ds->FieldByName("nodnumin")->AsInteger == 0)
      {
        ds->Next();
        continue;
       }
      M[IndMas].iz=ds->FieldByName("nodnumin")->AsInteger;
      M[IndMas].nn=nod; //это - в который входит
      M[IndMas].len=ds->FieldByName("lengthin")->AsFloat;
      M[IndMas].sign="";
      IndMas++;
      ds->Next();
    } //while
```

```
    return;
 }
//---------------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Button1Click(TObject *Sender)
{
 if(!IBClientDataSet1->Active)
   IBClientDataSet1->Open();
 IBClientDataSet1->ApplyUpdates(-1);
}
//---------------------------------------------------------------------------
void __fastcall TForm1::TabSheet1Show(TObject *Sender)
{
 if(!IBClientDataSet1->Active)
   IBClientDataSet1->Open();
 IBClientDataSet1->Edit();
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Button2Click(TObject *Sender)
{

   if(DBGrid1->Visible)
    {
     DBGrid1->Hide();
     Button2->Caption="Работа с табличным представлением";
    }
  else
   {
    Button2->Caption="Закрыть табличное представление";
    DBGrid1->Show();
   }
}
//---------------------------------------------------------------------------
```

```
void __fastcall TForm1::DBEdit1Enter(TObject *Sender)
{
 if(!IBClientDataSet1->Active)
   IBClientDataSet1->Open();
 IBClientDataSet1->Edit();
}
//---------------------------------------------------------------------------

void __fastcall TForm1::DBEdit1KeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if(Key == VK_RETURN)
   {
     DBEdit2->SetFocus();
   }
}
//---------------------------------------------------------------------------

void __fastcall TForm1::DBEdit2KeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
 if(Key == VK_RETURN)
   {
     DBEdit3->SetFocus();
   }
}
//---------------------------------------------------------------------------

void __fastcall TForm1::DBEdit3KeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
 if(Key == VK_RETURN)
   {
     DBNavigator1->SetFocus();
   }
}
```

```
//---------------------------------------------------------------------------
void __fastcall TForm1::Button3Click(TObject *Sender)
{
 Application->Terminate();
}
//---------------------------------------------------------------------------
void __fastcall TForm1::DBNavigator1Click(TObject *Sender,
      TNavigateBtn Button)
{
     DBEdit1->SetFocus();
}
//---------------------------------------------------------------------------
void __fastcall TForm1::TabSheet2Show(TObject *Sender)
{
 if(!IBClientDataSet1->Active)
   IBClientDataSet1->Open();
}
//---------------------------------------------------------------------------
void __fastcall TForm1::LabeledEdit2KeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
 if(Key != VK_RETURN)
  return;

 //выборка из схемы всех узлов, входящих в конечный узел J
  IBClientDataSet2->Close();
  IBClientDataSet2->CommandText="select * from inoutnods where
  nodnumber="
  + LabeledEdit2->Text;
  IBClientDataSet2->Open();
  int i=StrToInt(LabeledEdit1->Text); //начальная точка пути
  int j=StrToInt(LabeledEdit2->Text); //конечная точка пути
  int k,l;

  IBTable1->Close();
  IBTable1->Open();
```

```
 /*Все входящие узлы по мере их обработки
помещаем в массив вместе с их длинами
и начинаем просматривать выбранный массив по узлам вхождения:
если узел вхождения равен i (начальному узлу пути), то процесс
развертывания этого узла в его входящие завершается, и переходим
к следующему узлу (строке массива) для его возможной развертки.
 Если же узел вхождения не равен i, то он развертывается:
 Выбираются все узлы, входящие в него: формируется новая выборка
 */

  //просмотр IBClientDataSet2

   ds2=IBClientDataSet2;
   ds2->First();

   IndMas=0;
   while(!ds2->Eof)   //здесь выборка узлов, входящих в узел j
    {
      M[IndMas].nn  = StrToInt(LabeledEdit2->Text); //входит в j
     M[IndMas].iz  = ds2->FieldByName("nodnumin")->AsInteger;
      M[IndMas].len = ds2->FieldByName("lengthin")->AsFloat;
    //если узел развертывается, он помечается звездочкой:
      M[IndMas].sign="*";

    ds1=IBClientDataSet1;
    k=M[IndMas].iz;
//обработка на развертку
    unfolding(ds1,M,k);

    ds2->Next();
  } //while
 //пересылка структуры в Memo для проверки заполнения
 /* Memo1->Clear();
    int ii=0;
   while(M[ii].nn != EndOfArr)
    {
```

```
    AnsiString s=IntToStr(M[ii].iz)+ "***"
    +IntToStr(M[ii].nn)+ "***" +
     FloatToStr(M[ii].len)+ "***  " +M[ii].sign;

     Memo1->Lines->Add(s);
     ii++;
    }
  */
//здесь все узлы после 1-го шага развернуты и записаны в массив
//но в массиве могут быть узлы, которые надо развертывать дальше.
//Поэтому обрабатываем массив дальше до тех пор, пока все узлы
// не станут иметь пометку  "*", т. е. будут все развернуты

   k=0;
   while (1)
    {
     IndMas=0;
     while (M[IndMas].nn != EndOfArr)
      {
       AnsiString s=M[IndMas].sign;
       int bb, b=s.LastDelimiter("*");
       bb=M[IndMas].nn;
       if(b !=0 || bb ==0) //помечен - пропускаем
       {
        IndMas++;
        continue;
       }
      else    //не помечен - развертываем
        {
          k=1;
          M[IndMas].sign="*";
          ds1=IBClientDataSet1;
         //запомнить запись, с которой ушли:
           int sto=IndMas;
           int n=M[IndMas].iz;
```

```
          unfolding(ds1,M,n); //обработка на развертку
            //восстановить номер записи
          IndMas=sto;
          IndMas++;
            continue;
        }  //else
      } //while (!eof)
      if(k==1)     //условие окончания: больше нечего развертывать
      {
        k=0;
        continue;
      }
       else break;
    } //while(1)
  /* в этом месте в массиве все узлы на пути от i к j и среди них есть
   повторяющиеся*/
//пересылка структуры в Memo для проверки заполнения
/*
  Memo1->Clear();
   ii=0;
   while(M[ii].nn != EndOfArr)
    {
    AnsiString s=IntToStr(M[ii].iz)+ "***"
    +IntToStr(M[ii].nn)+ "***" +
     FloatToStr(M[ii].len)+ "***   " +M[ii].sign;

     Memo1->Lines->Add(s);
     ii++;
     }
*/


//Начало формирования минимального пути по данным массива.
/*Работаем, начиная с узла i - начального и выбираем из всех в него
входящих тот, у которого минимальная длина пути. Узлы формируем
в NewTable */
//Пересылка данных из массива структур в таблицу
```

```
IBTable1->EmptyTable();
IBTable1->ApplyUpdates();
IBTable1->Refresh();

int ii=0;
while(M[ii].nn != EndOfArr)
 {
   IBTable1->Append();
   IBTable1->FieldByName("iz")->AsInteger = M[ii].iz;
   IBTable1->FieldByName("nn")->AsInteger = M[ii].nn;
   IBTable1->FieldByName("len")->AsFloat  = M[ii].len;

   IBTable1->Post();
    //чтобы этот Post() работал, надо установить CashedUpdates=true
    ii++;
    IBTable1->Next();
 }
   IBTable1->ApplyUpdates();
   IBTable1->Refresh();

  IBTable1->IndexName="INDNEWTABLE";
//включаем индекс таблицы для сортировки по iz
//В таблице будут повторяющиеся строки. Но у них будут одинаковые номера
//узлов и длины, и поэтому они ничего не испортят

float su=0,min,cur;
AnsiString A="";
int nod=i,iz,izf,fix;
A+=IntToStr(nod);

while(1)
 {
  //поиск необходимой строки с данным узлом
  TLocateOptions Opts;
     Opts.Clear();
     Opts << loPartialKey;
```

```
   Variant locvalues[1];
   locvalues[0] = Variant(nod);
int i=IBTable1->Locate("iz",VarArrayOf(locvalues, 0), Opts);
if(i==0)
 {
 ShowMessage("По данному запросу путь не найден”);
 LabeledEdit1->SetFocus();
 return;
 }

  iz=IBTable1->FieldByName("iz")->AsInteger;
  min=IBTable1->FieldByName("len")->AsFloat;
  cur=IBTable1->FieldByName("len")->AsFloat;
  nod=IBTable1->FieldByName("nn")->AsInteger;
  fix=0;
  //поиск минимальной длины
  while(!IBTable1->Eof)
   {
     IBTable1->Next();
     izf=IBTable1->FieldByName("iz")->AsInteger;
     if(izf != iz) /*попали на новый узел и в min – минимальная длина,
                    а в nod – nn. Переход на поиск нового узла.*/
       {
        su+=min;
        A+=+ "->" +IntToStr(nod);
        fix=1;
        break;
       }
     else
      {
        cur=IBTable1->FieldByName("len")->AsFloat;
        if(min <= cur)
         continue;
        else
         {
          min=cur;
```

```
            nod=IBTable1->FieldByName("nn")->AsInteger;
            continue;
           }
        }//else
     } //while(!
     //обработка конца таблицы
      if(fix==0) //завершение по концу таблицы
       {
         su+=min;
         A+=+ "->" + IntToStr(nod);
       }
     if(nod==j)
      break;
 }//while(1)

 LabeledEdit3->Text=A;
 LabeledEdit4->Text=FloatToStrF(su,sffFixed,10,2);

}  //от обработчика
//---------------------------------------------------------------------------

void __fastcall TForm1::LabeledEdit1KeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
   if(Key == VK_RETURN)
   {
     LabeledEdit2->SetFocus();
   }
}
//---------------------------------------------------------------------------

void __fastcall TForm1::LabeledEdit1Enter(TObject *Sender)
{
 LabeledEdit1->Text="";
 LabeledEdit2->Text="";
```

```
 LabeledEdit3->Text="";
 LabeledEdit4->Text="";
 //инициализация M[]
 for(int i=0; i < MaxInd; i++)
  M[i].nn=EndOfArr;
  IBTable1->IndexName=""; //отключить индекс
}
//---------------------------------------------------------------------------
```

h-файл

```
//---------------------------------------------------------------------------

#ifndef Unit1H
#define Unit1H
//---------------------------------------------------------------------------
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
#include <ComCtrls.hpp>
#include <DB.hpp>
#include <DBClient.hpp>
#include <DBLocal.hpp>
#include <DBLocalI.hpp>
#include <IBDatabase.hpp>
#include <Provider.hpp>
#include <DBCtrls.hpp>
#include <Mask.hpp>
#include <DBGrids.hpp>
#include <ExtCtrls.hpp>
#include <Grids.hpp>
#include <IBCustomDataSet.hpp>
#include <IBTable.hpp>
#include <DBTables.hpp>
//---------------------------------------------------------------------------
class TForm1 : public TForm
```

```
{
__published:  // IDE-managed Components
        TIBDatabase *IBDatabase1;
        TIBTransaction *IBTransaction1;
        TIBClientDataSet *IBClientDataSet1;
        TDataSource *DataSource1;
        TPageControl *PageControl1;
        TTabSheet *TabSheet1;
        TTabSheet *TabSheet2;
        TLabel *Label1;
        TDBEdit *DBEdit1;
        TLabel *Label2;
        TDBEdit *DBEdit2;
        TLabel *Label3;
        TDBEdit *DBEdit3;
        TDBNavigator *DBNavigator1;
        TButton *Button1;
        TButton *Button2;
        TDBGrid *DBGrid1;
        TButton *Button3;
        TLabeledEdit *LabeledEdit1;
        TLabeledEdit *LabeledEdit2;
        TLabeledEdit *LabeledEdit3;
        TDataSource *DataSource2;
        TIBClientDataSet *IBClientDataSet2;
        TIBTable *IBTable1;
        TIBTransaction *IBTransaction2;
        TDataSource *DataSource3;
        TLabeledEdit *LabeledEdit4;
        TIntegerField *IBClientDataSet1NODNUMBER;
        TIntegerField *IBClientDataSet1NODNUMIN;
        TFloatField *IBClientDataSet1LENGTHIN;
        void __fastcall Button1Click(TObject *Sender);
        void __fastcall TabSheet1Show(TObject *Sender);
        void __fastcall Button2Click(TObject *Sender);
        void __fastcall DBEdit1Enter(TObject *Sender);
```

```
        void __fastcall DBEdit1KeyDown(TObject *Sender, WORD &Key,
          TShiftState Shift);
        void __fastcall DBEdit2KeyDown(TObject *Sender, WORD &Key,
          TShiftState Shift);
        void __fastcall DBEdit3KeyDown(TObject *Sender, WORD &Key,
          TShiftState Shift);
        void __fastcall Button3Click(TObject *Sender);
        void __fastcall DBNavigator1Click(TObject *Sender,
          TNavigateBtn Button);
        void __fastcall TabSheet2Show(TObject *Sender);
        void __fastcall LabeledEdit2KeyDown(TObject *Sender, WORD &Key,
          TShiftState Shift);
        void __fastcall LabeledEdit1KeyDown(TObject *Sender, WORD &Key,
          TShiftState Shift);
        void __fastcall LabeledEdit1Enter(TObject *Sender);
private:      // User declarations
public:       // User declarations
        __fastcall TForm1(TComponent* Owner);
};
//---------------------------------------------------------------------------
extern PACKAGE TForm1 *Form1;
//---------------------------------------------------------------------------
#endif
```

dfm-файл
(в нем указываются описания компонентов формы)

```
object Form1: TForm1
  Left = 210
  Top = 105
  Width = 540
  Height = 308
  Caption = 'Form1'
  Color = clWhite
  Font.Charset = DEFAULT_CHARSET
  Font.Color = clWindowText
  Font.Height = -11
```

```
Font.Name = 'MS Sans Serif'
Font.Style = []
OldCreateOrder = False
PixelsPerInch = 96
TextHeight = 13
object PageControl1: TPageControl
  Left = 8
  Top = 48
  Width = 513
  Height = 217
  ActivePage = TabSheet1
  Style = tsButtons
  TabIndex = 0
  TabOrder = 0
  object TabSheet1: TTabSheet
    Caption = 'Ввод схемы'
    OnShow = TabSheet1Show
    object Label1: TLabel
      Left = 0
      Top = 8
      Width = 131
      Height = 13
      Caption = 'Номер текущего узла'
      FocusControl = DBEdit1
      Font.Charset = DEFAULT_CHARSET
      Font.Color = clWindowText
      Font.Height = -11
      Font.Name = 'MS Sans Serif'
      Font.Style = [fsBold]
      ParentFont = False
    end
    object Label2: TLabel
      Left = 0
      Top = 48
      Width = 139
      Height = 13
```

```
    Caption = 'Номер входящего узла'
    FocusControl = DBEdit2
    Font.Charset = DEFAULT_CHARSET
    Font.Color = clWindowText
    Font.Height = -11
    Font.Name = 'MS Sans Serif'
    Font.Style = [fsBold]
    ParentFont = False
  end
  object Label3: TLabel
    Left = 0
    Top = 88
    Width = 185
    Height = 13
    Caption = 'Длина пути от входящего узла'
    FocusControl = DBEdit3
    Font.Charset = DEFAULT_CHARSET
    Font.Color = clWindowText
    Font.Height = -11
    Font.Name = 'MS Sans Serif'
    Font.Style = [fsBold]
    ParentFont = False
  end
  object DBEdit1: TDBEdit
    Left = 0
    Top = 24
    Width = 49
    Height = 24
    Color = clWhite
    DataField = 'NODNUMBER'
    DataSource = DataSource1
    Font.Charset = DEFAULT_CHARSET
    Font.Color = clRed
    Font.Height = -13
    Font.Name = 'MS Sans Serif'
    Font.Style = [fsBold]
```

```
    ParentFont = False
    TabOrder = 0
    OnEnter = DBEdit1Enter
    OnKeyDown = DBEdit1KeyDown
  end
  object DBEdit2: TDBEdit
    Left = 0
    Top = 64
    Width = 49
    Height = 24
    DataField = 'NODNUMIN'
    DataSource = DataSource1
    Font.Charset = DEFAULT_CHARSET
    Font.Color = clRed
    Font.Height = -13
    Font.Name = 'MS Sans Serif'
    Font.Style = [fsBold]
    ParentFont = False
    TabOrder = 1
    OnKeyDown = DBEdit2KeyDown
  end
  object DBEdit3: TDBEdit
    Left = 0
    Top = 104
    Width = 73
    Height = 24
    DataField = 'LENGTHIN'
    DataSource = DataSource1
    Font.Charset = DEFAULT_CHARSET
    Font.Color = clRed
    Font.Height = -13
    Font.Name = 'MS Sans Serif'
    Font.Style = [fsBold]
    ParentFont = False
    TabOrder = 2
```

```
    OnKeyDown = DBEdit3KeyDown
  end
  object DBNavigator1: TDBNavigator
    Left = 193
    Top = 0
    Width = 240
    Height = 25
    DataSource = DataSource1
    TabOrder = 3
    OnClick = DBNavigator1Click
  end
  object Button1: TButton
    Left = -3
    Top = 136
    Width = 193
    Height = 25
    Caption = 'Обновить таблицу '
    Font.Charset = DEFAULT_CHARSET
    Font.Color = clBlack
    Font.Height = -19
    Font.Name = 'MS Sans Serif'
    Font.Style = [fsBold]
    ParentFont = False
    TabOrder = 4
    OnClick = Button1Click
  end
  object Button2: TButton
    Left = 117
    Top = 23
    Width = 393
    Height = 25
    Caption = 'Работа с табличным представлением'
    Font.Charset = DEFAULT_CHARSET
    Font.Color = clWindowText
    Font.Height = -19
    Font.Name = 'MS Sans Serif'
```

```
    Font.Style = [fsBold]
    ParentFont = False
    TabOrder = 5
    OnClick = Button2Click
  end
  object DBGrid1: TDBGrid
    Left = 195
    Top = 50
    Width = 313
    Height = 127
    DataSource = DataSource1
    TabOrder = 6
    TitleFont.Charset = DEFAULT_CHARSET
    TitleFont.Color = clWindowText
    TitleFont.Height = -11
    TitleFont.Name = 'MS Sans Serif'
    TitleFont.Style = []
    Visible = False
    Columns = <
      item
        Expanded = False
        FieldName = 'NODNUMBER'
        Title.Caption = 'Номер тек. узла'
        Width = 72
        Visible = True
      end
      item
        Expanded = False
        FieldName = 'NODNUMIN'
        Title.Caption = 'Номер вход. узла'
        Width = 76
        Visible = True
      end
      item
        Expanded = False
        FieldName = 'LENGTHIN'
```

```
        Title.Caption = 'Длина пути от вх. узла'
        Visible = True
      end>
  end
end
object TabSheet2: TTabSheet
  Caption = 'Расчет пути'
  ImageIndex = 1
  OnShow = TabSheet2Show
  object LabeledEdit1: TLabeledEdit
    Left = 16
    Top = 16
    Width = 57
    Height = 24
    EditLabel.Width = 199
    EditLabel.Height = 13
    EditLabel.Caption = 'Введите номер начального узла пути'
    EditLabel.Color = clWhite
    EditLabel.Font.Charset = DEFAULT_CHARSET
    EditLabel.Font.Color = clBlack
    EditLabel.Font.Height = -11
    EditLabel.Font.Name = 'MS Sans Serif'
    EditLabel.Font.Style = [fsBold]
    EditLabel.ParentColor = False
    EditLabel.ParentFont = False
    Font.Charset = DEFAULT_CHARSET
    Font.Color = clRed
    Font.Height = -13
    Font.Name = 'MS Sans Serif'
    Font.Style = [fsBold]
    LabelPosition = lpAbove
    LabelSpacing = 3
    ParentFont = False
    TabOrder = 0
    OnEnter = LabeledEdit1Enter
```

```
  OnKeyDown = LabeledEdit1KeyDown
end
object LabeledEdit2: TLabeledEdit
  Left = 16
  Top = 56
  Width = 57
  Height = 24
  EditLabel.Width = 196
  EditLabel.Height = 13
  EditLabel.Caption = 'Введите номер конечного узла пути'
  EditLabel.Font.Charset = DEFAULT_CHARSET
  EditLabel.Font.Color = clWindowText
  EditLabel.Font.Height = -11
  EditLabel.Font.Name = 'MS Sans Serif'
  EditLabel.Font.Style = [fsBold]
  EditLabel.ParentFont = False
  Font.Charset = DEFAULT_CHARSET
  Font.Color = clRed
  Font.Height = -13
  Font.Name = 'MS Sans Serif'
  Font.Style = [fsBold]
  LabelPosition = lpAbove
  LabelSpacing = 3
  ParentFont = False
  TabOrder = 1
  OnKeyDown = LabeledEdit2KeyDown
end
object LabeledEdit3: TLabeledEdit
  Left = 0
  Top = 109
  Width = 457
  Height = 24
  EditLabel.Width = 454
  EditLabel.Height = 24
  EditLabel.Caption = 'Кратчайший путь между введенными узлами:'
  EditLabel.Font.Charset = DEFAULT_CHARSET
```

```
    EditLabel.Font.Color = clRed
    EditLabel.Font.Height = -19
    EditLabel.Font.Name = 'MS Sans Serif'
    EditLabel.Font.Style = [fsBold]
    EditLabel.ParentFont = False
    Font.Charset = DEFAULT_CHARSET
    Font.Color = clBlue
    Font.Height = -13
    Font.Name = 'MS Sans Serif'
    Font.Style = [fsBold]
    LabelPosition = lpAbove
    LabelSpacing = 3
    ParentFont = False
    TabOrder = 2
  end
  object LabeledEdit4: TLabeledEdit
    Left = 0
    Top = 157
    Width = 121
    Height = 24
    EditLabel.Width = 226
    EditLabel.Height = 24
    EditLabel.Caption = 'Суммарная длина пути'
    EditLabel.Font.Charset = DEFAULT_CHARSET
    EditLabel.Font.Color = clBlue
    EditLabel.Font.Height = -19
    EditLabel.Font.Name = 'MS Sans Serif'
    EditLabel.Font.Style = [fsBold]
    EditLabel.ParentFont = False
    Font.Charset = DEFAULT_CHARSET
    Font.Color = clRed
    Font.Height = -13
    Font.Name = 'MS Sans Serif'
    Font.Style = [fsBold]
    LabelPosition = lpAbove
    LabelSpacing = 3
```

```
      ParentFont = False
      TabOrder = 3
    end
  end
end
object Button3: TButton
  Left = 16
  Top = 8
  Width = 177
  Height = 25
  Caption = 'Выход из задачи'
  Font.Charset = DEFAULT_CHARSET
  Font.Color = clWindowText
  Font.Height = -19
  Font.Name = 'MS Sans Serif'
  Font.Style = [fsBold]
  ParentFont = False
  TabOrder = 1
  OnClick = Button3Click
end
object IBDatabase1: TIBDatabase
  Connected = True
  DatabaseName = 'D:\InterBase\DataBases\MINROAD.GDB'
  Params.Strings = (
    'user_name=sysdba')
  DefaultTransaction = IBTransaction1
  IdleTimer = 0
  SQLDialect = 3
  TraceFlags = []
  Left = 216
  Top = 8
end
object IBTransaction1: TIBTransaction
  Active = True
  AutoStopAction = saNone
  Left = 248
```

```
    Top = 8
  end
  object IBClientDataSet1: TIBClientDataSet
    CommandText = 'select * from INOUTNODS'
    Aggregates = <>
    Options = [poAllowCommandText]
    Params = <>
    DBConnection = IBDatabase1
    DBTransaction = IBTransaction1
    Left = 216
    Top = 40
    object IBClientDataSet1NODNUMBER: TIntegerField
      FieldName = 'NODNUMBER'
      Origin = 'INOUTNODS.NODNUMBER'
      Required = True
    end
    object IBClientDataSet1NODNUMIN: TIntegerField
      FieldName = 'NODNUMIN'
      Origin = 'INOUTNODS.NODNUMIN'
      Required = True
    end
    object IBClientDataSet1LENGTHIN: TFloatField
      FieldName = 'LENGTHIN'
      Origin = 'INOUTNODS.LENGTHIN'
      Required = True
    end
  end
  object DataSource1: TDataSource
    DataSet = IBClientDataSet1
    Left = 248
    Top = 40
  end
  object DataSource2: TDataSource
    DataSet = IBClientDataSet2
    Left = 328
```

```
  Top = 8
end
object IBClientDataSet2: TIBClientDataSet
  CommandText = 'select distinct * from NEWTABLE order by iz'
  Aggregates = <>
  Options = [poAllowCommandText]
  Params = <>
  DBConnection = IBDatabase1
  DBTransaction = IBTransaction1
  Left = 296
  Top = 8
end
object IBTable1: TIBTable
  Database = IBDatabase1
  Transaction = IBTransaction2
  BufferChunks = 1000
  CachedUpdates = True
  FieldDefs = <
    item
      Name = 'NN'
      DataType = ftInteger
    end
    item
      Name = 'LEN'
      DataType = ftFloat
    end
    item
      Name = 'SIGN'
      DataType = ftString
      Size = 2
    end
    item
      Name = 'IZ'
      DataType = ftInteger
    end>
```

```
    IndexDefs = <
      item
        Name = 'INDNEWTABLE'
        Fields = 'IZ'
      end>
    StoreDefs = True
    TableName = 'NEWTABLE'
    Left = 296
    Top = 40
  end
  object IBTransaction2: TIBTransaction
    Active = False
    DefaultDatabase = IBDatabase1
    AutoStopAction = saNone
    Left = 328
    Top = 40
  end
  object DataSource3: TDataSource
    DataSet = IBTable1
    Left = 360
    Top = 40
  end
end
```

Расчет кратчайшего пути

Для расчета кратчайшего пути между двумя точками в сети путей используется рабочая таблица:

```
Create Table NewTable
 (
   nn integer,  //номер узла, в который входит путь от другого узла
   len float,  //длина пути между двумя узлами
   iz integer,  //номер узла, из которого исходит путь в другой узел
   sign char(2)
```

```
    //здесь формируется вспомогательный пометочный символ
 );
```

Для сортировки данных по iz к таблице создан индекс:

```
 CREATE INDEX indnewtable
ON NewTable (iz);
```

Алгоритм

Машинный (формальный) алгоритм расчета кратчайшего пути, естественно, отличается от алгоритма ручного, приведенного ранее. Более подробный его вариант следует из пояснений к тексту листинга 4.3.

По номеру конечного узла пути из таблицы путей выбираются все узлы, входящие в этот узел. А далее в обратном порядке происходит развертка всех этих узлов до начального узла. При этом развертываемые узлы помечаются символом *. Все развертываемые узлы помещаются в специальный массив структур (он выбран просто с целью ускорения процесса расчета, что вводит ограничения на объем сети путей, хотя можно было бы взять и обычную IB-таблицу, которая это ограничение снимает за счет увеличения времени расчета). Процесс развертки итеративно повторяется до тех пор, пока все развертываемые узлы не окажутся помеченными. Что такой процесс будет сходиться — ясно хотя бы из конечности самой схемы путей. После полной развертки массив результатов переписывается в рабочую таблицу с целью сортировки данных по полю iz. Затем, собственно, и начинается формироваться кратчайший путь:

1. Среди всех раскрытых узлов находится заданный начальный.
2. Затем среди всех узлов, исходящих из начального, находится узел с минимальной длиной пути.
3. Затем в таблице находится запись с этим номером узла.
4. Процесс повторяется до тех пор, пока очередной найденный узел не совпадет с конечным заданным узлом.
5. Одновременно с поиском узлов накапливаются и соответствующие им длины путей.

Результаты расчетов на контрольном примере, рассмотренном ранее, показаны на рис. 4.9.

Тексты приложения показаны в листинге 4.3.

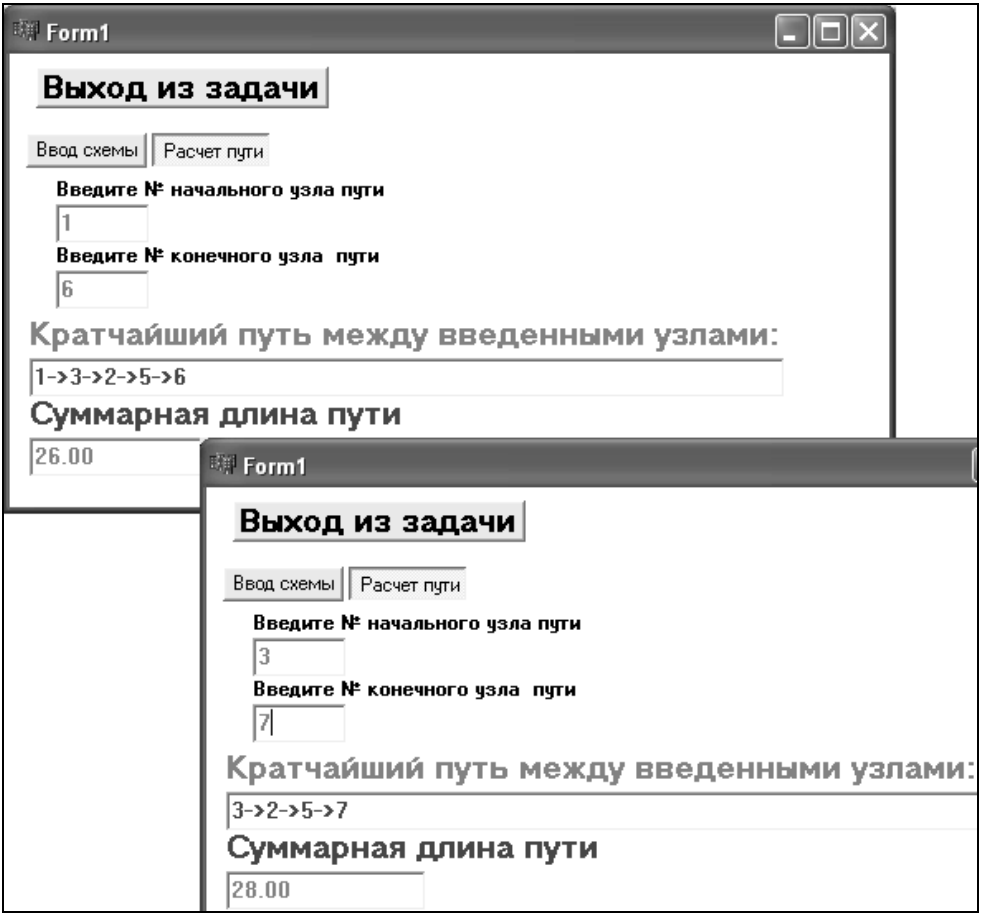


Рис. 4.9. Расчет кратчайшего пути между двумя точками

Листинг 4.3

cpp-файл

```
//---------------------------------------------------------------------------

#include <vcl.h>
#pragma hdrstop

#include "Unit1.h"
```

```
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"

#define MaxInd 1000 //максимальное количество узлов сети
#define EndOfArr 999999  //признак конца массива
TForm1 *Form1;

TIBClientDataSet *ds1,*ds2;
int IndMas;
 struct st
   {
     int nn;
     int iz;
     float len;
     AnsiString sign;
    }M[MaxInd];

enum TStringFloatFormat {sffGeneral, sffExponent, sffFixed, sffNumber,
sffCurrency };

// функция развертывания узла назад:
// записывает в массив tr все узлы, входящие в nod с длинами их путей
 TLocateOptions Opts;
void unfolding(TIBClientDataSet *ds,struct st M[],int nod);
//---------------------------------------------------------------------------
void unfolding(TIBClientDataSet *ds,struct st M[],int nod)
 {
  // подгонка массива
  IndMas=0;
  while(M[IndMas].nn != EndOfArr)
   IndMas++;

  int  nodnumin;
   ds->Close();
```

```
   ds->CommandText="select * from inoutnods where nodnumber="+ In-
tToStr(nod);
   ds->Open();

   if(ds->FieldByName("nodnumin")->AsInteger == 0)
    return; // в данный узел не входит ни один другой узел
   while (!ds->Eof)
    {
     nodnumin= ds->FieldByName("nodnumin")->AsInteger;
     if(ds->FieldByName("nodnumin")->AsInteger == 0)
      {
        ds->Next();
        continue;
       }
      M[IndMas].iz=ds->FieldByName("nodnumin")->AsInteger;
      M[IndMas].nn=nod; // это - в который входит
      M[IndMas].len=ds->FieldByName("lengthin")->AsFloat;
      M[IndMas].sign="";
      IndMas++;
      ds->Next();
    } //while
    return;
 }
//---------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------
void __fastcall TForm1::Button1Click(TObject *Sender)
{
 if(!IBClientDataSet1->Active)
   IBClientDataSet1->Open();
 IBClientDataSet1->ApplyUpdates(-1);
}
```

```
//---------------------------------------------------------------------------
void __fastcall TForm1::TabSheet1Show(TObject *Sender)
{
 if(!IBClientDataSet1->Active)
   IBClientDataSet1->Open();
 IBClientDataSet1->Edit();
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Button2Click(TObject *Sender)
{

   if(DBGrid1->Visible)
    {
     DBGrid1->Hide();
     Button2->Caption="Работа с табличным представлением";
    }
  else
   {
    Button2->Caption="Закрыть табличное представление";
    DBGrid1->Show();
   }
}
//---------------------------------------------------------------------------

void __fastcall TForm1::DBEdit1Enter(TObject *Sender)
{
 if(!IBClientDataSet1->Active)
   IBClientDataSet1->Open();
 IBClientDataSet1->Edit();
}
//---------------------------------------------------------------------------

void __fastcall TForm1::DBEdit1KeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if(Key == VK_RETURN)
```

```
   {
     DBEdit2->SetFocus();
   }
}
//---------------------------------------------------------------------------

void __fastcall TForm1::DBEdit2KeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
 if(Key == VK_RETURN)
   {
     DBEdit3->SetFocus();
   }
}
//---------------------------------------------------------------------------

void __fastcall TForm1::DBEdit3KeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
 if(Key == VK_RETURN)
   {
     DBNavigator1->SetFocus();
   }
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Button3Click(TObject *Sender)
{
 Application->Terminate();
}
//---------------------------------------------------------------------------


void __fastcall TForm1::DBNavigator1Click(TObject *Sender,
      TNavigateBtn Button)
{
```

```
     DBEdit1->SetFocus();
}
//---------------------------------------------------------------------------

void __fastcall TForm1::TabSheet2Show(TObject *Sender)
{
 if(!IBClientDataSet1->Active)
   IBClientDataSet1->Open();
}
//---------------------------------------------------------------------------

void __fastcall TForm1::LabeledEdit2KeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
 if(Key != VK_RETURN)
  return;

 // выборка из схемы всех узлов, входящих в конечный узел J
  IBClientDataSet2->Close();
  IBClientDataSet2->CommandText="select * from inoutnods where
  nodnumber="
  + LabeledEdit2->Text;
  IBClientDataSet2->Open();
  int i=StrToInt(LabeledEdit1->Text); // начальная точка пути
  int j=StrToInt(LabeledEdit2->Text); //конечная точка пути
  int k,l;

  IBTable1->Close();
  IBTable1->Open();
/*Все входящие узлы по мере их обработки
помещаем в массив вместе с их длинами
и начинаем просматривать выбранный массив по узлам вхождения:
если узел вхождения равен i (начальному узлу пути), то процесс
развертывания этого узла в его входящие завершается, и переходим
к следующему узлу (строке массива) для его возможной развертки.
```

```
Если же узел вхождения не равен i, то он развертывается:
выбираются все узлы, входящие в него: формируется новая выборка
 */
   ds2=IBClientDataSet2;
   ds2->First();

   IndMas=0;
   while(!ds2->Eof)   // здесь - выборка узлов, входящих в узел j
    {
      M[IndMas].nn  = StrToInt(LabeledEdit2->Text); // входит в j
      M[IndMas].iz  = ds2->FieldByName("nodnumin")->AsInteger;
      M[IndMas].len = ds2->FieldByName("lengthin")->AsFloat;
    // если узел развертывается, он помечается звездочкой:
      M[IndMas].sign="*";

    ds1=IBClientDataSet1;
    k=M[IndMas].iz;
// обработка на развертку
    unfolding(ds1,M,k);

    ds2->Next();
  } //while
 // пересылка структуры в Memo для проверки заполнения
 /* Memo1->Clear();
    int ii=0;
   while(M[ii].nn != EndOfArr)
    {
    AnsiString s=IntToStr(M[ii].iz)+ "***"
    +IntToStr(M[ii].nn)+ "***" +
     FloatToStr(M[ii].len)+ "***  " +M[ii].sign;

     Memo1->Lines->Add(s);
     ii++;
    }
  */
```

```
 //здесь все узлы после 1-го шага развернуты и записаны в массив
 //но в массиве могут быть узлы, которые надо развертывать дальше.
//Поэтому обрабатываем массив дальше до тех пор, пока все узлы
// не станут иметь пометку "*", т. е. будут все развернуты

   k=0;
   while (1)
    {
     IndMas=0;
     while (M[IndMas].nn != EndOfArr)
      {
       AnsiString s=M[IndMas].sign;
       int bb, b=s.LastDelimiter("*");
       bb=M[IndMas].nn;
       if(b !=0 || bb ==0) // помечен - пропускаем
       {
        IndMas++;
        continue;
       }
      else    // не помечен - развертываем
        {
          k=1;
          M[IndMas].sign="*";
          ds1=IBClientDataSet1;
         // запомнить запись, с которой ушли:
           int sto=IndMas;
           int n=M[IndMas].iz;
           unfolding(ds1,M,n); // обработка на развертку
             // восстановить номер записи
           IndMas=sto;
           IndMas++;
             continue;
        }  //else
      } //while (!eof)
      if(k==1)     // условие окончания: больше нечего развертывать
```

```
       {
         k=0;
         continue;
       }
        else break;
     } //while(1)
     } //while(1)
  /* в этом месте в массиве – все узлы на пути от i к j и среди них есть
   повторяющиеся*/
//пересылка структуры в Memo для проверки заполнения
/*
  Memo1->Clear();
   ii=0;
   while(M[ii].nn != EndOfArr)
    {
    AnsiString s=IntToStr(M[ii].iz)+ "***"
    +IntToStr(M[ii].nn)+ "***" +
     FloatToStr(M[ii].len)+ "***   " +M[ii].sign;

     Memo1->Lines->Add(s);
     ii++;
     }
*/



//Начало формирования минимального пути по данным массива.
/*Работаем, начиная с узла i - начального и выбираем из всех в него
входящих тот, у которого минимальная длина пути. Узлы формируем
в NewTable */
//Пересылка данных из массива структур в таблицу

IBTable1->EmptyTable();
IBTable1->ApplyUpdates();
IBTable1->Refresh();

int ii=0;
while(M[ii].nn != EndOfArr)
```

```
 {
   IBTable1->Append();
   IBTable1->FieldByName("iz")->AsInteger = M[ii].iz;
   IBTable1->FieldByName("nn")->AsInteger = M[ii].nn;
   IBTable1->FieldByName("len")->AsFloat  = M[ii].len;

   IBTable1->Post();
    // чтобы этот Post() работал, надо установить CashedUpdates=true
    ii++;
    IBTable1->Next();
 }
   IBTable1->ApplyUpdates();
   IBTable1->Refresh();


IBTable1->IndexName="INDNEWTABLE"; //включаем индекс таблицы для
//сортировки по iz
//В таблице будут повторяющиеся строки. Но у них будут одинаковые номера
//узлов и длины и поэтому они ничего не испортят
float su=0,min,cur;
AnsiString A="";
int nod=i,iz,izf,fix;
A+=IntToStr(nod);


while(1)
 {
  // поиск необходимой строки с данным узлом
  TLocateOptions Opts;
     Opts.Clear();
     Opts << loPartialKey;
     Variant locvalues[1];
     locvalues[0] = Variant(nod);
  int i=IBTable1->Locate("iz",VarArrayOf(locvalues, 0), Opts);
  if(i==0)
   {
   ShowMessage("По данному запросу путь не найден ");
   LabeledEdit1->SetFocus();
```

```
return;
}

 iz=IBTable1->FieldByName("iz")->AsInteger;
 min=IBTable1->FieldByName("len")->AsFloat;
 cur=IBTable1->FieldByName("len")->AsFloat;
 nod=IBTable1->FieldByName("nn")->AsInteger;
 fix=0;
 // поиск минимальной длины
 while(!IBTable1->Eof)
  {
    IBTable1->Next();
    izf=IBTable1->FieldByName("iz")->AsInteger;
    if(izf != iz) /*попали на новый узел и в min - минимальная длина,
                    а в nod - nn. Переход на поиск нового узла.*/
      {
       su+=min;
       A+=+ "->" +IntToStr(nod);
       fix=1;
       break;
      }
    else
     {
       cur=IBTable1->FieldByName("len")->AsFloat;
       if(min <= cur)
        continue;
       else
        {
         min=cur;
         nod=IBTable1->FieldByName("nn")->AsInteger;
         continue;
        }
     }//else
  } //while(!
  // обработка конца таблицы
   if(fix==0) // завершение по концу таблицы
```

```
       {
         su+=min;
         A+=+ "->" + IntToStr(nod);
       }
     if(nod==j)
      break;
 }//while(1)

 LabeledEdit3->Text=A;
 LabeledEdit4->Text=FloatToStrF(su,sffFixed,10,2);

}  // от обработчика

//---------------------------------------------------------------------------

void __fastcall TForm1::LabeledEdit1KeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
   if(Key == VK_RETURN)
   {
     LabeledEdit2->SetFocus();
   }
}
//---------------------------------------------------------------------------

void __fastcall TForm1::LabeledEdit1Enter(TObject *Sender)
{
 LabeledEdit1->Text="";
 LabeledEdit2->Text="";
 LabeledEdit3->Text="";
 LabeledEdit4->Text="";
 //инициализация M[]
 for(int i=0; i < MaxInd; i++)
  M[i].nn=EndOfArr;
  IBTable1->IndexName=""; // отключить индекс
}
```

```
//---------------------------------------------------------------------------
h-файл
//---------------------------------------------------------------------------

#ifndef Unit1H
#define Unit1H
//---------------------------------------------------------------------------
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
#include <ComCtrls.hpp>
#include <DB.hpp>
#include <DBClient.hpp>
#include <DBLocal.hpp>
#include <DBLocalI.hpp>
#include <IBDatabase.hpp>
#include <Provider.hpp>
#include <DBCtrls.hpp>
#include <Mask.hpp>
#include <DBGrids.hpp>
#include <ExtCtrls.hpp>
#include <Grids.hpp>
#include <IBCustomDataSet.hpp>
#include <IBTable.hpp>
#include <DBTables.hpp>
//---------------------------------------------------------------------------
class TForm1 : public TForm
{
__published:  // IDE-managed Components
        TIBDatabase *IBDatabase1;
        TIBTransaction *IBTransaction1;
        TIBClientDataSet *IBClientDataSet1;
        TDataSource *DataSource1;
        TPageControl *PageControl1;
        TTabSheet *TabSheet1;
```

```
TTabSheet *TabSheet2;
TLabel *Label1;
TDBEdit *DBEdit1;
TLabel *Label2;
TDBEdit *DBEdit2;
TLabel *Label3;
TDBEdit *DBEdit3;
TDBNavigator *DBNavigator1;
TButton *Button1;
TButton *Button2;
TDBGrid *DBGrid1;
TButton *Button3;
TLabeledEdit *LabeledEdit1;
TLabeledEdit *LabeledEdit2;
TLabeledEdit *LabeledEdit3;
TDataSource *DataSource2;
TIBClientDataSet *IBClientDataSet2;
TIBTable *IBTable1;
TIBTransaction *IBTransaction2;
TDataSource *DataSource3;
TLabeledEdit *LabeledEdit4;
TIntegerField *IBClientDataSet1NODNUMBER;
TIntegerField *IBClientDataSet1NODNUMIN;
TFloatField *IBClientDataSet1LENGTHIN;
void __fastcall Button1Click(TObject *Sender);
void __fastcall TabSheet1Show(TObject *Sender);
void __fastcall Button2Click(TObject *Sender);
void __fastcall DBEdit1Enter(TObject *Sender);
void __fastcall DBEdit1KeyDown(TObject *Sender, WORD &Key,
  TShiftState Shift);
void __fastcall DBEdit2KeyDown(TObject *Sender, WORD &Key,
  TShiftState Shift);
void __fastcall DBEdit3KeyDown(TObject *Sender, WORD &Key,
  TShiftState Shift);
void __fastcall Button3Click(TObject *Sender);
```

```
        void __fastcall DBNavigator1Click(TObject *Sender,
          TNavigateBtn Button);
        void __fastcall TabSheet2Show(TObject *Sender);
        void __fastcall LabeledEdit2KeyDown(TObject *Sender, WORD &Key,
          TShiftState Shift);
        void __fastcall LabeledEdit1KeyDown(TObject *Sender, WORD &Key,
          TShiftState Shift);
        void __fastcall LabeledEdit1Enter(TObject *Sender);
private:      // User declarations
public:       // User declarations
        __fastcall TForm1(TComponent* Owner);
};
//---------------------------------------------------------------------------
extern PACKAGE TForm1 *Form1;
//---------------------------------------------------------------------------
#endif
```

dfm-файл

```
object Form1: TForm1
  Left = 210
  Top = 105
  Width = 540
  Height = 308
  Caption = 'Form1'
  Color = clWhite
  Font.Charset = DEFAULT_CHARSET
  Font.Color = clWindowText
  Font.Height = -11
  Font.Name = 'MS Sans Serif'
  Font.Style = []
  OldCreateOrder = False
  PixelsPerInch = 96
  TextHeight = 13
  object PageControl1: TPageControl
    Left = 8
    Top = 48
    Width = 513
```

```
Height = 217
ActivePage = TabSheet2
Style = tsButtons
TabIndex = 1
TabOrder = 0
object TabSheet1: TTabSheet
  Caption = 'A'
  OnShow = TabSheet1Show
  object Label1: TLabel
    Left = 0
    Top = 8
    Width = 131
    Height = 13
    Caption = 'Номер текущего узла'
    FocusControl = DBEdit1
    Font.Charset = DEFAULT_CHARSET
    Font.Color = clWindowText
    Font.Height = -11
    Font.Name = 'MS Sans Serif'
    Font.Style = [fsBold]
    ParentFont = False
  end
  object Label2: TLabel
    Left = 0
    Top = 48
    Width = 139
    Height = 13
    Caption = 'Номер входящего узла'
    FocusControl = DBEdit2
    Font.Charset = DEFAULT_CHARSET
    Font.Color = clWindowText
    Font.Height = -11
    Font.Name = 'MS Sans Serif'
    Font.Style = [fsBold]
    ParentFont = False
  end
```

```
object Label3: TLabel
  Left = 0
  Top = 88
  Width = 185
  Height = 13
  Caption = 'Длина пути от входящего узла'
  FocusControl = DBEdit3
  Font.Charset = DEFAULT_CHARSET
  Font.Color = clWindowText
  Font.Height = -11
  Font.Name = 'MS Sans Serif'
  Font.Style = [fsBold]
  ParentFont = False
end
object DBEdit1: TDBEdit
  Left = 0
  Top = 24
  Width = 49
  Height = 24
  Color = clWhite
  DataField = 'NODNUMBER'
  DataSource = DataSource1
  Font.Charset = DEFAULT_CHARSET
  Font.Color = clRed
  Font.Height = -13
  Font.Name = 'MS Sans Serif'
  Font.Style = [fsBold]
  ParentFont = False
  TabOrder = 0
  OnEnter = DBEdit1Enter
  OnKeyDown = DBEdit1KeyDown
end
object DBEdit2: TDBEdit
  Left = 0
  Top = 64
  Width = 49
```

```
    Height = 24
    DataField = 'NODNUMIN'
    DataSource = DataSource1
    Font.Charset = DEFAULT_CHARSET
    Font.Color = clRed
    Font.Height = -13
    Font.Name = 'MS Sans Serif'
    Font.Style = [fsBold]
    ParentFont = False
    TabOrder = 1
    OnKeyDown = DBEdit2KeyDown
  end
  object DBEdit3: TDBEdit
    Left = 0
    Top = 104
    Width = 73
    Height = 24
    DataField = 'LENGTHIN'
    DataSource = DataSource1
    Font.Charset = DEFAULT_CHARSET
    Font.Color = clRed
    Font.Height = -13
    Font.Name = 'MS Sans Serif'
    Font.Style = [fsBold]
    ParentFont = False
    TabOrder = 2
    OnKeyDown = DBEdit3KeyDown
  end
  object DBNavigator1: TDBNavigator
    Left = 193
    Top = 0
    Width = 240
    Height = 25
    DataSource = DataSource1
    TabOrder = 3
```

```
    OnClick = DBNavigator1Click
  end
  object Button1: TButton
    Left = -3
    Top = 136
    Width = 193
    Height = 25
    Caption = 'Обновить таблицу '
    Font.Charset = DEFAULT_CHARSET
    Font.Color = clBlack
    Font.Height = -19
    Font.Name = 'MS Sans Serif'
    Font.Style = [fsBold]
    ParentFont = False
    TabOrder = 4
    OnClick = Button1Click
  end
  object Button2: TButton
    Left = 117
    Top = 23
    Width = 393
    Height = 25
    Caption = 'Работа с табличным представлением'
    Font.Charset = DEFAULT_CHARSET
    Font.Color = clWindowText
    Font.Height = -19
    Font.Name = 'MS Sans Serif'
    Font.Style = [fsBold]
    ParentFont = False
    TabOrder = 5
    OnClick = Button2Click
  end
  object DBGrid1: TDBGrid
    Left = 195
    Top = 50
    Width = 313
```

```
    Height = 127
    DataSource = DataSource1
    TabOrder = 6
    TitleFont.Charset = DEFAULT_CHARSET
    TitleFont.Color = clWindowText
    TitleFont.Height = -11
    TitleFont.Name = 'MS Sans Serif'
    TitleFont.Style = []
    Visible = False
    Columns = <
      item
        Expanded = False
        FieldName = 'NODNUMBER'
        Title.Caption = 'Номер узла'
        Width = 72
        Visible = True
      end
      item
        Expanded = False
        FieldName = 'NODNUMIN'
        Title.Caption = 'Номер вход. узла'
        Width = 76
        Visible = True
      end
      item
        Expanded = False
        FieldName = 'LENGTHIN'
        Title.Caption = 'Длина пути от вх. узла'
        Visible = True
      end>
  end
end
object TabSheet2: TTabSheet
  Caption = 'Расчет пути'
  ImageIndex = 1
  OnShow = TabSheet2Show
```

```
object LabeledEdit1: TLabeledEdit
  Left = 16
  Top = 16
  Width = 57
  Height = 24
  EditLabel.Width = 199
  EditLabel.Height = 13
  EditLabel.Caption = 'Введите номер начального узла пути'
  EditLabel.Color = clWhite
  EditLabel.Font.Charset = DEFAULT_CHARSET
  EditLabel.Font.Color = clBlack
  EditLabel.Font.Height = -11
  EditLabel.Font.Name = 'MS Sans Serif'
  EditLabel.Font.Style = [fsBold]
  EditLabel.ParentColor = False
  EditLabel.ParentFont = False
  Font.Charset = DEFAULT_CHARSET
  Font.Color = clRed
  Font.Height = -13
  Font.Name = 'MS Sans Serif'
  Font.Style = [fsBold]
  LabelPosition = lpAbove
  LabelSpacing = 3
  ParentFont = False
  TabOrder = 0
  OnEnter = LabeledEdit1Enter
  OnKeyDown = LabeledEdit1KeyDown
end
object LabeledEdit2: TLabeledEdit
  Left = 16
  Top = 56
  Width = 57
  Height = 24
  EditLabel.Width = 196
  EditLabel.Height = 13
  EditLabel.Caption = 'Введите номер конечного узла пути'
```

```
    EditLabel.Font.Charset = DEFAULT_CHARSET
    EditLabel.Font.Color = clWindowText
    EditLabel.Font.Height = -11
    EditLabel.Font.Name = 'MS Sans Serif'
    EditLabel.Font.Style = [fsBold]
    EditLabel.ParentFont = False
    Font.Charset = DEFAULT_CHARSET
    Font.Color = clRed
    Font.Height = -13
    Font.Name = 'MS Sans Serif'
    Font.Style = [fsBold]
    LabelPosition = lpAbove
    LabelSpacing = 3
    ParentFont = False
    TabOrder = 1
    OnKeyDown = LabeledEdit2KeyDown
  end
  object LabeledEdit3: TLabeledEdit
    Left = 0
    Top = 109
    Width = 457
    Height = 24
    EditLabel.Width = 454
    EditLabel.Height = 24
    EditLabel.Caption = 'Кратчайший путь между введенными узлами: '
    EditLabel.Font.Charset = DEFAULT_CHARSET
    EditLabel.Font.Color = clRed
    EditLabel.Font.Height = -19
    EditLabel.Font.Name = 'MS Sans Serif'
    EditLabel.Font.Style = [fsBold]
    EditLabel.ParentFont = False
    Font.Charset = DEFAULT_CHARSET
    Font.Color = clBlue
    Font.Height = -13
    Font.Name = 'MS Sans Serif'
    Font.Style = [fsBold]
```

```
      LabelPosition = lpAbove
      LabelSpacing = 3
      ParentFont = False
      TabOrder = 2
    end
    object LabeledEdit4: TLabeledEdit
      Left = 0
      Top = 157
      Width = 121
      Height = 24
      EditLabel.Width = 226
      EditLabel.Height = 24
      EditLabel.Caption = 'Суммарная длина пути'
      EditLabel.Font.Charset = DEFAULT_CHARSET
      EditLabel.Font.Color = clBlue
      EditLabel.Font.Height = -19
      EditLabel.Font.Name = 'MS Sans Serif'
      EditLabel.Font.Style = [fsBold]
      EditLabel.ParentFont = False
      Font.Charset = DEFAULT_CHARSET
      Font.Color = clRed
      Font.Height = -13
      Font.Name = 'MS Sans Serif'
      Font.Style = [fsBold]
      LabelPosition = lpAbove
      LabelSpacing = 3
      ParentFont = False
      TabOrder = 3
    end
  end
end
object Button3: TButton
  Left = 16
  Top = 8
  Width = 177
```

```
  Height = 25
  Caption = 'Выход из задачи'
  Font.Charset = DEFAULT_CHARSET
  Font.Color = clWindowText
  Font.Height = -19
  Font.Name = 'MS Sans Serif'
  Font.Style = [fsBold]
  ParentFont = False
  TabOrder = 1
  OnClick = Button3Click
end
object IBDatabase1: TIBDatabase
  Connected = True
  DatabaseName = 'D:\InterBase\DataBases\MINROAD.GDB'
  Params.Strings = (
    'user_name=sysdba')
  DefaultTransaction = IBTransaction1
  IdleTimer = 0
  SQLDialect = 3
  TraceFlags = []
  Left = 216
  Top = 8
end
object IBTransaction1: TIBTransaction
  Active = True
  AutoStopAction = saNone
  Left = 248
  Top = 8
end
object IBClientDataSet1: TIBClientDataSet
  CommandText = 'select * from INOUTNODS'
  Aggregates = <>
  Options = [poAllowCommandText]
  Params = <>
  DBConnection = IBDatabase1
```

```
    DBTransaction = IBTransaction1
    Left = 216
    Top = 40
    object IBClientDataSet1NODNUMBER: TIntegerField
      FieldName = 'NODNUMBER'
      Origin = 'INOUTNODS.NODNUMBER'
      Required = True
    end
    object IBClientDataSet1NODNUMIN: TIntegerField
      FieldName = 'NODNUMIN'
      Origin = 'INOUTNODS.NODNUMIN'
      Required = True
    end
    object IBClientDataSet1LENGTHIN: TFloatField
      FieldName = 'LENGTHIN'
      Origin = 'INOUTNODS.LENGTHIN'
      Required = True
    end
  end
  object DataSource1: TDataSource
    DataSet = IBClientDataSet1
    Left = 248
    Top = 40
  end
  object DataSource2: TDataSource
    DataSet = IBClientDataSet2
    Left = 328
    Top = 8
  end
  object IBClientDataSet2: TIBClientDataSet
    CommandText = 'select distinct * from NEWTABLE order by iz'
    Aggregates = <>
    Options = [poAllowCommandText]
    Params = <>
    DBConnection = IBDatabase1
```

```
  DBTransaction = IBTransaction1
  Left = 296
  Top = 8
end
object IBTable1: TIBTable
  Database = IBDatabase1
  Transaction = IBTransaction2
  BufferChunks = 1000
  CachedUpdates = True
  FieldDefs = <
    item
      Name = 'NN'
      DataType = ftInteger
    end
    item
      Name = 'LEN'
      DataType = ftFloat
    end
    item
      Name = 'SIGN'
      DataType = ftString
      Size = 2
    end
    item
      Name = 'IZ'
      DataType = ftInteger
    end>
  IndexDefs = <
    item
      Name = 'INDNEWTABLE'
      Fields = 'IZ'
    end>
  StoreDefs = True
  TableName = 'NEWTABLE'
  Left = 296
```

```
    Top = 40
  end
  object IBTransaction2: TIBTransaction
    Active = False
    DefaultDatabase = IBDatabase1
    AutoStopAction = saNone
    Left = 328
    Top = 40
  end
  object DataSource3: TDataSource
    DataSet = IBTable1
    Left = 360
    Top = 40
  end
end
```

Приложение

Описание данных на компакт-диске

На компакт-диске к книге размещены шесть архивов данных:

- ❒ база данных по задаче управления кадрами, рассмотренной в *главе 2*;
- ❒ тексты программ в среде Borland C++Builder 6 по задаче управления кадрами, рассмотренной *в главе 2*;
- ❒ база данных по задаче учета движения материалов на складах, рассмотренной в *главе 3*;
- ❒ тексты программ в среде Borland C++Builder 6 по задаче учета движения материалов на складах, рассмотренной в *главе 3*;
- ❒ база данных по задаче нахождения оптимального пути между двумя пунктами, рассмотренной в *главе 4*;
- ❒ тексты программ в среде Borland C++Builder 6 по задаче нахождения оптимального пути между двумя пунктами, рассмотренной в *главе 4*.

Базы данных по задачам должны располагаться в каталоге InterBase\DataBases. Программы-приложения должны располагаться в каталоге InterBase\Applications. Чтобы воспользоваться прилагаемым к книге программным обеспечением, следует выполнить следующие действия:

1. Установить на своем компьютере продукт Borland C++Builder 6.
2. Распаковать данные тексты программ по задаче управления кадрами в папку InterBase\Applications\Учет кадров-карточка.
3. Распаковать данные тексты программ по задаче учета движения материалов на складах в папку InterBase\Applications \Склад.
4. Распаковать данные по задаче нахождения оптимального пути между двумя пунктами в папку InterBase\Applications \Расчет кратчайшего пути.

Пример расположения программ показан на рис. П1.

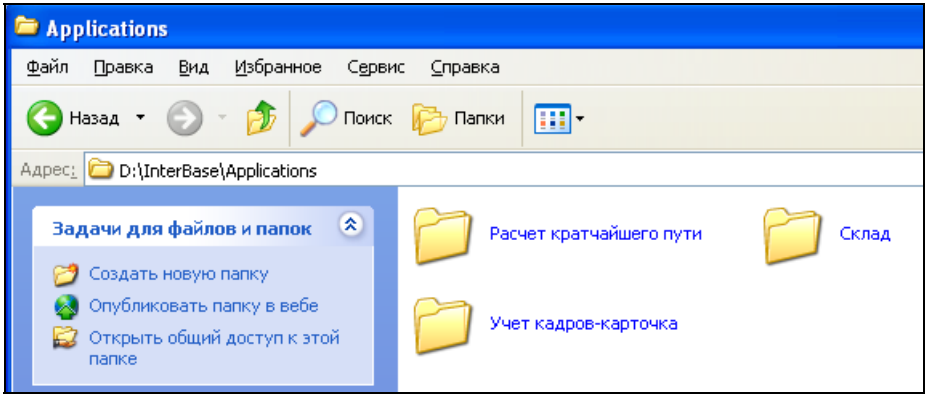


Рис. П1. Программы, расположенные в папке D:\InterBase\Applications

Пример расположения баз данных показан на рис. П2.

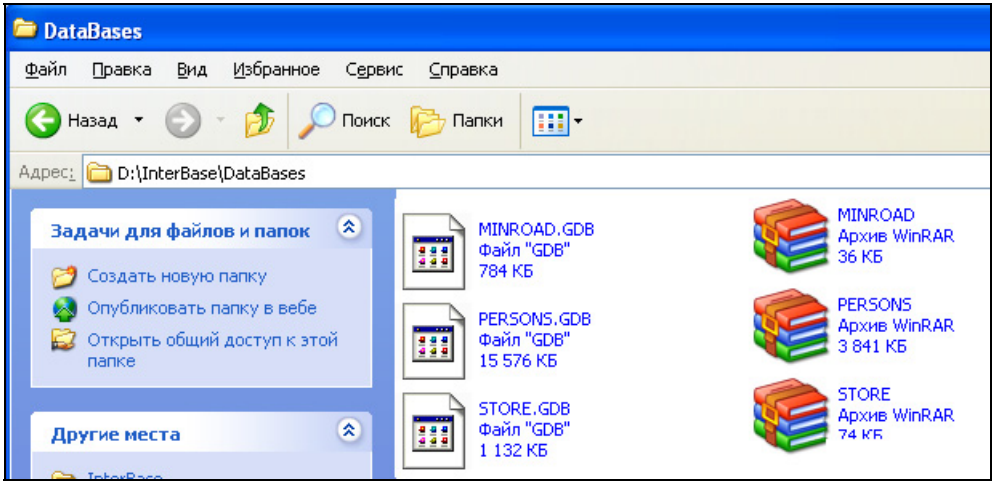


Рис. П2. Пример расположения баз данных

5. Запустить утилиту IBConsole командой **Пуск | Программы | Interbase | IBConsole**, щелкнуть мышью на строке **Local Server** в окне IBConsole, выполнить команду **Server/Login** и в появившемся окне **Server Login** в поле **password** набрать пароль **masterkey**. Откроется перечень баз данных, которые надо подключить к работе согласно рис. П3.

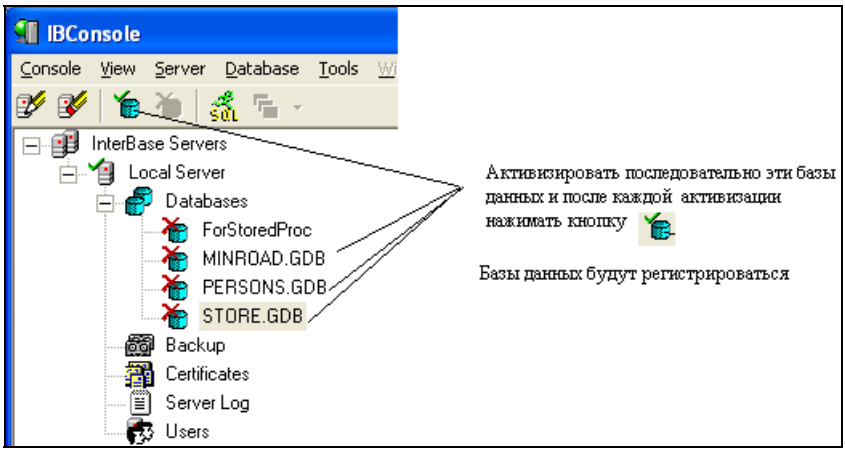


Рис. П3. Перечень баз данных, предлагаемых к регистрации

После выполнения указанных действий подключенные базы данных станут выглядеть так, как показано на рис. П4.

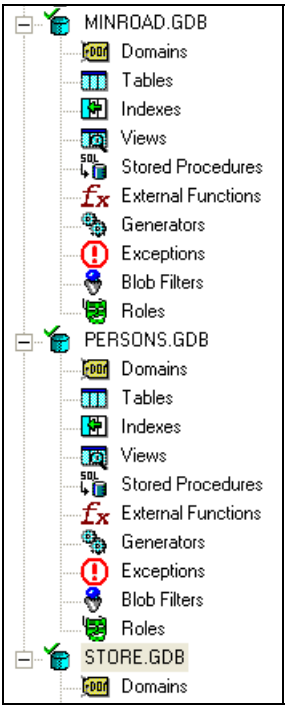


Рис. П4. Зарегистрированные базы данных

Для любой из трех задач для ее чтения в среде Borland C++Builder 6 надо открыть файл Project1.bpr в соответствующем каталоге. Перед появлением текста программы (задачи) появится окно для запроса пароля на доступ к соответствующей базе данных. Во всех случаях в качестве пароля следует набрать значение **masterkey**: с таким паролем создавались все базы данных. Если пароль набран верно, на экране появится текст программы-приложения. Далее с ним можно поступать в соответствии с желанием пользователя: корректировать, компилировать и запускать. В предложенных пользователю вариантах баз данных приведены контрольные примеры, которые пользователь вправе изменить и дополнить по своему усмотрению.

Пользователь вправе изменять каталоги размещения программ-приложений по своему усмотрению, однако соблюдая при этом меры предосторожности, так как при неаккуратном изменении каталогов cpp-модули могут перестать подключаться к основному проекту Project1.cpp.

Предметный указатель