Михаил Перов

ОРГАНИЗАЦИЯ РАБОТЫ С ДРЕВОВИДНЫМИ СПИСКАМИ
На примере тарификационных деревьев для средств связи

Эта статья посвящена вопросам работы с тарификационным деревом шаблонов для телефонных номеров. Автор попытался передать здесь свой собственный опыт и ни в коей мере не претендует на классическое изложение приемов обработки древовидной информации. Несмотря на прикладной характер проблемы, описываемый в статье круг вопросов может быть полезен для решения многих задач под любые программные платформы. В статье приводятся фрагменты программного кода, проверенного в среде Delphi 2006 for Microsoft .NET Framework (.NET 1.1.), которая, наверно, так и останется экзотической, но приводимая информация может быть также полезна для создания приложений в Delphi for win32 и в MS Visual Studio for .NET.

Дерево тарифных зон (по-другому тарифное или тарификационное дерево)—двоичный файл, записываемый в ограниченную область памяти какого-либо портативного компактного устройства тарифицирования, в частности платного телефонного аппарата—таксофона. Для краткости будем называть его просто дерево. Общепринятое назначение дерева—определить номер тарифной зоны для междугородного/международного телефонного номера, набранного абонентом. Необходимость использования тарифного дерева обусловлена тем, что некоторые конкурентоспособные на российском рынке программируемые средства тарифицирования (пример—междугородные карточные таксофоны) не всегда поддерживают большие объемы памяти для хранения данных и тем более редко находятся в постоянном подключении к системе управления верхнего уровня. Поэтому дерево стало неким компромиссным уровнем между программно-аппаратными средствами и обслуживающим их персоналом.

Платная тарифная зона определяется программой тарифицирующего устройства путем последовательного анализа цифр, входящих в набираемый телефонный номер. При этом происходит перемещение по так называемым *узлам* тарифного дерева, цифра за цифрой, по ветви, соответствующей анализируемому номеру.

На рис. 1 показан фрагмент тарифного дерева, на котором хорошо видно, как при возможном наборе абонентом телефонного номера 8–10–1–24 и т.д. тарифицирующее устройство автоматически попадает на шаблон 24(4-5), где к набору на этом уровне разрешен диапазон кодов от 244 до 245. Уже за этим конкретным шаблоном закреплено определенное тарифное правило, и аппарат точно будет знать, сколько денег должен абонент за минуту такого телефонного звонка. Подобные «конечные» шаблоны будем называть *формулами*. То есть формула—это узел дерева с однозначно определенными тарифными правилами. Формула как узел не имеет дочерних веток, а узлы, которые могут иметь в качестве «детей» формулы или другие узлы (ветки), будем называть директориями и графически обозначать пиктограммами в виде папок. В отличие от фиксированной стоимости тарифных зон, не существует каких-то жестких правил, которые бы регламентировали необходимость отнести определенный фрагмент шаблона набора номера к директории или к формуле. Другими словами, окончательное решение по построению дерева может оставаться за оператором, если, конечно в системе не предусмотрено его автоматическое построение. Единственное явное ограничение -– это необходимость обеспечения выхода на конечную формулу, чтобы соотнести ее с конкретной тарифной зоной.

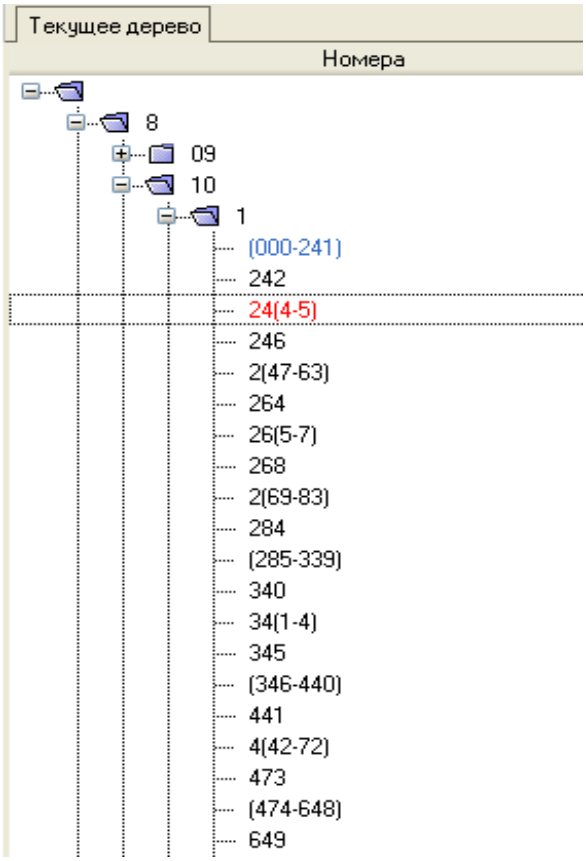

Рис. 1. Графическое отображение фрагмента тарифного дерева

Тарификационные деревья в зависимости от региона их использования (даже в одной и той же области) могут принципиально различаться. Например, имея идентичную структуру, крона одного из них может начинаться с дополнительного префикса.

Таким образом, у нас есть три сущности: *тарифное дерево*, *тарифная директория* и *тарифная формула*. Существует также достаточно условное понятие—ветка тарифного дерева (или просто ветка). Это фрагмент тарифного дерева, начинающийся с одной из его директорий и в частном случае (при начале с корневой директории) являющийся самим деревом. Ветка может быть использована для операций экспорта-импорта при создании новых деревьев или редактировании одинаковых фрагментов разных по структуре деревьев. Поэтому в целях применения универсальных алгоритмов обработки при создании методов необходимо ориентироваться именно на ветки. А теперь посмотрим на упрощенные вопросы, которые приходится решать при реализации программного комплекса, ориентированного на работу с тарифными деревьями:

- Визуализация: форма ввода, визуальное представление тарифного дерева и т.д.
- Обработка: преобразование форматов представления дерева, чтение, запись, экспорт, импорт.
- Хранение: долговременное хранение тарифных деревьев с учетом оптимизации задач по чтению-записи деревьев (или их фрагментов—веток).

Объектная модель

Очевидно, что задача делится как минимум на две части—создание клиентского и серверного приложений. От идеи создания какого-либо сервера приложений можно отказаться, переложив основную нагрузку на клиентскую программу—редактор тарифных деревьев, выполняющий функции ввода новых данных, редактирования имеющихся, экспорта-импорта и работы с базой. И тут есть соблазн сразу броситься в проектирование визуальной части с использованием компонентов TTreeView, TDBTreeView или TTreeList. Да, это вполне возможно. Но у нас как раз тот случай, когда проектирование объектной модели способно сильно упростить жизнь, и уже после можно будет с полезным багажом окунуться в мир указателей и приведения типов. Будем считать, что мы уже достаточно знаем о предметной области тарифных деревьев, и попробуем создать наброски самой объектной модели. Очевидно, что у директорий и формул существуют общие или сходные по назначению свойства, которые не ограничиваются свойствами наподобие Text или Caption, поэтому создадим для них общий родительский класс (см. листинг 1).

Реализации описываемых методов достаточно тривиальны и приводятся в полном тексте исходных кодов тестового проекта. Основными тут являются следующие свойства класса:

Parent—идентификатор родительского узла;

ID—собственный идентификатор узла, уникальный для данного дерева;

Comment—строковый комментарий, произвольно задаваемый оператором;

Template—то, ради чего все это делается,—шаблон набора номера. Для директорий используется одна или несколько цифр, для формул—особый формат записи, указывающий на возможные диапазоны наборов.

С точки зрения клиентской программы, имеющей готовые механизмы визуализации деревьев, в описываемых классах будут (или могут) присутствовать явные «излишества»: уникальные идентификаторы, идентификаторы родительских директорий, строки со списком дочерних формул и директорий, а также самодостаточные коллекции для дочерних объектов.

Такая избыточность позволит достичь необходимой гибкости в представлении различных форм дерева или на этапах развития самого проекта, а также высокой скорости обработки данных при различных преобразованиях. Ведь по опыту известно, что предусмотреть все требования к проекту просто невозможно. На этом шаге важно как можно меньше привязывать функциональность к возможностям, предоставляемым визуальным компонентом, каким бы хорошим он ни был, ибо в противном случае по прошествии некоторого времени можно стать заложником собственной «гениальности».

Листинг 1. Общий родительский класс для директорий и формул тарифного дерева

```
///<summary> Базовый класс для описания объектов тарифного дерева
/// </summary>

TXObject = class

 private

  _ID:Integer;
  _Comment:String;
  _Template:String;
  _Path:String;
  _PARENT:Integer;
  _CreateDate:TDateTime;
  _Changed:Boolean;

 protected

   _ISBUILD:Boolean;

    procedure SetCreateDate(const Value: TDateTime);
    procedure SetChanged(const Value: Boolean);
    procedure SetComment(const Value: String);
    procedure SetID(const Value: Integer);
    procedure SetPARENT(const Value: Integer);
    procedure SetPath(const Value: String);
    procedure SetTemplate(const Value: String);

    property CreateDate:TDateTime read _CreateDate write SetCreateDate;

 public

    function TestChanged:Boolean;
    procedure StopBuild;
    procedure StartBuild;
    procedure SetData(const NewComment,NewTemplate:String);

  property Parent:Integer read _PARENT write SetPARENT;
  property ID:Integer read _ID write SetID;
  property Comment:String read _Comment write SetComment;
  property Template:String read _Template write SetTemplate;
  property Path:String read _Path write SetPath;
  property Changed:Boolean read _Changed;

 end;
```

Теперь рассмотрим наследника нашего основного класса—класс TXDirectory (см. листинг 2).

Основные свойства класса:

CallType—идентификатор типа звонка, который будет присвоен при записи в базу информации об абонентском соединении для всех формул, принадлежащих данной директории;

EnableStat—флаг, указывающий на необходимость включения части набранного номера, относящейся к данной директории, в отчет, т.е. в данные по транзакции. К примеру, если абонент наберет номер 8-495-236-35-… и т.д. и при этом на директории «495» данный флаг будет выставлен в «0», то набранный номер запишется в базу данных уже без «8-495» и будет начинаться с «236..». Хитрость эта была придумана, конечно, не от хорошей жизни, а ради спасения свободного пространства в памяти тарифицирующего устройства;

Pause—технологический параметр;

BranchName—очередное полезное «излишество», связанное с предположением, что каждая директория в перспективе может стать самостоятельной веткой (или даже целым деревом), у которой должно быть свое имя, но в общем случае это поле пустует;

MAX_ID—статическое свойство класса для получения текущего максимального значения идентификатора директории, в дальнейшем будет полезно при записи дерева в реляционную БД.

Листинг 2. Описание класса для директории тарифного дерева

```
[XTreeINFO('20.10.2007 00:00:00', 'Директория тарифного дерева')]

 TXDirectory = class sealed(TXObject)

 strict private

 class var MAX_ID_NODE:Integer;

 private

  _Pause:Double;
  _EnableStat:Integer;
  _CallType:Integer;
  _BranchName:String;

 public

    SubDirs:TStringList;
    Formulas:TStringList;
    DIR_items:ArrayList;
    FRM_items:ArrayList;

    procedure SetBranchName(const Value: String);
    procedure SetCallType(const Value: Integer);
    procedure SetEnableStat(const Value: Integer);
    procedure SetPause(const Value: Double);

  class function TestMAX_ID_NODE(value:Integer):Integer; static;
  class function NextGen_ID_NODE:Integer;static;

  constructor Create(tf:TIniFile;SectionID:Integer;DS:TSQLQuery);

  function AsXML:String;

  property Pause:Double read _Pause write SetPause;
  property EnableStat:Integer read _EnableStat write SetEnableStat;
  property CallType:Integer read _CallType write SetCallType;
  property BranchName:String read _BranchName write SetBranchName;

  class property MAX_ID:Integer read MAX_ID_NODE;

 end;
```

Основные методы и функции класса:

class function NextGen_ID_NODE:Integer;static—генерация следующего максимального значения идентификатора для директорий;

constructor Create(tf:TIniFile;SectionID:Integer;DS:TSQLQuery)—конструктор класса директории, обеспечивающий важную стратегическую задачу—создание директории для нескольких возможных случаев: 1) чтение данных о директории из текстового INI-файла; 2) чтение директории из записи выборки базы данных; 3) создание директории при отсутствии начальных данных (к примеру, при ее добавлении в редакторе тарифного дерева). Свойство SectionID при наличии INI-файла задает номер секции в файле с описанием данных о директории.

Описание класса для формулы имеет похожий вид (см. листинг 3).

Основные свойства класса:

MAX_ID—статическое свойство класса для получения текущего максимального значения идентификатора формулы;

DialLength, Tariff1, Tariff2, TRCount, TR—технологические параметры.

Листинг 3. Описание класса для формулы тарифного дерева

```
[XTreeINFO('20.10.2007 00:00:00', 'Формула тарифного дерева')]

 TXFormula = class sealed(TXObject)

 strict private

 class var MAX_ID_NODE:Integer;

 private

  _DialLength:Integer;
  _Tariff1:Integer;
  _Tariff2:Integer;
  _TRCount:Integer;
  _TR:String;

 protected

    procedure SetDialLength(const Value: Integer);
    procedure SetTariff1(const Value: Integer);
    procedure SetTariff2(const Value: Integer);
    procedure SetTR(const Value: String);
    procedure SetTRCount(const Value: Integer);

 public

  class function TestMAX_ID_NODE(value:Integer):Integer; static;
  class function NextGen_ID_NODE:Integer;static;

  constructor Create(tf:TINIFile;SectionID:Integer;DS:TDataSet);
  function AsXML:String;

  property DialLength:Integer read _DialLength write SetDialLength;

  property Tariff1:Integer read _Tariff1 write SetTariff1;
  property Tariff2:Integer read _Tariff2 write SetTariff2;
  property TRCount:Integer read _TRCount write SetTRCount;
  property TR:String read _TR write SetTR;

  class property MAX_ID:Integer read MAX_ID_NODE;

 end;
```

Свойство **Tariff1** используем для одной из основных задач—привязки к шаблону конкретного номера тарифной зоны, что позволит определять стоимость звонков абонентов в зависимости от набранного номера и длительности разговора, так как таблица тарифных зон—это обычный набор для перевода секунд (или минут) в рубли. Отдельной задачей будет сортировка всех формул имеющегося дерева по номерам привязанных к ним тарифных зон с группировкой по директориям. Что поделать, если именно в таком линейном виде дерево может быть пригодно для визуального контроля на соответствие требованиям по тарификации.

Основные методы и функции класса:

class function NextGen_ID_NODE:Integer;static—генерация следующего максимального значения идентификатора для директорий;

constructor Create(tf:TIniFile;SectionID:Integer;DS:TSQLQuery)—конструктор для получения новой формулы, аналогичный конструктору для директории.

Оба класса имеют функции AsXML исключительно для демонстрации простоты формирования узлов XML, а в итоге и представления всего дерева в формате XML.

Визуальная часть

Визуализация дерева—это, возможно, самый сложный и захватывающий раздел нашей «древесной» темы. Дело в том, что выбор нужного компонента может сильно повлиять на остальные части кода. Часто именно так и случается. И тогда уже не помогает никакая программистская «диалектика» о первичности описываемых сущностей в соответствии с предъявляемыми к ним требованиями. Автору так и не удалось найти бесплатный компонент типа TTreeList для VCL.NET (BDS 4.0). А ведь в бурном 21-м веке давно уже мало отображать на экране список объектов, требуется обязательно там же и тут же видеть их свойства. Сегодня в Интернете без особых усилий можно подобрать понравившийся компонент схожего типа и для MS Visual Studio, и для Delphi for win32, более того, многие из них совсем бесплатны. В последнем случае можно порекомендовать уже достаточно стабильный пакет компонентов LMD ElPack SE 4.05 (win32) для работы с древовидным списком. Когда к давно известному компоненту успела прилипнуть знаменитая марка LMD, наверное, не так важно, как сам факт его теперешней стабильной работы. Но, возвращаясь к нашим условиям, приходится заключить, что у разработчика будут два самых ближайших пути—это строить дерево на базе старого доброго TTreeView (см. рис. 2) или просто выбросить Delphi for .NET! (VCL). Справедливости ради стоит вспомнить про Delphi for Windows Forms Application (где существует возможность работы со всеми имеющимися в системе для WinForms визуальными объектами напрямую, без VCL), но на данный момент есть большие сомнения в перспективности этой среды разработки...

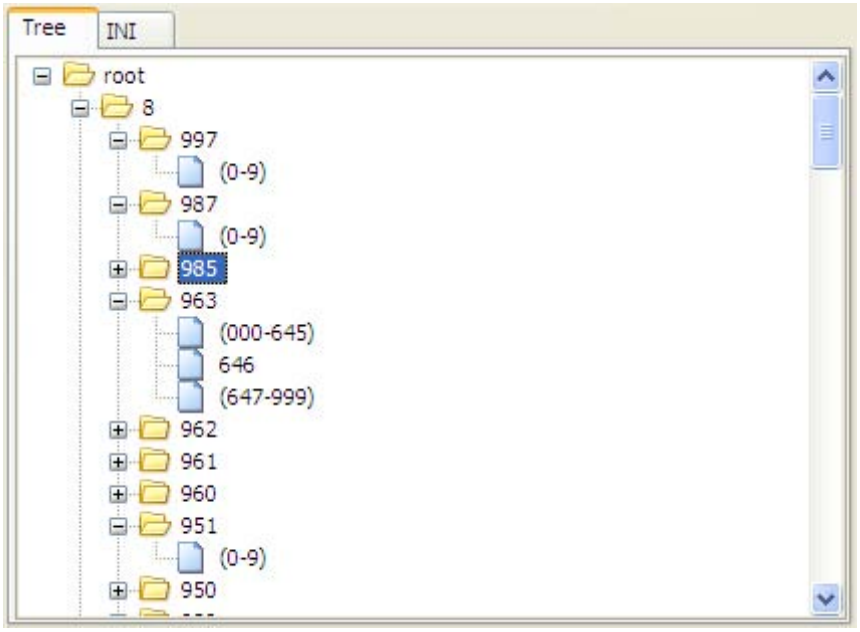

Рис. 2. Тарифное дерево, построенное с помощью VCL.NET

Хранение древовидных данных в базе

Пожалуй, один из самых развернутых русскоязычных ресурсов, посвященных этому вопросу, можно найти на сайте www.ibase.ru (http://www.ibase.ru/develop.htm#prog). Первоисточником для некоторых из имеющихся там статей является серия статей от Джо Селко (http://www.celko.com/). Пропуская все упомянутые там варианты сохранения деревьев, мы рассмотрим подробнее, пожалуй, наиболее распространенный и простой вариант, заметив прежде, что для хранения дерева в реляционной базе совсем не обязательно раскладывать по полочкам его узлы. Но сначала рассмотрим структуру организации базы (см. листинг 4) для хранения тарифного дерева в РСУБД FireBird(http://www.firebirdsql.org).

Тем, кто привык работать с внешними ключами (FOREIGN KEY), рекомендуется конечно же их использовать. Но особой необходимости в этом нет: всю «грязную» работу (защита от получения входных данных, результатом которых может стать нарушение отношений, и, разумеется, каскадные изменения) будет делать клиентское приложение, да и дерево у нас всего одно, и оно постоянно полностью перезаписывается. Таким образом, роль РСУБД в данном случае незначительна, что иногда может быть даже полезным. Наибольшую пользу этот метод хранения данных может принести, когда требуется получить не все дерево, а только его часть—какую-то

выбранную ветку. При этом чем больше в дереве папок и формул, тем полезнее окажется СУБД благодаря возможности индексации информации, что значительно облегчит извлечение отдельных веток дерева, редактирование дерева, находящегося в базе, и генерацию отчетов с сортировкой по параметрам папок и директорий. При работе с маленьким деревом польза от индексации по вполне понятным причинам будет не видна.

Листинг 4. Скрипт создания БД для хранения тарификационного дерева в формате СУБД FireBird 2.1, сгенерирован в программе InterBase/FireBird Development Studio , www.sqlly.com

```
SET TERM ^ ;

SET SQL DIALECT 3^

/* CREATE DATABASE...*/


CREATE GENERATOR GEN_FORMULA_ID^

SET GENERATOR GEN_FORMULA_ID TO 463^


COMMIT WORK^

CREATE OR ALTER PROCEDURE NODELIST (STARTID Integer,
     GETROOT Integer)
returns (ID Integer)
AS
BEGIN EXIT; END^

CREATE OR ALTER PROCEDURE SP_GEN_FORMULA_ID
returns (ID Integer)
AS
BEGIN EXIT; END^


COMMIT WORK^

SET TERM ; ^

CREATE TABLE FOLDER (
     ID Integer NOT NULL,    /*Идентификатор дерева*/
     ENABLED Integer NOT NULL,    /*Флаг активности узла*/
     TEMPLATE Varchar(64) CHARACTER SET WIN1251 NOT NULL,    /*Шаблон набора*/
     CALLTYPE Integer NOT NULL,    /*Тип звонка*/
     PAUSE Varchar(8) CHARACTER SET WIN1251 NOT NULL,    /*Технологический параметр*/
     ENABLESTAT Integer NOT NULL,    /*Технологический параметр*/
     COMMENT Varchar(255) CHARACTER SET WIN1251,    /*Комментарий к директории*/
     PARENT Integer NOT NULL,    /*Родительская директория*/
     CREATETS TIMESTAMP,
     NAME Varchar(64) CHARACTER SET WIN1251,  /*Имя тарифного дерева или ветки*/
     CONSTRAINT PK_FOLDER PRIMARY KEY (ID)
);


/* Порядковый номер формулы*/
CREATE TABLE FORMULA (
     LOGNO Integer NOT NULL,    /*Порядковый номер формулы*/
     FOLDER Integer NOT NULL,    /*Указатель на принадлежность директории*/
     TEMPLATE Varchar(64) CHARACTER SET WIN1251 NOT NULL,    /*Шаблон набора*/
     DIALLENGTH Integer NOT NULL,    /*Технологический параметр*/
     TARIFF1 Integer NOT NULL,    /*Технологический параметр*/
     TARIFF2 Integer NOT NULL,    /*Технологический параметр*/
     TRCOUNT Integer NOT NULL,    /*Технологический параметр*/
     TR Varchar(32) CHARACTER SET WIN1251 NOT NULL,    /*Технологический параметр*/
     COMMENT Varchar(255) CHARACTER SET WIN1251,    /*Комментарий*/
     ENABLED Integer NOT NULL,    /*Флаг активности*/
      CREATETS TIMESTAMP,
      CONSTRAINT PK_FORMULA PRIMARY KEY (LOGNO)
);
```

```
/* Произвольный список глобальных параметров(КЛЮЧ-ЗНАЧЕНИЕ)*/
CREATE TABLE GLOBALPARAMS (
     PARAMNAME Varchar(128) CHARACTER SET WIN1251 NOT NULL,
     PARAMVALUE Varchar(1024) CHARACTER SET WIN1251 DEFAULT '',
     CONSTRAINT INTEG_29 PRIMARY KEY (PARAMNAME)
);


CREATE INDEX FOLDER_IDX1 ON FOLDER(TEMPLATE);
CREATE INDEX FOLDER_IDX2 ON FOLDER(CALLTYPE,ENABLESTAT);
CREATE INDEX FOLDER_IDX3 ON FOLDER(PARENT,ENABLED);
CREATE INDEX FOLDER_IDX4 ON FOLDER(CREATETS);
CREATE INDEX FOLDER_IDX5 ON FOLDER(NAME);

CREATE INDEX FORMULA_IDX1 ON FORMULA(TEMPLATE);
CREATE INDEX FORMULA_IDX2 ON FORMULA(DIALLENGTH,ENABLED);


SET TERM ^ ;

COMMIT WORK^


ALTER PROCEDURE NODELIST (STARTID Integer,
     GETROOT Integer)
returns (ID Integer)
AS

begin
 if (:GETROOT<>0) then
 begin
   ID=:startid; SUSPEND;
 end

 for select ID FROM folder
 where PARENT=:startid INTO :id
  do for select ID from NODELIST(:id, 1) into :id
   DO SUSPEND;
end^

ALTER PROCEDURE SP_GEN_FORMULA_ID
returns (ID Integer)
AS

begin
  ID = GEN_ID(GEN_FORMULA_ID, 1);
  SUSPEND;
end^

CREATE OR ALTER TRIGGER FOLDER_BIU0 FOR FOLDER
ACTIVE BEFORE  INSERT OR UPDATE POSITION 0
AS

begin

    if (new.CREATETS is null) then new.CREATETS = 'now';

  /* Trigger text */

end^

CREATE OR ALTER TRIGGER FORMULA_BI FOR FORMULA
ACTIVE BEFORE INSERT POSITION 0
AS

begin
  IF (NEW.LOGNO IS NULL) THEN
    NEW.LOGNO = GEN_ID(GEN_FORMULA_ID,1);

end^
```

```
CREATE OR ALTER TRIGGER FORMULA_BIU0 FOR FORMULA
ACTIVE BEFORE  INSERT OR UPDATE POSITION 0
AS

begin


   if (new.CREATETS is null) then new.CREATETS = 'now';

end^

COMMIT WORK^
/*Descriptions*/
COMMENT ON TABLE FORMULA IS 'Порядковый номер формулы'^
COMMENT   ON   TABLE   GLOBALPARAMS   IS   'Произвольный   список   глобальных
параметров(КЛЮЧ-ЗНАЧЕНИЕ)'^
```

Теперь несколько слов о выборе самой СУБД. Не вдаваясь в сравнительные исследования и стараясь всеми силами не отражать на этих страницах элементы «религиозных войн», просто приведу ряд свойств СУБД FireBird:

- возможность хранения данных в одном файле и замены данных в формате этого же рабочего файла без промежуточных операций;
- работа сервера под Windows, Linux, Solaris, FreeBSD, Mac OS;
- полная бесплатность;
- наличие встраиваемой версии, которая позволяет работать с файлом базы без инсталляции самой СУБД;
- хорошие бесплатные оконные инструментарии для разработки и администрирования;
- очень скромные требования к системным ресурсам (в сравнении с СУБД похожего уровня);
- существование живых русскоязычных форумов по FireBird и большое количество учебных материалов на русском языке;
- неофициальная совместимость с «родными» для Delphi компонентами для работы с InterBase;
- наличие пакетов компонентов для Delphi и Lazarus, специально предназначенных для работы с FireBird;
- поддержка .NET 1.1/2.0/MONO.

Однако, как уже упоминалось, совсем не обязательно хранить дерево в «открытом» табличном формате. Целиком его можно записать в одно-единственное поле формата BLOB, при этом, как и в других вариантах, не привязывая его к архитектуре визуального компонента. Ведь конечное представление тарифного дерева—это некий бинарный файл данных, поэтому ничто не мешает, к примеру, записать в BLOB или текстовое поле все дерево одной длинной HEX-строкой, формат которой уже приближен к требованиям конечного устройства. Самый явный недостаток подобного решения—такой вариант серьезно затруднит возможность получения из базы отдельных фрагментов дерева. Преимуществом записи дерева в одно поле будет достаточно естественный способ разграничения одного дерева от другого: каждая строка таблицы содержит очередное дерево.

Хранение дерева в INI-файле

Нет ничего проще, чем записать дерево в обычный INI-файл. Но при этом будет использован немного другой принцип связи объектов, чем тот, на котором мы остановились при сохранении в базе. Дело в том, что свойство PARENT (для формул и директорий) в рассматриваемом объекте нам не поможет. Точнее, это не лучший вариант для задания связей при сохранении в INI-файле. Разумнее всего будет в каждой секции, созданной для описания определенной директории, записывать поля SubDirs и Formulas, выкладывая в список через запятую идентификаторы дочерних директорий и формул, принадлежащих данной директории (см. листинг 5).

Листинг 5. Фрагмент тарифного дерева в формате INI, где каждой секции соответствует одна директория или формула

```
[DIR0]
Comment=
Template=
DialLength=
Pause=0
EnableStat=3
CallType=0
SubDirs=1
Formulas=2,3,4,5,6,7,8,9,10

[FRM2]
Comment=
Template=0(1-5)
DialLength=2
Tariff1=0
Tariff2=0
TRCount=0
TR=0

[FRM3]
Comment=
Template=0605
DialLength=4
Tariff1=0
Tariff2=0
TRCount=0
TR=0

[FRM4]
Comment=
Template=0(61-71)
DialLength=3
Tariff1=0
Tariff2=0
TRCount=0
TR=0

[FRM5]
Comment=
Template=0850
DialLength=4
Tariff1=0
Tariff2=0
TRCount=0
TR=0

[FRM6]
Comment=
Template=0880
DialLength=4
Tariff1=0
Tariff2=0
TRCount=0
TR=0
```

Наверно, сложно найти более универсальный формат, чем этот, несмотря на то что в таком варианте каждая директория должна «отследить» перед записью в файл все свои дочерние объекты. Тем не менее INI-файл может быть успешно прочитан на любой программной платформе любыми современными программными средствами. При работе с визуальным компонентом это можно сделать во время выстраивания дерева, редактирования или в самый последний момент—при записи в INI-файл. Недостаток подобного формата хранения дерева начнет проявляться на больших размерах при построении, когда будет идти постоянное обращение к устройству хранения данных. Однако именно в таком виде удобнее всего хранить разные веточки и маленькие деревца (см. листинг 6).

Листинг 6. Функция записи ветки тарифного дерева из редактора тарифных деревьев в INI-файл

```
///<summary> Запись ветки в INI-файл
/// </summary>
function TXTree.SaveDirToIni(TN: TTreeNode):Integer;
var i:Integer;


    //Текущий объект-директория
    CurrentDir:TXDirectory;

    //Списки дочерних директорий и формул
    SelfSubDirs,SelfFormulas,
    DirSec:String;

    // Запись дочерней формулы, на входе - элемент визуального компонента,
    // на выходе - ее идентификатор
    function SaveFrmlToIni(FRML:TTreeNode):Integer;
    var FrmlSec:String;
        CurrentFormula:TXFormula;
    begin

       //Задаем имя секции для формулы
       FrmlSec:='FRM'+FrmlCnt.ToString;

       //Считываем данные для формулы
       CurrentFormula:=(FRML.Data as TXFormula);

        with FreeINIF,CurrentFormula do begin

         //Записываем данные по формуле в секцию

          WriteInteger(FrmlSec,'DialLength',DialLength);
          WriteInteger(FrmlSec,'Tariff1',Tariff1);
          WriteInteger(FrmlSec,'Tariff2',Tariff2);
          WriteInteger(FrmlSec,'TRCount',TRCount);
          WriteString(FrmlSec,'TR',TR);
          WriteString(FrmlSec,'Comment',Comment);
          WriteString(FrmlSec,'Template',Template);
          WriteString(FrmlSec,'Path',Path);

        end;

       result:=FrmlCnt;
       inc(FrmlCnt);

    end;

begin

  try

     SelfSubDirs:=''; SelfFormulas:='';

    //Считываем данные по текущей директории
    CurrentDir:=(TN.Data as TXDirectory);

    //Задаем имя секции для директории
    DirSec:='DIR'+FolderCnt.ToString;

     with FreeINIF,CurrentDir do begin
```

```
      //Записываем данные по формуле в секцию

      WriteString(DirSec,'Comment',Comment);
      WriteString(DirSec,'Template',Template);
      WriteInteger(DirSec,'EnableStat',EnableStat);
      WriteInteger(DirSec,'CallType',CallType);
      WriteFloat(DirSec,'Pause',Pause);

     end;


      result:=FolderCnt;
      inc(FolderCnt);

       if  TN.Count>0 then begin
         for i:=0 to TN.Count-1 do begin
           if TN.Item[i].ImageIndex<>2 then begin

              //Записываем дочерние директории
              if Trim(SelfsubDirs)>'' then SelfsubDirs:=SelfSubDirs+',';
              SelfsubDirs:=SelfSubDirs+SaveDirToIni(TN.Item[i]).ToString;

           end else begin

              //Записываем дочерние формулы
              if Trim(SelfFormulas)>'' then SelfFormulas:=SelfFormulas+',';
              SelfFormulas:=SelfFormulas+SaveFrmlToIni(TN.Item[i]).ToString;

           end;

         end;
       end;

   // Сохраняем в текущей секции информацию по дочерним объектам
   with FreeINIF,CurrentDir do begin
      WriteString(DirSec,'SubDirs',SelfSubDirs);
      WriteString(DirSec,'Formulas',SelfFormulas);
   end;

   except on e:Exception do begin
     PrepareError(e,'TXTree.SaveDirToIni');
   end;
  end;
end;
```

Хранение древовидных данных в объектном виде

Теперь давайте попробуем в действии всю мощь Delphi for .NET, а заодно продемонстрируем возможность сохранения дерева в объектном виде. Для этого в класс директории тарифного дерева мы добавляем **DIR_items** и **FRM_items**—массивы с динамически увеличивающимся размером. В эти массивы мы соберем все дочерние директории для текущей рассматриваемой папки и соответственно все принадлежащие ей формулы. В конце нашего «собирания» (снятия текущей информации с визуального компонента TTreeView) мы получим целое дерево.

Достоинство массива этого типа состоит в том, что он является стандартным классом для .NET и принадлежит пространству имен System.Collections, которое надо указать в разделе uses. Поэтому при использовании такого класса у нас не должно возникнуть проблем при работе с базой, имеющей обычный интерфейс для программ под .NET (т.е. без поддержки специфичной для Delphi объектной модели). В качестве такой базы выберем достаточно компактную динамично развивающуюся объектную базу данных db4o. Это СУБД с открытыми исходными текстами, а обращаться к ней можно как из .NET, так и из Java. Разработчики утверждают, что неповторимый «родной» для объектов дизайн делает эту базу просто идеальным решением для встроенных систем и различных мобильных устройств.

Листинг 7. Функция считывания «чистой» ветки объектов из визуального компонента для последующей записи в объектную базу

```
function TdmTree.GetBranch_AsOBJECT(TN: TTreeNode;
                                    const TreeName: String):TXDirectory;
var CURRENT_FOLDER,SUB_FOLDER:TXDirectory;
    i:Integer;
begin

    //Основной (рекурсивный) метод для добавления ветки

    CURRENT_FOLDER:=(TN.Data as TXDirectory);
    CURRENT_FOLDER.DIR_items:=ArrayList.Create;
    CURRENT_FOLDER.FRM_items:=ArrayList.Create;

    if TN.Parent=nil then CURRENT_FOLDER.Parent:=-1;

     CURRENT_FOLDER.BranchName:=TreeName;

          if TN.Count>0 then begin
            for i:=0 to TN.Count-1 do begin
              if TN.Item[i].ImageIndex<>2 then begin

                SUB_FOLDER:=GetBranch_AsOBJECT(TN.Item[i],'');

                CURRENT_FOLDER.DIR_items.Add(SUB_FOLDER);

              end else begin

                CURRENT_FOLDER.FRM_items.Add(TN.Item[i].Data as TXFormula);

              end;
            end;
          end;


     result:=CURRENT_FOLDER;

end;
```

Теперь, после считывания ветки дерева (или всего дерева, при указании начального узла), попробуем сохранить ее в базе данных.

Листинг 8. Метод сохранения ветки тарифного дерева в объектной базе данных

```
procedure TdmTree.AddBranch_AsDb4o_2(const DB, TreeName: String; TN: TTreeNode);
begin

   OpenAsdb4o(DB);

     db_FILE.&Set(GetBranch_AsOBJECT(TN,TreeName));

   Closedb4o;

end;
```

Тут **DB**—это имя файла базы данных, а **db_FILE**—указатель несколько необычного для «дельфиста» типа, который и указывает на эту самую объектную базу.

Листинг 9. Методы открытия и закрытия объектной базы тарифного дерева

```
function TdmTree.OpenAsdb4o(DB: String): Integer;
begin
 result:=-1;
 try
  if FileExists(DB) then DeleteFile(DB);

  db_FILE:=Db4oFactory.OpenFile(DB); result:=0;

 except on e:Exception do begin
   result:=-3;
   PrepareError(e,'TdmTree.OpenAsdb4o');
 end;
 end;
end;


procedure TdmTree.Closedb4o;
begin
  db_FILE.Close;
end;
```

Теперь рассмотрим методы, позволяющие прочитать из файла наше тарифное дерево, а точнее говоря, ветку и «прицепить» ее к визуальному компоненту (см. листинг 10).

Листинг 10. Методы построения тарифного дерева из объектной базы формата db4o (для .NET) в среде Delphi for the Microsoft.NET Framework

```
procedure TdmTree.OBJECT2TreeView(TreeObj: TXDirectory; TV: TTreeView;
  TN: TTreeNode);
var CURRENT_FOLDER:TTreeNode;
    CURRENT_FRML:TTreeNode;
    FRML:TXFormula;
    FLDR:TXDirectory;
begin
   try
     //Строим директорию

      CURRENT_FOLDER:=TV.Items.AddChildFirst(TN,TreeObj.Template);
      CURRENT_FOLDER.Data:=TreeObj as TXDirectory;

     //Итерации по дочерним поддиректориям

     for FLDR in TreeObj.DIR_items do begin
         OBJECT2TreeView(FLDR,TV,CURRENT_FOLDER);
     end;

     //Строим формулы
     for FRML in TreeObj.FRM_items do begin
         CURRENT_FRML:=TV.Items.AddChild(CURRENT_FOLDER,FRML.Template);
         CURRENT_FRML.ImageIndex:=2;
         CURRENT_FRML.Data:=FRML as TXFormula;
         Tree.AddToZONES(FRML);
     end;

   except on e:Exception do begin
      PrepareError(e,'TdmTree.OBJECT2TreeView');
   end;
   end;
  try
      if TN=nil then begin
             TV.Items[0].Text:='root';
             TV.Items[0].Expand(false);
             TV.Items[1].Expand(false);
      end;
  except
  end;
end;
```

```
procedure TdmTree.LoadObjectToTreeView(const DB, TreeName: String;
  TV: TTreeView; TN: TTreeNode);
var ROOT_FOLDER: TXDirectory;
    query:IQuery;
begin

     //Открываем базу данных
     db_FILE:=Db4oFactory.OpenFile(DB);

     //Запрашиваем из базы объект класса TXDirectory
     query:=db_FILE.Query();
     query.Constrain(typeof(TXDirectory));

     ROOT_FOLDER:=(query.Execute()).Item[0] as TXDirectory;

     //Отображаем дерево в виде объекта класса TXDirectory на визуальный компонент TV
     OBJECT2TreeView(ROOT_FOLDER,TV,TN);

     //Закрываем базу данных
     Closedb4o;
end;
```

В этом фрагменте метод OBJECT2TreeView предназначен для отображения указанной на входе ветки в виде TreeObj на визуальный компонент TV с указанным для начала построения узлом TN.

Метод LoadObjectToTreeView аккумулирует все наши прежние объектные наработки по работе с базой db4o и выстраиванию визуального компонента.

Теперь для выстраивания дерева из известного файла базы можно будет выполнить:

```
dmTree.LoadObjectToTreeView(GetCurrentDir+'\BKP.db4','root',TreeView1,nil);
```

На самом деле с базой формата db4o можно производить все общепринятые манипуляции, свободно ориентируясь в объектной модели. Подробнее об этом можно узнать на сайте разработчиков или начать со статьи на известном портале: http://www.codeproject.com/useritems/OOP_with_db4o.asp

Более того, на сайте разработчиков выложена программа загадочного менеджера объектов, но ознакомиться с ним так и не удалось по причине явно некорректной работы сайта на базе ASP.NET и не вполне ясных правил авторизации.

Список использованной литературы

1. Либерти Дж. Программирование на C#. Символ-Плюс, 2003.
2. Рихтер Дж. CLR via C#. Программирование на платформе Microsoft .NET FRAMEWORK 2.0 на языке C#. Русская редакция, 2007.
3. Дубцов А. Microsoft .NET в подлиннике. СПб.: БХВ-Петербург, 2004.
4. Фаронов В. Искусство создания компонентов на базе Delphi. Спб.: Питер, 2005.
5. Борри Х. FireBird. Руководство разработчика. Спб.:БХВ-Петербург, 2006.

Приложение. Демонстрационная программа для работы с тарификационным деревом.