Министерство науки и высшего образования Российской Федерации

УНИВЕРСИТЕТСКИЙ КОЛЛЕДЖ

# федерального государственного бюджетного образовательного учреждения

### высшего образования «Оренбургский государственный университет»

Отделение информационных технологий

Предметно-цикловая комиссия информационных технологий

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА**

### Специальность 09.02.03 Программирование в информационных системах

**Программа формирования графика движения подвижных составов**

Пояснительная записка

Университетский колледж ОГУ 09.02.03. 1220.032ПЗ

Председатель

предметно-цикловой комиссии

информационных технологий \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ М.Г. Таспаева

(подпись, дата)

Руководитель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Н.А. Уйманова

(подпись, дата)

Студент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ В.В. Кравец

(подпись, дата)

Оренбург 2020

**Аннотация**

Изм.

Лист

№ докум.

Подп.

Дата ДДата

Лист

3

*Университетский колледж ОГУ 09.02.03. 1220.032 ПЗ*

Разраб.

*Кравец В.В.*

Пров.

Уйманова Н.А.

Н.контр.

Уйманова Н.А.

Пред. ПЦК

*Таспаева М.Г.*

*Программа формирования графика движения подвижных составов*

Лит.

Листов

70

*16ПКС-2*

Выпускная квалификационная работа посвящена разработке программы построения графика движения подвижных составов.

Целью работы является упрощение контроля движения подвижных составов.

Предпосылкой для написания данной работы стало то, что в современном многофункциональном предприятии работникам не удобно чертить графики от руки.

Актуальность темы настоящей работы обусловлена тем, что программа на основе заполненных данных, составляет график, тем самым эффективно распределяя производственное время.

Структура работы представлена введением, шестью главами, заключением, списком литературы и приложениями.

Работа содержит 70 листов текста, 36 рисунков, 3 таблицы, 3 приложения.

**Содержание**

Введение 4

1. Техническое задание 6
   1. Введение технического задания 6
   2. Назначение разработки 6
   3. Требования к программе или программному изделию 6
   4. Требования к программной документации 7
2. Проектирование и разработка программного обеспечения 9
   1. Методология проектирования программного обеспечения и анализа бизнес-процессов 9
   2. Инструменты и технологии разработки программного обеспечения 11
   3. Инфологическая модель данных 18
3. Тестирование программного обеспечения 21
   1. Основные термины и их соотношение 21
   2. Статические методы устранения ошибок 22
   3. Функциональное тестирование 25
   4. Структурное тестирование 33
4. Руководство системного программиста 36
   1. Общие сведения о программе 36
   2. Структура программы 36
   3. Настройка программы 38
   4. Сообщения системному программисту 39
5. Руководство оператора 40
   1. Выполнение программы 40
   2. Сообщения оператору 44
6. Охрана труда и безопасность жизнедеятельности 45

Заключение 48

Список использованных источников 50

Приложение А (справочное) Входные и выходные документы 51

Приложение Б(рекомендуемое) Диаграммы моделирования и анализа бизнес-процессов 53

Приложение В(обязательное) Текст программы 53

**Введение**

Информатизация - это сложный социальный процесс, связанный со значительными изменениями в образе жизни населения. Он требует серьезных усилий на многих направлениях, включая ликвидацию компьютерной неграмотности, формирование культуры использования новых информационных технологий.

Движущей силой развития общества должно стать производство информационного, а не материального продукта. В информационном обществе изменяется не только производство, но и весь уклад жизни, система ценностей, возрастает значимость культурного досуга по отношению к материальным ценностям. В информационном обществе производятся и потребляются интеллект, знания, что приводит к увеличению доли умственного труда. От человека потребуется способность к творчеству, возрастает спрос на знания. Материальной и технологической базой информации общества, станут различного рода системы, на базе компьютерной техники и компьютерных сетей, информационной технологии, телекоммуникационной связи.

Информатизация общества - это организованный социально - экономический и научно-технический процесс создания оптимальных условий для удовлетворения информационных потребностей и реализации прав граждан, органов государственной власти, органов местного самоуправления, организаций, общественных объединений на основе формирования и использования информационных ресурсов

Информационное общество - общество, в котором большинство работающих занято производством, хранением, переработкой и реализацией информации, особенно высшей ее формы - знаний. Цель информатизации - улучшение качества жизни людей за счет повышения производительности и облегчения условий их труда.

Сервисные компании создают технологические решения для различных организаций-заказчиков, отдавая приоритет более сложным и долгосрочным проектам. При такой модели компания сама отвечает за весь процесс разработки, предлагая комплексные услуги, включая бизнес-анализ, менеджмент проектов, консалтинг. Все эти направления открыты для специалистов без технического образования.

В настоящее время существует огромное количество программ, позволяющих работать предприятиям с клиентами и дающих возможность сопровождать заказ на любом этапе его выполнения. Однако большинство из них или слишком сложны для рядового пользователя, или используют сложные функции, или напротив, многие их функции оказываются не нужны при формировании заказов. Зачастую приходится тратить много времени для того, чтобы обучить персонал, что приводит к значительным затратам времени. Все это говорит о необходимости создания программы, предоставляющей малому предприятию возможность выполнять необходимые ему действия более простым способом.

Объектом проектирования в выпускной квалификационной работе по профилю специальности является компания ГУП «Оренбургремдорстрой».

Программный продукт должен представлять собой БД для построения графика движения подвижных составов ГУП «Оренбургремдорстрой».

Цель программы – упростить контроль движения подвижных составов.

Задачи исследования:

* выполнить анализ предметной области и описать характеристику объекта информатизации;
* описать техническое задание на разработку;
* спроектировать и разработать программное обеспечение;
* составить руководство системного программиста.

Методология исследования:

* изучение теоретического материала;
* изучение теории программирования;
* изучение аналитического материала на предприятии.

## 1 Техническое задание

**1.1 Введение технического задания**

Программа для работы с базой данных (БД)– это специальное прикладное программное обеспечение, призванное ускорить и сделать более удобной работу с базами данных. Среди базовых возможностей, должна иметь функции удобного добавления, редактирования, удаления, поиска, фильтрации записей. Помимо вышеперечисленного функционала в продвинутых программах есть возможность создавать отчёты, документы, работать с медиа файлами, и т.д.

Один из главных принципов организации данных – построение взаимосвязей между всеми элементами, что и отличает базу данных от простого набора таблиц. Т.е. информация в таблицах реляционной базы данных должна быть соответствующим образом организована. Еще один важнейший принцип — нормализация таблиц, которая сводится к устранению недостатков структуры базы данных, приводящих к различным аномалиям и нарушениям целостности данных.

**1.2 Назначение разработки**

Назначением разработки является создание программы для работы с базой данных, содержащей информацию о станциях, подвижном составе и маршрутах, которая позволит облегчить процесс построения графика движения подвижных составов.

**1.3 Требования к программе или программному изделию**

1.3.1 Требования к функциональным характеристикам

Система должна обеспечивать возможность выполнения следующих функций:

* выполнение основных действий, связанных с БД (добавление, удаление, редактирование записей), сортировка, фильтрация и поиск записей в таблицах;
* программа должна иметь простой, но в то же время понятный и наглядный интерфейс, который не должен перегружать ресурсы компьютера;
* вывод на печать необходимых записей из таблиц;
* вывод на печать графика;
* построение графика на основе данных из БД;
* программа должна иметь возможность выхода из какой-либо процедуры, не нарушая ход программы;
* пользователь должен иметь возможность видеть выполняемые им действия и полученный результат;
* программа не должна занимать большой объем памяти и не должна требовать установки на жесткий диск компьютера;

1.3.2 Требования к надежности

При некорректных действия пользователя программа не должна приостанавливать работу, а соответствующим образом обрабатывать эти действия, поддерживать нормальный и корректный ход программы. Пользователь должен работать только с активным окном и, только закрыв его, он может перейти к работе предыдущим окном.

Все устройства на ПК, где осуществляется функционирование программного продукта, должны находиться в рабочем состоянии и не должны иметь каких-либо физических изломов либо логических ошибок. Должны быть правильно установлены соответствующие драйвера для всех устройств, входящих в состав компьютера. Поддержание функций операционной системы, где работает программный продукт. Проверка правильности вводимых и выводимых данных.

При соблюдении всех этих требований не совсем гарантирована безопасная и корректная работа программы.

Создаваемый программный продукт можно хранить и транспортировать на любом, удобном для пользователя, носителя информации.

1.3.3 Требования к составу и параметрам технических средств

Система должна работать на IBM совместимых персональных компьютерах. Минимальная конфигурация:

* процессор: Intel® Pentium® 2.00 GHz;
* ОЗУ: 1024 МБ;
* видеокарта: любой видеоадаптер, способный работать с Windows;
* операционная система: WindowsXP / Vista / 7 / 8 / 8.1 / 10.

1.3.4 Требования к информационной и программной совместимости

Система должна обеспечивать удобный для пользователей программы интерфейс, удовлетворяющий следующим требованиям:

* единый стиль оформления для пользовательских интерфейсов;
* должна быть удобная, интуитивно понятная навигация в интерфейсе пользователя;
* должна быть установлена программа InterBase(6.5).

**1.4 Требования к программной документации**

Программная документация включает:

* + техническое задание – содержит требования, предъявляемые к программе и программному изделию, необходимые стадии и сроки разработки, виды испытаний;
  + тексты программ – записи программ с необходимыми комментариями;
  + руководство системного программиста – эксплуатационный программный документ на программное обеспечение;
  + руководство пользователя – обеспечение пользователям возможности самостоятельно решать все основные задачи, на которые нацелена программа.

**2 Проектирование программного обеспечения**

**2.1Методология проектирования программного обеспечения и анализа бизнес-процессов**

В настоящее время для описания бизнес-процессов используется несколько методологий. К числу наиболее распространенных относятся методологии создания моделей структурного типа, методологии описания потоков работ (Work Flow\*) и методологии описания потоков данных (Data Flow Modeling).

Давно известная и широко используемая методология структурного описания бизнес-процессов — стандарт США IDEF0. Подход IDEF0 разработан на основе методологии структурного анализа и проектирования SADT в 1963 году. С момента разработки стандарт не претерпел существенных изменений. В настоящее время развитие методологии IDEF0 сопряжено с развитием поддерживающих ее инструментов — программных продуктов для моделирования бизнес-процессов (например, Casewise, Business Studio\*\* и т. д.).

Методология IDEF0 предоставляет аналитику прекрасные возможности для описания бизнеса организации на верхнем уровне с акцентом на управление процессами. Нотация позволяет отражать в модели процесса обратные связи различного типа: по информации, по управлению, движение материальных ресурсов. Продуманные механизмы декомпозиции модели процесса в IDEF0 существенно упрощают работу аналитика. Еще обратите внимание, что модели в нотации IDEF0 являются структурными и предназначены для описания бизнеса на верхнем уровне. Их основное преимущество, на наш взгляд, состоит в возможности создавать модель верхнего уровня и описывать управление процессами организации.

Вторая важнейшая методология описания процессов — Work Flow Modeling\*. Существует несколько методологий, в которых можно формировать модели типа Work Flow. Одна из первых методологий такого типа — IDEF3 — предназначена для описания рабочих процессов, или, иными словами, потоков работ. Методология описания IDEF3 очень близка к алгоритмическим методам построения схем процессов и стандартным средствам построения блок-схем (см., например, построение блок-схемы в программе MS Word). Следует отметить, что спецификация IDEF3 включает два существенно различающихся метода описания процессов. В данной книге мы рассмотрим получивший наибольшее распространение метод. Основа методологии IDEF3 состоит в построении моделей процессов по принципу последовательно выполняемых во времени работ (функций, операций). Можно обоснованно утверждать, что принципы, заложенные в IDEF3, лежат в основе многих современных подходов к созданию моделей типа Work Flow, в том числе методологий ARIS eEPC и BPMN (Business Process Model and Notation — нотация и модель бизнес-процессов).

Именно поэтому, несмотря на то что на момент выхода данного издания книги нотация IDEF3 не поддерживается основными программными продуктами, методология описания потоков работ будет рассматриваться на примерах в данной нотации.

Еще одна группа методологий, активно используемых на практике, — нотация DFD (Data Flow Diagramming). Эта нотация предназначена для описания потоков данных. Она позволяет отразить последовательность работ, выполняемых по ходу процесса, и потоки информации, циркулирующие между этими работами. Кроме того, нотация DFD описывает потоки документов (документооборот) и материальных ресурсов (например, движение материалов от одной работы к другой).

Методология DFD может эффективно использоваться для описания процессов при внедрении процессного подхода к управлению организацией, так как позволяет максимально снизить субъективность описания бизнес-процессов. Схемы процессов в DFD позволяют выявить основные потоки данных в организации. Это важно для последующего создания моделей структуры данных и разработки требований к информационной системе организации.

Одна из современных методологий описания процессов — ARIS (Architecture of Integrated Information Systems — архитектура интегрированных информационных систем). Методология была разработана немецкой компанией IDS Scheer AG. Основа методологии состоит в том, что любая организация рассматривается как сложная система, описание которой строится из четырех основных групп моделей: моделей организационной структуры, моделей функций, моделей данных и объединяющих эти три группы моделей бизнес-процессов.

Архитектура ARIS включает большое количество типов моделей, использующих различные типы графических объектов и различные типы связей для построения разносторонних моделей организации. Однако следует подчеркнуть, что на практике используется очень ограниченное число нотаций архитектуры ARIS. К числу наиболее практически важных относится основная нотация eEPC, что означает «расширенная цепочка процесса, управляемого событиями».

По сути, данная нотация действительно является расширением методологии IDEF3 за счет использования понятия «событие» (Event).

Кроме нотации eEPC, ARIS предоставляет аналитику и другие средства описания процессов организации. Сегодня аналогичными возможностями обладают программные продукты Casewise и Business Studio. Даже в MS Visio достаточно много возможностей для создания моделей бизнес-процессов.

Отметим, что в последние годы существенное развитие получила методология BPMN. Есть все основания полагать, что со временем она вытеснит нотацию ARIS eEPC с рынка, так как все больше программных продуктов позволяют не только автоматизировать процессы с использованием нотации BPMN, но и разрабатывать комплексную систему процессов организации.

Помимо указанных выше методологий, существуют и другие методологии, предложенные различными частными фирмами-производителями программных продуктов.

В заключение краткого описания существующих методологий следует отметить, что бизнес-процессы предприятия могут быть описаны при помощи стандартных блок-схем. По сути, блок-схемы основаны на методологии нотации IDEF3, но при этом они содержат некоторые дополнительные специальные графические объекты.

Использование этих графических объектов позволяет сделать блок-схемы процессов более наглядными и понятными для исполнителей [9].

**2.2 Инструменты и технологии разработки программного обеспечения**

Для проектирования программного обеспечения и базы данных использовался язык программирования SQL.

SQL (Structured Query Language — «язык структурированных запросов») — декларативный язык программирования, применяемый для создания, модификации и управления данными в реляционной базе данных, управляемой соответствующей системой управления базами данных.

Является, прежде всего, информационно-логическим языком, предназначенным для описания, изменения и извлечения данных, хранимых в реляционных базах данных. SQL считается языком программирования, в общем случае (без ряда современных расширений) не является тьюринг-полным, но вместе с тем стандарт языка спецификацией SQL/PSM предусматривает возможность его процедурных расширений.

Изначально SQL был основным способом работы пользователя с базой данных и позволял выполнять следующий набор операций:

* создание в базе данных новой таблицы;
* добавление в таблицу новых записей;
* изменение записей;
* удаление записей;
* выборка записей из одной или нескольких таблиц (в соответствии с заданным условием);
* изменение структур таблиц.

Со временем SQL усложнился — обогатился новыми конструкциями, обеспечил возможность описания и управления новыми хранимыми объектами (например, индексы, представления, триггеры и хранимые процедуры) — и стал приобретать черты, свойственные языкам программирования.

При всех своих изменениях SQL остаётся самым распространённым лингвистическим средством для взаимодействия прикладного программного обеспечения с базами данных. В то же время современные СУБД, а также информационные системы, использующие СУБД, предоставляют пользователю развитые средства визуального построения запросов.[6]

В начале 1970-х годов в одной из исследовательских лабораторий компании IBM была разработана экспериментальная реляционная СУБД IBM System R, для которой затем был создан специальный язык SEQUEL, позволявший относительно просто управлять данными в этой СУБД. Аббревиатура SEQUEL расшифровывалась как Structured English QUEry Language — «структурированный английский язык запросов». Позже по юридическим соображениям язык SEQUEL был переименован в SQL. Когда в 1986 году первый стандарт языка SQL был принят ANSI (American National Standards Institute), официальным произношением стало [,es kju:' el] — эс-кью-эл. Несмотря на это, англоязычные специалисты зачастую продолжают читать SQL как сиквел (по-русски часто говорят «эс-ку-эль»).

Целью разработки было создание простого непроцедурного языка, которым мог воспользоваться любой пользователь, даже не имеющий навыков программирования. Собственно разработкой языка запросов занимались Дональд Чэмбэрлин (Donald D. Chamberlin) и Рэй Бойс (Ray Boyce). Пэт Селинджер (Pat Selinger) занималась разработкой стоимостного оптимизатора (cost-based optimizer), Рэймонд Лори (Raymond Lorie) занимался компилятором запросов.

SEQUEL был не единственным языком подобного назначения. В Калифорнийском Университете Беркли была разработана некоммерческая СУБД Ingres (являвшаяся дальним прародителем популярной сейчас некоммерческой СУБД PostgreSQL), которая являлась реляционной СУБД, но использовала свой собственный язык QUEL, который, не выдержал конкуренции по количеству поддерживающих его СУБД по сравнению с языком SQL. В качестве альтернативного подхода для обеспечения пользовательского доступа к базам данных также рассматривался появившийся в 1970-е годы метод QBE, впоследствии в видоизменённой форме появившийся в ряде интегрированных сред управления данными, но так и не заменивший, а лишь дополнивший SQL.

Первыми СУБД, поддерживающими новый язык, стали в 1979 году Oracle V2 для машин VAX от компании Relational Software Inc. (впоследствии ставшей компанией Oracle) и System/38 от IBM, основанная на System/R.

Поскольку к началу 1980-х годов существовало несколько вариантов СУБД от разных производителей, причём каждый из них обладал собственной реализацией языка запросов, было принято решение разработать стандарт языка, который будет гарантировать переносимость ПО с одной СУБД на другую (при условии, что они будут поддерживать этот стандарт).

В 1983 году Международная организация по стандартизации (ISO) и Американский национальный институт стандартов (ANSI) приступили к разработке стандарта языка SQL. По прошествии множества консультаций и отклонения нескольких предварительных вариантов, в 1986 году ANSI представил свою первую версию стандарта, описанную в документе ANSI X3.135-1986 под названием «Database Language SQL» (Язык баз данных SQL). Неофициально этот стандарт SQL-86 получил название SQL1.

Год спустя была завершена работа над версией стандарта ISO 9075-1987 под тем же названием. Разработка этого стандарта велась под патронажем Технического Комитета TC97 (англ. Technical Committee TC97), областью деятельности которого являлись процессы вычисления и обработки информации (англ. Computing and Information Processing). Именно его подразделение, именуемое как Подкомитет SC21 (англ. Subcommittee SC21), курировало разработку стандарта, что стало залогом идентичности стандартов ISO и ANSI для SQL1 (SQL-86).

Стандарт SQL1 разделялся на два уровня. Первый уровень представлял собой подмножество второго уровня, описывавшего весь документ в целом. То есть, такая структура предусматривала, что не все спецификации стандарта SQL1 будут относиться к Уровню 1. Тем самым поставщик, заявлявший о поддержке данного стандарта, должен был заявлять об уровне, которому соответствует его реализация языка SQL. Это значительно облегчило принятие и поддержку стандарта, поскольку производители могли реализовывать его поддержку в два этапа.

Со временем к стандарту накопилось несколько замечаний и пожеланий, особенно с точки зрения обеспечения целостности и корректности данных, в результате чего в 1989 году данный стандарт был расширен, получив название SQL89. В частности, в него была добавлена концепция первичного и внешнего ключей. ISO-версия документа получила название ISO 9075:1989 «Database Language SQL with Integrity Enhancements» (Язык баз данных SQL с добавлением контроля целостности). Параллельно была закончена и ANSI-версия.

Сразу после завершения работы над стандартом SQL1 в 1987 году была начата работа над новой версией стандарта, который должен был заменить стандарт SQL89, получив название SQL2, поскольку дата принятия документа на тот момент была неизвестна. Таким образом, фактически SQL89 и SQL2 разрабатывались параллельно. Новая версия стандарта была принята в 1992 году, заменив стандарт SQL89. Новый стандарт, озаглавленный как SQL92, представлял собой по сути расширение стандарта SQL1, включив в себя множество дополнений, имевшихся в предыдущих версиях инструкций.

Как и SQL1, SQL92 также был разделён на несколько уровней, однако, во-первых, число уровней было увеличено с двух до трёх, а во-вторых, они получили названия вместо порядковых цифр: начальный (англ. entry), средний (англ. intermediate), полный (англ. full).

Уровень «полный», как и Уровень 2 в SQL1 подразумевал весь стандарт целиком. Уровень «начальный» представлял собой подмножество уровня «средний», в свою очередь, представлявшего собой подмножество уровня «полный». Уровень «начальный» был сравним с Уровнем 2 стандарта SQL1, но спецификации этого уровня были несколько расширены. Таким образом, цепочка включений уровней стандартов выглядела примерно следующим образом: SQL1 Уровень 1 → SQL1 Уровень 2 → SQL92 «Начальный» → SQL92 «Средний» → SQL92 «Полный».

После принятия стандарта SQL92 к нему были добавлены ещё несколько документов, расширявших функциональность языка. Так, в 1995 году был принят стандарт SQL/CLI (Call Level Interface, интерфейс уровня вызовов), впоследствии переименованный в CLI95. На следующий год был принят стандарт SQL/PSM (Persistent Stored Modules, постоянно хранимые модули), получивший название PSM-96.[7]

Следующим стандартом стал SQL:1999 (SQL3). В настоящее время действует стандарт, принятый в 2003 году (SQL:2003) с небольшими модификациями, внесёнными позже (SQL:2008).

По традиции, как и со многими стандартами в IT-индустрии, с языком SQL возникла проблема: на каком-то этапе многие производители использующего SQL программного обеспечения решили, что функциональность в текущей (на тот момент времени) версии стандарта недостаточна, и её желательно расширить. В результате у разных производителей систем управления базами данных (СУБД) в ходу разные диалекты SQL, в общем случае между собой несовместимые.

До 1996 года вопросами соответствия коммерческих реализаций SQL стандарту занимался в основном Национальный институт стандартов и технологий (NIST), который и устанавливал уровень соответствия стандарту. Поздне́е подразделение, занимавшееся СУБД, было расформировано, и на текущий момент все усилия по проверке СУБД на соответствие стандарту ложатся на её производителя.

Впервые понятие «уровня соответствия» было предложено в стандарте SQL-92. А именно, ANSI и NIST определяли четыре уровня соответствия реализации этому стандарту:

* Entry (базовый);
* Transitional (переходный) — проверку на соответствие этому уровню проводил только NIST;
* Intermediate (промежуточный);
* Full (полный). [14]

Легко можно понять, что каждый последующий уровень соответствия заведомо подразумевал соответствие предыдущему уровню. Далее, согласно данной «лесенке» стандартов любая СУБД, которая соответствовала уровню Entry, могла заявлять себя как «SQL-92 compliant» («совместимая с SQL-92»), хотя на самом деле переносимость и соответствие стандарту ограничивалось набором возможностей, входящих в этот уровень.

Положение изменилось с введением стандарта SQL:1999. Отныне стандарт приобрёл модульную структуру — основная часть стандарта была вынесена в раздел «SQL/Foundation», все остальные были выведены в отдельные модули. Соответственно, остался только один уровень совместимости — Core, что означало поддержку этой основной части. Поддержка остальных возможностей оставлена на усмотрение производителей СУБД. Аналогичное положение имело место и с последующими версиями стандарта.

Для проектирования программного обеспечения использовался язык программирования Delphi 7[1].

Delphi - императивный, структурированный, объектно-ориентированный, высокоуровневый язык программирования со строгой статической типизацией переменных. Основная область использования — написание прикладного программного обеспечения[2].

Этот язык программирования является диалектом языка Object Pascal. Изначально язык Object Pascal относился к несколько другому языку, который был разработан в фирме Apple в 1986 году группой Ларри Теслера. Однако, начиная с Delphi 7 в официальных документах компании Borland название Delphi стало использоваться для обозначения языка, ранее известного как Object Pascal.

Изначально среда разработки Delphi была предназначена исключительно для разработки приложений Microsoft Windows, затем был реализован вариант для платформ Linux (под торговой маркой Kylix), однако после выпуска в 2002 году Kylix 3 его разработка была прекращена, и вскоре было объявлено о поддержке Microsoft .NET, которая, в свою очередь, была прекращена с выходом Delphi 2007.[3]

В настоящее время, наряду с поддержкой разработки 32 и 64-разрядных программ для Windows, реализована возможность создавать приложения для Apple Mac OS X (начиная с Embarcadero Delphi XE2), iOS (включая симулятор, начиная с XE4 посредством собственного компилятора), Google Android (начиная с Delphi XE5), а также Linux Server x64 (начиная с версии 10.2 Tokyo).

Независимая, сторонняя реализация среды разработки проектом Lazarus (Free Pascal, в случае компиляции в режиме совместимости с Delphi) позволяет использовать его для создания приложений на Delphi для таких платформ, как Linux, Mac OS X и Windows CE.

Также предпринимались попытки использования языка в проектах GNU (например, Notepad GNU) и написания компилятора для GCC (GNU Pascal).

Для проектирования и документирования базы данных использовалась программа Erwin Data Modeler[4].

Erwin Data Modeler - это компьютерная программа для проектирования и документирования баз данных. Модели данных помогают визуализировать структуру данных, обеспечивая эффективный процесс организации, управления и администрирования таких аспектов деятельности предприятия, как уровень сложности данных, технологий баз данных и среды развертывания. Изначально разработанный компанией Logic Works, erwin был приобретен рядом компаний, прежде чем был выделен частной инвестиционной фирмой Parallax Capital Partners, которая приобрела и объединила его в качестве отдельной организации, erwin, Inc., с генеральным директором Адамом Famularo.

Движок программного обеспечения основан на методе IDEF1X, хотя теперь он также поддерживает диаграммы отображаемые с альтернативной инженерной нотацией для информационных технологий, а также нотацию для размерного моделирования.

ERwin был создан компанией Logic Works, которая базировалась в Принстоне, штат Нью-Джерси. В мае 1993 года компания Logic Works выпустила ERwin / ERX версию инструмента, предназначенную для работы совместно с PowerBuilder̩. Модели баз данных, созданные с использованием ERwin, могли быть переведены в программное обеспечение, созданное с помощью интегрированной среды разработки PowerBuilder (IDE).

Erwin также разработалa программную платформу erwin EDGE с EDGE в качестве аббревиатуры от «опыта управления корпоративными данными», намереваясь использовать управление данными для анализа данных, чтобы помочь в достижении организационных целей. В феврале 2018 года erwin выпустилa свой первый отчет о состоянии управления данными с UBM. В августе 2018 года erwin приобрелa компанию по управлению метаданными и управлению данными AnalytiX DS.

По своей сути, erwin имеет инструмент для разработки программного обеспечения (или инструмент CASE). Пользователи могут использовать erwin Data Modeler как способ создания концептуальной модели данных и создания логической модели данных, которая не зависит от конкретной технологии базы данных. Эта схематическая модель может быть использована для создания физической модели данных. Затем пользователи могут направить инженерный язык определения данных, необходимый для создания схемы для ряда систем управления базами данных. Программное обеспечение включает в себя функции для графической модификации модели, в том числе диалоговые окна для указания количества взаимосвязей сущностей, ограничений базы данных, индексов и уникальности данных. Эрвин поддерживает три языка моделирования данных: IDEF1X , вариант разработки информационных технологий, разработанный Джеймсом Мартином, и форму обозначения размерного моделирования .

Программное обеспечение также позволяет пользователям создавать модели данных путем обратного проектирования уже существующих баз данных, основанных на нескольких различных форматах. Еще одна включенная функция — это способность Эрвина создавать многократно используемые стандарты проектирования: «включая стандарты именования, стандарты типов данных, шаблоны моделей и т. Д.». Программное обеспечение включает в себя несколько функций для изменения способа отображения модели данных, включая параметры для нескольких цветов, шрифты, схемы, тематические области и макеты.

Функция полного сравнения erwin позволяет пользователю сравнивать две версии модели, отображает различия и позволяет объединять и обновлять данные в любом направлении. По состоянию на март 2016 года в комплект программного обеспечения также входит собственный конструктор отчетов. Обновление erwin DM 2018 включало Netezza, MySQL 8.x, PostgreSQL 10.4 и Hive; модель подсчитывает отчеты; и поддержка PII. Обновление 2019 года включало DB2 z / OS v12, SQL Server 2017, Teradata v16.20 и PostgreSQL 11.2 в дополнение к улучшениям отчетности, таким как определяемые пользователем свойства и фильтры.

Для проектирования базы данных использовалась программа InterBase(6.5).

Interbase — реляционная система управления базами данных, разрабатывающаяся компанией Embarcadero, появилась в середине 1980-х годов, принадлежала самостоятельной одноимённой компании, Ashton-Tate, Borland. Код Interbase стал основой свободно распространяемой системы управления базами данных Firebird.

Основой Interbase стал самостоятельный проект сотрудника DEC Джима Старки (Jim Starkey), работавшего в то время над сетевой СУБД Datatrive, в котором он реализовал реляционную систему с поддержкой многопользовательской работы. Первая версия проекта создана в 1984 году, он получил название JRD (англ. Jim’s relational database). По-видимому, за основу была взята архитектура Rdb, так как Старки был одним из разработчиков этой СУБД в DEC. В 1985 году Старки, его жена Анн Харрисон и Дон Депальма (Don Depalma) основали компанию Groton Database Systems (файлы базы данных Interbase долгое время сохраняли традиционное расширение gdb — от Groton Database).

После ряда перепродаж и изменения наименования компании в Interbase Software Corporation в 1986 году был выпущен InterBase 2. Система использовалась в основном во встраиваемых системах военного и специального назначения — например, она до сих пор используется в системе управления американской системы залпового огня MLPRS, а также в компании Боинг для специальных расчётов жёсткости крыльев.

В 1988 году компания Ashton-Tate приобретает 51 % акций Interbase, а в 1991 году Borland покупает Ashton-Tate. В том же году выходит InterBase 3.

Большую популярность Interbase приобрёл с выходом версии 4 в 1994 году, для того времени это была очень мощная СУБД, конкурировавшая по возможностям и производительности с MS SQL (6.5) и Sybase ASE.

В 1997 году выходит Interbase 5, а в 1998 году Interbase 5.1.1 был включён в дистрибутив Delphi 4, что в значительной мере предопределило его популярность среди разработчиков на Delphi и C++ Builder.[5]

В конце 1999 года 3 ключевых сотрудника Interbase - Билл Карвин (Bill Karwin), Пол Бич (Paul Beach) и Уэйн Остигай (Wayne Ostiguy) увольняются из Borland, в интернет-конференциях начинаются волнения, австралийская активистка Хелен Борри (Helen Borrie) создает список рассылки под названием «Спасём Interbase», с целью не допустить закрытия проекта, позднее образуется инициативная группа IBDI (англ. Interbase developer’s initiative) для защиты разработчиков-пользователей Interbase, основатели — Хелен Борри, Джейсон Уортон (Jason Wharton) и Дэлтон Кэлфорд (Dalton Calford).

В 2000 годуBorland выпустила версию InterBase 6.0 в открытых кодах — InterBase 6 OpenSourceEdition, под специальной InterbasePublicLicense (IPL). К релизу не было выпущено ни документации, ни системы тестирования, ни системы сборки проекта, некоторые исходные коды не компилировались. Фактически Borland в тот момент отказался от дальнейшего развития системы.

31 июля 2000 года инициативная группа, отчаявшись добиться от Borland поддержки или хотя бы внятной позиции, скопировала исходные коды Interbase 6 и образовала проект Firebird — свободный проект, основанный на кодах Interbase 6 Open Source, активно развивающийся независимо: начиная с версии Interbase 7.5 (2007) и Firebird 1.5 СУБД далеки от полной совместимости. В 2001 году благодаря открытию исходных кодов в СУБД был обнаружен бэкдор, позволявший получить доступ к любому экземпляру СУБД, существовавший с 1994 года.

В 2001 году Borland вновь решила развивать InterBase, директором подразделения был назначен Джон Артур (John Arthur), а ведущим разработчиком — Чарли Каро (Charlie Caro), были выпущены проприетарные версии 6.0 и 6.5, а несколько позже официально полностью была прекращена поддержка Interbase Open Source Edition.[11]

В 2006 году Borland выделила подразделения по созданию средства разработки и Interbase в дочернюю фирму CodeGear, разработка которую продала Embarcadero в 2008 году. В 2009 году вышла версия Interbase 2009 , в которой появилась поддержка юникода и шифрование AES/DES.

Interbase обладает целым рядом преимуществ, выгодно отличающих его от остальных СУБД:

* обновляемые представления View;
* двухфазное подтверждение транзакций;
* эффективный механизм триггеров;
* серверная обработка BLOB-полей (BLOB-filters);
* события (уведомления);
* шифрование сетевого трафика, базы данных, файлов бекапа и отдельных столбцов БД [13].

**2.3 Инфологическая модель данных**

ER-модель (от англ. entity-relationship model, модель «сущность — связь») — модель данных, позволяющая описывать концептуальные схемы предметной области.

ER-модель используется при высокоуровневом (концептуальном) проектировании баз данных. С её помощью можно выделить ключевые сущности и обозначить связи, которые могут устанавливаться между этими сущностями.

Во время проектирования баз данных происходит преобразование ER-модели в конкретную схему базы данных на основе выбранной модели данных (реляционной, объектной, сетевой или др.).

ER-модель представляет собой формальную конструкцию, которая сама по себе не предписывает никаких графических средств её визуализации. В качестве стандартной графической нотации, с помощью которой можно визуализировать ER-модель, была предложена диаграмма «сущность-связь» (англ. entity-relationship diagram, ERD, ER-диаграмма).

Понятия «ER-модель» и «ER-диаграмма» часто не различают, хотя для визуализации ER-моделей могут быть использованы и другие графические нотации, либо визуализация может вообще не применяться (например, использоваться текстовое описание).

Модель была предложена в 1976 году Питером Ченом, им же предложена и самая популярная графическая нотация для модели.

На рисунке 1 изображена ER-диаграмма БД.

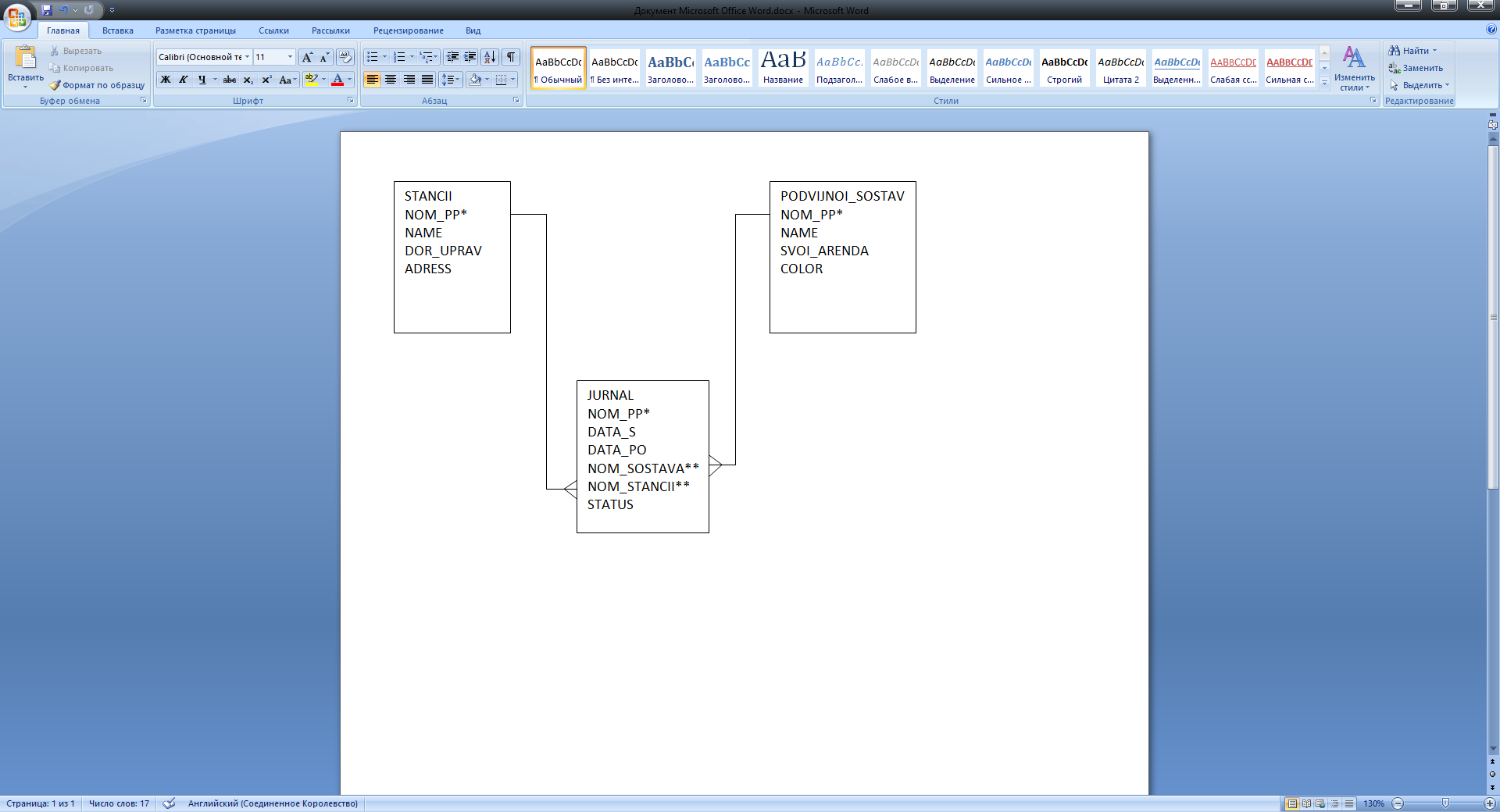


Рисунок 1 – ER-диаграмма предметной области

Логическая структура данных представлена в таблицах 1-3.

Таблица 1 – Сущность «Станции»

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Перевозки | | | | | | |
| Поле | | Признак ключа | Формат поля | | | |
| Имя (обозначение) | Полное наименование | Тип | Длина | Точность | N/  NN |
| NOM\_PP | NOM\_PP | PK | Счетчик | - | - | NN |
| NAME | Название |  | Текстовый | 20 | - | NN |
| DOR\_UPRAV | Дорожное управление |  | Числовой | - | - | NN |
| ADRESS | Адрес |  | Числовой | - | - | NN |

Таблица 2 – Сущность «Составы»

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Города | | | | | | |
| Поле | | Признак ключа | Формат поля | | | |
| Имя (обозначение) | Полное наименование | Тип | Длина | Точность | N/  NN |
| NOM\_PP | NOM\_PP | PK | Счетчик | - | - | NN |
| NAME | Название |  | Текстовый | 20 | - | NN |
| Имя (обозначение) | Полное наименование | Признак ключа | Тип | Длина | Точность | N/  NN |
| SVOI\_ARENDA | Свой/аренда |  | Числовой | - | - | NN |
| COLOR | Цвет |  | Числовой | - | - | NN |

Таблица 3 – Сущность «Журнал»

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Груз | | | | | | |
| Поле | | Признак ключа | Формат поля | | | |
| Имя (обозначение) | Полное наименование | Тип | Длина | Точность | N/  NN |
| NOM\_PP | NOM\_PP | PK | Счетчик | - | - | NN |
| DATA\_S | Дата с |  | Дата | - | - | NN |
| DATA\_PO | Дата по |  | Дата | - | - | NN |
| NOM\_SOSTAVA | Название состава | FK | Числовой | - | - | NN |
| NOM\_STANCII | Название станции | FK | Числовой | - | - | NN |
| STATUS | Состояние |  | Числовой | - | - | NN |

**3 Тестирование программного обеспечения**

**3.1 Основные термины и их соотношение**

Тестирование — процесс выполнения программы с целью обнаружения ошибок. Шаги процесса задаются тестами. Другое название теста — тестовый вариант. Каждый тест определяет:

* набор исходных данных и условий для запуска программы;
* набор ожидаемых результатов работы программы.

Полную проверку программы гарантирует исчерпывающее тестирование. Оно требует проверить все наборы исходных данных, все варианты их обработки и включает большое количество тестовых вариантов. Исчерпывающее тестирование в большинстве случаев невозможно из-за ресурсных ограничений (прежде всего, ограничений по времени).

Хорошим считают тестовый вариант с высокой вероятностью обнаружения ещё не раскрытой ошибки. Успешным называют тест, который обнаруживает до сих пор не раскрытую ошибку.

Целью проектирования тестовых вариантов является систематическое обнаружение различных классов ошибок при минимальных затратах времени и стоимости.

Тестирование обеспечивает:

* обнаружение ошибок;
* демонстрацию соответствия функций программы её назначению;
* демонстрацию реализации требований к характеристикам программы;
* отображение надёжности как индикатора качества программы.

Тестирование не может показать отсутствия дефектов (оно может показать только присутствие дефектов).

Условно процесс тестирования можно представить как результат взаимодействия нескольких информационных потоков.

Модель тестирования выглядит следующим образом. На вход процесса тестирования поступают три потока:

* текст программы;
* исходные данные для запуска программы;
* ожидаемые результаты.

Выполняются все тесты, полученные результаты оцениваются сравнением реальных результатов тестов с ожидаемыми. При обнаружении несовпадения фиксируется ошибка — начинается отладка. Процесс отладки непредсказуем по времени. На поиск места дефекта и исправление может потребоваться час, день, месяц. Неопределённость в отладке приводит к большим трудностям в планировании действий.

После сбора и оценивания результатов тестирования начинается отображение качества и надёжности ПО. Если регулярно встречаются серьёзные ошибки, требующие проектных изменений, то качество и надёжность ПО подозрительны, констатируется необходимость усиления тестирования. С другой стороны, если функции ПО реализованы правильно, а обнаруженные ошибки легко исправляются, может быть сделан один из двух выводов:

* качество и надёжность ПО удовлетворительны;
* тесты не способны обнаруживать серьёзные ошибки.

В конечном счёте, если тесты не обнаруживают ошибок, появляется сомнение в том, что тестовые варианты достаточно продуманы и что в ПО нет скрытых ошибок. Такие ошибки будут, в конечном итоге, обнаруживаться пользователями и корректироваться разработчиком на этапе сопровождения, когда стоимость исправления значительно возрастает (по сравнению с этапом разработки.)

Результаты, накопленные в ходе тестирования, оцениваются на основании различных моделей надёжности ПО, выполняющих прогноз надёжности по реальным данным об интенсивности потока ошибок.

**3.2 Статические методы устранения ошибок**

Эти методы основываются на анализе кода и документации программы.

Первым статическим методом устранения ошибок является «проверка за столом».

Этот метод наиболее традиционен, но при этом совершенно необходим, хотя такая проверка не вызывает восторга и ее иногда стараются избежать.

Термин «проверка за столом» характеризует неавтоматизированную деятельность, к которой наиболее часто относят:

* анализ текста программы на наличие ошибок;
* выполнение программы вручную («сухую прокрутку»).

Коллективная проверка (контроль программы, сквозной просмотр)

Это процесс, при котором бригада программистов тщательно просматривает программу или ее часть в порядке инспекции. Программа проверяется с разных точек зрения с целью поиска ошибок, ускользнувших при менее широком подходе.

Коллективную проверку лучше проводить после исключения синтаксических ошибок программы и, возможно, после предварительного тестирования. Предпочтительно, чтобы коллектив проверяющих включал трех-четырех разработчиков (не руководство!), заинтересованных в качестве разрабатываемой программы (например, проверяющие разрабатывали бы программу, взаимодействующую с проверяемой).

Последовательность действий по проверке вырабатывается самими участниками.

Пример метода «проверка за столом» показан на рисунке 2.

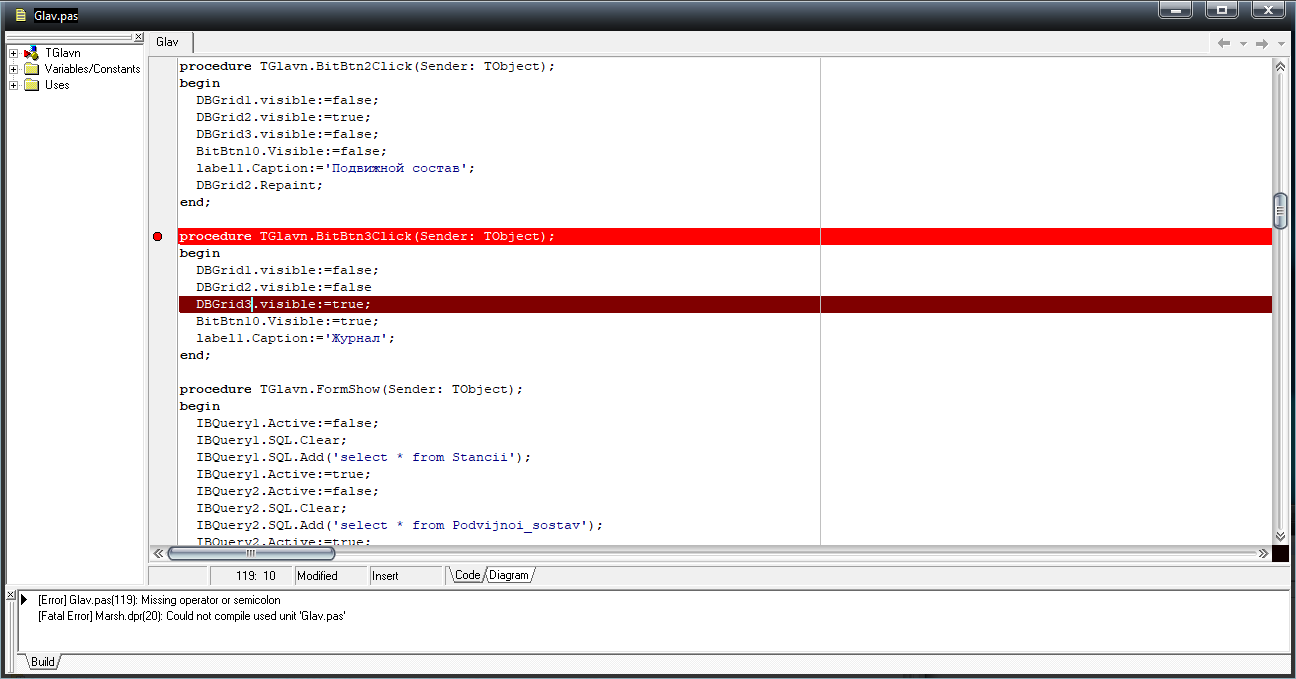


Рисунок 2 – Пример метода «проверка за столом»

Устранение данной ошибки показано на рисунке 3.

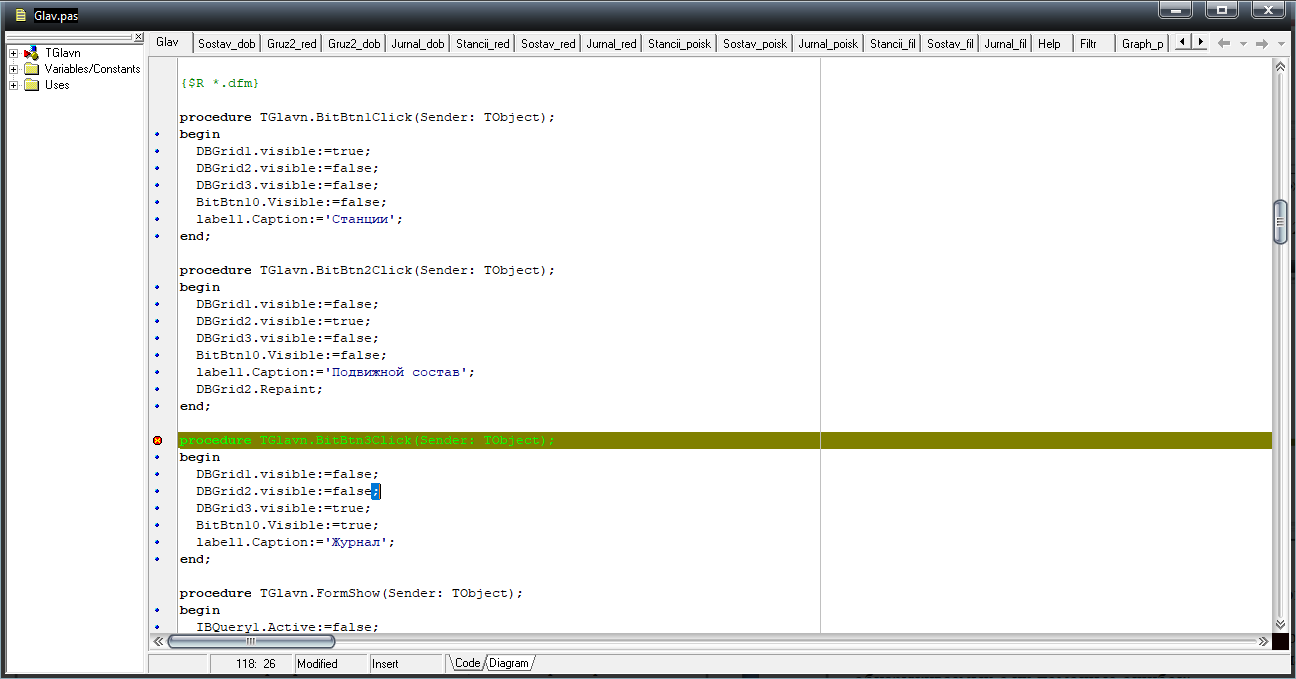


Рисунок 3 – Устранение ошибки

Вторым статическим методом устранения ошибок является «структурный анализ».

Это автоматизированный анализ текста программы с помощью специальных программ – структурных анализаторов. Примеры обнаруживаемых с их помощью ошибок:

* переменные не описаны или описаны неправильно;
* переменные используются прежде, чем им было присвоено значение, или присвоенное значение никогда не используется;
* используются недопустимые языковые формы;
* неправильное вложение конструкций;
* несоответствие формального и фактического параметров процедур;
* невыполнимые условия; потенциально бесконечные циклы и др.

Как самостоятельное средство поиска ошибок такого рода программы широкого распространения не получили, однако большинство перечисленных функций включено в современные компиляторы.

Пример метода «структурный анализ» показан на рисунке 4.

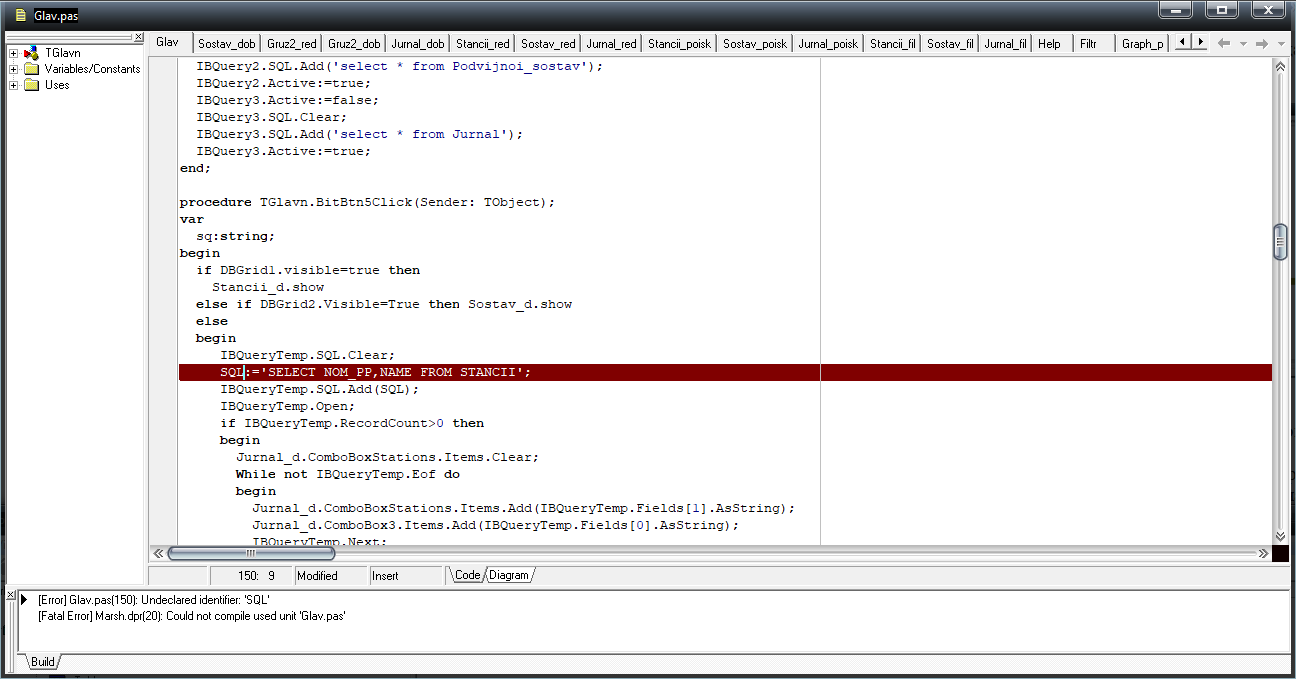


Рисунок 4 – Пример метода «структурный анализ»

Устранение данной ошибки показано на рисунке 4.

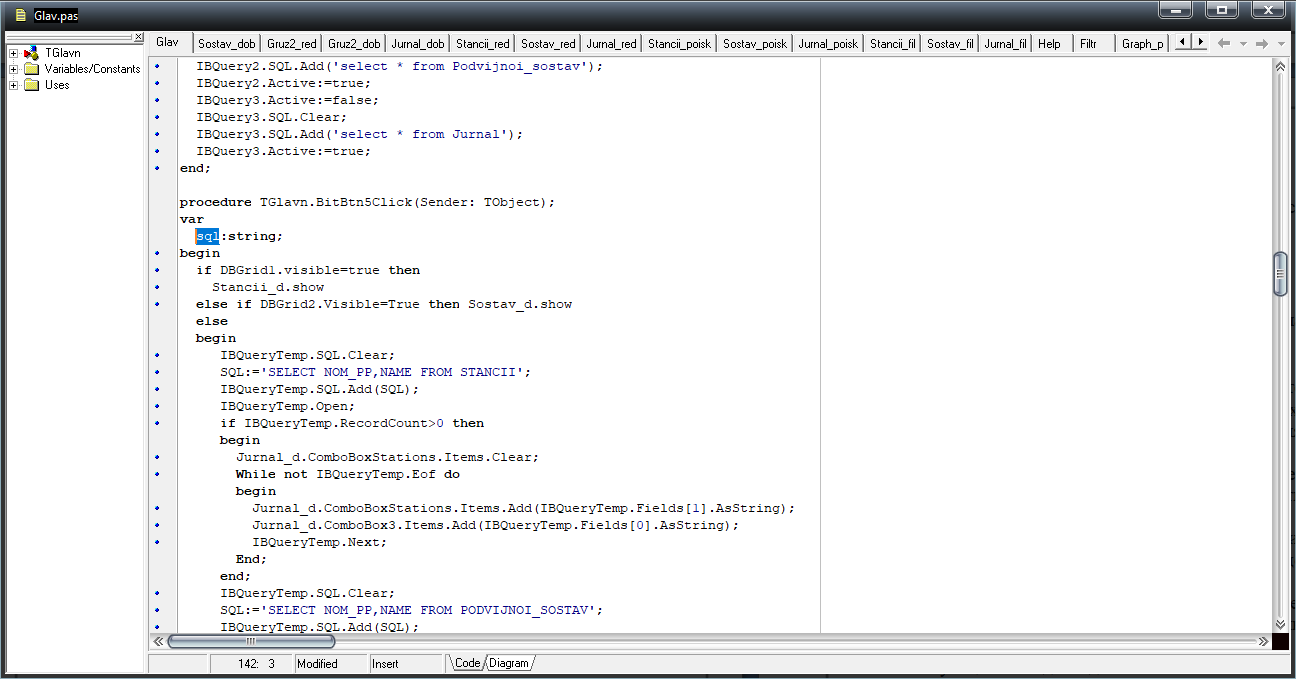


Рисунок 4 – Устранение ошибки

Третьим статическим методом устранения ошибок является «доказательство корректности (верификация)».

Это процесс, имеющий целью показать с помощью математического аппарата соответствие проекта и программы предъявляемым к ним требованиям. Программа разбивается на логические сегменты, определяются входные и выходные утверждения для каждого сегмента и доказывается, что если все входные утверждения при работе программы истинны, то и все выходные утверждения также будут истинны. Входное утверждение описывает характеристики входных данных сегмента, а выходное утверждение – характеристики результатов.

Реально метод может быть применен только для доказательства правильности небольших структурированных программ и представляет скорее теоретический интерес.

Пример метода «доказательство корректности (верификация)» показан на рисунке 5-6. Все номера рисунков проверь

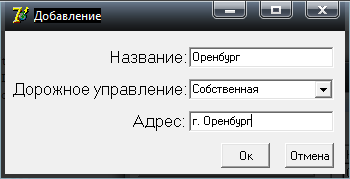


Рисунок 5 - Пример метода «доказательство корректности (верификация)»

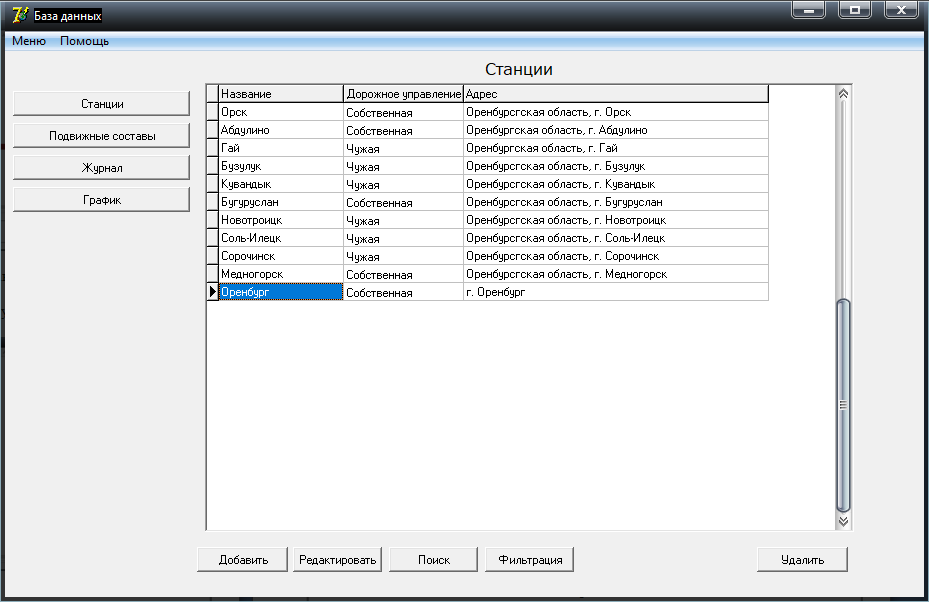


Рисунок 6 - Пример метода «доказательство корректности (верификация)»

**3.3 Функциональное тестирование**

При функциональном тестировании выявляются следующие категории ошибок: (привести свои примеры)

* некорректность или отсутствие функций;
* ошибки интерфейса;
* ошибки в структурах данных;
* ошибки машинных характеристик (нехватка памяти и др.);
* ошибки инициализации и завершения.

Техника тестирования ориентирована:

* на сокращение необходимого количества тестовых вариантов;
* на выявление классов ошибок, а не отдельных ошибок.

Первым способом функционального тестирования является «разбиение на классы эквивалентности».

Это самый популярный способ. Его суть заключается в разделении области входных данных программы на классы эквивалентности и разработке для каждого класса одного тестового варианта.

Класс эквивалентности – набор данных с общими свойствами, в силу чего при обработке любого набора данных этого класса задействуется один и тот же набор операторов.

Классы эквивалентности определяются по спецификации программы. Тесты строятся в соответствии с классами эквивалентности, а именно: выбирается вариант исходных данных некоторого класса и определяются соответствующие выходные данные.

Самыми общими классами эквивалентности являются классы допустимых и недопустимых (аномальных) исходных данных. Описание класса строится как комбинация условий, описывающих каждое входное данное.

Условия допустимости или недопустимости данных задают возможные значения данных и могут описывать:

* некоторое конкретное значение; определяется один допустимый и два недопустимых класса эквивалентности: заданное значение, множество значений меньше заданного, множество значений больше заданного;
* диапазон значений; определяется один допустимый и два недопустимых класса эквивалентности: множество значений в границах диапазона; множество значений, выходящих за левую границу диапазона; множество значений, выходящих за правую границу диапазона;
* множество конкретных значений; определяется один допустимый и один недопустимый класс эквивалентности: заданное множество и множество значений, в него не входящих.

Такие классы можно описать языком логики, например, языком исчисления предикатов. Описания более сложных условий и соответствующих классов (например, элементы массива должны находиться в некотором диапазоне и при этом массив не должен содержать нулевых элементов) могут быть построены на основании приведенных выше условий.

В примере, приводимом в вопросе 2 темы 3, при построении тестов неформально использовался описанный метод. В методических целях были выделены только основные класс тестов. Кроме того, исходя из условия задачи, были выделены классы эквивалентности внутри класса правильных данных.

Вторым способом функционального тестирования является «анализ граничных значений».

Этот способ построения тестов дополняет предыдущий и предполагает анализ значений, лежащих на границе допустимых и недопустимых данных. Построение таких тестов часто диктуется интуицией.

Основные правила построения тестов:

* если условие правильности данных задает диапазон, то строятся тесты для левой и правой границы диапазона; для значений чуть левее левой и чуть правее правой границы;
* если условие правильности данных задает дискретное множество значений, то строятся тесты для минимального и максимального значений; для значений чуть меньше минимума и чуть больше максимума;
* если используются структуры данных с переменными границами (массивы), то строятся тесты для минимального и максимального значения границ.

Согласно этим правилам для упомянутого примера следовало бы построить тесты для значений данных, лежащих на границах диапазонов; кроме того, следовало бы рассмотреть массив из одного и из 50 элементов и, соответственно, матрицу из такого же количества строк и столбцов.

Третьим способом функционального тестирования является «диаграммы причин-следствий».

Взаимосвязь классов эквивалентности и соответствующих им действий описывается формально в виде графа на основе автоматного подхода. Граф преобразуется в таблицу решений, столбцы которой в свою очередь преобразуются в тестовые варианты.

На рисунках 7-18 показано тестирование всех процедур программы.

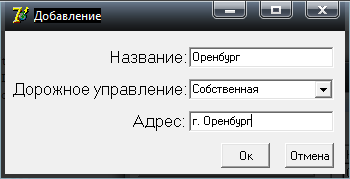


Рисунок 7 – Процедура добавления

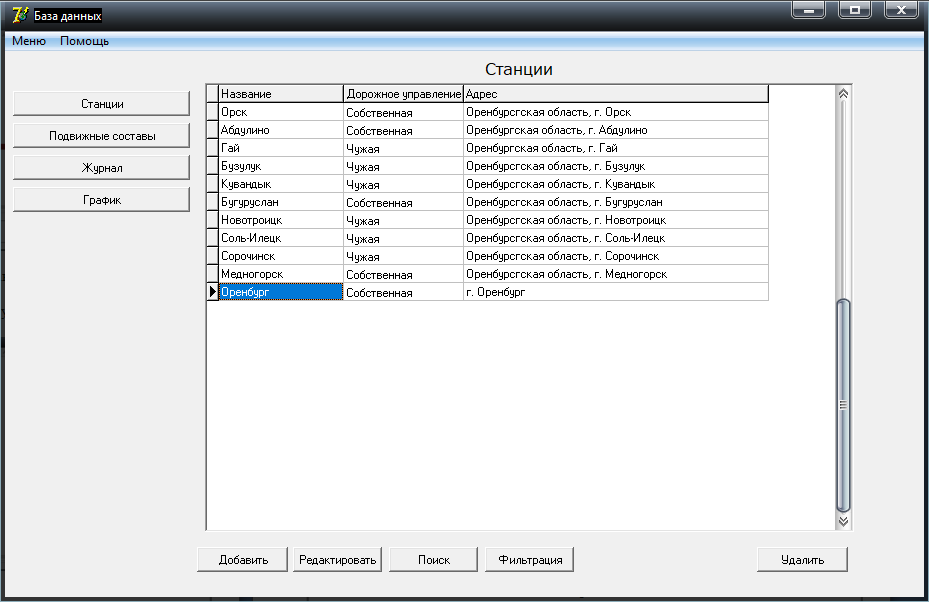


Рисунок 8 – Успешное добавление записи

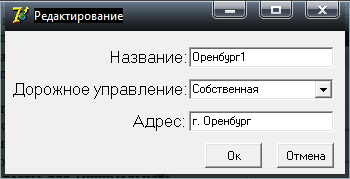


Рисунок 9 – Процедура редактирования

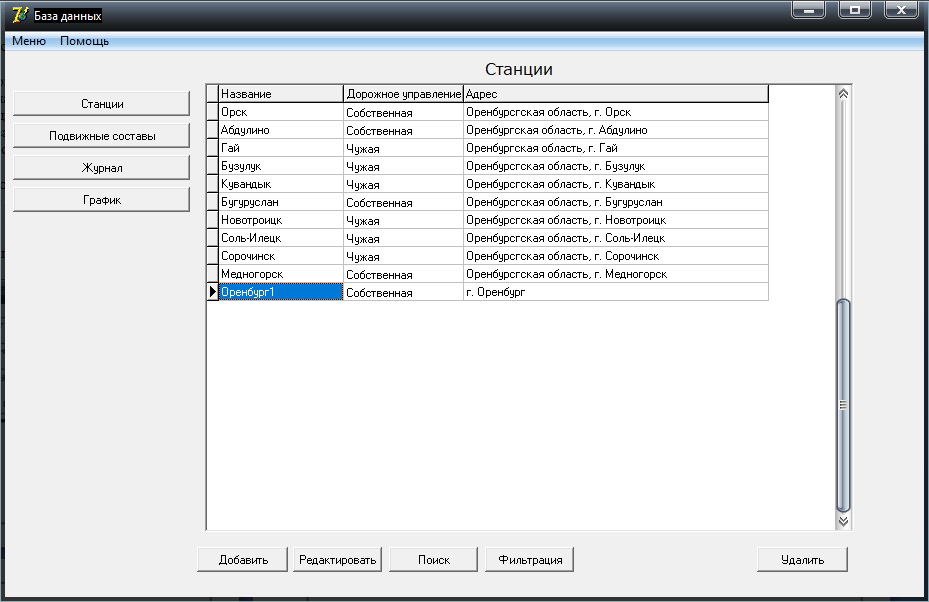


Рисунок 10 – Успешное редактирование записи

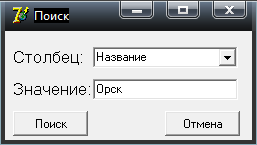


Рисунок 11 – Процедура поиска

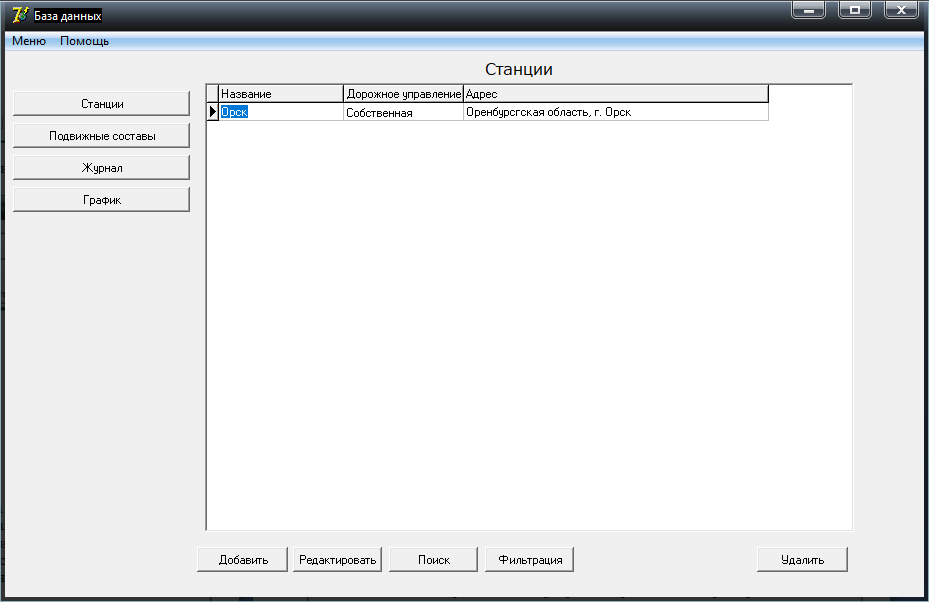


Рисунок 12 – Успешный поиск записи

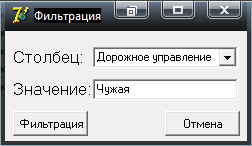


Рисунок 13 – Процедура фильтрации

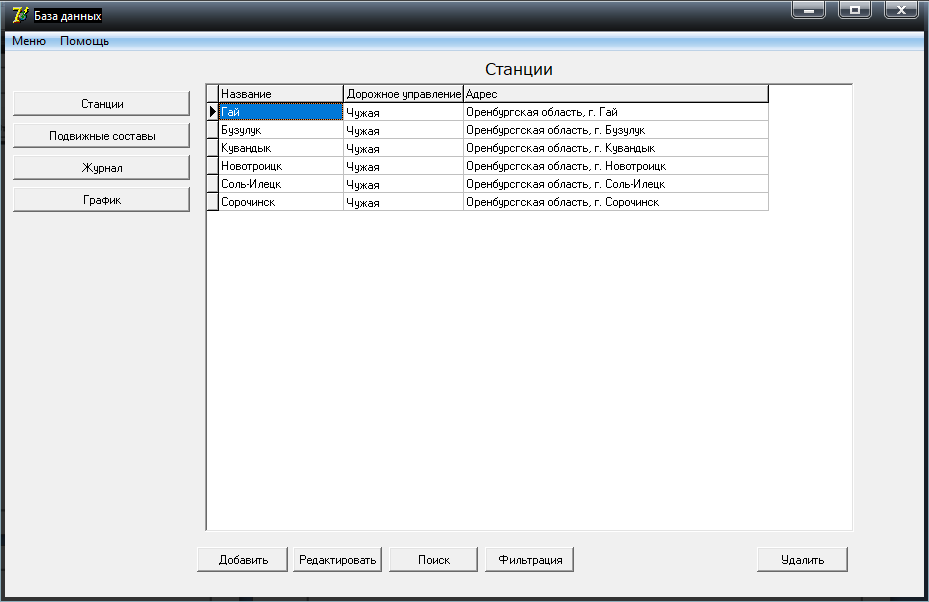


Рисунок 14 – Успешная фильтрация записей

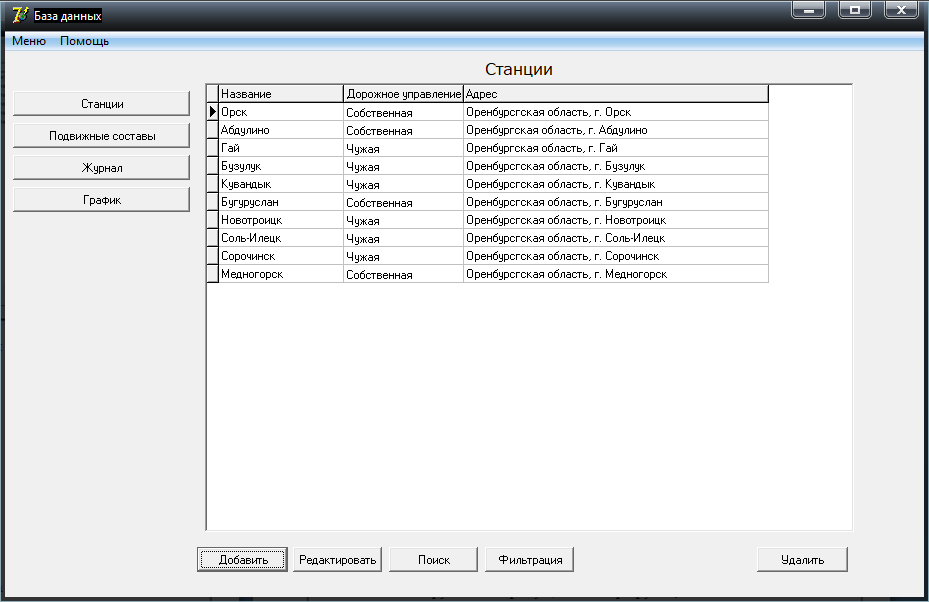


Рисунок 15 – Процедура удаления

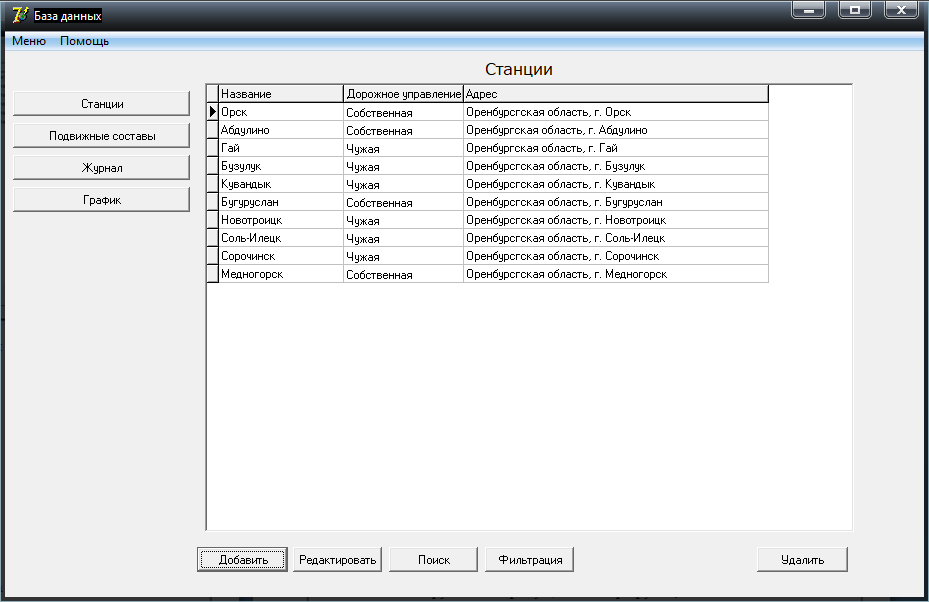


Рисунок 16 – Успешное удаление записи

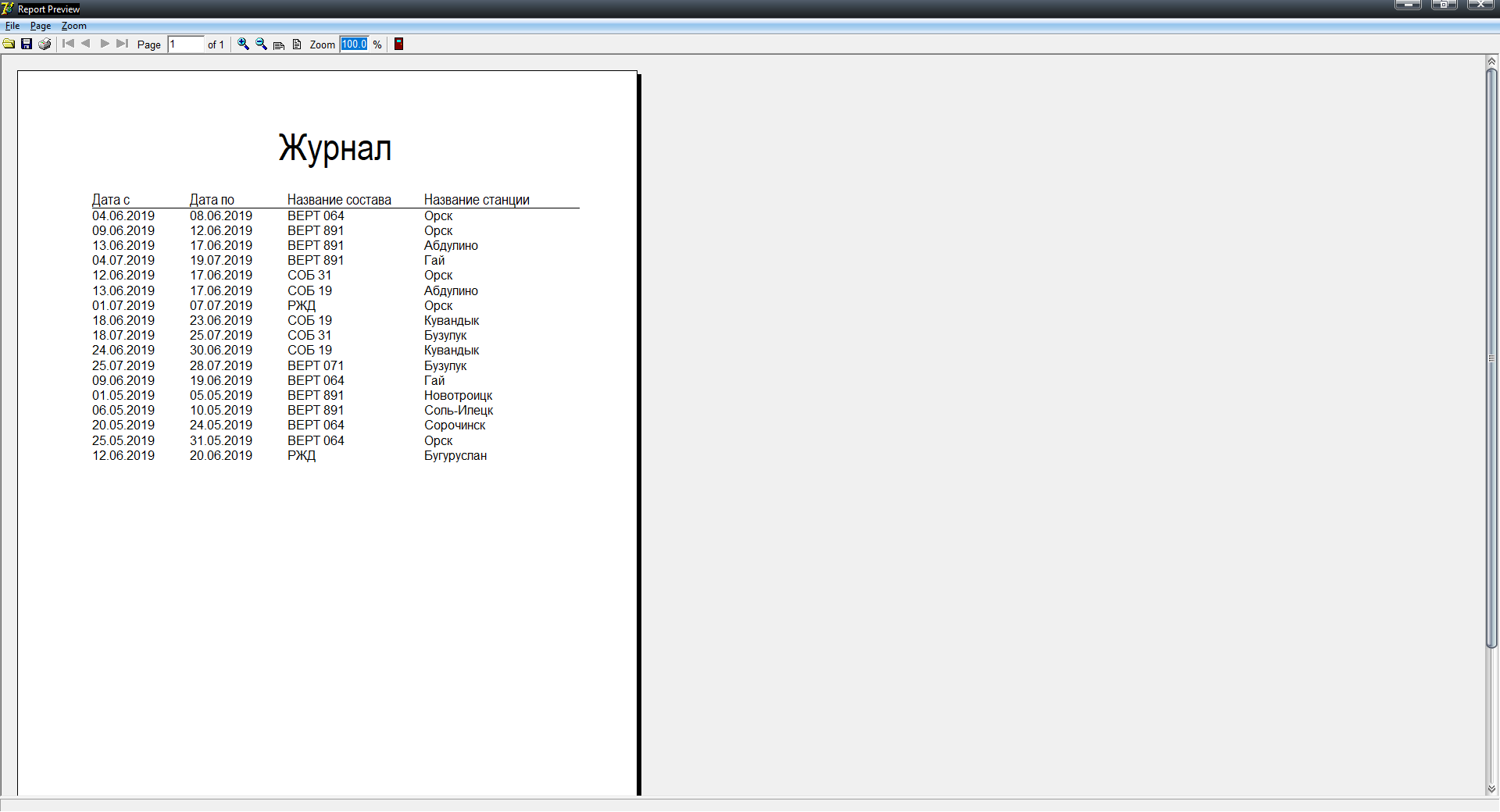


Рисунок 17 – Вывод на печать

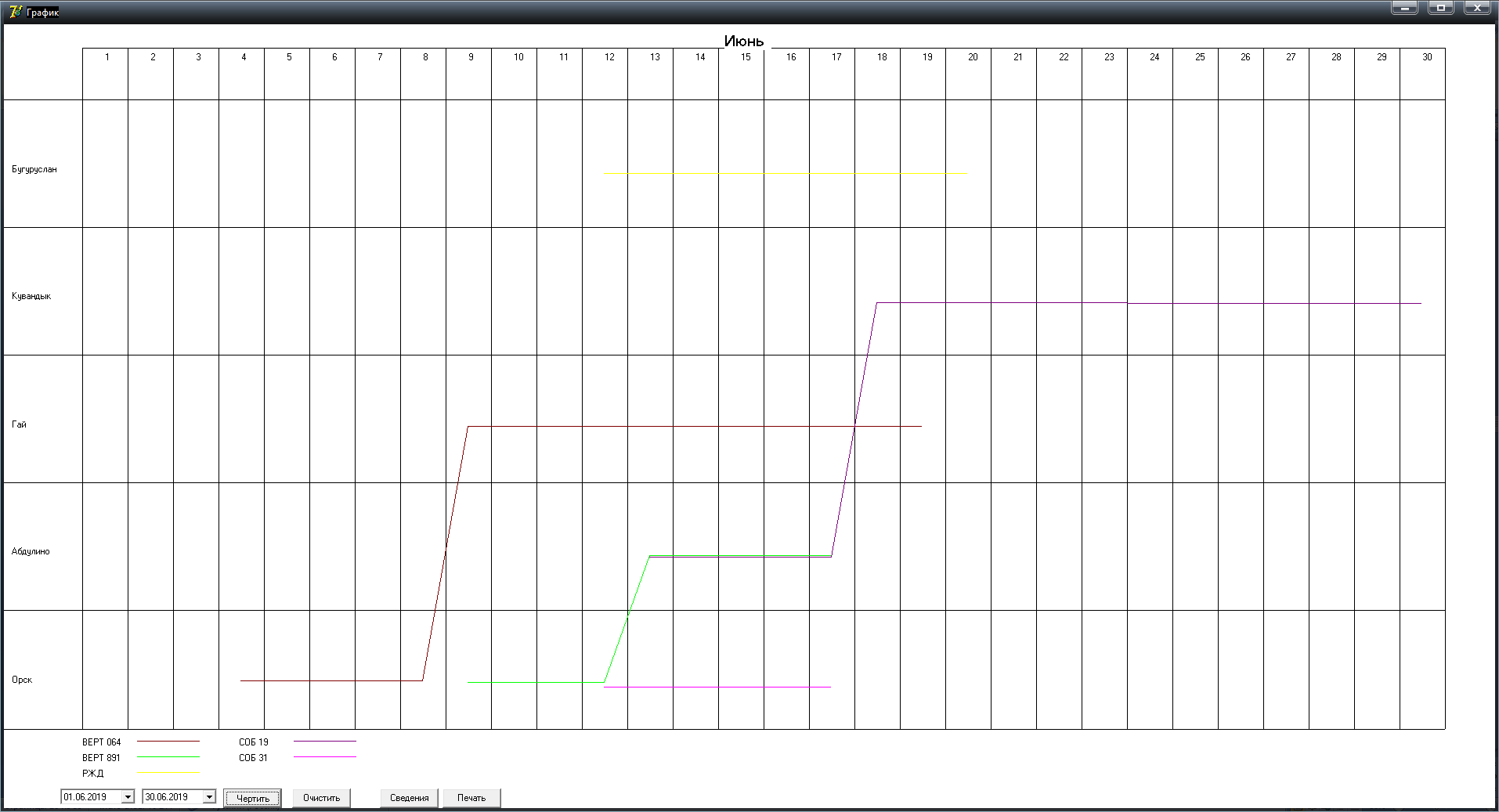


Рисунок 18 – Построение графика

**3.4 Структурное тестирование**

Структурное тестирование основывается на детальном изучении логики программы и подборе тестов, обеспечивающих максимально возможное число проверяемых операторов, логических ветвлений и условий. Его еще называют «тестирование по маршрутам». Под маршрутом понимают последовательности операторов программы, которые выполняются при конкретном варианте исходных данных.

При построении тестов используют следующие критерии:

* покрытие операторов путем выбора набора данных, обеспечивающего выполнение каждого оператора в программе по крайней мере один раз.
* покрытие условий путем подбора наборов данных, обеспечивающих в узлах ветвления с более чем одним условием принятие каждым условием значения "истина" или "ложь" хотя бы по одному разу.
* комбинаторное покрытие условий путем подбора тестов, обеспечивающих в узлах ветвления с более чем одним условием перебор всех возможных сочетаний значений условий в одном узле ветвления.

Считают, что программа проверена полностью, если с помощью тестов удается осуществить выполнение программы по всем возможным маршрутам передач управления. Однако нетрудно видеть, что даже в программе среднего уровня сложности число неповторяющихся маршрутов может быть очень велико и, следовательно, полное или исчерпывающее тестирование маршрутов, как правило, невозможно.

При структурном тестировании учитывается логика программы.

Примеры тестов приведены ниже.

Для тестирования программы с помощью критерия покрытия операторов в простейшем случае достаточно проверки программы на следующих тестах:

* a = 1 b = 4 (b<5);
* a = 1 b = 6 (b>5).

Видно, что в этих тестах ни разу не будет рассмотрен случай, когда а равно 0.

Для тестирования программы с помощью критерия покрытия условий можно использовать следующий набор тестов:

* а = 1 b = 6;
* a = 0 b = 1.

Видно, что в этих тестах ни разу не будет проверено выполнение вычисления, хотя тестовые наборы и удовлетворяют критерию.

Для тестирования программы с помощью критерия комбинаторного покрытия условий тестовые наборы данных могут выглядеть, например, следующим образом

* a = 1 b = 4;
* a = 0 b = 4;
* a = 1 b = 6;
* a = 0 b = 6.

Структурный подход к тестированию имеет ряд недостатков. Так тестовые наборы, построенные по данной стратегии:

* не обнаруживают пропущенных маршрутов;
* не обнаруживают ошибок, зависящих от обрабатываемых данных, например, в оператореif (a - b) < eps - пропуск функции абсолютного значения abs проявится только, если а < b;
* не дают гарантии, что программа правильна, например, если вместо сортировки по убыванию реализована сортировка по возрастанию [8].

Пример структурного тестирования показан на рисунке 19.

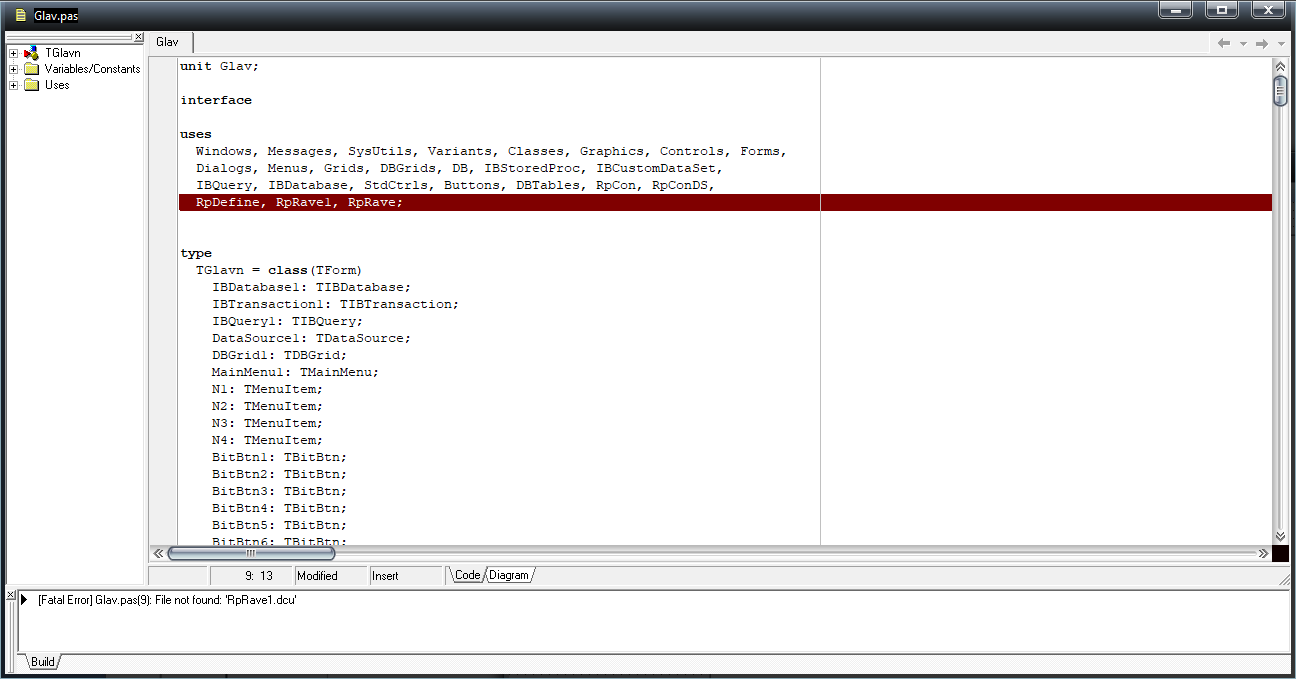


Рисунок 19 – Ошибка при подключении библиотеки

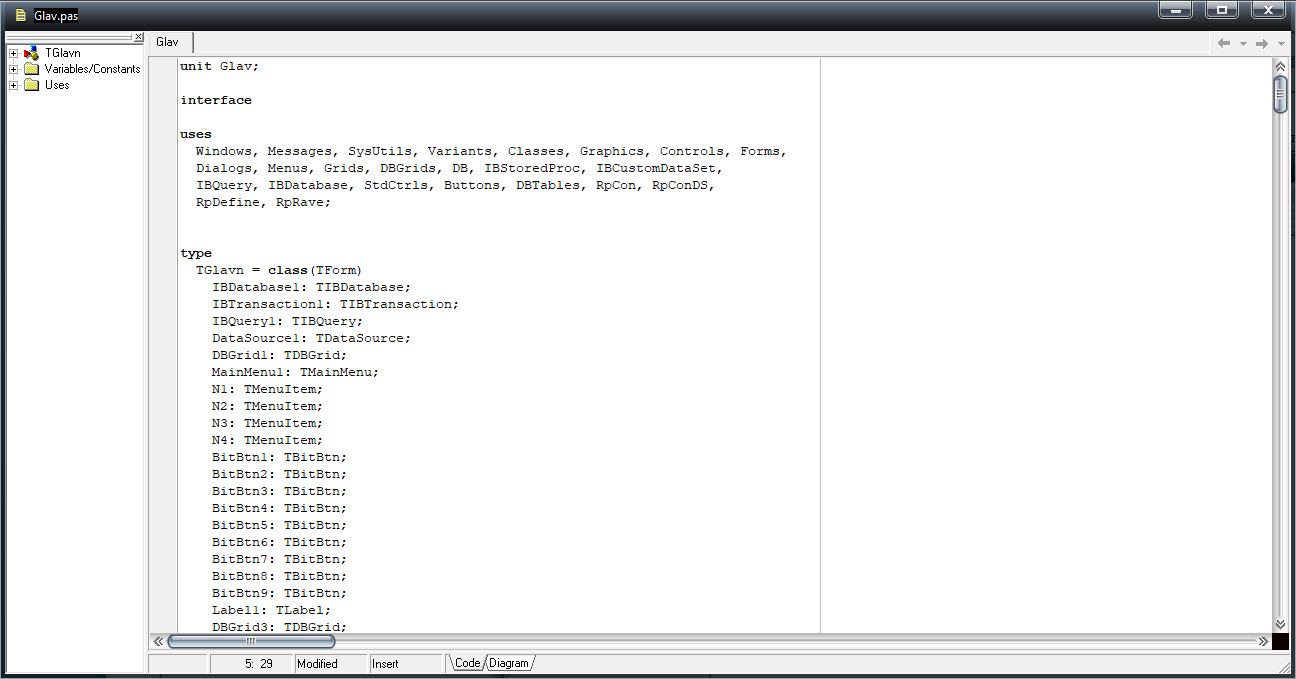


Рисунок 20 – Устранение ошибки

**4 Руководство системного программиста**

**4.1 Общие сведения о программе**

Программа предназначена для ведения базы данных и построения графика подвижных составов. БД содержит информацию о станциях, составах, маршрутах.

Чтобы начать работу с программой необходимо запустить файл с именем Marsh.exe. После этого открывается главная форма проекта, на которой расположены таблица, кнопки и меню для управления программой.

Кнопка «Станции» меняет таблицу на данную.

Кнопка «Подвижные составы» меняет таблицу на данную.

Кнопка «Журнал» меняет таблицу на данную.

Кнопка «График» открывает форму для построения графика.

Кнопка «Добавить» открывает форму для добавления записи в таблицу.

Кнопка «Редактировать» открывает форму для редактирования записи в таблице.

Кнопка «Поиск» открывает форму для поиска записи в таблице.

Кнопка «Фильтрация» открывает форму для фильтрации записей в таблице.

Кнопка «Удалить» удаляет выбранную запись из таблицы.

В верхнем меню программы имеются 2 кнопки: «Выход», «Справка».

Подключение базы данных к программе происходит с помощью компонента IBDataBase, где уже заранее выбран путь к БД. Сама БД находится в той же папке, что и программа.

Для редактирования данного ПО на компьютере должны быть установлены такие программы, как:

* Delphi;
* InterBase.

**4.2 Структура программы**

Структурная схема программы представлена на рисунке 21.

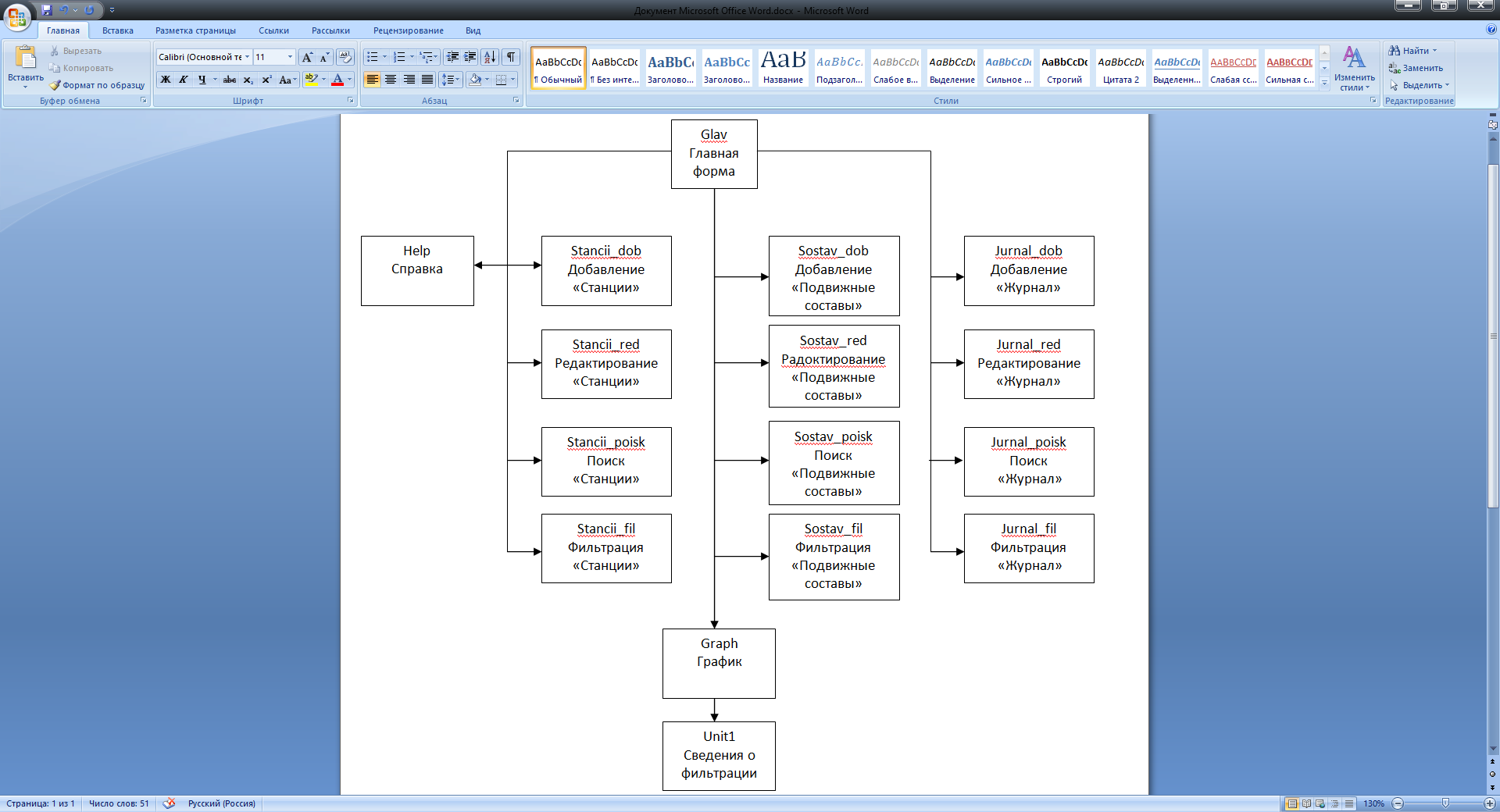


Рисунок 21 – Структура программы

Программа состоит из следующих модулей:

* модульGlav–главный модуль программы, где описаны все основные действия программы;
* модуль Help – модуль с справкой к программе;
* модуль Stancii\_dob – модуль с функцией добавления записи в таблицу «Станции»;
* модуль Stancii\_red – модуль с функцией редактирования записи в таблице «Станции»;
* модуль Stancii\_poisk – модуль с функцией поиска записи в таблице «Станции»;
* модуль Stancii\_fil – модуль с функцией фильтрации записей в таблице «Станции»;
* модуль Sostav\_dob – модуль с функцией добавления записи в таблицу «Подвижные составы»;
* модуль Sostav\_red – модуль с функцией редактирования записи в таблице «Подвижные составы»;
* модуль Sostav\_posik – модуль с функцией поиска записи в таблице «Подвижные составы»;
* модуль Sostav\_fil – модуль с функцией фильтрации записей в таблице «Подвижные составы»;
* модуль Jurnal\_dob – модуль с функцией добавления записи в таблицу «Журнал»;
* модуль Jurnal\_red – модуль с функцией редактирования записи в таблице «Журнал»;
* модуль Jurnal\_poisk – модуль с функцией поиска записи в таблице «Журнал»;
* модуль Jurnal\_fil – модуль с функцией фильтрации записей в таблице «Журнал»;
* модульGraph– модуль программы для построения графика;
* модульUnit1–модуль программы, где при построении графика выводятся отфильтрованные записи.

**4.3 Настройка программы**

На рисунке 22 показан компонент для подключения базы данных - IBDataBase.

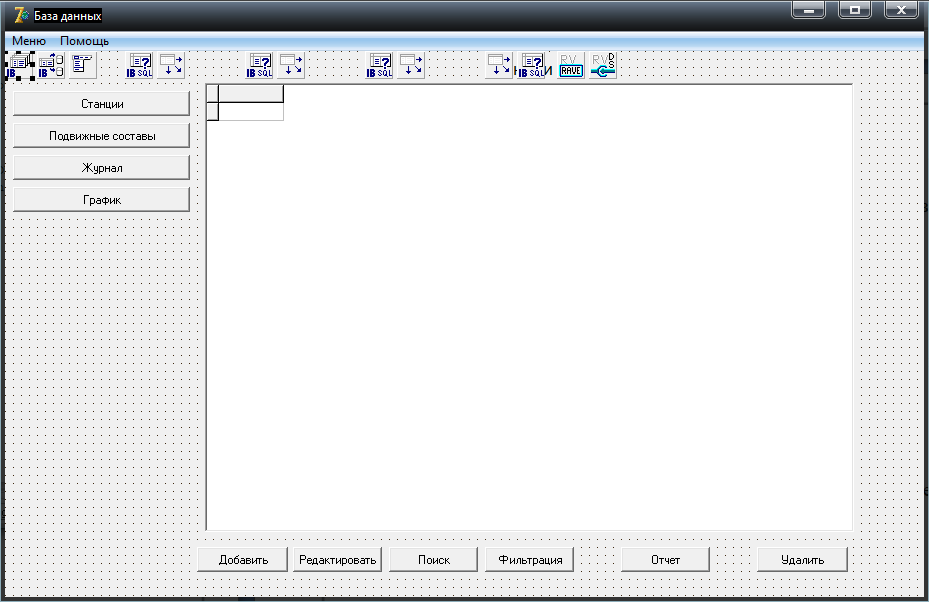


Рисунок 22 – Компонент IBDataBase

На рисунке 23 показан прописанный путь к БД.

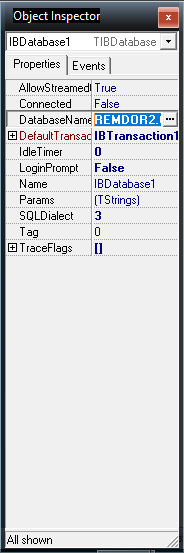


Рисунок 23 – Путь к БД

При нажатии на кнопку, выводится окно для выбора БД, также путь можно прописать вручную.

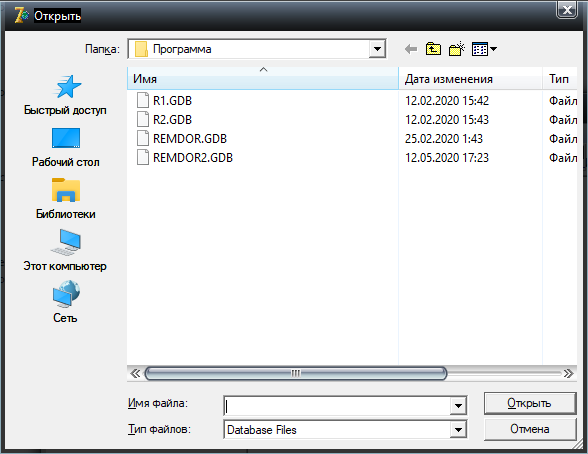


Рисунок 24 – Окно выбора БД

**4.4Сообщение системному программисту**

На рисунке 25 показана ошибка, возникающая при неверном прописанном пути к базе данных.

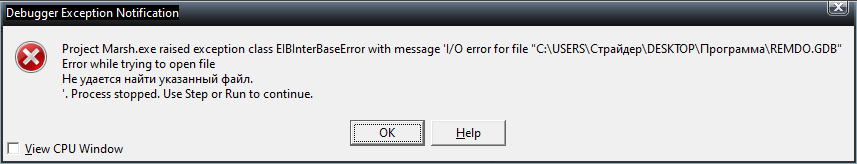
****

Рисунок 25 *–* Ошибка

**5 Руководство оператора**

**5.1 Выполнение программы**

Главная форма на которой содержатся все таблицы и компоненты предоставлена на рисунке 26.

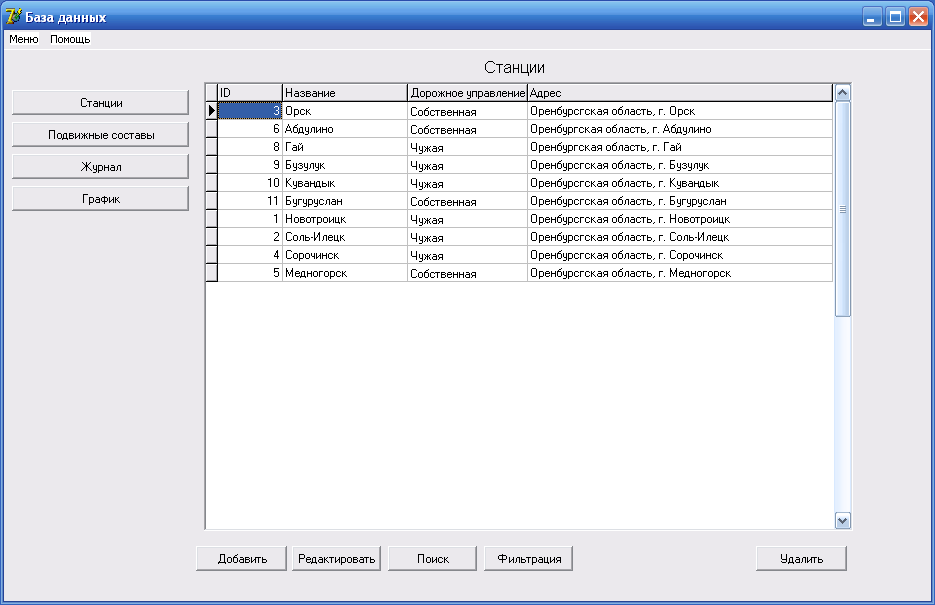


Рисунок 26 – Главная форма программы

На рисунке представлена фильтрация записей.

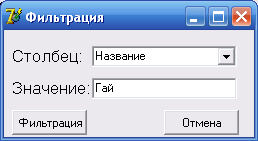


Рисунок 27 – Фильтрация записей.

На рисунке 28 представлен поиск записей.

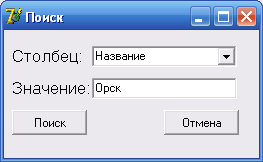


Рисунок 28 – Поиск записи

На рисунке 29 представлена таблица «Станции».

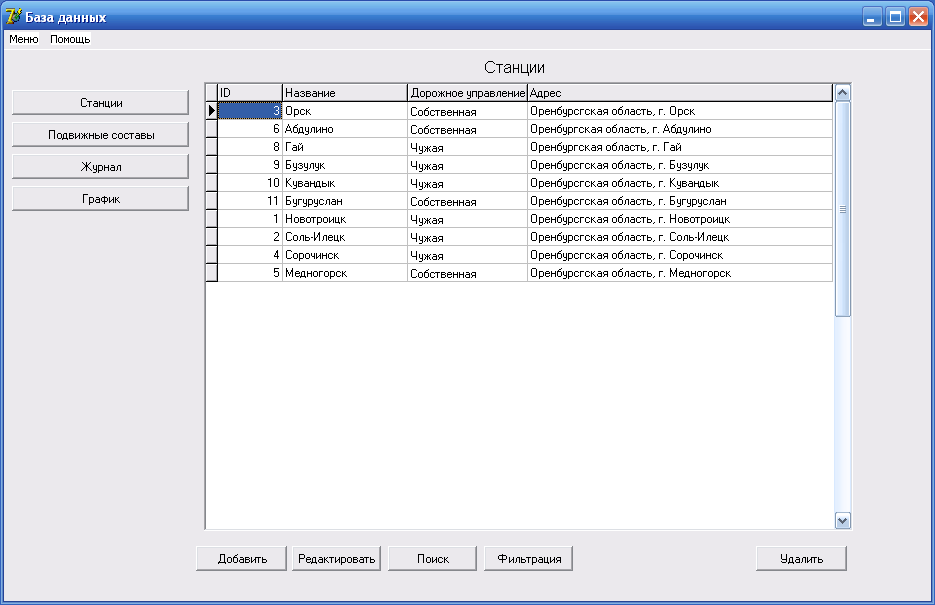


Рисунок 29 – Таблица «Станции»

На рисунке 30 представлена таблица «Подвижные составы».

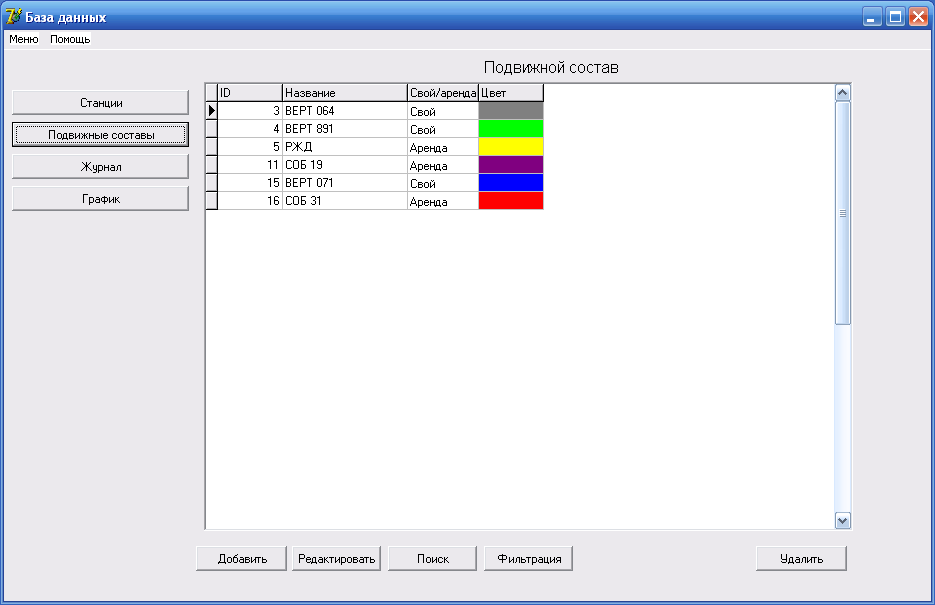


Рисунок 30 – Таблица «Подвижные составы»

На рисунке 31 таблица «Журнал».

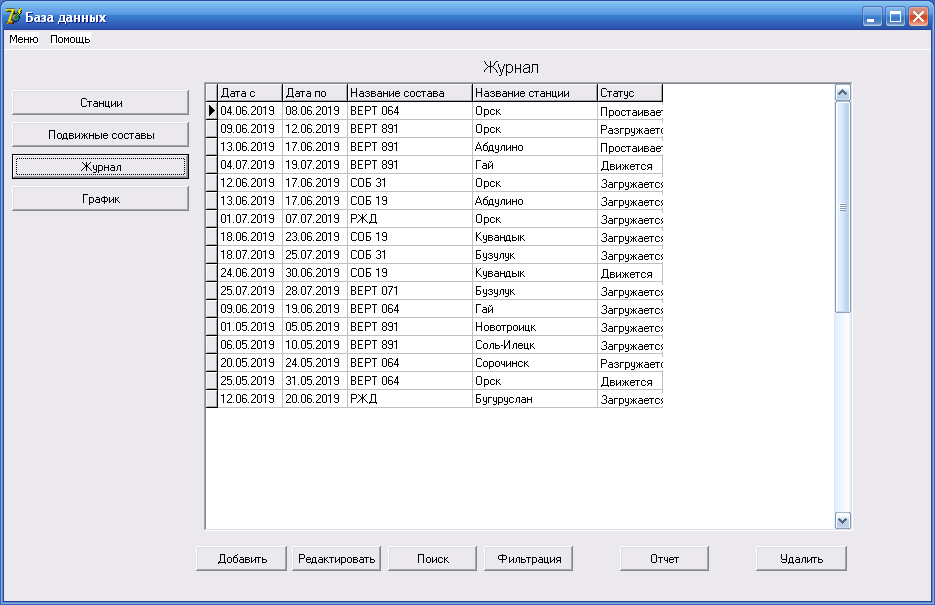


Рисунок 31 – Таблица «Журнал»

На рисунке 32 представлено добавление записи в таблицу.

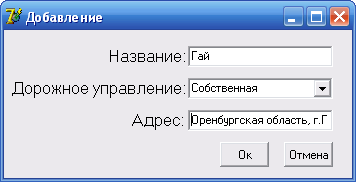


Рисунок 32 – Добавление записи в таблицу

На рисунке 33 представлено редактирование записи.

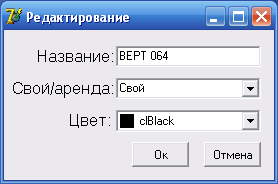


Рисунок 33 – Редактирование записи

На рисунке 34 представлен график движения подвижных составов.

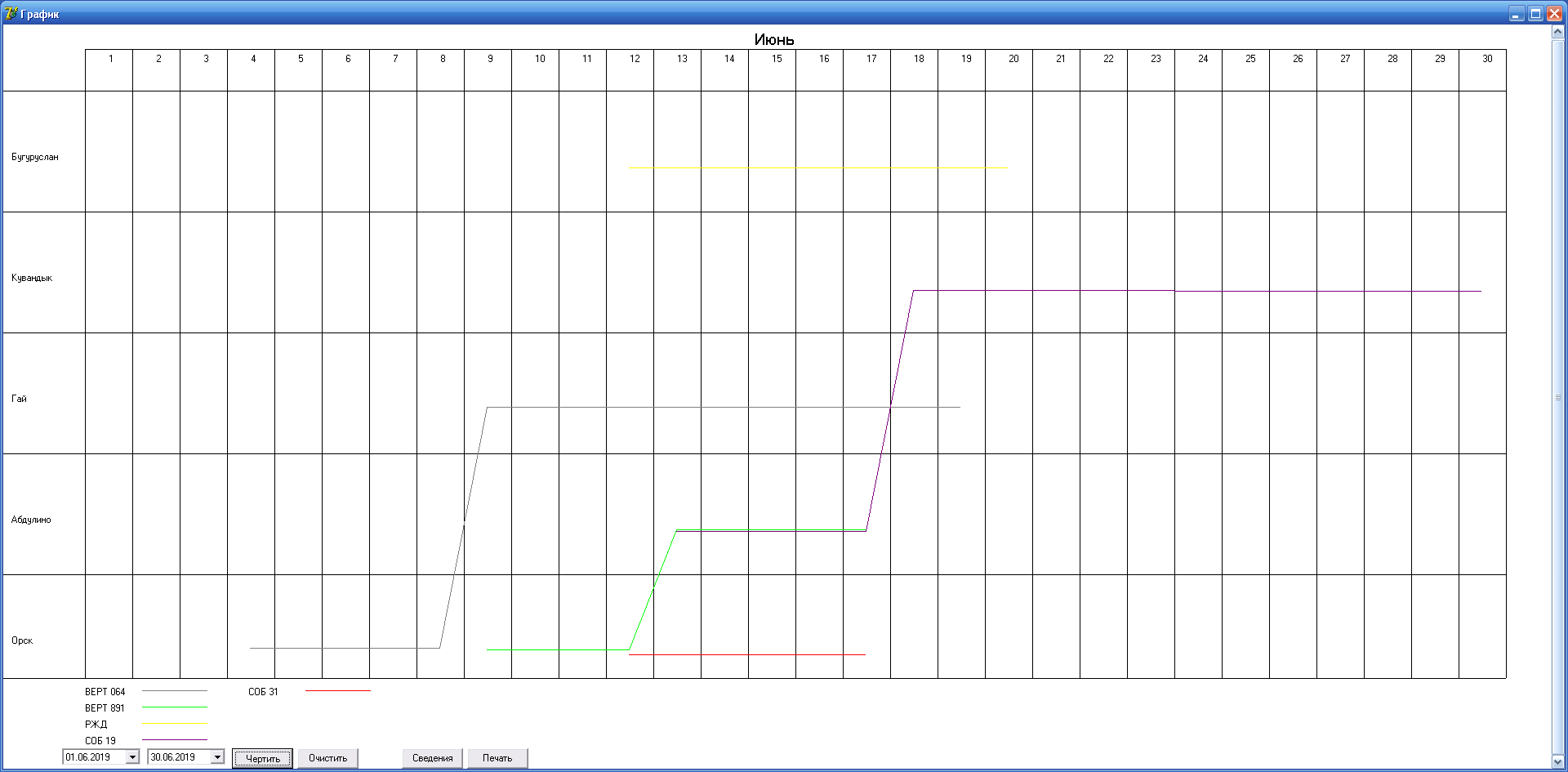


Рисунок 34 – График движения подвижных составов

**5.2 Сообщения оператору**

На рисунке 35 показано сообщение при успешном редактировании записи базы данных.

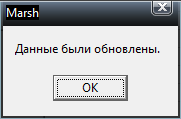


Рисунок 35 *–* Сообщение при успешном редактировании записи

На рисунке 36 показано сообщение при построении графика.

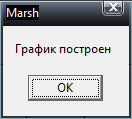


Рисунок 36 *–* Сообщение при построении графика

**6 Охрана труда и безопасность жизнедеятельности**

В процессе эксплуатации электрооборудования существует вероятность поражения электрическим током. Дать оценку потенциальной опасности поражения электрическим током проектируемого оборудования.

Электрооборудование должно быть выбрано и установлено таким образом, чтобы не могло привести к повреждению оборудования, возникновению короткого замыкания или замыканию на землю, а также причинить вред обслуживающему персоналу.

При написании данного раздела необходимо использовать следующие нормативные документы:

* правила устройства электроустановок;
* правила эксплуатации электроустановок потребителей;
* межотраслевые правила по охране труда (правила безопасности) при эксплуатации электроустановок ПОТ РМ-016-2001 РД 153-34.0-03.150-00.

Мероприятия по обеспечению электробезопасности разрабатываются в соответствии с ГОСТ 12.1.019-79\* ССБТ «Электробезопасность. Общие требования» и ПУЭ.

Для повышения производительности труда, путем снижения утомляемости и травматизма, работодатель обязан обеспечивать рациональное освещение рабочего помещения. Для этого необходимо подобрать соответствующий вид освещения, которые подразумевают искусственное и естественное, а также структуру системы освещения, сюда входят общее и комбинированное. Необходимую нормируемую освещенность определяют по СНиП 23-05-95 «Естественное и искусственное освещение» в зависимости от характеристики зрительной работы (наименьшего размера объекта различения), фона, контраста, вида и системы освещения или СанПиН 2.2.1/2.1.1.1278-03 «Гигиенические требования к естественному, искусственному и совмещенному освещению жилых и общественных зданий»

В случаи выбора искусственного освещения, работодателю необходимо позаботиться о типе светильников, где выбор будет зависеть от условий среды (нормальная, пыльная, влажная, жаркая и т.п.).

Также необходимо указать нормируемую освещенность для аварийного освещения и тип светильника. При проектировании ГПП предусмотреть наружное освещение территории.

Для расчёта общего искусственного производственного освещения вполне применим метод коэффициента использования светового потока. При этом учитывается как прямой свет от светильника, так и свет, отраженный от стен и потолка: Требуемый световой поток ламп можно рассчитать по формуле (1):

(лм) , (1)

где F- световой поток каждой из ламп в люменах;

S –площадь помещения в м2;

k— коэффици­ент запаса, учитывающий старение ламп, запыление и загрязнение светильников;

z — коэффициент неравномерности освещения, отношение средней освещенности к минимальной (в большинст­ве случаев z=1,1 —1,2);

N — число светильников;

n— коэффициент использования светового потока (в долях единицы), т.е. отношение потока, падающего на рас­четную поверхность, к суммарному потоку всех ламп, находится в зависимости от типа светильника, величины индекса помеще­ния (i) и коэффициента отражения потолка и стен.

Выбирают способ размещения светильников, который может быть симметричным или локализованным. При симметричном размещении светильники располагаются как вдоль, так и поперек помещения на одинаковом расстоянии, по углам прямоугольника или в шахматном порядке. Симметричное размещение светильников обеспечивает одинаковое освещение оборудования, станков, рабочих мест и проходов, но требует большого расхода электроэнергии. При локализованном расположении светильники размещают с учетом местонахождения станков, машин, оборудования, мест контроля и рабочих мест. Такое расположение светильников, сокращающее расход электроэнергии, применяют в цехах с несимметричным размещением оборудования.

Далее определяют отношение расстояния между светильниками L к высоте их подвеса Hс. В зависимости от типа светильника это отношение L/Hc при расположении светильников прямоугольником может быть принято равным 1,4-2,0, а при шахматном расположении -1,7-2,5.

Высота расположения светильника над освещаемой поверхностью определяется по формуле (2):

(2)

где Н - общая высота помещения, м;

hc - высота от потолка до нижней части светильника, м;

hр - высота от пола до освещаемой поверхности, м.

Чтобы уменьшить ослепляющее действие светильников общего освещения, высоту подвеса их над уровнем пола устанавливают не менее 2,5-4 м при лампах мощностью до 200 Вт и не менее 3-6 м при лампах большей мощности.

Потребное число светильников (ламп) определяют по формуле (3):

(при La=Lb) (3)

На следующем этапе расчета определяют индекс помещения i по формуле:

(лм) (4)

где Hc - высота подвески светильника над рабочей поверхностью, м;

S - площадь помещения, м2;

A и В — длина и ширина помещения, м

**А где расчеты то по формулам???**

**Заключение**

В ходе исследования, проведенного во время выпускной квалификационной работы, были закреплены навыки по созданию и модификации управления данными в реляционной базе данных, было спроектировано и разработано приложение, осуществляющее создание и просмотр БД с построением графика движения подвижных составов.

Результатом выпускной квалификационной работы является программный продукт. В ходе выполнения выпускной квалификационной работы была проведена следующая работа:

* сформулированы основные задачи и ключевые направления разработки. Назначением разработки является создание программы для работы с базой данных, содержащей информацию о станциях, подвижном составе и маршрутах, которая позволит облегчить процесс построения графика движения подвижных составов. Проведенный анализ предметной области позволил выделить основные технические и функциональные особенности будущего программного продукта;
* произведен выбор прикладного программного обеспечения. В качестве основной среды разработки выбрана среда визуального программирования Delphi. База данных проектировалась средствами сервера удаленных баз данных InterBase;
* проведено проектирование программы. Основной функционал программного продукта соответствует заявленным задачам и концептуальным требованиям предприятия;
* проведено тестирование программы. Модель тестирования выглядела следующим образом. На вход процесса тестирования поступали три потока: текст программы; исходные данные для запуска программы; ожидаемые результаты. Было проведено структурное, функциональное и статическое тестирование. Статическое тестирование предполагало применение нескольких методов, одним из которых являлся «структурный анализ». Это автоматизированный анализ текста программы с помощью специальных программ – структурных анализ. Функциональное тестирование включало в себя проверку основных функций, описанных в техническом задании. Структурное тестирование основывалось на детальном изучении логики программы и подборе тестов, обеспечивающих максимально возможное число проверяемых операторов, логических ветвлений и условий. Это позволило выявить и устранить ошибки или недочеты в работе программы.
* составлено руководство системного программиста и оператора, которое содержит подробную инструкцию по работе с программой. Руководство системного программиста охарактеризовывает процесс установки программы. Чтобы начать работу с программой необходимо запустить файл с именем Marsh.exe. После этого открывается главная форма проекта, на которой расположены таблица, кнопки и меню для управления программой.

По итогу выполнения выпускной квалификационной работы, можно сделать вывод, что при разработке программного обеспечения был пройден полный цикл проектирования программы от постановки задачи до введения выходного результата на исполнение и эксплуатацию.

При выполнение данной работы, можно сделать вывод, что программное обеспечение значительно упрощает работу предприятия и повышает быстроту и эффективность работы сотрудников.

**Список использованных источников**

1 Delphi C/S 2. Русскоязычная документация; Borland Press - М., 2015. - 321 c.

2 Архангельский, А.Я. Программирование в Delphi 6; Бином - М., 2018. - 258 c.3

Архангельский, А.Я. Программирование в Delphi. Учебник по классическим версиям Delphi (+ дискета); Бином - М., 2017. - 583 c.

4 Бобровский, С. Delphi 5 Учебный курс; СПб: Питер - М., 2017. - 640 c.

5 Бобровский, Сергей Delphi 7. Учебный курс; СПб: Питер - М., 2018. – 736 c.

6 Бондарь Александр InterBase и Firebird. Практическое руководство для умных пользователей и начинающих разработчиков (+ CD-ROM); БХВ-Петербург - Москва, 2007. - 592 c.

7 Бондарь Александр InterBase и Firebird. Практическое руководство для умных пользователей и начинающих разработчиков (+ CD-ROM); БХВ-Петербург - Москва, 2012. - 592 c.

8/ Википедия [Электронный ресурс]. Тестирование программного обеспечения – Режим доступа: https://ru.wikipedia.org/wiki/Тестирование\_программного\_обеспечения.

9 Википедия [Электронный ресурс]. IDEF0 – Режим доступа : https https://ru.wikipedia.org/wiki/IDEF0.

10 Григорьев, А.Б. О чем не пишут в книгах по Delphi; БХВ-Петербург - М., 2016.-576c.

11 Дарахвелидзе, П.Г.; Марков, Е.П. Delphi 2005 для Win32 наиболее полное руководство; БХВ-Петербург - М., 2018. - 234 c.

12 Калверт, Ч. Базы данных в Delphi 4; Киев: ДиаСофт - М., 2013. - 464 c.

13 Ковязин, А.Н.; Востриков, С.М. Архитектура, администрирование и разработка приложений баз данных в InterBase/FireBird/Yaffil; М.: Кудиц-образ; Издание 4-е - Москва, 2006. – 496 c.

14 Культин, Никита Основы программирования в Delphi 7; СПб: БХВ - М., 2014. -608c.

15 Марков, Е.П.; Никифоров, В.В. Delphi 2005 для .NET; БХВ-Петербург - М., 2017.-896c.

16 Понамарев, В. Базы данных в Delphi 7. Самоучитель; СПб: Питер - М., 2015. -224c.

17 Сван, Том Секреты 32-разрядного программирования в Delphi (+ дискета); Диалектика-М., 2015.-480c.

18 Сухарев, М.В. Основы Delphi. Профессиональный подход; Наука и техника - М., 2018.-600c.

19 Федоров, А. Delphi 2.0 для всех; Компьютер-пресс - М., 2013. - 464 c.

20 Шумаков, П.В. Delphi 3 и разработка приложений баз данных; Нолидж - М., 2018. - 704 c.

**Приложение А**

*(справочное)*

**Входные и выходные документы**

Входная информация:

* информация о станциях;
* информация о составах;
* информация о маршрутах;

Выходная информация:

* отчет по журналу;
* график движения подвижных составов;

**Приложение Б**

*(рекомендуемое)*

**Диаграммы моделирования и анализа бизнес-процессов**

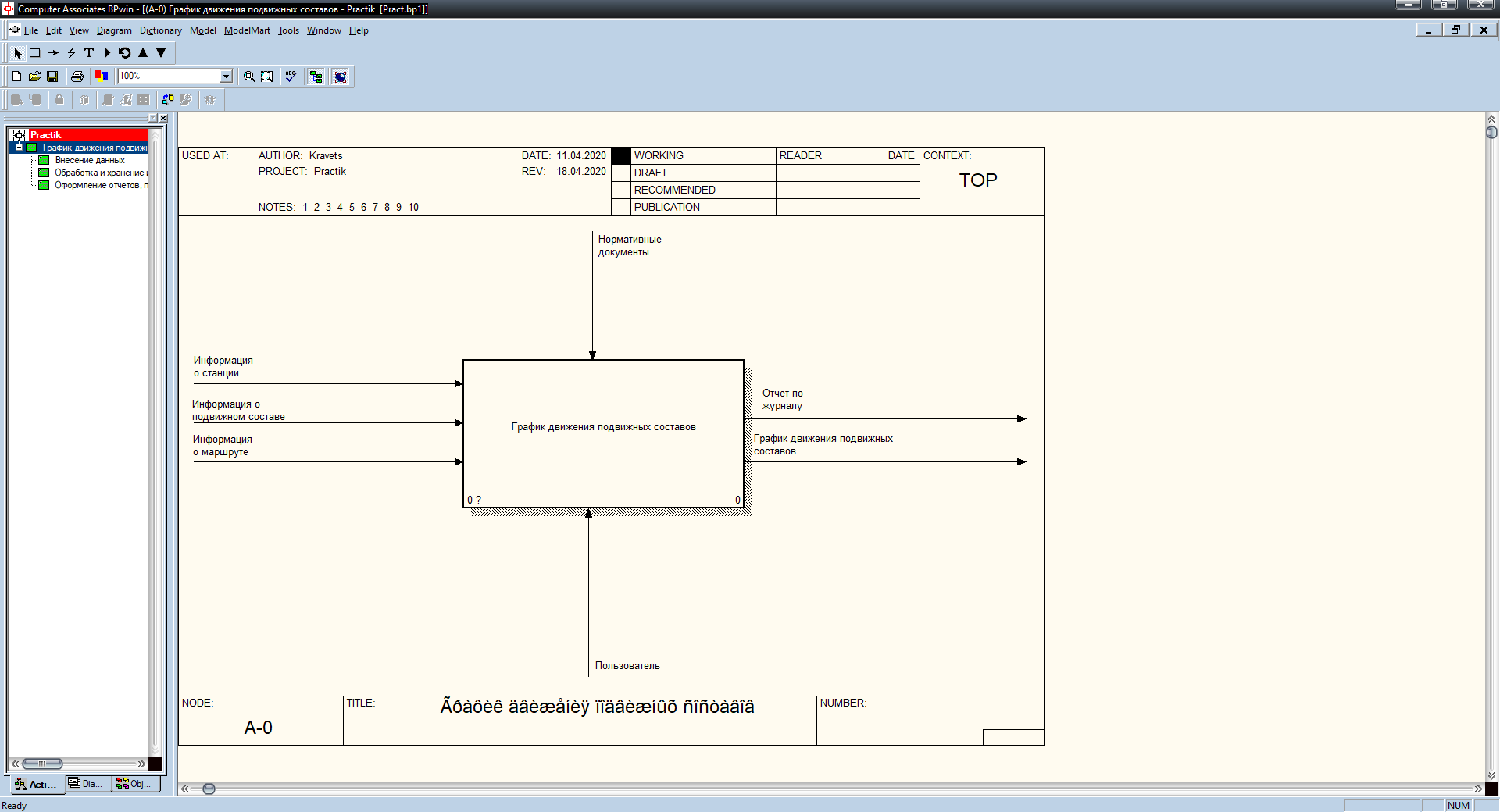


Рисунок Б.1 – Контекстная диаграмма

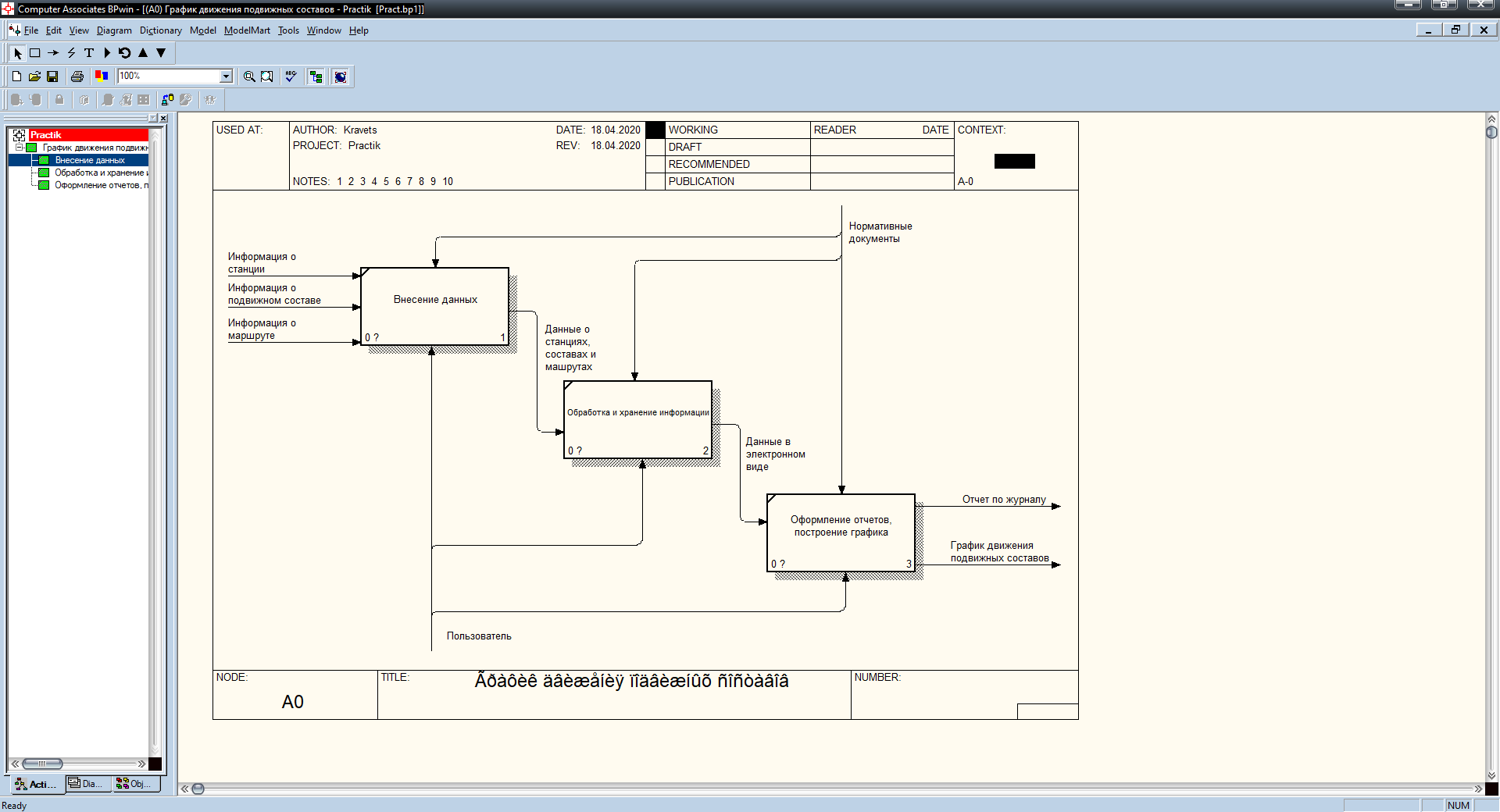


Рисунок Б.2 – Диаграмма декомпозиции

**Приложение В**

*(обязательное)*

**Текст программы**

unit Glav;

interface

uses

Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,

Dialogs, Menus, Grids, DBGrids, DB, IBStoredProc, IBCustomDataSet,

IBQuery, IBDatabase, StdCtrls, Buttons, DBTables, RpCon, RpConDS,

RpDefine, RpRave;

type

TGlavn = class(TForm)

IBDatabase1: TIBDatabase;

IBTransaction1: TIBTransaction;

IBQuery1: TIBQuery;

DataSource1: TDataSource;

DBGrid1: TDBGrid;

MainMenu1: TMainMenu;

N1: TMenuItem;

N2: TMenuItem;

N3: TMenuItem;

N4: TMenuItem;

BitBtn1: TBitBtn;

BitBtn2: TBitBtn;

BitBtn3: TBitBtn;

BitBtn4: TBitBtn;

BitBtn5: TBitBtn;

BitBtn6: TBitBtn;

BitBtn7: TBitBtn;

BitBtn8: TBitBtn;

BitBtn9: TBitBtn;

Label1: TLabel;

DBGrid3: TDBGrid;

IBQuery2: TIBQuery;

DataSource2: TDataSource;

IBQuery3: TIBQuery;

DataSource3: TDataSource;

IBQuery3NOM\_PP: TIntegerField;

IBQuery3DATA\_S: TDateField;

IBQuery3DATA\_PO: TDateField;

IBQuery3NOM\_SOSTAVA: TIntegerField;

IBQuery3NOM\_STANCII: TIntegerField;

IBQuery3STATUS: TSmallintField;

IBQuery2NOM\_PP: TIntegerField;

IBQuery2NAME: TIBStringField;

IBQuery2SVOI\_ARENDA: TSmallintField;

IBQuery2COLOR: TIntegerField;

IBQuery1NOM\_PP: TIntegerField;

IBQuery1NAME: TIBStringField;

IBQuery1DOR\_UPRAV: TSmallintField;

IBQuery1ADRESS: TIBStringField;

IBQuery3Name\_sostava: TStringField;

IBQuery3Name\_stancii: TStringField;

DBGrid2: TDBGrid;

DataSourceTemp: TDataSource;

IBQueryTemp: TIBQuery;

BitBtn10: TBitBtn;

RvProject3: TRvProject;

RvDataSetConnection3: TRvDataSetConnection;

procedure BitBtn1Click(Sender: TObject);

procedure BitBtn2Click(Sender: TObject);

procedure BitBtn3Click(Sender: TObject);

procedure FormShow(Sender: TObject);

procedure BitBtn5Click(Sender: TObject);

procedure BitBtn6Click(Sender: TObject);

procedure BitBtn7Click(Sender: TObject);

procedure BitBtn8Click(Sender: TObject);

procedure BitBtn9Click(Sender: TObject);

procedure DBGrid4DrawDataCell(Sender: TObject; const Rect: TRect;

Field: TField; State: TGridDrawState);

procedure DBGrid3DrawDataCell(Sender: TObject; const Rect: TRect;

Field: TField; State: TGridDrawState);

procedure DBGrid1DrawDataCell(Sender: TObject; const Rect: TRect;

Field: TField; State: TGridDrawState);

procedure BitBtn10Click(Sender: TObject);

procedure N2Click(Sender: TObject);

procedure N4Click(Sender: TObject);

procedure BitBtn4Click(Sender: TObject);

private

{ Private declarations }

public

{ Public declarations }

end;

var

Glavn: TGlavn;

implementation

uses Stancii\_dob, Sostav\_dob, Jurnal\_dob, Stancii\_red, Sostav\_red,

Jurnal\_red, Stancii\_poisk, Sostav\_poisk, Jurnal\_poisk, Stancii\_fil,

Sostav\_fil, Jurnal\_fil, Help, Graph\_p;

{$R \*.dfm}

procedure TGlavn.BitBtn1Click(Sender: TObject);

begin

DBGrid1.visible:=true;

DBGrid2.visible:=false;

DBGrid3.visible:=false;

BitBtn10.Visible:=false;

label1.Caption:='Ñòàíöèè';

end;

procedure TGlavn.BitBtn2Click(Sender: TObject);

begin

DBGrid1.visible:=false;

DBGrid2.visible:=true;

DBGrid3.visible:=false;

BitBtn10.Visible:=false;

label1.Caption:='Ïîäâèæíîé ñîñòàâ';

DBGrid2.Repaint;

end;

procedure TGlavn.BitBtn3Click(Sender: TObject);

begin

DBGrid1.visible:=false;

DBGrid2.visible:=false;

DBGrid3.visible:=true;

BitBtn10.Visible:=true;

label1.Caption:='Æóðíàë';

end;

procedure TGlavn.FormShow(Sender: TObject);

begin

IBQuery1.Active:=false;

IBQuery1.SQL.Clear;

IBQuery1.SQL.Add('select \* from Stancii');

IBQuery1.Active:=true;

IBQuery2.Active:=false;

IBQuery2.SQL.Clear;

IBQuery2.SQL.Add('select \* from Podvijnoi\_sostav');

IBQuery2.Active:=true;

IBQuery3.Active:=false;

IBQuery3.SQL.Clear;

IBQuery3.SQL.Add('select \* from Jurnal');

IBQuery3.Active:=true;

end;

procedure TGlavn.BitBtn5Click(Sender: TObject);

var

sql:string;

begin

if DBGrid1.visible=true then

Stancii\_d.show

else if DBGrid2.Visible=True then Sostav\_d.show

else

begin

IBQueryTemp.SQL.Clear;

SQL:='SELECT NOM\_PP,NAME FROM STANCII';

IBQueryTemp.SQL.Add(SQL);

IBQueryTemp.Open;

if IBQueryTemp.RecordCount>0 then

begin

Jurnal\_d.ComboBoxStations.Items.Clear;

While not IBQueryTemp.Eof do

begin

Jurnal\_d.ComboBoxStations.Items.Add(IBQueryTemp.Fields[1].AsString);

Jurnal\_d.ComboBox3.Items.Add(IBQueryTemp.Fields[0].AsString);

IBQueryTemp.Next;

End;

end;

IBQueryTemp.SQL.Clear;

SQL:='SELECT NOM\_PP,NAME FROM PODVIJNOI\_SOSTAV';

IBQueryTemp.SQL.Add(SQL);

IBQueryTemp.Open;

if IBQueryTemp.RecordCount>0 then

begin

Jurnal\_d.ComboBoxSostav.Items.Clear;

While not IBQueryTemp.Eof do

begin

Jurnal\_d.ComboBoxSostav.Items.Add(IBQueryTemp.Fields[1].AsString);

Jurnal\_d.ComboBox2.Items.Add(IBQueryTemp.Fields[0].AsString);

IBQueryTemp.Next;

End;

end;

Jurnal\_d.show;

end;

end;

procedure TGlavn.BitBtn6Click(Sender: TObject);

var sql:string;

begin

if DBGrid1.visible=true then

begin

Stancii\_r.show;

Stancii\_r.Edit1.text:=IBquery1.fieldvalues['Name'];

Stancii\_r.Edit2.text:=IBquery1.fieldvalues['Adress'];

Stancii\_r.Edit3.text:=IBquery1.fieldvalues['Nom\_pp'];

Stancii\_r.combobox1.itemindex:=IBquery1.fieldvalues['Dor\_uprav'];

end

else if DBGrid2.Visible=True then begin

Sostav\_r.show;

Sostav\_r.Edit1.text:=IBquery2.fieldvalues['Name'];

Sostav\_r.colorbox1.ItemIndex:=IBquery2.fieldvalues['Color'];

Sostav\_r.Edit2.text:=IBquery2.fieldvalues['Nom\_pp'];

Sostav\_r.combobox1.itemindex:=IBquery2.fieldvalues['Svoi\_arenda'];

end

else

begin

IBQueryTemp.SQL.Clear;

SQL:='SELECT NOM\_PP,NAME FROM STANCII';

IBQueryTemp.SQL.Add(SQL);

IBQueryTemp.Open;

if IBQueryTemp.RecordCount>0 then

begin

Jurnal\_r.ComboBoxStations.Items.Clear;

While not IBQueryTemp.Eof do

begin

Jurnal\_r.ComboBoxStations.Items.Add(IBQueryTemp.Fields[1].AsString);

Jurnal\_r.ComboBox3.Items.Add(IBQueryTemp.Fields[0].AsString);

IBQueryTemp.Next;

End;

end;

IBQueryTemp.SQL.Clear;

SQL:='SELECT NOM\_PP,NAME FROM PODVIJNOI\_SOSTAV';

IBQueryTemp.SQL.Add(SQL);

IBQueryTemp.Open;

if IBQueryTemp.RecordCount>0 then

begin

Jurnal\_r.ComboBoxSostav.Items.Clear;

While not IBQueryTemp.Eof do

begin

Jurnal\_r.ComboBoxSostav.Items.Add(IBQueryTemp.Fields[1].AsString);

Jurnal\_r.ComboBox2.Items.Add(IBQueryTemp.Fields[0].AsString);

IBQueryTemp.Next;

End;

end;

Jurnal\_r.show;

Jurnal\_r.Combobox2.text:=IBquery3.fieldvalues['Nom\_sostava'];

Jurnal\_r.datetimepicker1.date:=IBquery3.fieldvalues['Data\_s'];

Jurnal\_r.datetimepicker2.date:=IBquery3.fieldvalues['Data\_po'];

Jurnal\_r.Combobox3.text:=IBquery3.fieldvalues['Nom\_stancii'];

Jurnal\_r.ComboboxSostav.text:=IBquery3.fieldvalues['Name\_sostava'];

Jurnal\_r.ComboboxStations.text:=IBquery3.fieldvalues['Name\_stancii'];

Jurnal\_r.Edit3.text:=IBquery3.fieldvalues['Nom\_pp'];

Jurnal\_r.combobox1.itemindex:=IBquery3.fieldvalues['Status'];

end;

end;

procedure TGlavn.BitBtn7Click(Sender: TObject);

begin

if DBGrid1.visible=true then Stancii\_p.show

else if DBGrid2.Visible=True then Sostav\_p.show

else Jurnal\_p.show;

end;

procedure TGlavn.BitBtn8Click(Sender: TObject);

begin

if DBGrid1.visible=true then Stancii\_f.show

else if DBGrid2.Visible=True then Sostav\_f.show

else Jurnal\_f.show;

end;

procedure TGlavn.BitBtn9Click(Sender: TObject);

begin

if DBGrid1.visible=true then

begin

IBQueryTemp.Close;

IBQueryTemp.SQL.Clear;

IBQueryTemp.SQL.Add('DELETE from Stancii where Nom\_pp='+IntToStr(IBQuery1.fieldbyname('Nom\_pp').asInteger));

IBQueryTemp.ExecSQL;

IBQuery1.Active:=false;

IBQuery1.Active:=true;

end

else if DBGrid2.Visible=True then

begin

IBQueryTemp.Close;

IBQueryTemp.SQL.Clear;

IBQueryTemp.SQL.Add('DELETE from Podvijnoi\_sostav where Nom\_pp='+IntToStr(IBQuery2.fieldbyname('Nom\_pp').asInteger));

IBQueryTemp.ExecSQL;

IBQuery2.Active:=false;

IBQuery2.Active:=true;

end

else

begin

IBQueryTemp.Close;

IBQueryTemp.SQL.Clear;

IBQueryTemp.SQL.Add('DELETE from Jurnal where Nom\_pp='+IntToStr(IBQuery3.fieldbyname('Nom\_pp').asInteger));

IBQueryTemp.ExecSQL;

IBQuery3.Active:=false;

IBQuery3.Active:=true;

end;

end;

procedure TGlavn.DBGrid4DrawDataCell(Sender: TObject; const Rect: TRect;

Field: TField; State: TGridDrawState);

var

img:TBitmap;

begin

if Field.DisplayName = 'Öâåò' then

begin

with (sender as tdbgrid).canvas do

begin

Img := TBitmap.Create;

Brush.Color := field.AsInteger;

FillRect(Rect);

Draw(round((Rect.Left + Rect.Right - Img.Width) / 2), Rect.Top, Img);

Img.Destroy;

end;

end;

if Field.DisplayName = 'Ñâîé/àðåíäà' then

begin

with (sender as tdbgrid).canvas do

begin

fillrect(rect);

CASE field.AsInteger OF

0:textout(rect.left+2,rect.top+3,'Ñâîé');

1:textout(rect.left+2,rect.top+3,'Àðåíäà');

End;

end;

end;

end;

procedure TGlavn.DBGrid3DrawDataCell(Sender: TObject; const Rect: TRect;

Field: TField; State: TGridDrawState);

begin

if Field.DisplayName = 'Ñòàòóñ' then

begin

with (sender as tdbgrid).canvas do

begin

fillrect(rect);

CASE field.AsInteger OF

0:textout(rect.left+2,rect.top+3,'Çàãðóæàåòñÿ');

1:textout(rect.left+2,rect.top+3,'Ðàçãðóæàåòñÿ');

2:textout(rect.left+2,rect.top+3,'Ïðîñòàèâàåò');

3:textout(rect.left+2,rect.top+3,'Äâèæåòñÿ');

End;

end;

end;

end;

procedure TGlavn.DBGrid1DrawDataCell(Sender: TObject; const Rect: TRect;

Field: TField; State: TGridDrawState);

begin

if Field.DisplayName = 'Äîðîæíîå óïðàâëåíèå' then

begin

with (sender as tdbgrid).canvas do

begin

fillrect(rect);

CASE field.AsInteger OF

0:textout(rect.left+2,rect.top+3,'Ñîáñòâåííàÿ');

1:textout(rect.left+2,rect.top+3,'×óæàÿ');

End;

end;

end;

end;

procedure TGlavn.BitBtn10Click(Sender: TObject);

begin

RvProject3.Execute;

end;

procedure TGlavn.N2Click(Sender: TObject);

begin

Glavn.Close;

end;

procedure TGlavn.N4Click(Sender: TObject);

begin

Spravka.Show;

end;

procedure TGlavn.BitBtn4Click(Sender: TObject);

begin

Graph.Show;

end;

end.

unit Stancii\_dob;

interface

uses

Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,

Dialogs, StdCtrls, Buttons;

type

TStancii\_d = class(TForm)

Edit1: TEdit;

Label1: TLabel;

Label2: TLabel;

Label3: TLabel;

ComboBox1: TComboBox;

BitBtn1: TBitBtn;

Edit2: TEdit;

BitBtn2: TBitBtn;

procedure BitBtn1Click(Sender: TObject);

procedure ComboBox1KeyPress(Sender: TObject; var Key: Char);

procedure BitBtn2Click(Sender: TObject);

private

{ Private declarations }

public

{ Public declarations }

end;

var

Stancii\_d: TStancii\_d;

implementation

uses Glav;

{$R \*.dfm}

procedure TStancii\_d.BitBtn1Click(Sender: TObject);

begin

Glavn.IBQuery1.Active:=false;

Glavn.IBQuery1.SQL.Clear;

Glavn.IBQuery1.SQL.Add('Insert into Stancii (Name,Dor\_uprav,Adress) values ('+#39+edit1.Text+#39+','+IntToStr(combobox1.itemindex)+','+#39+edit2.text+#39+')');

Glavn.IBQuery1.ExecSQL;

Glavn.IBQuery1.SQL.Clear;

Glavn.IBQuery1.SQL.Add('select \* from Stancii');

Glavn.IBQuery1.Active:=true;

Stancii\_d.Close;

Edit1.text:=''; Edit2.text:='';

end;

procedure TStancii\_d.ComboBox1KeyPress(Sender: TObject; var Key: Char);

begin

key:=Chr(0);

end;

procedure TStancii\_d.BitBtn2Click(Sender: TObject);

begin

Stancii\_d.Close;

end;

end.

unit Stancii\_red;

interface

uses

Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,

Dialogs, StdCtrls, Buttons;

type

TStancii\_r = class(TForm)

Edit1: TEdit;

Label1: TLabel;

Label2: TLabel;

Label3: TLabel;

ComboBox1: TComboBox;

BitBtn1: TBitBtn;

Edit2: TEdit;

Edit3: TEdit;

BitBtn2: TBitBtn;

procedure BitBtn1Click(Sender: TObject);

procedure ComboBox1KeyPress(Sender: TObject; var Key: Char);

procedure BitBtn2Click(Sender: TObject);

private

{ Private declarations }

public

{ Public declarations }

end;

var

Stancii\_r: TStancii\_r;

implementation

uses Glav, Stancii\_dob;

{$R \*.dfm}

procedure TStancii\_r.BitBtn1Click(Sender: TObject);

begin

Glavn.IBQuery1.Active:=false;

Glavn.IBQuery1.SQL.Clear;

Glavn.IBQuery1.SQL.Add('Update Stancii Set Name = '#39+edit1.Text+#39',Dor\_uprav = '+IntToStr(combobox1.itemindex)+',Adress = '#39+edit2.text+#39' where Nom\_pp = '+edit3.text+'');

Glavn.IBQuery1.ExecSQL;

Glavn.IBQuery1.SQL.Clear;

Glavn.IBQuery1.SQL.Add('select \* from Stancii');

Glavn.IBQuery1.Active:=true;

Stancii\_r.Close;

end;

procedure TStancii\_r.ComboBox1KeyPress(Sender: TObject; var Key: Char);

begin

key:=Chr(0);

end;

procedure TStancii\_r.BitBtn2Click(Sender: TObject);

begin

Stancii\_r.close;

end;

end.

unit Stancii\_poisk;

interface

uses

Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,

Dialogs, StdCtrls, Buttons;

type

TStancii\_p = class(TForm)

Edit1: TEdit;

ComboBox1: TComboBox;

Label1: TLabel;

Label2: TLabel;

BitBtn1: TBitBtn;

BitBtn2: TBitBtn;

procedure BitBtn1Click(Sender: TObject);

procedure BitBtn2Click(Sender: TObject);

procedure ComboBox1KeyPress(Sender: TObject; var Key: Char);

private

{ Private declarations }

public

{ Public declarations }

end;

var

Stancii\_p: TStancii\_p;

implementation

uses Glav;

{$R \*.dfm}

procedure TStancii\_p.BitBtn1Click(Sender: TObject);

var s:string;

begin

if combobox1.text = 'Íàçâàíèå' then s:='Name'

else if combobox1.text = 'Äîðîæíîå óïðàâëåíèå' then s:='Dor\_uprav'

else if combobox1.text = 'Àäðåñ' then s:='Adress';

Glavn.IBQuery1.Active:=false;

Glavn.IBQuery1.SQL.Clear;

Glavn.IBQuery1.SQL.Add('select \* from Stancii where '+s+' ='+#39+edit1.Text+#39);

Glavn.IBQuery1.Active:=true;

end;

procedure TStancii\_p.BitBtn2Click(Sender: TObject);

begin

Glavn.IBQuery1.Active:=false;

Glavn.IBQuery1.SQL.Clear;

Glavn.IBQuery1.SQL.Add('select \* from Stancii');

Glavn.IBQuery1.Active:=true;

end;

procedure TStancii\_p.ComboBox1KeyPress(Sender: TObject; var Key: Char);

begin

key:=Chr(0);

end;

end.

unit Stancii\_fil;

interface

uses

Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,

Dialogs, StdCtrls, Buttons;

type

TStancii\_f = class(TForm)

Edit1: TEdit;

ComboBox1: TComboBox;

Label1: TLabel;

Label2: TLabel;

BitBtn1: TBitBtn;

BitBtn2: TBitBtn;

procedure BitBtn2Click(Sender: TObject);

procedure BitBtn1Click(Sender: TObject);

procedure ComboBox1KeyPress(Sender: TObject; var Key: Char);

private

{ Private declarations }

public

{ Public declarations }

end;

var

Stancii\_f: TStancii\_f;

implementation

uses Glav;

{$R \*.dfm}

procedure TStancii\_f.BitBtn2Click(Sender: TObject);

begin

Glavn.IBQuery1.Active:=false;

Glavn.IBQuery1.SQL.Clear;

Glavn.IBQuery1.SQL.Add('select \* from Stancii');

Glavn.IBQuery1.Active:=true;

end;

procedure TStancii\_f.BitBtn1Click(Sender: TObject);

var s:string;

begin

if combobox1.text = 'Íàçâàíèå' then s:='Name'

else if combobox1.text = 'Äîðîæíîå óïðàâëåíèå' then s:='Dor\_uprav'

else if combobox1.text = 'Àäðåñ' then s:='Adress';

Glavn.IBQuery1.Active:=false;

Glavn.IBQuery1.SQL.Clear;

Glavn.IBQuery1.SQL.Add('select \* from Stancii where '+s+' like '#39+edit1.Text+#39);

Glavn.IBQuery1.Active:=true;

end;

procedure TStancii\_f.ComboBox1KeyPress(Sender: TObject; var Key: Char);

begin

key:=Chr(0);

end;

unit Sostav\_dob;

interface

uses

Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,

Dialogs, ExtCtrls, StdCtrls, Buttons;

type

TSostav\_d = class(TForm)

Edit1: TEdit;

Label1: TLabel;

Label2: TLabel;

Label3: TLabel;

ComboBox1: TComboBox;

BitBtn1: TBitBtn;

ColorBox1: TColorBox;

BitBtn2: TBitBtn;

procedure BitBtn1Click(Sender: TObject);

procedure ComboBox1KeyPress(Sender: TObject; var Key: Char);

procedure BitBtn2Click(Sender: TObject);

private

{ Private declarations }

public

{ Public declarations }

end;

var

Sostav\_d: TSostav\_d;

implementation

uses Glav;

{$R \*.dfm}

procedure TSostav\_d.BitBtn1Click(Sender: TObject);

begin

Glavn.IBQuery2.Active:=false;

Glavn.IBQuery2.SQL.Clear;

Glavn.IBQuery2.SQL.Add('Insert into Podvijnoi\_sostav (Name,Svoi\_arenda,Color) values ('+#39+edit1.Text+#39+','+IntToStr(combobox1.itemindex)+','+IntToStr(colorbox1.selected)+')');

Glavn.IBQuery2.ExecSQL;

Glavn.IBQuery2.SQL.Clear;

Glavn.IBQuery2.SQL.Add('select \* from Podvijnoi\_sostav');

Glavn.IBQuery2.Active:=true;

Sostav\_d.Close;

Edit1.text:='';

end;

procedure TSostav\_d.ComboBox1KeyPress(Sender: TObject; var Key: Char);

begin

key:=Chr(0);

end;

procedure TSostav\_d.BitBtn2Click(Sender: TObject);

begin

Sostav\_d.close;

end;

end.

unit Sostav\_fil;

interface

uses

Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,

Dialogs, StdCtrls, Buttons;

type

TSostav\_f = class(TForm)

Label1: TLabel;

Label2: TLabel;

Edit1: TEdit;

ComboBox1: TComboBox;

BitBtn1: TBitBtn;

BitBtn2: TBitBtn;

procedure BitBtn1Click(Sender: TObject);

procedure BitBtn2Click(Sender: TObject);

procedure ComboBox1KeyPress(Sender: TObject; var Key: Char);

private

{ Private declarations }

public

{ Public declarations }

end;

var

Sostav\_f: TSostav\_f;

implementation

uses Glav;

{$R \*.dfm}

procedure TSostav\_f.BitBtn1Click(Sender: TObject);

var s:string;

begin

if combobox1.text = 'Íàçâàíèå' then s:='Name'

else if combobox1.text = 'Ñâîé/àðåíäà' then s:='Svoi\_arenda';

Glavn.IBQuery2.Active:=false;

Glavn.IBQuery2.SQL.Clear;

Glavn.IBQuery2.SQL.Add('select \* from Podvijnoi\_sostav where '+s+' like '+#39+edit1.Text+#39);

Glavn.IBQuery2.Active:=true;

end;

procedure TSostav\_f.BitBtn2Click(Sender: TObject);

begin

Glavn.IBQuery2.Active:=false;

Glavn.IBQuery2.SQL.Clear;

Glavn.IBQuery2.SQL.Add('select \* from Podvijnoi\_sostav');

Glavn.IBQuery2.Active:=true;

end;

procedure TSostav\_f.ComboBox1KeyPress(Sender: TObject; var Key: Char);

begin

key:=Chr(0);

end;

end.

unit Sostav\_poisk;

interface

uses

Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,

Dialogs, StdCtrls, Buttons;

type

TSostav\_p = class(TForm)

Edit1: TEdit;

ComboBox1: TComboBox;

Label1: TLabel;

Label2: TLabel;

BitBtn1: TBitBtn;

BitBtn2: TBitBtn;

procedure BitBtn2Click(Sender: TObject);

procedure BitBtn1Click(Sender: TObject);

procedure ComboBox1KeyPress(Sender: TObject; var Key: Char);

private

{ Private declarations }

public

{ Public declarations }

end;

var

Sostav\_p: TSostav\_p;

implementation

uses Glav;

{$R \*.dfm}

procedure TSostav\_p.BitBtn2Click(Sender: TObject);

begin

Glavn.IBQuery2.Active:=false;

Glavn.IBQuery2.SQL.Clear;

Glavn.IBQuery2.SQL.Add('select \* from Podvijnoi\_sostav');

Glavn.IBQuery2.Active:=true;

end;

procedure TSostav\_p.BitBtn1Click(Sender: TObject);

var s:string;

begin

if combobox1.text = 'Íàçâàíèå' then s:='Name'

else if combobox1.text = 'Ñâîé/àðåíäà' then s:='Svoi\_arenda';

Glavn.IBQuery2.Active:=false;

Glavn.IBQuery2.SQL.Clear;

Glavn.IBQuery2.SQL.Add('select \* from Podvijnoi\_sostav where '+s+' ='+#39+edit1.Text+#39);

Glavn.IBQuery2.Active:=true;

end;

procedure TSostav\_p.ComboBox1KeyPress(Sender: TObject; var Key: Char);

begin

key:=Chr(0);

end;

end.

unit Sostav\_red;

interface

uses

Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,

Dialogs, ExtCtrls, StdCtrls, Buttons;

type

TSostav\_r = class(TForm)

Edit1: TEdit;

Label1: TLabel;

Label2: TLabel;

Label3: TLabel;

ComboBox1: TComboBox;

BitBtn1: TBitBtn;

ColorBox1: TColorBox;

Edit2: TEdit;

BitBtn2: TBitBtn;

procedure BitBtn1Click(Sender: TObject);

procedure ComboBox1KeyPress(Sender: TObject; var Key: Char);

procedure BitBtn2Click(Sender: TObject);

private

{ Private declarations }

public

{ Public declarations }

end;

var

Sostav\_r: TSostav\_r;

implementation

uses Glav;

{$R \*.dfm}

procedure TSostav\_r.BitBtn1Click(Sender: TObject);

begin

Glavn.IBQuery2.Active:=false;

Glavn.IBQuery2.SQL.Clear;

Glavn.IBQuery2.SQL.Add('Update Podvijnoi\_sostav Set Name = '#39+edit1.Text+#39',Svoi\_arenda = '+IntToStr(combobox1.itemindex)+',Color = '+Inttostr(colorbox1.selected)+' where Nom\_pp = '+edit2.text+'');

Glavn.IBQuery2.ExecSQL;

Glavn.IBQuery2.SQL.Clear;

Glavn.IBQuery2.SQL.Add('select \* from Podvijnoi\_sostav');

Glavn.IBQuery2.Active:=true;

Sostav\_r.Close;

end;

procedure TSostav\_r.ComboBox1KeyPress(Sender: TObject; var Key: Char);

begin

key:=Chr(0);

end;

procedure TSostav\_r.BitBtn2Click(Sender: TObject);

begin

Sostav\_r.close;

end;

end.

unit Help;

interface

uses

Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,

Dialogs, StdCtrls;

type

TSpravka = class(TForm)

Memo1: TMemo;

private

{ Private declarations }

public

{ Public declarations }

end;

var

Spravka: TSpravka;

implementation

{$R \*.dfm}

end.

unit Filtr;

interface

uses

Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,

Dialogs, Grids, DBGrids;

type

TSved = class(TForm)

DBGrid1: TDBGrid;

private

{ Private declarations }

public

{ Public declarations }

end;

var

Sved: TSved;

implementation

uses Graph\_p;

{$R \*.dfm}

end.

unit Jurnal\_dob;

interface

uses

Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,

Dialogs, Grids, StdCtrls, Buttons, ComCtrls, DB, IBCustomDataSet,

IBStoredProc, IBQuery, DBGrids;

type

TJurnal\_d = class(TForm)

Label1: TLabel;

DateTimePicker1: TDateTimePicker;

Label2: TLabel;

DateTimePicker2: TDateTimePicker;

Label3: TLabel;

Label4: TLabel;

Label5: TLabel;

ComboBox1: TComboBox;

BitBtn1: TBitBtn;

DBGrid1: TDBGrid;

BitBtn2: TBitBtn;

BitBtn3: TBitBtn;

BitBtn4: TBitBtn;

BitBtn5: TBitBtn;

BitBtn6: TBitBtn;

DataSource1: TDataSource;

IBQuery1: TIBQuery;

IBStoredProc1: TIBStoredProc;

IBQuery1NOM\_PP: TIntegerField;

IBQuery1NOM\_MAT: TIntegerField;

IBQuery1NOM\_JUR: TIntegerField;

IBQuery1MASSA: TFloatField;

IBQuery2: TIBQuery;

IBQuery2NOM\_PP: TIntegerField;

IBQuery2NAME: TIBStringField;

IBQuery1Name\_mat: TStringField;

ComboBoxStations: TComboBox;

ComboBoxSostav: TComboBox;

Combobox2: TComboBox;

ComboBox3: TComboBox;

BitBtn7: TBitBtn;

procedure BitBtn1Click(Sender: TObject);

procedure ComboBox1KeyPress(Sender: TObject; var Key: Char);

procedure ComboBoxSostavChange(Sender: TObject);

procedure ComboBoxStationsChange(Sender: TObject);

procedure BitBtn7Click(Sender: TObject);

procedure ComboBoxSostavKeyPress(Sender: TObject; var Key: Char);

procedure ComboBoxStationsKeyPress(Sender: TObject; var Key: Char);

private

{ Private declarations }

public

{ Public declarations }

end;

var

Jurnal\_d: TJurnal\_d;

implementation

uses Glav, Gruz2\_dob, Gruz2\_red;

{$R \*.dfm}

procedure TJurnal\_d.BitBtn1Click(Sender: TObject);

begin

Glavn.IBQuery3.Active:=false;

Glavn.IBQuery3.SQL.Clear;

Glavn.IBQuery3.SQL.Add('Insert into Jurnal (Data\_s,Data\_po,Nom\_sostava, Nom\_stancii, status) values ('+#39+DatetoStr(Datetimepicker1.date)+#39+','+#39+DatetoStr(Datetimepicker2.date)+#39+','+ComboBox2.Text+','+ComboBox3.Text+','+IntToStr(combobox1.itemindex)+')');

Glavn.IBQuery3.ExecSQL;

Glavn.IBQuery3.SQL.Clear;

Glavn.IBQuery3.SQL.Add('select \* from Jurnal');

Glavn.IBQuery3.Active:=true;

Jurnal\_d.Close;

ComboBoxStations.text:=''; ComboBoxSostav.text:='';

end;

procedure TJurnal\_d.ComboBox1KeyPress(Sender: TObject; var Key: Char);

begin

key:=Chr(0);

end;

procedure TJurnal\_d.ComboBoxSostavChange(Sender: TObject);

var i:integer;

begin

i:=comboboxsostav.ItemIndex;

combobox2.ItemIndex:=i;

end;

procedure TJurnal\_d.ComboBoxStationsChange(Sender: TObject);

var i:integer;

begin

i:=comboboxstations.ItemIndex;

combobox3.ItemIndex:=i;

end;

procedure TJurnal\_d.BitBtn7Click(Sender: TObject);

begin

Jurnal\_d.close;

end;

procedure TJurnal\_d.ComboBoxSostavKeyPress(Sender: TObject; var Key: Char);

begin

key:=Chr(0);

end;

procedure TJurnal\_d.ComboBoxStationsKeyPress(Sender: TObject;

var Key: Char);

begin

key:=Chr(0);

end;

end.

unit Jurnal\_fil;

interface

uses

Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,

Dialogs, StdCtrls, Buttons;

type

TJurnal\_f = class(TForm)

Label1: TLabel;

Label2: TLabel;

Edit1: TEdit;

ComboBox1: TComboBox;

BitBtn1: TBitBtn;

BitBtn2: TBitBtn;

procedure BitBtn1Click(Sender: TObject);

procedure BitBtn2Click(Sender: TObject);

procedure ComboBox1KeyPress(Sender: TObject; var Key: Char);

private

{ Private declarations }

public

{ Public declarations }

end;

var

Jurnal\_f: TJurnal\_f;

implementation

uses Glav;

{$R \*.dfm}

procedure TJurnal\_f.BitBtn2Click(Sender: TObject);

begin

Glavn.IBQuery3.Active:=false;

Glavn.IBQuery3.SQL.Clear;

Glavn.IBQuery3.SQL.Add('select \* from Jurnal');

Glavn.IBQuery3.Active:=true;

end;

procedure TJurnal\_f.BitBtn1Click(Sender: TObject);

var s:string;

begin

if combobox1.text = 'Äàòà ñ' then s:='Data\_s'

else if combobox1.text = 'Äàòà ïî' then s:='Data\_po'

else if combobox1.text = 'Íîìåð ñîñòàâà' then s:='Nom\_sostava'

else if combobox1.text = 'Íîìåð ñòàíöèè' then s:='Nom\_stancii'

else if combobox1.text = 'Ñòàòóñ' then s:='Status';

Glavn.IBQuery3.Active:=false;

Glavn.IBQuery3.SQL.Clear;

Glavn.IBQuery3.SQL.Add('select \* from Jurnal where '+s+' like '+#39+edit1.Text+#39);

Glavn.IBQuery3.Active:=true;

end;

procedure TJurnal\_f.ComboBox1KeyPress(Sender: TObject; var Key: Char);

begin

key:=Chr(0);

end;

end.

unit Jurnal\_poisk;

interface

uses

Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,

Dialogs, StdCtrls, Buttons;

type

TJurnal\_p = class(TForm)

Edit1: TEdit;

ComboBox1: TComboBox;

Label1: TLabel;

Label2: TLabel;

BitBtn1: TBitBtn;

BitBtn2: TBitBtn;

procedure BitBtn1Click(Sender: TObject);

procedure BitBtn2Click(Sender: TObject);

procedure ComboBox1KeyPress(Sender: TObject; var Key: Char);

private

{ Private declarations }

public

{ Public declarations }

end;

var

Jurnal\_p: TJurnal\_p;

implementation

uses Glav;

{$R \*.dfm}

procedure TJurnal\_p.BitBtn1Click(Sender: TObject);

var s:string;

begin

if combobox1.text = 'Äàòà ñ' then s:='Data\_s'

else if combobox1.text = 'Äàòà ïî' then s:='Data\_po'

else if combobox1.text = 'Íîìåð ñîñòàâà' then s:='Nom\_sostava'

else if combobox1.text = 'Íîìåð ñòàíöèè' then s:='Nom\_stancii'

else if combobox1.text = 'Ñòàòóñ' then s:='Status';

Glavn.IBQuery3.Active:=false;

Glavn.IBQuery3.SQL.Clear;

Glavn.IBQuery3.SQL.Add('select \* from Jurnal where '+s+' ='+#39+edit1.Text+#39);

Glavn.IBQuery3.Active:=true;

end;

procedure TJurnal\_p.BitBtn2Click(Sender: TObject);

begin

Glavn.IBQuery3.Active:=false;

Glavn.IBQuery3.SQL.Clear;

Glavn.IBQuery3.SQL.Add('select \* from Jurnal');

Glavn.IBQuery3.Active:=true;

end;

procedure TJurnal\_p.ComboBox1KeyPress(Sender: TObject; var Key: Char);

begin

key:=Chr(0);

end;

end.

unit Jurnal\_red;

interface

uses

Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,

Dialogs, Grids, StdCtrls, Buttons, ComCtrls;

type

TJurnal\_r = class(TForm)

Label1: TLabel;

DateTimePicker1: TDateTimePicker;

Label2: TLabel;

DateTimePicker2: TDateTimePicker;

Label3: TLabel;

Label4: TLabel;

Label5: TLabel;

ComboBox1: TComboBox;

BitBtn1: TBitBtn;

Edit3: TEdit;

BitBtn2: TBitBtn;

ComboBoxStations: TComboBox;

ComboBoxSostav: TComboBox;

Combobox2: TComboBox;

ComboBox3: TComboBox;

procedure BitBtn1Click(Sender: TObject);

procedure ComboBox1KeyPress(Sender: TObject; var Key: Char);

procedure BitBtn2Click(Sender: TObject);

procedure ComboBoxSostavChange(Sender: TObject);

procedure ComboBoxStationsChange(Sender: TObject);

procedure ComboBoxStationsKeyPress(Sender: TObject; var Key: Char);

procedure ComboBoxSostavKeyPress(Sender: TObject; var Key: Char);

private

{ Private declarations }

public

{ Public declarations }

end;

var

Jurnal\_r: TJurnal\_r;

implementation

uses Glav;

{$R \*.dfm}

procedure TJurnal\_r.BitBtn1Click(Sender: TObject);

begin

Glavn.IBQuery3.Active:=false;

Glavn.IBQuery3.SQL.Clear;

Glavn.IBQuery3.SQL.Add('Update Jurnal Set Data\_s = '+#39+DatetoStr(Datetimepicker1.date)+#39+',Data\_po = '+#39+DatetoStr(Datetimepicker2.date)+#39+',Nom\_sostava = '+combobox2.text+',Nom\_stancii = '+combobox3.text+',status = '+IntToStr(combobox1.itemindex)+' where Nom\_pp = '+edit3.text+'');

Glavn.IBQuery3.ExecSQL;

Glavn.IBQuery3.SQL.Clear;

Glavn.IBQuery3.SQL.Add('select \* from Jurnal');

Glavn.IBQuery3.Active:=true;

Jurnal\_r.Close;

end;

procedure TJurnal\_r.ComboBox1KeyPress(Sender: TObject; var Key: Char);

begin

key:=Chr(0);

end;

procedure TJurnal\_r.BitBtn2Click(Sender: TObject);

begin

Jurnal\_r.close;

end;

unit Graph\_p;

interface

uses

Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,

Dialogs, ExtCtrls, TeeProcs, TeEngine, Chart, StdCtrls, Series, DB,

IBCustomDataSet, IBTable, DbChart, Glav, IBQuery, ComCtrls, Buttons,

Grids, DBGrids, ExtDlgs;

type

TGraph = class(TForm)

DataSource1: TDataSource;

IBQuery1: TIBQuery;

Label1: TLabel;

BitBtn1: TBitBtn;

DateTimePicker1: TDateTimePicker;

DateTimePicker2: TDateTimePicker;

DBGrid1: TDBGrid;

IBQueryTemp: TIBQuery;

DataSource2: TDataSource;

Label2: TLabel;

DBGrid2: TDBGrid;

Button1: TButton;

IBQueryColor: TIBQuery;

ComboBox2: TComboBox;

Button2: TButton;

Button3: TButton;

SaveDialog1: TSaveDialog;

SavePictureDialog1: TSavePictureDialog;

procedure BitBtn1Click(Sender: TObject);

procedure FormPaint(Sender: TObject);

procedure Button1Click(Sender: TObject);

procedure DBGrid2DrawDataCell(Sender: TObject; const Rect: TRect;

Field: TField; State: TGridDrawState);

procedure Button2Click(Sender: TObject);

procedure Button3Click(Sender: TObject);

private

{ Private declarations }

public

{ Public declarations }

end;

var

Graph: TGraph;

implementation

uses Math, Filtr;

{$R \*.dfm}

function idtonum(yss:integer):integer;

var i:integer;

s:string;

begin

for i:=0 to Graph.ComboBox2.Items.Count-1 do

begin

s:=Graph.ComboBox2.Items[i];

If yss=StrToInt(s) then

begin

idtonum:=i+1;

break;

end;

end;

end;

procedure TGraph.BitBtn1Click(Sender: TObject);

var i,h,l,h1,h2,xs,xe,ys,fl,rec,m,n,c,x,y,j,t,k,p,p1,f,start,yn,xn,ps:integer;

d1,d2,s:string;

G:array of integer;

begin

IBQueryTemp.Active:=false;

IBQueryTemp.SQL.Clear;

IBQueryTemp.SQL.Add('Delete from Graph');

IBQueryTemp.Active:=true;

IBQuery1.Active:=false;

IBQuery1.SQL.Clear;

IBQuery1.SQL.Add('select \* from Jurnal where Data\_s >= '+#39+datetostr(datetimepicker1.date)+#39+' and Data\_po <= '+#39+datetostr(datetimepicker2.date)+#39);

IBQuery1.Active:=true;

p:=1;

IBQuery1.First;

While not IBQuery1.Eof do

begin

d1:=IBQuery1.Fields[1].asstring;

d2:=IBQuery1.Fields[2].asstring;

m:=IBQuery1.Fields[3].asinteger;

n:=IBquery1.Fields[4].asinteger;

IBQueryColor.Active:=false;

IBQueryColor.SQL.Clear;

IBQueryColor.SQL.Add('select Color from Podvijnoi\_sostav where Nom\_pp = '+inttostr(m));

IBQueryColor.Active:=true;

c:=ibquerycolor.Fields[0].asinteger;

IBQueryColor.Next;

IBQueryTemp.Active:=false;

IBQueryTemp.SQL.Clear;

IBQueryTemp.SQL.Add('Insert into Graph (Data\_ss,Data\_ppo,ID\_sost,ID\_stan,Colorr) values ('+#39+d1+#39+','+#39+d2+#39+','+inttostr(m)+','+inttostr(n)+','+inttostr(c)+')');

IBQueryTemp.ExecSQL;

IBQueryTemp.SQL.Clear;

IBQueryTemp.SQL.Add('select \* from Graph');

IBQueryTemp.Active:=true;

IBQuery1.Next;

p:=p+1;

End;

t:=1;

SetLength(G, p+1);

While not IBQueryTemp.Eof do

begin

G[t]:=IBqueryTemp.Fields[3].asinteger;

IBQueryTemp.Next;

t:=t+1;

end;

p1:=1;

i:=1;

while i<=t do

begin

j:=i+1;

while j<=t do

begin

if G[j]=G[i] then

begin

for k:=j to t-1 do

G[k]:=G[k+1];

t:=t-1;

end else j:=j+1;

end;

i:=i+1;

p1:=p1+1;

end;

p1:=p1-1;

for I := 1 to p1-2 do

for j := 1 to p1-2 do

if G[j] > G[j+1] then

begin

x := G[j+1];

G[j+1] := G[j];

G[j] := x;

end;

l:=1;

for i:=1 to p1-1 do

begin

ComboBox2.Items.Add(inttostr(G[i]));

l:=l+1;

end;

if FormatDateTime('mm', DateTimePicker1.Date) = '01' then

begin

label2.caption:='ßíâàðü';

f:=31;

end;

if FormatDateTime('mm', DateTimePicker1.Date) = '02' then

begin

label2.caption:='Ôåâðàëü';

f:=29;

end;

if FormatDateTime('mm', DateTimePicker1.Date) = '03' then

begin

label2.caption:='Ìàðò';

f:=31;

end;

if FormatDateTime('mm', DateTimePicker1.Date) = '04' then

begin

label2.caption:='Àïðåëü';

f:=30;

end;

if FormatDateTime('mm', DateTimePicker1.Date) = '05' then

begin

label2.caption:='Ìàé';

f:=31;

end;

if FormatDateTime('mm', DateTimePicker1.Date) = '06' then

begin

label2.caption:='Èþíü';

f:=30;

end;

if FormatDateTime('mm', DateTimePicker1.Date) = '07' then

begin

label2.caption:='Èþëü';

f:=31;

end;

if FormatDateTime('mm', DateTimePicker1.Date) = '08' then

begin

label2.caption:='Àâãóñò';

f:=31;

end;

if FormatDateTime('mm', DateTimePicker1.Date) = '09' then

begin

label2.caption:='Ñåíòÿáðü';

f:=30;

end;

if FormatDateTime('mm', DateTimePicker1.Date) = '10' then

begin

label2.caption:='Îêòÿáðü';

f:=31;

end;

if FormatDateTime('mm', DateTimePicker1.Date) = '11' then

begin

label2.caption:='Íîÿáðü';

f:=30;

end;

if FormatDateTime('mm', DateTimePicker1.Date) = '12' then

begin

label2.caption:='Äåêàáðü';

f:=31;

end;

PatBlt(Graph.Canvas.Handle, 0, 0, Graph.ClientWidth, Graph.ClientHeight, WHITENESS);

Canvas.Pen.Color := clblack;

xn:=Round(graph.ClientWidth/f-5);

yn:=Round(graph.ClientHeight/p1-4);

Canvas.MoveTo(100,30);

Canvas.LineTo(100,800);

Canvas.MoveTo(100,30);

Canvas.LineTo(1840,30);

Canvas.MoveTo(0,800);

Canvas.LineTo(1840,800);

start:=100;

for i:=1 to f do

begin

Canvas.MoveTo(start+xn,30);

Canvas.LineTo(start+xn,800);

start:=start+xn;

Canvas.TextOut (Round(start-xn/2),35,IntToStr(i));

end;

start:=graph.ClientHeight-90;

for i:=1 to p1-1 do

begin

Canvas.MoveTo(0,start-yn);

Canvas.LineTo(1840,start-yn);

if Glavn.ibquery3.Locate('Nom\_stancii', G[i],[loCaseInsensitive, loPartialKey]) then s:=Glavn.IBQuery3.Fields[4].asstring;

Canvas.TextOut (10,Round(start-yn/2),s);

start:=start-yn;

end;

ibQueryTemp.Active:=false;

ibQueryTemp.SQL.Clear;

ibQueryTemp.SQL.Add('Select \* from graph order By id\_sost asc, data\_ss asc');

ibQueryTemp.Active:=True;

rec:=0;

ibQueryTemp.First;

for I := 1 to p1-2 do

for j := 1 to p1-2 do

if G[j] < G[j+1] then

begin

x := G[j+1];

G[j+1] := G[j];

G[j] := x;

end;

h:=0;

h1:=0;

h2:=20;

x:=100;

y:=790;

repeat

ps:=ibQueryTemp.FieldByName('ID\_sost').asinteger;

xs:=StrToInt(FormatDateTime('d',ibQueryTemp.FieldByName('data\_ss').AsDateTime));

Canvas.MoveTo(xs\*xn+70,graph.ClientHeight-ys\*yn);

fl:=0;

while ibQueryTemp.FieldByName('id\_sost').asinteger = ps do

begin

Canvas.Pen.Color:=ibQueryTemp.fieldbyname('Colorr').AsInteger;

xs:=StrToInt(FormatDateTime('d',ibQueryTemp.FieldByName('data\_ss').AsDateTime));

xe:=StrToInt(FormatDateTime('d',ibQueryTemp.FieldByName('data\_ppo').AsDateTime));

t:=ibQueryTemp.FieldByName('id\_stan').asinteger;

ys:=idtonum(t);

if fl<>0 then Canvas.LineTo(xs\*xn+70,graph.ClientHeight-ys\*yn+h);

Canvas.MoveTo(xs\*xn+70,graph.ClientHeight-ys\*yn+h);

Canvas.LineTo(xe\*xn+70,graph.ClientHeight-ys\*yn+h);

ibQueryTemp.Next;

fl:=1;

rec:=rec+1;

h:=h+1;

if rec=ibQueryTemp.RecordCount then Break;

end;

if y>=860 then

begin

h1:=h1+200;

h2:=20;

y:=790;

end;

x:=x+h1;

y:=y+h2;

if Glavn.ibquery3.Locate('Nom\_sostava', ps,[loCaseInsensitive, loPartialKey]) then s:=Glavn.IBQuery3.Fieldbyname('Name\_sostava').asstring;

Canvas.MoveTo(x+70,y+5);

Canvas.LineTo(x+150,y+5);

Canvas.TextOut(x,y,s);

until rec=ibQueryTemp.RecordCount

end;

procedure TGraph.FormPaint(Sender: TObject);

begin

Canvas.Pen.Color := clblack;

Canvas.MoveTo(100,0);

Canvas.LineTo(100,800);

Canvas.MoveTo(0,800);

Canvas.LineTo(1900,800);

Canvas.MoveTo(0,400);

Canvas.LineTo(1900,400);

Canvas.TextOut(30, 380, 'Ñòàíöèè');

Canvas.TextOut(900, 30, 'Äàòà');

Canvas.MoveTo(950,0);

Canvas.LineTo(950,800);

end;

procedure TGraph.Button1Click(Sender: TObject);

begin

IBQueryTemp.Active:=false;

IBQueryTemp.SQL.Clear;

IBQueryTemp.SQL.Add('Delete from Graph');

IBQueryTemp.Active:=true;

PatBlt(Graph.Canvas.Handle, 0, 0, Graph.ClientWidth, Graph.ClientHeight, WHITENESS);

Canvas.Pen.Color := clblack;

Canvas.MoveTo(100,0);

Canvas.LineTo(100,800);

Canvas.MoveTo(0,800);

Canvas.LineTo(1900,800);

Canvas.MoveTo(0,400);

Canvas.LineTo(1900,400);

Canvas.TextOut(30, 380, 'Ñòàíöèè');

Canvas.TextOut(900, 30, 'Äàòà');

Canvas.MoveTo(950,0);

Canvas.LineTo(950,800);

end;

procedure TGraph.DBGrid2DrawDataCell(Sender: TObject; const Rect: TRect;

Field: TField; State: TGridDrawState);

var

img:TBitmap;

begin

if Field.displayName = 'COLORR' then

begin

with (sender as tdbgrid).canvas do

begin

Img := TBitmap.Create;

Brush.Color := field.AsInteger;

FillRect(Rect);

Draw(round((Rect.Left + Rect.Right - Img.Width) / 2), Rect.Top, Img);

Img.Destroy;

end;

end;

end;

procedure TGraph.Button2Click(Sender: TObject);

begin

Sved.Show;

end;

procedure TGraph.Button3Click(Sender: TObject);

var b: TBitmap;

i: integer;

begin

b := TBitmap.Create();

b.Width := Graph.ClientWidth;

b.Height := 860;

b.Canvas.CopyRect(Rect(0, 0, b.Width, b.Height), Graph.Canvas,Graph.ClientRect);

if not SavePictureDialog1.Execute then Exit;

b.SaveToFile(SavePictureDialog1.FileName);

b.Free;

end;

end.