Нижегородский государственный университет им.Н.И.Лобачевского
Институт информационных технологий, математики и механики
Центр информатики и интеллектуальных информационных технологий

СУБД ГИС Терра Plus
Руководство программиста

Нижний Новгород
2019

СОДЕРЖАНИЕ

1. ВВЕДЕНИЕ

В документе описан интерфейс прикладного программиста (API, application programming interface), предназначенный для разработки приложений с использованием средств СУБД ГИС Терра Plus [1].

Работа ведется с базами данных в формате интегрального файла с использованием СУБД PostgreSQL [2] и ее расширения PostGIS [2].

Данный API может быть использован для создания VCL-приложений для ОС Windows и FMX-приложений для ОС Windows и ОС Android.

Ниже приведены основные параметры баз данных, обрабатываемых СУБД ГИС Терра Plus.

Структура объекта СУБД ГИС Терра [4, 5, 6, 7] полностью сохранена.

В настоящее время поддерживается следующие типы метрического описания:

- Точка имеет две или три координаты;
- Тип координат – целый (размер координаты 4 байта);
- Тип координаты – вещественный (размер координаты 4 или 8 байт).

Все координаты одной точки имеют одинаковый тип и размер.

Каждый объект базы данных может иметь свое индивидуальное метрическое описание.

Дискретный объект имеет поле метрики – в него переносятся координаты точек привязки характеристики, являющейся точкой привязки дискретного объекта. Это сделано с целью возможности использования геометрических функций PostGIS и пространственного индекса СУБД PostgreSQL.

Изменен подход к организации хранения мультимедийной информации объекта. Теперь при считывании объекта из базы данных в структуре объекта (GBASE_OBJECT) не формируется поле мультимедиа (Video). А пользователю предоставляется законченный набор функций для работы с элементами мультимедиа объекта (они хранятся в отдельных от объекта таблицах).

Каждый объект базы данных дополнительно имеет следующие атрибуты:

- Название объекта;
- Коэффициент масштаба;
- Уникальный идентификатор типа источника поступления объекта в базу данных (например, 1 для планшетов гидрографической съемки (ПГС). Здесь и далее примеры будут приводиться для предметной области – морская картография);
- Уникальный идентификатор источника поступления объекта в базу данных (например, идентификатор морской навигационной карты (МНК) с адмиралтейским номером ‘21008’);
- Статус объект (0 – актуальный, 1 – удаленный, 2 – заблокированный). При выполнении функции удаления объекта объект физически не удаляется из базы данных;
- Дату создания (или обновления) объекта в базе данных;
- Флажки обновляемых полей объекта.

Для дискретного объекта (и только для одного в используемой предметной области) имеется возможность его представления не виде объекта в формате интегрального файла (ИФ), а виде записей реляционной таблицы. Это имеет смысл только для объектов, имеющих очень большое количество, например, отметки глубин (они могут исчисляться сотнями тысяч). Для объектов подобного типа предусмотрен особый раздел API.

В базе данных обеспечивается хранение и обработка источников поступления объектов в базу данных. Каждый источник может иметь метаданные (их состав зависит от типа источника) и любые дополнительные материалы (файлы): изображения, тексты, базы данных и т.п.

Каждый тип источника представлен своей таблицей данных. Их имена могут потребоваться программисту при реализации поиска информации с условиями отбора на источники.

Их имена для морской картографии:

- if_pgs – ПГС;
- if_s57 – файлы S57;
- if_mnk – МНК системы АСОИМК (ИФ);
- if_sxf – файлы в формате SXF;
- if_dm – файлы в формате DM;
- if_none – тип источника не известен.

Их имена для топографических карт и планов:

- if_topmap – топокарта;
- if_topplan – топоплан;
- if_cityplan – план города;
- if_kls – классификатор;
- if_none – тип источника не известен.

В базе данных имеется возможность многоверсионного хранения объекта. При обновлении объекта старая версия объекта сохраняется. В дальнейшем имеется возможность выборки версий объекта в обратном хронологическом порядке. Эта возможность устанавливается перед подключением к базе данных.

Для каждого изменяемого поля объекта запоминается идентификатор выполненной операции (обновление, добавление, удаление).

2. УСЛОВИЯ ИСПОЛЬЗОВАНИЯ

Разработка приложений должна производиться в среде системы программирования Embarcadero RAD Studio [8] версии, начиная с XE7.

При создании приложения для Windows могут использоваться следующие компоненты доступа к базам данных:

-стороннего разработчика – UniDac. В этом случае дополнительно должен быть установлен UNIDAC (Universal Data Access Components), версии не ниже 6.1;

-интегрированный в Embarcadero FireDac.

При создании приложений для Андроид может использоваться только UniDac.

Базы данных создаются и обрабатываются в среде СУБД PostgreSQL с расширением PostGIS.

Структура папок и файлов API:

Головная папка API_TERRA_PLUS содержит папки:

INCLUDE

LIB

FMX

Папка INCLUDE содержит следующик header-файлы:

ifpg_fmx.h

ifpg_serv.h

ifpg_uni.h

ifpg_vcl.h

if_def.h

if_feat.h

if_multilib.h

Папка VCL_LIB содержит статические библиотеки VCL-приложения:

lib_plus_u.lib при использовании UniDac

lib_plus.lib – при использовании FireDac

Папка FMX папки OBJ и O с объектными файлами для создания FMX-приложения (только с использованием UniDac).

Папка FMX\OBJ:

ifpg_serv.obj

ifpg_uni.obj

if_feat.obj

if_multilib.obj

if_main.obj

Папка FMX\O:

ifpg_serv.o

ifpg_uni.o

if_feat.o

if_multilib.o

if_main.o

Для работы с объектно-ориентированной топологической структурированной моделью (ООТСМ) объекта разработаны три версии СУБД ГИС Терра:

- СУБД ГИС Терра [4-7] – локальная СУБД для ОС Windows;
- СУБД ГИС Терра Plus – клиент-серверная СУБД для ОС Windows и ОС Android [1];
- СУБД ГИС Терра Mobile – локальная СУБД для ОС Android.

Все эти СУБД имеют один общий API (application programming interface) – работа с объектами в оперативной памяти, а также каждый свой API для соответствующей платформы доступа к данным:

- Для СУБД ГИС Терра – это оригинальный древовидный метод доступа;
- Для СУБД ГИС Терра Plus – это СУБД PostgreSQL с расширением PostGIS;
- Для СУБД ГИС Терра Mobile – это локальная встраиваемая СУБД SQlite.

Кроме того этот общий API используется для работы с базой данных, выгруженной в обменных формат СУБД ГИС Терра.

2.1. Создание VCL-приложения

2.1.1. Для UniDac

1) Создать проект VCL-приложения.
2)В опциях проекта для директорий установить путь к папке API_TERRA_PLUS/INCLUDE.
3) Переписать в папку проекта из папки API_TERRA_PLUS/INCLUDE файл IFPG_VCL.h и переименовать его в IFPG.H.
4) Включить в H-файл главного окна приложения:

```
#include "ifpg.h"
#include "if_def.h"
#include "ifpg_uni.h"
#include "if_feat.h"   // Если потребуется работа с характеристиками объекта
#include "if_multilib.h"  // Если потребуется работа с библиотекой запросов
#include <Data.DB.hpp>
#include "UniDacVcl.hpp"
```

5) Включить в проект библиотеку LIB_PLUS_U.LIB:
***) В h-файле ifpg_uni.h должен присутствовать оператор #define __IFPG_UNIDAC__

2.1.2. Для FireDac

1) Создать проект VCL-приложения.
2)В опциях проекта для директорий установить путь к папке API_TERRA_PLUS/INCLUDE.
3) Переписать в папку проекта из папки API_TERRA_PLUS/INCLUDE файл IFPG_VCL.h и переименовать его в IFPG.H.
4) Включить в H-файл главного окна приложения:

```
#include "ifpg.h"
#include "if_def.h"
#include "ifpg_uni.h"
#include "if_feat.h"   // Если потребуется работа с характеристиками объекта
#include "if_multilib.h"  // Если потребуется работа с библиотекой запросов
#include <FireDAC.Phys.PG.hpp>
#include <FireDAC.VCLUI.Login.hpp>
```

5) Включить в проект библиотеку LIB_PLUS_U.LIB:
***) В h-файле ifpg_uni.h должен отсутствовать оператор #define __IFPG_UNIDAC__

2.2. Создание FMX-приложения (только для UniDac)

2.2.1. Для Windows

1) Создать проект FMX.
2) В опциях проекта установить путь к папке API_TERRA_PLUS/INCLUDE.

3) Переписать в папку проекта из папки API_TERRA_PLUS/INCLUDE файл IFPG_FMX.h и переименовать его в IFPG.H.

4) Включить в H-файл главного окна:

```
#include "ifpg.h"
#include "if_def.h"
#include "ifpg_serv.h"
#include "ifpg_uni.h"
#include "if_feat.h"
#include "if_multilib.h"
#include "DBAccess.hpp"
#include "Uni.hpp"
#include <Data.DB.hpp>
```

5) Включить в проект объектные файлы из папки API_TERRA_PLUS\FMX\OBJ:

```
ifpg_serv.obj
ifpg_main.obj
ifpg_uni.obj
if_feat.obj
if_multilib.obj
```

***) В h-файле ifpg_uni.h первым должен идти оператор #define __IFPG_UNIDAC__

6) Включить в CPP главного окна:

```
#pragma link "DBAccess"
#pragma link "Uni"
```

2.2.2. Для Android

1) Создать проект FMX.

2) В опциях проекта установить путь к папке API_TERRA_PLUS/INCLUDE.

3) Переписать в папку проекта из папки API_TERRA_PLUS/INCLUDE файл IFPG_FMX.h и переименовать его в IFPG.H.

4) Включить в H-файл главного окна:

```
#include "ifpg.h"
#include "if_def.h"
#include "ifpg_serv.h"
#include "ifpg_uni.h"
#include "if_feat.h"
#include "if_multilib.h"
#include "DBAccess.hpp"
#include "Uni.hpp"
#include <Data.DB.hpp>
```

5) Включить в проект объектные файлы из папки API_TERRA_PLUS\FMX\O:

```
ifpg_serv.o
ifpg_main.o
ifpg_uni.o
if_feat.o
if_multilib.o
```

6)Включить в проект файл libpgprovider210.a из установки Embarcadero (\LIB\ANDROID\RELEASE).

7) Включить в CPP главного окна:

```
#pragma link "DBAccess"
#pragma link "Uni"
```

2.3. Создание совместного VCL-приложения

Можно создать приложение работающее одновременно с базами данных СУБД ГИС Терра и СУБД ГИС Терра Plus (на примере использования FireDac).

1) Создать проект VCL-приложения.

2)В опциях проекта установить путь к папке API_TERRA_PLUS/INCLUDE и путь к папке INCLUDE API СУБД ГИС Терра.

3) Переписать в папку проекта из папки API_TERRA_PLUS/INCLUDE файл IFPG_VCL.h и переименовать его в IFPG.H.

4) Включить в H-файл главного окна приложения:

```
#include "b01md.h"
#include "ifpg.h"
#include "if_def.h"
#include "ifpg_uni.h"
#include "if_feat.h"
#include "if_multilib.h"
#include <FireDAC.Phys.PG.hpp>
#include <FireDAC.VCLUI.Wait.hpp>
```

5) Включить в проект библиотеку LIB_PLUS.LIB:

6) Включить в проект библиотеку СУБД ГИС Терра ‘BASELOAD_2006.LIB’.

8) В H-файл главной формы приложения включить переменную:

HINSTANCE hdll01

9) В событие OnCreate главной формы приложения включить код:

```
int err;
hdll01=Base01LoadFun1(&err,NULL);
```

10) В событие OnActivate главной формы приложения вставить код, проверяющий загрузку функций API СУБД ГИС Терра:

```
if (hdll01 == NULL)
{
 Close();
 return;
}
```

При программировании такого типа приложения особое внимание надо уделить правильному использованию функций двух API: работу с объектом, читаемым или записываемым в базу данных СУБД ГИС Терра производить ТОЛЬКО функциями его API (у них префикс _Base). Нарушение этого правила может привести к краху приложения, поскольку память под объект в СУБД ГИС Терра будет выделяться в DLL-библиотеке этой СУБД, а под объект в СУБД ГИС Терра Plus будет выделяться в приложении.

Для облегчения создания таких приложений разработан специальный универсальный класс чтения объектов TTerraReadObject (п.4.5).

3. СТРУКТУРЫ ДАННЫХ

Ниже приведены новые (по сравнению с СУБД ГИС Терра) структуры данных, используемые в функциях API. Они находятся в файле 'if_def.h'. Остальные структуры данных либо такие же как в СУБД ГИС Терра, либо аналогичные с заменой префикса _Base на IF.

3.1. Паспорт базы данных

typedef struct

```
{
 double Xmin;                  // Углы охватывающего прямоугольника
 double Ymin;
 double Xmax;
 double Ymax;
 BASE_INT CodePasport[10];     // Код паспорта
 BASE_INT CodeRamka[10];       // Код внутренней рамки
 BASE_INT CodeScale[10];       // Код масштаба
 BASE_INT CodeTP[10];          // Код точки привязки
 BASE_INT CodeSource[10];      // Код характеристики "источник"
 BASE_INT CodeTypeDoc[10];     // Код типа документа (обобщенный)
 BASE_INT CodeCoeff[10];       // Код нормирующего множителя
 int CountItemsMet;
 int SizeItemMet;
 int TypeItemMet;
 int UnitMet;
 double CoeffGrad;             // Коэффициент умножения для градусов
 double Distance;              // Радиус поиска в единицах базы
 int TypeCoord;                // Тип координат:
                               // 1 - градусы
                               // 0 - система листа
 int TypeDB;                   // Тип базы данных:
                               // 0 - база данных
                               // 1 - классификатор
                               // 2 - модель
                               // 3 - библиотека условных знаков
 String NameDB;
 String Database;             // Имя базы данных
 String NameClass;            // Имя классификатора (например: SEAMAP2)
} IFPG_PASPORT;
```

3.2. Объект

```
typedef struct
{
 GBASE_OBJECT Object;    // Объект в формате интегрального файла ГИС Терра
 int Key;                // Ключ доступа к объекту (0 – 32767)
 String NameCode;        // Название объекта по классификатору
 int CountItemsMet;      // Размерность (1-256)
 int SizeItemMet;        // Размер координаты (1-256)
 int TypeItemMet;        // Тип координаты:
                         // BASE_MET_INTEGER - целый
                         // BASE_MET_FLOAT - вещественный
                         // BASE_MET_LOGICAL - логический
 int Units;              // Тип представления метрики
                         // 0 - координаты в системе листа
                         // 1 - координаты в градусах
 int Resolution;         // Дискретность метрики (кол-во точек в 1 мм)
```

```
 int Scale;              // Коэффициент масштаба
 unsigned int ScaleObj; // Шкала объекта
 int CountVersions;      // Количество версий объекта
 int NumberVersion;      // Номер версии объекта (0 – актуальная)
 int CountSem;           // Количество характеристик (первого уровня)
 int CountSv;            // Количество связей
 int CountPr;            // Количество прерываний
 int CountMet;           // Колчество точек метрики
 int Color               // Цвет отображения объекта (0 – по умолчанию)
 int TypeUpdate;         // Маска режимов обновления последней версии
 int Status;             // 0 - актуальный, 1- версия, 2 - удаленный
 TDateTime DateObj;      // Дата создания/обновления/удаления
 int IdSource;      // Уникальный идентификатор источника данного объекта
 int TypeDoc;                // Уникальный идентификатор типа источника
 int IdCode;                 // Уникальный идентификатор кода объекта
 unsigned long Number;       // Номер объекта
 unsigned long LocalNumber;  // Локальный номер в базе источника
 int FlagNumber;             // Только для записи объекта
                  // 1 - Объект будет добавлен (записан) с
                  // идентификатором (IdCode) и номером (Number)
                  // заданными в этой структуре
                  // 0 - идентификатор кода и номер
                  // определяются автоматически
}IFPG_OBJECT;
```

3.3. Отметка глубины

```
typedef struct
{
 unsigned int IdDepth; // Уникальный идентификатор отметки глубины
 double X;             // Координаты
 double Y;
 double Value;         // Значение
 int IdSource;         // Идентификатор источника
 int TypeDoc;          // Тип источника
 int Scale;            // Коэффициент масштаба
 int Version;          // Номер версии (0 - актуальная)
 int NumVersions;      // Количество версий
 int Status;           // Статус отметки
                       //  0 - актуальная
                       //  1 - удаленная
                       //  2 - заюлокированная
 BASE_INT *pSem;       // При добавлении или обновлении отметки глубины
                       // можно подать поле семантики объекта
                       // в формате ИФ
                       // или NULL
 unsigned long LocalNumber; // Локальный номер в базе источника
} IFPG_DEPTH;
```

3.4. Метаданные источника информации

```
typedef struct
{
 int IdSource;        // Уникальный идентификатор исходного документа
 String NameSource;   // Имя исходного документа (из паспорта)
 String TypeSource;   // Тип исходного документа (ПГС,МНК ...)
 String FileName;     // Имя файла исходного документа
 String Ramka;        // Рамка исходного документа (полигон)
} IFPG_META;
```

4. КЛАССЫ

4.1. База данных

4.1.1. Конструктор

TIFPGdatabase ()

4.1.2. Подключение к базе данных

4.1.2.1. Подключение обычного пользователя

int OpenDataBase(String NameBase,String Server,int Port,String MainUser,String PasswordServer,String VendorHome,String Login,String Password,bool Prompt,String Debug)

NameBase - Имя базы данных
Server - Адрес сервера (например, 192.168.39.10)
Port - Порт сервера (например, 5432)
MainUser - Имя встроенного пользователя сервера (обычно 'postgres')
PasswordServer - Пароль доступа к серверу
Vendorhome - Путь к вендорным библиотекам PostreSQL (например, d:\baseutil). Нужно задавать только при использовании FireDac
Login - Имя пользователя
Password - Пароль пользователя
Prompt - true - включить диалог коннекта
Debug - Полное имя файла для отладочной печати SQL-запросов ("" - нет печати)
UserOper - Возвращает маску (суперпозицию) режимов доступа к операциям
OPER_VIEW – просмотр
OPER_IMPORT - импорт
OPER_EXPORT - экспорт
OPER_EDIT - редактирорвание

Функция возвращает:
-1 - нет коннекта к базе данных
0 - у пользователя (Login) нет прав доступа к базе данных
>0 - идентификатор пользователя

4.1.2.2. Подключение суперпользователя

bool open(String NameBase)

NameBase – имя базы данных.
Возвращаемое значение:
true – есть подключение;
false – нет подключения.

Перед подключением к базе данных в этом режиме необходимо выполнить следующие методы**.**

4.1.2.2.1. Задание номера порта

void SetPort(int Port)

Port – номер порта для подключения к базе данных (установлен в PostgreSQL). По умолчанию используется порт 5432.

4.1.2.2.2. Задание имени сервера

void SetServer(String Server)

Server – имя сервера с базой данных.
По умолчанию – 'localhost'.

4.1.2.2.3. Задание имени суперпользователя пользователя

void SetUserName(String Username)

Username – имя суперпользователя базы данных. По умолчанию – 'postgres'.

4.1.2.2.4. Задание пароля доступа к серверу

void SetPassword(String Password)

Password – пароль. По умолчанию – '123'.

4.1.2.2.5. Установка режима диалога при коннекте

void SetLoginPrompt(bool Prompt)

Prompt :
true – включить режим диалога при подключении к базе данных (установлен по умолчанию);
false – выключить.

4.1.2.2.6. Установление режима отладочной печати

void SetDebug(String Filedebug)

Filedebug – полное имя файла, в который будут выводиться все формируемые SQL-запросы.

Если Filedebug="", то режим отладочной печати будет отменен. По умолчанию режим отладочной печати отменен.

4.1.2.2.7. Задание папки с вендорными библиотеками FireDac

void SetVendorHome(String Folder)

Folder – папка с вендоными библиотеками (например, d:\baseutil). В этой папке в подпапке BIN должны находитьтся вендорные библиотеки POSTGRESQLЖ

libeay32.dll
libintl.dll
libpq.dll
ssleay32.dll

4.1.2.3. Установка режима использования хранимых процедур

void SetFunctions(bool UseFunction)

UseFunction:

true – включить режим использования хранимых функций (на сервере) отбора по мультифильтру (установлен по умолчанию);

false – отключить. Отбор по мультифильтру будет производиться на компьютере клиента.

Необходимо выполнить до открытия (коннекта) базы данных.

Если будет установлен режим использования хранимых процедур, то отбор объектов по не-SQL запросам (запросам СУБД ГИС Терра) будет производиться на сервере. Этот режим установлен по умолчанию. В этом случае необходимо наличие бибилиотеки IF_QUERY.DLL в папке LIB СУБД PostgreSQL.

4.1.2.4. Установление режима версионности

void SetRegVersions(bool RegVer)

RegVer:

true – включить режим версионности (запоминание версий полей объекта при их изменении);

false – отключить (установлен по умолчанию).

Необходимо выполнить до открытия (коннекта) базы данных.

4.1.2.5. Отключение от базы данных

void Close()

4.1.3. Получение параметров подключения

4.1.3.1. Получение номера порта

int GetPort()

Возвращает номер порта подключения.

4.1.3.2. Получение имени сервера

String GetServer()

Возвращает имя сервера.

4.1.3.3. Получение пароля

String GetPassword()

Возвращает пароль подключения.

4.1.3.4. Получение имени пользователя

String GetUserName(

Возвращает имя пользователя.

4.1.3.5. Получение имени базы данных

String GetDatabase()

Возвращает имя подключенной базы данных.

4.1.4. Управление транзакциями

Обычно эти функции должны использоваться при массовых операциях обновления информации в базе данных, которые должны быть гарантированно выполнены целиком (например, загрузка в базу данных информации с одного документа целиком). По умолчанию транзакции объявляются на запись/обновление одной единицы информации (например, объекта базы данных).

4.1.4.1. Старт транзакции

bool StartTransaction()

Функция возвращает:
true – транзакция открыта;
false – предыдущая транзакция не завершена.

4.1.4.2. Завершение транзакции

bool EndTransaction()

Функция возвращает:
true – транзакция закрыта (все изменения сделаны в базн данных);
false – транзакция не открыта.

4.1.4.3. Откат транзакции

bool RollbackTransaction()

Функция возвращает:
true – произведен откат транзакции;
false – транзакция не была открыта.

Пример использования данных функций.

```
IFPG *TIFPFdatabase=new TIFPGdatabase();
if (IFPG->Open(”База данных”)
{
 ……
 StartTransaction();
 ….
 операции с базой данных
 …..
 if(обнаружена ошибка)
 {
  IFPG->RollbackTransaction();
  IFPG-Close();
  delete IFPG;
  return;
 }
 else
 {
  IFPG->EndTRansaction();
 }
 IFPG->Close();
}
delete IFPG;
```

4.1.5. Работа с паспортом базы данных

4.1.5.1. Получение паспорта базы данных

IFPG_PASPORT * GetPasport()

Функция возвращает указатель на структуру паспорта или NULL в случае ошибки.

4.1.5.2. Обновление паспорта базы данных

int UpdatePasport(IFPG_PASPORT *PS)

PS – указатель на структуру с обновляемыми данными паспорта.
Функция возвращает:
BERR_OK – нормальное завершение;
BERR_BASE – обнаружены ошибки.
Можно обновить только координаты минимального охватывающего прямоугольника базы данных. Должна использоваться при записи новых объектов в базу данных для обновления минимального охватывающего прямоугольника базы данных.

4.1.6. Работа с источниками поступления информации

4.1.6.1. Получение списка типов документов источников

TStrings *GetListDocs()

Функция возвращает список названий типов документов источников (например, ”ПГС”,”S57”,”МНК(ИФ)” и т.д.) или NULL в случае обнаружения ошибки.

4.1.6.2. Получение идентификатора типа документа источника

int GetIdDoc(String NameDoc)

NameDoc – имя типа документа (например, ”ПГС”).
Функция возвращает уникальный идентификатор документа источника или 0 в случае обнаружения ошибки или отсутствии указанного типа.

4.1.6.3. Получение списка источников документа

TStrings *GetListSource(int IdDoc)

IdDoc – уникальный идентификатор типа источника (например, 1 – это тип данных ПГС).
Функция возвращает список названий источников (например, ”21008”,”21009”,”32029” и т.д.) или NULL в случае обнаружения ошибки.

4.1.6.4. Получение идентификатора источника

int GetIdSource(int IdDoc,String NameSource)

IdDoc – уникальный идентификатор типа источника (например, 1 – это тип данных ПГС);
NameSource – имя источника (например, “21008”).
Функция возвращает уникальный идентификатор источника или 0 в случае обнаружения ошибки.

4.1.6.5. Получение метаданных источника

IFPG_META *GetMetaDataObject(int IdDoc, int IdSource)

IdDoc – уникальный идентификатор типа источника (например, 1 – это тип данных ПГС);

IdSource – уникальный идентификатор источника (напимер, морской навигационной карты с адмиралтейским номером ‘21008’).

Значения указанных параметров получаются при чтении объектов из базы данных.

Функция возвращает указатель на структуру метаданных или NULL в случае обнаружения ошибки.

4.1.6.6. Получение имени таблицы источников данного типа

String GetNameDBDoc(String NameDoc)

NameDoc - имя типа документа (например, ”ПГС”).

Функция возвращает имя таблицы источников для данного типа документа или пустую строку в случае обнаружения ошибки.

4.1.6.7. Запись дополнительных данных в источники информации

int AddImages(int IdDoc,int IdSource,bool Flag,String FileName,String Caption)

IdDoc – уникальный идентификатор типа источника (например, 1 – это тип данных ПГС);

IdSource – уникальный идентификатор источника (напимер, морской навигационной карты с адмиралтейским номером ‘21008’);

Flag – признак: true – один файл, false – все файлы из папки:

FileName – полное имя файла или имя папки с файлами;

Caption – название, присваиваемое файлу или всем файлам из папки.

Функция возвращает:

BERR_OK – нормальное завершение;

BERR_PARAM – ошибка в параметрах;

BERR_BASE – ошибка в базе данных/

4.1.6.8. Получение файлов источника информации

Tstrings *GetFiles(int IdDoc,int IdSource,String Folder,String Filter)

IdDoc – уникальный идентификатор типа источника (например, 1 – это тип данных ПГС);

IdSource – уникальный идентификатор источника (напимер, морской навигационной карты с адмиралтейским номером ‘21008’);

Folder – папка, в которую помещаются файлы;

Filter – фильтр на расширение файлов (например, ”.BMP”).

Функция возвращает список имен выбранных файлов или NULL в случае обнаружения ошибки или отсутствия файлов.

4.1.7. Покоординатный поиск объектов

4.1.7.1. Поиск объекта в базе данных

int FindObjects(IF_FIND_CODE *FC,intSizeFC,double R,int X, int Y,int *IdCode,unsigned int *Number,double *D,IF_MULTI_FILTER *Filter,String AddBase,String AddSQL)

FC – указатель на массив структур найденных объектов;

```
typedef struct
{
 int IdCode;            // Идентификатор кода объекта
 unsigned int Number;   // Номер объекта
 double Distance;       // Расстояние до объекта в единицах базы
 BASE_INT Code[10];     // Код объекета
 String NameCode;       // Название объекта
} IF_FIND_CODE;
```

SizeFC – максимальный размер массива в элементах массива (например, 100 – на сто найденных объектов);

R – размер (в единицах базы данных) прямоугольника (квадрата) для поиска объектов (половина стороны квадрата);

X, Y – координаты центра квадрата поиска;

IdCode, Number – выдается уникальный идентификатор и номер ближайшего найденного объекта (объект в дальнейшем может прочитан методом Read из класса TIFPGobject);

D – расстояние (в единицах базы данных) до ближайшего найденного объекта;

Filter – мультифильтр для отбора объектов при поиске;

AddSQL – дополнительный SQL-запрос на отбор объектов или пустая строка. Пример: ”if_scale=500000 and if_status=1” – дополнительно отбирать только объекты масштаба 1:500000 и имеющие статус 1 (удаленные). Если нужен дополнительный фильтр на метаданные источников информации объектов, то дополнительно нужно сформировать параметр AddBase (см. ниже). Пример: ”and if_scale=500000 and if_name_source='RU3LFHD0'” – дополнительно отбирать объекты масштаба 1:500000, поступившие с источника информации RU3LFHD0 (файл S57);

AddBase – имя таблицы для данного типа источников информации. Пример: ”if_s57” – файлы в формате S57. Это параметр требуется только для установления отбора по данным источников информации. В остальных случаях должна быть задана пустая строка.

Функция возвращает количество объектов полученных в массиве FC.

Алгоритм поиска следующий: сначала отыскиваются все объекты, попадающие внутрь или пересекающие заданный параметром R прямоугольник. Информация о них заносится в массив структур FC. Если количество объектов превышает размер массива, то дальнейший поиск объектов прекращается. Затем, среди отобранных объектов ищется наиболее близкий к заданной точке (X, Y). Для площадного объекта поиск производится относительно центра тяжести объекта.

Следует учитывать, что время поиска и количество обрабатываемых объектов сильно зависит от правильного задания параметра R.

4.1.7.2. Поиск глубины в базе данных

int FindDepths(IF_FIND_CODE *FC,intSizeFC,double R,int X, int Y,int *IdDEpth,double *D,IF_MULTI_FILTER *Filter,String AddBase,String AddSQL)

FC – указатель на массив структур найденных объектов;

SizeFC – максимальный размер массива в элементах массива (например, 100 – на сто найденных объектов);

R – размер (в единицах базы данных) прямоугольника (квадрата) для поиска объектов (половина стороны квадрата);

X, Y – координаты центра квадрата поиска;

IdCoder – выдается уникальный идентификатор глубины ближайшей найденной глубины (в дальнейшем может быть прочитана методом Read из класса TIFPdepth);

D – расстояние (в единицах базы данных) до ближайшей глубины;

Filter – SQL-запрос для отбора глубин по значению и масштабу. Пример: “and value=100 and scale=500000” – глубины со значением 100 с масштаба 1:500000;

AddSQL – дополнительный SQL-запрос на метаданные источников информации глубин. Пример: ”and if_name_source='RU3LFHD0'” – дополнительно отбирать глубины, поступившие с источника информации RU3LFHD0 (файл S57). Если AddSQL задано (не пустая строка), то обязательно должен быть задан параметр AddBase;

AddBase – имя таблицы для данного типа источников информации. Пример: ”if_s57” – файлы в формате S57. Это параметр требуется только для установления отбора по данным источников информации. Если AddSQL пустая строка, то в AddBase должна быть задана пустая строка.

Функция возвращает количество глубин, полученных в массиве FC.

Алгоритм поиска аналогичен поиску объектов в базе данных (см.п.4.1.2.28).

4.2. Объект

4.2.1. Конструктор

TIFPGobject (TIFPFdatabase *DB)

DB – указатель на экземпляр объекта класса базы данных.

4.2.2. Установление параметров чтения объектов

4.2.2.1. Установление мультифильтра

bool SetMultiFilter(IF_MULTI_MKO *Filter)

Filter – указатель на сформированный мультифильтр (мультифильтр или сложный запрос СУБД ГИС Терра) или NULL (для его отключения).

Функция возвращает:
true – нормальное завершение;
false – ошибка в фильтре.
По умолчанию мультифильтр отключен.

4.2.2.2. Установление статуса читаемых объектов

bool SetObjectsStatus(int Status)

Status – статус объектов:
0 – актуальные (по умолчанию)
1- удаленные
2- заблокированные
Функция возвращает:
true – нормальное завершение;
false – ошибка в статусе.

4.2.2.3. Установление масштаба

bool SetScale(int Scale)

Scale – будут выбираться объекты только этого масштаба. 0 - читать объекты всех масштабов (по умолчанию).

Функция возвращает:
true - нормальное завершение
false - ошибка в параметре

4.2.2.4. Установление ключа доступа к объектам

bool SetKey(int Key)

Key – ключ доступа (>= 0)./ Если Key < Object.Key то объект не выдается
Функция возвращает:
true - нормальное завершение
false - ошибка в параметре

4.2.2.5. Установление отбора по метрике

bool SetMetricSelect(int Type,int CountPoints,MET_DOUBLE *Met)

Type - тип отбора (суперпозиция):
IFPG_METRIC_IN - строго внутри
IFPG_METRIC_CROSS - пересекает
CountPoints - количество точек области. Точек должно быть не менее 4-х
Met - координаты точек области. Область должна быть замкнута (первая точка совпадать с последней).
Функция возвращает:
true - нормальное завершение
false - ошибка в параметре

4.2.2.6. Установление дополнительного SQL-запроса

void SetAddSQL(String AddSQL)

Пример: ”if_scale=500000 and if_status=1” – отбирать только объекты масштаба 1:500000 и имеющие статус 1 (удаленные).

Атрибуты объекта, по которым можно сформировать дополнительный SQL-запрос:

if_scale – коэффициент масштаба (целое число);

if_status – статус объекта (целое число: 0 – актуальный, 1 – удаленный, 2 - заблокированный);

if_type – тип объекта (целое число: 1 – линейный, 2 – площадной, 3 – дискретный, 0 – без метрики).

4.2.2.7. Установление дополнительного SQL-запроса по метаданным

void SetAddBaseSQL(String AddBase, String AddSQL)

AddBase - имя таблицы для данного типа источников информации. Пример: ”if_s57” – файлы в формате S57;

AddSQL- условия отбора на метаданные. Пример: ”and if_name_source='RU3LFHD0'” – дополнительно отбирать объекты, поступившие с источника информации RU3LFHD0 (файл S57).

4.2.2.8. Сброс дополнительного SQL-запроса

void ClearAddSQL()

4.2.2.9. Сброс дополнительного SQL-запроса по метаданным

void ClearAddBaseSQL()

4.2.2.10. Установка количества записей для чтения из базы данных

void SetRowSetSize(int Size)

Size – количество записей в считываемой порции (по умолчанию – 1000). Увеличение количества записей приводит к убыстрению процесса чтения (если позволяют ресурсы оперативной памяти).

4.2.3. Чтение объектов из базы данных

4.2.3.1. Чтение объектов без отбора

IFPG_OBJECT * Open(BASE_INT *Code,unsigned int Number,int Type,int Parts)

Code – код объекта:

полный (при Number=0 будут читаться все объекты данного полного кода);

обобщенный (Code[0]=0 – все объекты). В этом случае должен быть Number=0;

Number – номер объекта (только когда Code полный);

Type - OPEN_CODE - чтение всех объектов указанного полного кода, OPEN_TREE - чтение всех объектов из дерева указанного обобщенным кодом (обход дерева произвольный);

Parts - режим чтения полей объекта (суперпозиция)

OBJ_CODE - только постоянная часть объекта (код,номер,шкала,ключ доступа)

OBJ_SEM - поле характеристик

OBJ_SV - поле связей

OBJ_PR - поле прерываний

OBJ_MET - поле метрики

OBJ_TEXT- поле текста

OBJ_FULL - все поля объекта

Функция возвращает указатель на описатель первого объекта или NULL если объектов нет. По этой функции устанавливается запрос на чтение объектов без отбора по мультифильтру. Обор по метрическому описанию и дополнительные SQL-запросы могут быть задействованы.

4.2.3.2. Чтение следующего объекта по установленному запросу

IFPG_OBJECT Next()

Функция возвращает указатель на очередной объект или NULL, если объекты по установленному запросу закончились.

4.2.3.3. Повторное чтение объекта

IFPG_OBJECT Retry()

Функция читает заново только что прочитанный объект.

Функция возвращает указатель на объект или NULL в случае ошибки.

4.2.3.4. Чтение предыдущих версий объекта

IFPG_OBJECT PreviousVersion()

Для только что прочитанного объекта можно организовать чтение всех предыдущих версий объектов. Данная функция будет в обратном хронологическом порядке выдавать предыдущие версии, пока они не закончатся и функция не вернет NULL.

4.2.3.5. Принудительное закрытие запроса на чтение

void Close()

Во всех открытых запросах на чтение объектов происходит автоматическое закрытие запроса при исчерпании объектов. Данная функция позволяет закрыть запрос при необходимости его принудительного завершения.

4.2.3.6. Чтение объектов с отбором по мультифильтру

bool OpenMulti(int Parts)

Parts – читаемые части объекта (аналогично функции Open).

Открывается запрос на чтение объектов по установленному мультифильтру (функция SetMultiFilter). Если фильтр не установлен или он пустой, то будут читаться все объекты. Первый объект не читается.

Функция возвращает:

true - нормальное завершение

false - ошибка в базе данных

4.2.3.7. Чтение первого и очередных объектов по мультифильтру

IFPG_OBJECT *NextMulti()

Функция возвращает указатель на первый или очередной объект или NULL, если объекты по установленному запросу закончились.

4.2.3.8. Повторное чтение объекта по мультифильтру

IFPG_OBJECT RetryMulti()

Функция читает заново только что прочитанный объект.

Функция возвращает указатель на объект или NULL в случае ошибки.

4.2.3.9. Принудительное закрытие запроса на чтение по мультифильтру

void CloseMulti()

Во всех открытых запросах на чтение объектов происходит автоматическое закрытие запроса при исчерпании объектов. Данная функция позволяет закрыть запрос при необходимости его принудительного завершения.

4.2.3.10. Чтение объекта по его номеру

IFPG_OBJECT *Read(unsigned int Number, int Parts)

Number – номер объекта;

Parts – читаемые части объекта (как в функции Open).

Функция возвращает указатель на объект или NULL, если объекта нет.

4.2.3.11. Получение идентификатора существующего кода объекта

int GetIdCode(BASE_INT *Code,unsigned long *Number)

Code – код объекта;

Number - выдается максимальный номер объекта данного кода.

Функция возвращает уникальный идентификатор существующего кода объекта или 0;

4.2.4. Запись, обновление и удаление объектов

4.2.4.1. Добавление нового объекта

unsigned int Add(IFPG_OBJECT *NewObject)

NewObject – указатель на структуру сформированного объекта.
Должны обязательно быть сформированы следующие поля структуры:
Object – объект в формате интегрального файла
CountItemsMet
SizeItemMet
TypeItemMet
Units
Resolution
Scale
Status=0
FlagNumber=1 - Объект будет добавлен (записан) с/ идентификатором (IdCode) и номером (Number), заданными в этой структуре
FlagNumber=0 - идентификатор кода и номер определяются автоматически
IdCode (при FlagNumber=1)
Number (при FlagNumber=1)
IdSource - уникальный идентификатор источника данного объекта (если неизвестен, то 0)
TypeDoc - уникальный идентификатор типа источника (если неизвестен, то 0)
Функция возвращает номер записанного объекта или 0 при обнаружении ошибки.

4.2.4.2. Обновление объекта

unsigned int Update(IFPG_OBJECT *Object)

Object – указатель на структуру обновляемого объекта.
Функция возвращает номер обновленного объекта или 0 при обнаружении ошибки.

4.2.4.3. Удаление частей объекта

int DeleteParts(BASE_INT *Code,unsigned int Number,int Parts,int Reg,int Status)

Code - код объекта (полный или обобщенный)
Number - номер объекта (если 0 - обрабатываются все объекты указанного кода)
Parts - суперпозиция частей объекта (как в функции Open)
Reg – режим работы: 0 - логическое удаление, 1 - физическое удаление
Status – статус обрабатываемых объектов (как в функции SetStatus).
Функция возвращает:
BERR_OK – нормальное завершение;
BERR_NOINF – нет информации;
BERR_PARAM – ошибка в параметрах;
BERR_BASE – ошибка в базе данных.

4.2.5. Работа с полем мультимедиа объекта

4.2.5.1. Запись мультимедийного элемента из файла

int AddMediaFromFile(BASE_INT *Code,unsigned int Number, String FileName,String Name)

Code, Number – код и номер объекта;
FileName – имя файла (в объект может быть помещена информация из файла любого типа);
Name – имя, которое присваивается записываемому элементу.
Функция возвращает:
BERR_OK - нормальное завершение
BERR_NOINF - нет объекта
BERR_PARAM - файл не найден
BERR_BASE - ошибка в базе данных

4.2.5.2. Запись мультимедийного элемента из массива

int AddMediaFromArray(BASE_INT *Code,unsigned int Number,String Name, TByteDynArray Array,String Ext)

Code, Number – код и номер объекта;
Name – имя элемента;
Array – динамический массив с данными;
Ext – строка, присваивая элементу в виде расширения файла или ””.
Функция возвращает:
BERR_OK - нормальное завершение
BERR_NOINF - нет объекта
BERR_BASE - ошибка в базе данных

4.2.5.3. Удаление элемента мультимедиа

int DeleteMedia(BASE_INT *Code,unsigned int Number,String Name)

Code, Number – код и номер объекта;
Name - имя элемента мультимедиа. Если Name="" удаляются все элементы
Функция возвращает:
BERR_OK - нормальное завершение
BERR_NOINF - нет объекта
BERR_PARAM - нет элемента
BERR_BASE - ошибка в базе данных

4.2.5.4. Выборка элемента мультимедиа в файл

String GetMedia(BASE_INT *Code,unsigned int Number,String Name, String FileName)

Code, Number – код и номер объекта;
Name – имя элемента мультимедиа;
FileName – имя файла (без расширения), в который выдается элемент.
Функция возвращает:
BERR_OK - нормальное завершение
BERR_NOINF - нет объекта

BERR_PARAM - нет элемента
BERR_BASE - ошибка в базе данных

4.2.5.5. Выборка элемента мультимедиа в массив

int GetMediaToArray(BASE_INT *Code,unsigned int Number,String Name, TByteDynArray *Array,String *Ext)

Code, Number – код и номер объекта;
Name – имя элемента мультимедиа;
Array – указатель на динамический массив для записи в него элемента;
Ext – для выдачи расширения файла (если оно есть).
Функция возвращает:
BERR_OK - нормальное завершение
BERR_NOINF - нет объекта
BERR_PARAM - нет элемента
BERR_BASE - ошибка в базе данных

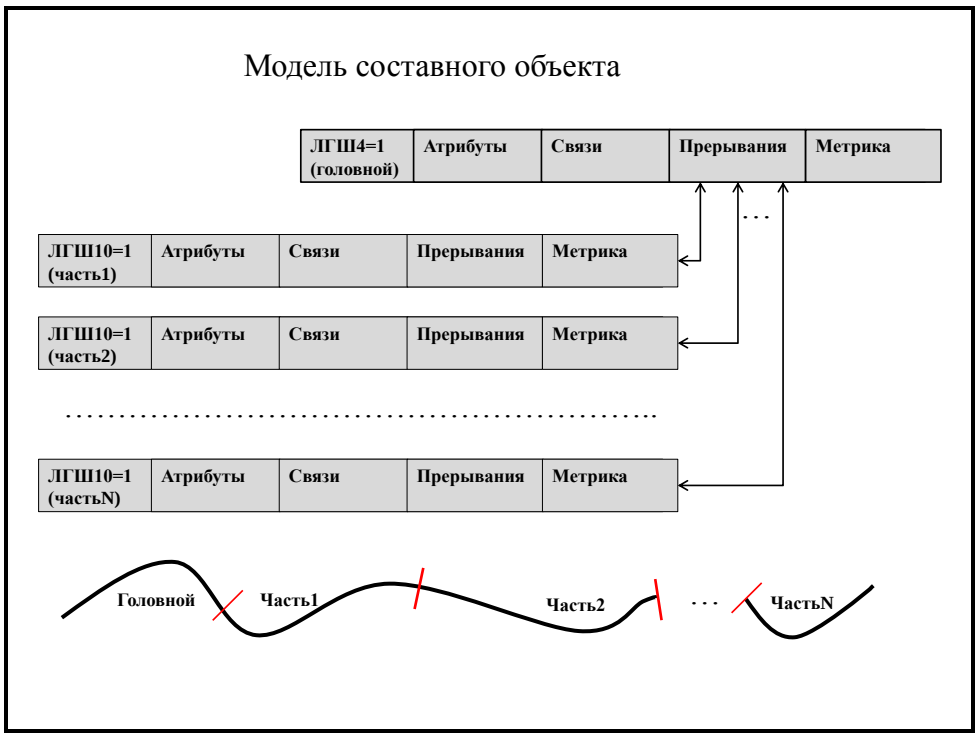
4.2.5.6. Получение списка элементов мультимедиа

TStrings *GetListMedia(BASE_INT *Code,unsigned int Number)

Code, Number – код и номер объекта.
Функция возвращает список имен элементов мультимедиа или NULL при их отсутствии.

4.2.6. Работа с составным объектом

Модель составного объекта используется при создании и ведении базы данных на большую территорию. В этом случае могут появиться очень большие по протяженности объекты, которые трудно, а иногда и невозможно обработать в оперативной памяти ЭВМ. Для представления таких объектов в базе данных используется модель составного объекта, приведенная ниже на рисунке:



Составной объект состоит из головного объекта и его частей.

Головной объект имеет следующую структуру:

В логической шкале объекта включен бит № 4 – ”Составной объект”;

- Поле характеристик содержит характеристики головной части составного объекта;
- Поле связей содержит связи головного объекта с другими объектами базы данных. Для площадного составного объекта дополнительно поле связей содержит все связи на объекты, являющимися внутренними контурами данного составного объекта;
- Поле прерываний содержит все прерывания головного объекта. Дополнительно поле прерываний содержит прерывания-ссылки на части данного составного объекта, идущие в порядке обхода метрики данного составного объекта. Эти прерывания ”привязаны” к последней точке в метрике головного объекта. У каждого из таких прерываний в логической шкале должны быть включены биты № 13 и № 14;
- Поле метрики содержит метрическое описание головного объекта.

Каждая часть составного объекта имеет следующую структуру:

- В логической шкале объекта включен бит № 10 – ”Часть составного объекта”;
- Поле характеристик содержит характеристики данной части составного объекта;
- Поле связей содержит связи данной части составного объекта с другими объектами базы данных;
- Поле прерываний содержит все прерывания данной части составного объекта. Дополнительно поле прерываний содержит прерывание-ссылку на головной объект данного составного объекта. Это прерывание ”привязано” к нулевой точке в метрике данной части составного объекта. У этого прерывания в логической шкале должны быть включены биты № 15 и № 16;
- Поле метрики содержит метрическое описание данной части составного объекта

Первая точка данного составного объекта должна совпадать с последней точкой предыдущего составного объекта или головного объекта (если данная часть составного объекта является его первой частью).

4.2.6.1 Открытие на чтение потоком частей составного объекта, начиная с головного

IFPG_OBJECT *ReadSostOpenByMain(BASE_ZAP_SP *Q,IFPG_OBJECT *ObjMain,int Parts)

Q - структура запроса для чтения прерываний;

ObjMain - прочитанный головной объект составного объекта;

Parts - суперпозиция частей читаемых объектов (прерывания читаются всегда).

Функция возвращает указатель на структуру прочитанного головного объекта или NULL при ошибке.

4.2.6.2. Чтение очередной части составного объекта

IFPG_OBJECT *ReadSostCont(BASE_ZAP_SP *Q,IFPG_OBJECT *ObjMain,int Parts)

Q - структура запроса для чтения прерываний (та же, что была указана в функции ReadSostOPenByMain);

ObjMain – ранее прочитанный головной объект составного объекта;

Parts - суперпозиция частей читаемых объектов (прерывания читаются всегда).

Функция возвращает указатель на структуру очередной части головного объекта или NULL при их исчерпании.

4.2.6.3. Получение номера головного объекта по любой его части

unsigned int GetSostMainByPart(IFPG_OBJECT *ObjPart)

ObjPart – прочитанная часть составного объекта.

Функция возвращает номер головного объекта или 0 при ошибке.

Ниже приведен пример работы с составным объектом.

```
// Number – это номер объекта
// Form1->IFPG – это указатель на открытую ранее базу данных
BASE_ZAP_SP Q;
unsigned int NumberMain;
TIFPGobject *obj, *sost;
IFPG_OBJECT *Obj1, *Obj2;
obj=new TIFPGobject(Form1->IFPG);
sost=new TIFPGobject(Form1->IFPG);
// Чтение объекта по его номеру
Obj1=obj->Read(Number,OBJ_SEM|OBJ_PR|OBJ_METRIC,0,0);
again1:
if (Obj1 != NULL)
{
 if (Obj1->Object.Scale & OBJ_SCALE_04) // Это составной объект ?
 {
  // Открытие на чтение потоком частей составного объекта
  // Читается головной объект составного объекта
  Obj2=sost->ReadSostOpenByMain(&Q,Obj1,OBJ_FULL);
  if (Obj2 == NULL) goto fin;
  // Здесь может быть обработка прочитанного объекта
  // Продолжение чтения частей составного объекта
  Obj2=obj->ReadSostCont(&Q,Obj1,OBJ_FULL);
  while (Obj2 != NULL)
  {
   // Здесь может быть обработка прочитанного объекта
   // Продолжение чтения частей составного объекта
   Obj2=sost->ReadSostCont(&Q,Obj1,OBJ_FULL);
  }
 }
 else
 {
  if (Obj1->Object.Scale & OBJ_SCALE_10) //Это часть составного объекта ?
  {
   // Получение номера головного объекта по прочитанной его части
   NumberMain=sost->GetSostMainByPart(Obj1);
   if (NumberMain != 0)
   {
    // Чтение головного объекта
    Obj1=obj->Read(NumberMain,OBJ_SEM|OBJ_PR|OBJ_METRIC,0,0);
    goto again1; // Переход к обработке головного объекта
   }
   else
   {
    ShowMessage("Не найден головной объект");
   }
  }
  else
  {
   ShowMessage("Это не составной объект");
  }
 }
}
fin:
delete obj;
delete sost;
```

4.2.7. Дополнительные функции для работы с объектом

4.2.7.1. Печать объекта в файл

int PrintObjectToFile(char *FileName,int RegOpenFile,IFPG_OBJECT *Object,int Parts,TIFPGdatabase *CLS)

FileName – имя файла;
RegOpenFile – режим открытия файла: 0 – “wt” (запись заново), 1 – “at” (добавление в файл);
Object – объект
Parts – печатаемые части объекта (как в функции Open);
CLS – указатель на объект базы данных классификатора или NULL.
Функция возвращает:
BERR_OK – нормальное завершение;
BERR_BASE – ошибка в объекте.

4.2.7.2. Проверка наличия объекта

unsigned int CheckObject(BASE_INT *Code,unsigned intNumber,int Status)

Code, Number – код и номер объекта
Status – статус объекта (как в функции SetStatus).
Функция возвращает уникальный идентификатор кода объекта или 0, если объекта нет.

4.2.7.3. Получение идентификатора кода и нового номера объекта

unsigned int GetNewNumber(BASE_INT *Code,unsigned int *Number)

Code – код объекта;
Number – возвращает новый номер объекта.
Функция возвращает идентификатор классификационного кода объекта и новый номер объекта или 0 при обнаружении ошибки.

4.2.7.4. Вычисление расстояния между объектом и точкой

double GetDistance(BASE_INT *Code,unsigned int Number,int NumVersion,double X, double Y)

Code, Number – код и номер объекта;
NumVersion – номер версии объекта (0 – актуальная);
X, Y – координаты точки в единицах базы данных.
Функция возвращает расстояние в единицах базы данных. Для площадного объекта вычисляется расстояние до центра тяжести.

4.2.7.5. Вычисление расстояния между прочитанным объектом и точкой

double GetDistanceObject(double X,double Y)

4.2.7.6. Инициализация структуры объекта

void InitObject(IFPG_OBJECT *Object)

Object – указатель на структуру объекта.

При открытии любого запроса на чтение объектов структура объекта инициализируется автоматически.

Функция требуется при подготовке нового объекта для записи в базу данных.

4.2.7.7. Очистка структуры объекта

void CloseObject(IFPG_OBJECT *Object*)

Object – указатель на структуру объекта.

4.2.8. Работа с объектом в оперативной памяти

В СУБД ГИС Терра Plus работа с объектом в оперативной памяти (в структуре GBASE_OBJECT) производится аналогично СУБД ГИС Терра.

4.2.8.1. Работа с метрическим описанием объекта

В метрическое описание объекта в оперативной памяти можно добавлять новые точки, удалять и обновлять указанные точки с помощью функций:

IFAddMetric – добавление точек

IFUpdateMetric – обновление точек

IFDelMetric – удаление точек

Описание указанных выше функций полностью соответствует аналогичным функциям СУБД ГИС Терра [7]. Названия функций отличаются префиксом (IF вместо Base).

4.2.8.2. Работа с семантическим описанием объекта

Работа с семантическим описанием объекта (прерывания или связи) обеспечивается функциями:

IFReadFeature – чтение характеристики

IFReadFeatureComp – чтение характеристики из состава сложной характеристики

IFDeleteFeature – удаление характеристики

IFDeleteFeatureComp – удаление характеристики из состава сложной характеристики

IFAddFeature – добавление характеристики

IFAddFeatureComp – добавление характеристики в состав сложной характеристики

IFUpdateFeature – замена характеристики

IFUpdateFeatureComp – замена характеристики в составе сложной характеристики

IFFeatureDelFull – удаление первой встреченной характеристики (простой или сложной) с указанным кодом

IFAddFeatureCompUniversal – добавление характеристики в состав сложной даже при отсутствии этой сложной характеристики

Описание указанных выше функций полностью соответствует аналогичным функциям СУБД ГИС Терра [7]. Названия функций отличаются префиксом (IF вместо Base).

4.2.8.3. Работа со связями и прерываниями объекта

Работа сосвязями и прерываниями объекта в оперативной памяти обеспечивается функциями:

IFReadSvPr – чтение прерываний/связей потоком

IFAddSvPr – добавление прерывания/связи

IFDeleteSvPr – удаление прерывания/связи

IFGetAdrSvPrForNumber – чтение прерывания/связи по указанному порядковому номеру

Описание указанных выше функций полностью соответствует аналогичным функциям СУБД ГИС Терра [7]. Названия функций отличаются префиксом (IF вместо _Base).

В отличии от СУБД ГИС Терра поддерживается только один способ выделения памяти под связи/прерывания – динамическое взятие памяти единым массивом функциям СУБД.

4.3. Документ

Этот класс предназначен для организации хранения и обработки в базе данных так называемых ДОКУМЕНТОВ. Может использоваться только в варианте API с использованием FireDac.

ДОКУМЕНТ представляет собой произвольный набор файлов, объединенных в некоторую единую сущность с помощью регистрационной карточки документа (файла специальной структуры). Структура регистрационной карточки приведена ниже.

```
[MAIN]
Name=<Название документа> (Например, Центральная часть НН)
Type=<Тип документа> (Например, Электронная карта)
MainFile=<Имя основного файла документа> (Например, 1_2000.ini)
DateCreate=10.07.2006
Region=<Название региона> (Например, Нижний Новгород)
LatitudeSW=<Широта юго-западного угла в градусах>
LongitudeSW=<Долгота юго-западного угла в градусах>
LatitudeNW=<Широта северо-западного угла в градусах>
LongitudeNW=<Долгота северо-западного угла в градусах>
LatitudeNE=<Широта северо-восточног угла в градусах>
LongitudeNE=<Долгота северо-восточного угла в градусах>
LatitudeSE=<Широта юго-восточного угла в градусах>
LongitudeSE=<Долгота юго-восточного угла в градусах>
[ADDFILE]
File1=1_2000.big
File2=1_2000.idx
File3=1_2000.dpf
```

MAIN – основные данные карточки.

Name – имя документа (любой текст до 256 символов).

Type – тип документа (в базе данных документы структурируются по их типам).

MainFile – имя основного файла документа (без пути).

DateCreate – дата записи документа в базу данных.

Region – район местности, в котором позиционируется документ (может отсутствовать).

Latitude и Longitude – долготы и широты минимального охватывающего прямоугольника документа в градусах (могут отсутствовать).

ADDFILE – секция дополнительных файлов:

FileI – имя очередного дополнительного файла документа (без пути). I – меняется от 1 через 1.

Перед записью в базу данных все файлы документа (в том числе и файл регистрационной карточки) должны находиться в одной папке.

4.3.1. Конструктор

TImageGSD(TIFPGdatabase *IFPG)

IFPG – указатель на экземпляр объекта открытой база данных

4.3.2. Запись документа в базу данных

TABIMAGEGSD * WriteImageToDB(String FileIRC)
FileIRC – полное имя файла регистрационной карточки

Функция возвращает указатель на структуру типа ***TABIMAGEGS*** (или NULL)***:***

```
typedef struct
{
 int Id;           // Уникальный идентификатор документа
 TDate Date;       // Дата записи документа
 String Name;      // Имя документа
 String MainFile;  // Имя основного файла
 String Type;      // Тип документа
 String Comment;   // Комментарий
 String Region;    // Имя региона
 MET_LONG_2 Coord[100];//Координаты охватывающего прямоугольника в единицах базы
 int CountPoints;   // Количество точек прямоугольника
 TStringList *List; // Список имен дополнительных файлов
} TABIMAGEGSD;
```

4.3.3. Выборка документов

4.3.3.1. Установление атрибутивного фильтра на чтение документов

void SetAttrFilter(String Filter)
Filter – SQL запрос на атрибуты документа:
name – имя документа
date – дата записи в базу
region – имя региона
type - тип
comment - комментарий

4.3.3.2. Установление координатного фильтра на чтение документов

void SetCoordFilter(String Filter)
Filter – отбор по координатам на языке SQL

4.3.3.3. Открытие запроса на чтение документов

TABIMAGEGSD * Open()
Выдается первый документ по запросу или NULL.

4.3.3.4. Чтение следующего документа по запросу

TABIMAGEGSD * Next()
Выдается следующий документ по запросу или NULL.

4.3.3.5. Принудительное закрытие открытого ранее запроса

void Close()
Примечание: при исчерпании документов по открытому запросу запрос автоматически закрывается.

4.4. Отметка глубины

Отметка глубины в базе данных имеет следующие атрибуты:
- value – значение глубины (вещественное число);
- scale – коэффициент масштаба (целое число);

- version – номер версии отметки.

Названия полей этих атрибутов могут потребоваться при установлении фильтра на отбор объектов из базы данных.

4.4.1. Конструктор

TIFPGdepth(TIFPFdatabsae *DB)

DB – указатель на экземпляр объекта класса базы данных.

4.4.2. Установление режимов выборки отметок глубин

4.4.2.1. Установление отбора по метрике

bool SetMetricSelect(int Type,int CountPoints,MET_DOUBLE *Met)

Type - тип отбора (суперпозиция):
IFPG_METRIC_IN - строго внутри
IFPG_METRIC_CROSS – на границе
CountPoints - количество точек области. Точек должно быть не менее 4-х
Met - координаты точек области. Область должна быть замкнута (первая точка совпадать с последней).
Функция возвращает:
true - нормальное завершение
false - ошибка в параметре

4.4.2.2. Установление статуса читаемых отметок глубин

int SetDepthStatus(int Status)

Status - статус :
0 - актуальные
1 - удаленные
2 - заблокированные

4.4.2.3. Установить режим чтения поля семантики

int SetSemIF(BASE_INT *pSem,int MaxSizeSem)

pSem - указатель на массив для поля семантики (если pSem=NULL, то режим отключен)

MaxSizeSem - размер в байтах массива. Если массива будет недостаточно или поле семантики в объекте отсутствует, то будет pSem[0]=0.

4.4.2.4. Установление фильтра

void SetFilter(String Filter)

Filter – это условия на языке SQL на отбор отметок глубин по их атрибутам в таблице.

Например: Filter=”scale=500000 and value=100” устанавливает отбор отметок глубин масштаба 1:500000 со значением 100.

4.4.2.5. Очистка фильтра

void ClearFilter()

4.4.2.6. Установление фильтра на метаданные источника

void SetAddMetaSQL(String AddBase,StringAddSQL)

AddBase - имя таблицы для данного типа источников информации. Пример: ”if_s57” – файлы в формате S57;

AddSQL- условия отбора на метаданные. Пример: ”and if_name_source='RU3LFHD0'” – дополнительно отбирать объекты, поступившие с источника информации RU3LFHD0 (файл S57).

4.4.2.7. Удаление фильтра на метаданные источника

void ClearAddMetaSQL()

4.4.2.8. Установка количества записей для чтения из базы данных

void SetRowSetSize(int Size)

Size – количество записей (по умолчанию – 1000). Увеличение количества записей приводит к убыстрению процесса чтения (если позволяют ресурсы оперативной памяти).

4.4.3. Чтение отметок глубин

4.4.3.1 Открытие запроса на чтение

IFPG_DEPTH *Open()

Выдается первая отметка глубины или NULL (обрабатываются только актуальные версии).

4.4.3.2. Чтение очередной отметки глубины

IFPG_DEPTH *Next()

Выдается очередная отметка глубины или NULL при их исчерпании. При NULL запрос автоматически закрывается.

4.4.3.3 Принудительное закрытие запроса на чтение

void Close()

4.4.3.4. Индивидуальное чтение

IFPG_DEPTH *Read(long int IdDepth,int Version)

IdDepth – уникальный идентификатор отметки глубины;
Version – номер версии (>=0).

4.4.3.5. Получение прочитанной отметку в формате ИФ

bool GetObjectIFRetry(GBASE_OBJECT *Object)

Object – указатель на структуру для объекта
Функция возвращает:
True – объект сформирован

false - отметка не найдена

4.4.3.6. Получение отметки глубины в формате ИФ

bool GetObjectIF(long int IdDepth,GBASE_OBJECT *Object,int Version)

IdDepth – уникальный идентификатор отметки;
Object – указатель на структуру для объекта;
Version – номер версии отметки.
Функция возвращает:
true – объект сформирован
false - отметка не найдена

4.4.3.7. Вычисление расстояния до точки

double GetDistance(long int IdDepth,double X,double Y)

IdDepth – уникальный идентификатор отметки глубины;
X, Y - координаты точки в единицах базы данных.

4.4.3. Добавление, обновление и удаление

4.4.3.1. Добавление

bool Add(IFPG_DEPTH *Depth)

Depth – указатель на структуру с данными отметки глубины.
Функция возвращает:
true – отметка записана;
false – ошибка в базе данных.
В структуре Depth должны быть заданы:
X, Y – Координаты (если задано pSem, то берутся оттуда при наличии)
Value – Значение (если задано pSem, то берется оттуда при наличии)
IdSource - Идентификатор источника (если имеется, если нет то задать 0)
TypeDoc - Тип источника (если имеется, если нет то задать 9)
Scale - Коэффициент масштаба
pSem - поле семантики объекта (в формате ИФ) или NULL

4.4.3.2. Полное удаление отметки глубины

bool DeleteFull(long int IdDepth,String Filter)

IdDepth – уникальный идентификатор отметки глубины (0 - удаление всех);
Filter - SQL-фильтр на отбор отметок глубины (только при IdDepth=0) или пустая строка. Правила задания как в функции SetFilter.
Функция возвращает:
true – отметка (отметки) удалена;
false – ошибка в базе данных.

4.4.3.3. Удаление актуальной версии с сохранением

bool Delete(long int IdDepth,String Filter)
IdDepth – уникальный идентификатор отметки глубины (0 - удаление всех);
Filter - SQL-фильтр на отбор отметок глубины (только при IdDepth=0) или пустая строка. Правила задания как в функции SetFilter.

Функция возвращает:
true – отметка (отметки) удалена;
false – ошибка в базе данных.

4.4.3.4. Физическое удаление отметок указанного статуса

bool Clear(int Status)

Status – статус (0, 1, 2).

4.4.3.5. Обновление прочитанной отметки глубины

bool Update(IFPG_DEPTH *Depth,int Flag)

Depth – указатель на структуру с прочитанной отметкой, в ней нужно задать изменяемые значения. Остальные не трогать;

Flag - режим работы:
1 - сохранять предыдущую версию
0 - не сохранять

Функция возвращает:
true – отметка обновлена;
false – ошибка в базе данных.

4.5. Универсальный класс чтения объектов

Данный класс предназначен для организации совместной работы с базами данных ГИС Терра и ГИС Терра Plus одними и темиже средствами.

4.5.1. Конструктор класса

TTerraReadObject(int Reg, void *Base)

Reg - Тип базы данных:
BASE_TERRA – база данных ГИС Терра
BASE_TERRA_PLUS – база данных ГИС Терра Plus

Base - указатель на базу данных ГИС Терра или ГИС TEppa PLus

4.5.2. Открытие запроса на чтение объектов по дереву

int ReadOpenTree(GBASE_OBJECT **Object)

Object – адрес указателя на структуру объекта
Возвращает значение:
BERR_OK – запрос открыт и прочитан первый объект
BERR_NOINF – нет объектов по запросу
BERR_BASE – ошибка в базе данных

4.5.3. Открытие запроса на чтение объектов по коду объекта

int ReadOpenCode(GBASE_OBJECT **Object)

Object – адрес указателя на структуру объекта
Возвращает значение:
BERR_OK – запрос открыт и прочитан первый объект
BERR_NOINF – нет объектов по запросу
BERR_BASE – ошибка в базе данных

4.5.4. Чтение следующего объекта по запросу

int ReadNext(GBASE_OBJECT **Object)
Object – адрес указателя на структуру объекта
Возвращает значение:
BERR_OK – прочитан очередной объект
BERR_NOINF – больше нет объектов по запросу
BERR_BASE – ошибка в базе данных

4.5.5. Повторное чтение только что прочитанного объекта

int Retry(GBASE_OBJECT **Object)
Object – адрес указателя на структуру объекта
Возвращает значение:
BERR_OK – прочитан объект
BERR_BASE – ошибка в базе данных

4.5.6. Индивидуальное чтение одного объекта

int Read(unsigned int Number, GBASE_OBJECT **Object)
Number – номер объекта. Для базы данных ГИС Терра предварительно неоходимо задать полный код объекта функцией SetCode (см. ниже)
Object – адрес указателя на структуру объекта
Возвращает значение:
BERR_OK – прочитан объект
BERR_NOINF – объект не найден
BERR_BASE – ошибка в базе данных

4.5.7. Получение полной информации объекта

void GetAdditionInfo(IFPG_OBJECT **Obj)
Obj – адрес указателя на полную структуру объекта

Перед тем, как начать читать объекты можно установить условия на их выборку:
- по мультифильтру;
- по области поиска;
- по обобщенному или полному коду;
- по дополнительному SQL запросу.

Для этих целей имеется ряд методов этого класса, приведенный ниже.

4.5.8. Задание мультифильтра

void SetFilter(void *Filter)
Filter - указатель на мультифильтр (GBASE_MULTI_MKO для ГИС Терра или IF_MULTI_MKO для ГИС Терра Plus). NULL по умолчанию.

4.5.9. Установка режима чтения частей объекта

void SetParts(int Parts)
Parts - Суперпозиция частей обхъекта (весь объект OBJ_FULL по умолчанию)

4.5.10. Задание кода объекта или группы объектов

int SetCode(BASE_INT *Code)
Code - классификационный код объекта (Code[0]=0 - все объекты)

4.5.11. Задание области поиска объектов

int SetRegion(int CountMet,MET_DOUBLE_2 *Region,int Type)

CountMet - количество точек (не менее четырех, первая должна совпадать с последней)

Region - точки области

Type - тп поиска (суперпозиция):

IFPG_METRIC_IN - строго внутри
IFPG_METRIC_OUT - строго снаружи
IFPG_METRIC_CROSS - пересекает

Функция возвращает значение:
BERR_OK – нормальное завершение
BERR_PARAM – ошибка в параметрах

4.5.12. Задание дополнительного SQL запроса

void SetAddSql(String AddSql)

AddSql - текст SQL запроса

5. ВСПОМОГАТЕЛЬНЫЕ ФУНКЦИИ

В этом разделе приведены дополнительные вспомогательные функции:
IFStrToCode – преобразование классификационного кода из строки в массив чисел;
IFCodeToStr – преобразование классификационного кода из массива чисел в строку;
IFCompCode – сравнение классификационных кодов.

IFStrToCode(String StrCode,BASE_INT *Code)

StrCode – классификационный код (перечень чисел, разделенных точкой. Например: 2.1.1.2);
Code - искомый классификационный код.

IFCodeToStr(BASE_INT *Code,String StrCode)

Code – классификационный код в виде массива чисел. Code[0] – количество позиций кода;
StrCode – искомый классификационный код в виде перечня чисел, разделенных точкой.

int IFCompCode(BASE_INT *Code1,BASE_INT *Code2)

Code1, Code2 – сравниваемые классификационные коды (на полное совпадение).
Функция возвращает:
1 – коды совпали
0 – коды не совпали

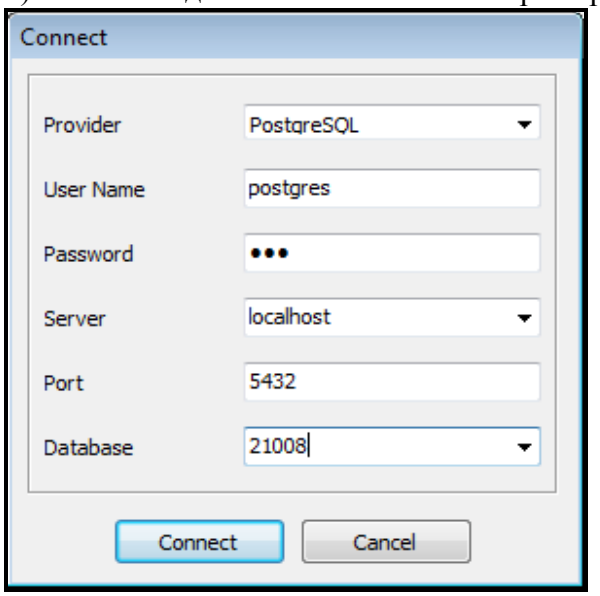
6. ПРИМЕРЫ

6.1. Коннект к базе данных

1) Коннект без диалогового окна выбора параметров базы данных

```
String BaseName="21008";
TIFPGdatabase *IFPG;
IFPG_PASPORT *PS;
IFPG=new TIFPGdatabase();
IFPG->SetPort=5432;              // Номер порта
IFPG->SetServer("localhost");    // Имя сервера
IFPG->SetPassword("123");        // Пароль
IFPG->SetUserName("postrgres"); // Имя пользователя
IFPG->LoginPrompt(false);        // Без диалога коннекта
if (IFPG->Open(BaseName))
{
 PS=IFPG->GetPasport();          // Чтение паспорта базы данных
 IFPG->Close();
}
delete IFPG;
```

2) Коннект с диалоговым окном выбора параметров базы данных




```
TIFPGdatabase *IFPG;
IFPG_PASPORT *PS;
IFPG=new TIFPGdatabase("");
IFPG->LoginPrompt(true);        // Без диалога коннекта
IFPG->SetDebug("c:\\temp");     // Режим отладочной печати в файл
if (IFPG->Open(BaseName))
{
 PS=IFPG->GetPasport();          // Чтение паспорта базы данных
 IFPG->Close();
}
delete IFPG;
```

6.2. Выборка всех объектов из базы данных

Файл Unit1.h

```
#ifndef Unit1H
#define Unit1H
#include <System.Classes.hpp>
#include <Vcl.Controls.hpp>
#include <Vcl.StdCtrls.hpp>
#include <Vcl.Forms.hpp>
#include "ifpg.h"
#include "if_def.h"
#include "ifpg_uni.h"
#include <Data.DB.hpp>
#include "UniDacVcl.hpp"
//---------------------------------------------------------------------------
class TForm1 : public TForm
{
__published:   // IDE-managed Components
 TButton *Button1;
 TListBox *ListBox1;
 void __fastcall Button1Click(TObject *Sender);
private:       // User declarations
public:        // User declarations
 __fastcall TForm1(TComponent* Owner);
};
//---------------------------------------------------------------------------
extern PACKAGE TForm1 *Form1;
//---------------------------------------------------------------------------
#endif
```

Файл Unit1.cpp

```
#include <vcl.h>
#pragma hdrstop
#include "Unit1.h"
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
//---------------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
 : TForm(Owner)
{
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Button1Click(TObject *Sender)
{
 TIFPGdatabase *IFPG;
 IFPG_PASPORT *PS;
 TIFPGobject *obj;
 IFPG_OBJECT *O;
 BASE_INT Code[10]={0};
 ListBox1->Clear();
 IFPG=new TIFPGdatabase();
 IFPG->SetServer("localhost");
 IFPG->SetUserName("postgres");
 IFPG->SetPassword("123");
 IFPG->SetVendorHome("d:\\baseutil");
 IFPG->SetPortDB(5432);
 IFPG->SetLoginPrompt(false);
 if (IFPG->Open("o3603") == true)
 {
  obj=new TIFPGobject(IFPG);
  O=obj->Open(Code,0,OPEN_TREE,OBJ_FULL);
  while (O != NULL)
  {
   ListBox1->Items->Add(String(O->Number));
   O=obj->Next();
  }
  delete obj;
  IFPG->Close();
```

```
 }
 else
 {
  ShowMessage("Database not opened");
 }
 delete IFPG;
}
```

ЛИТЕРАТУРА

1.Yu.G.Vasin, Yu.V.Yasakov. Object-Oriented Topological Management System of Spatially-Distributed Databases. ISSN 1054-6618. 2016, Vol. 26, No. 4, pp. 734-741. © Pleiades Publishing, Ltd., 2016. DOI: 10.1134/S1054661816040180

2.PostgreSQL 9.3 Documentation [Электронный ресурс]. – http://www.postgresql.org/docs/9.3/static/index.html

3.PostGIS 2.1 Manual [Электронный ресурс]. –http://postgis.net/docs/manual-2.13.

4.Ю.Г.Васин, Ю.В.Ясаков. Система управления базами пространственно-распределенных данных на основе объектно-ориентированной топологической модели. Свидетельство государственной регистрации программы для ЭВМ. № 2016616992. Дата государственной регистрации в Реестре программ для ЭВМ 23 июня 2016 г.

5. Ясаков Ю.В. Методическое пособие «СУБД для обработки ПРД». - Нижний Новгород, 2012 г. -34 стр.

6. Yu.G. Vasin, Yu.V. Yasakov. GIS Terra: A graphic database management system // Pattern recognition and image analysis. 2004. Vol. 14. No. 4. P. 579-586.

7. СУБД ГИС Терра. Руководство пользователя. Книга 1.

8.Embarcadero RAD Studio XE7. https://www.embarcaderot.com